ModelArts

用户指南（Standard）

文档版本 01

发布日期 2025-01-07

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 ModelArts Standard 使用流程

本章节旨在帮助您了解ModelArts Standard的基本使用方法，帮助您快速上手ModelArts服务。

面向熟悉代码编写和调测，熟悉常见AI引擎的开发者，ModelArts不仅提供了在线代码开发环境，还提供了从数据准备、模型训练、模型管理到模型部署上线的端到端开发流程（即AI全流程开发）。

本文档介绍了如何在ModelArts管理控制台完成AI开发，如果您习惯使用API或者SDK进行开发，建议查看《**ModelArts SDK参考**》和《**ModelArts API参考**》获取帮助。

使用AI全流程开发的端到端示例，请参见《**快速入门**》和《**最佳实践**》。

Standard 使用场景介绍

ModelArts Standard是面向AI开发者的一站式开发平台，提供了简洁易用的管理控制台，包含自动学习、数据管理、开发环境、模型训练、模型管理、部署上线等端到端的AI开发工具链。

- Standard的自动学习可以帮助用户零代码构建AI模型。自动学习功能根据标注数据自动设计模型、自动调参、自动训练、自动压缩和部署模型。开发者无需专业的开发基础和编码能力，只需上传数据，通过自动学习界面引导和简单操作即可完成模型训练和部署。具体请参见**自动学习简介**。
- Standard的Workflow是一套低代码的AI开发流水线工具，覆盖数据标注、数据处理、模型开发、训练、模型评估、部署上线等步骤，提供可视化的工作流运行方式。具体请参见**什么是Workflow**。
- Standard的开发环境Notebook提供了云上JupyterLab环境和本地IDE插件，方便用户编写训练推理代码，并使用云上资源进行代码调试。具体请参见**Notebook使用场景**。
- Standard的模型训练功能提供了界面化的训练调试环境和生产环境，用户可以使用自己的数据和算法，利用Standard提供的计算资源开展模型训练。具体请参见**使用ModelArts Standard训练模型**。
- Standard的推理部署功能提供了界面化的推理部署生产环境，AI模型开发完成后，在Standard中可以纳管AI模型并快速部署为推理服务，您可以进行在线推理预测，也可以通过调用API把AI推理能力集成到自己的IT平台。具体请参见**推理部署使用场景**。

Standard 使用流程说明

ModelArts Standard平台提供了从数据准备到模型部署的AI全流程开发，兼容开发者的使用习惯，支持多种引擎和用户场景，使用自由度较高。针对AI开发的每个环节，Standard功能使用相对自由，您可以根据实际需要选择其中的环节。下文介绍使用ModelArts平台，从准备数据到完成模型开发上线的全流程。

图 1-1 Standard 使用流程

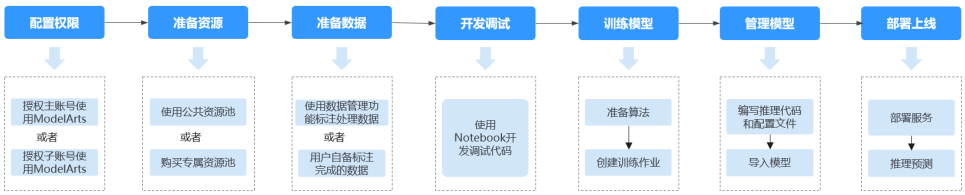


表 1-1 使用流程说明

流程	子任务	说明	详细指导
配置权限	配置ModelArts委托授权	ModelArts功能使用还依赖与其他云服务的交互，需要先配置委托授权，允许ModelArts访问相关依赖服务。	快速配置ModelArts委托授权
创建OBS桶（可选）	创建OBS桶用于ModelArts存储数据	由于ModelArts本身没有数据存储的功能，使用Modelarts Standard进行AI开发过程中的输入数据、输出数据、中间缓存数据都可以在OBS桶中进行存储、读取。因此，建议您在使用ModelArts之前先创建一个OBS桶。 创建OBS桶可以提前完成，也可以在后续使用到时再创建。	创建OBS桶用于ModelArts存储数据
准备资源（可选）	创建Standard专属资源池	ModelArts Standard支持公共资源池和专属资源池。 公共资源池：方便快捷，无需创建，创建训练推理任务时直接选择即可。此时，忽略此步骤。 专属资源池：用户独占资源，需要先购买创建。如果使用专属资源池，需要完成此步骤。	创建Standard专属资源池

流程	子任务	说明	详细指导
准备数据（可选）	创建数据集	ModelArts Standard提供了数据管理功能，用户可以在ModelArts Standard中创建数据集，用于管理、预处理、标注数据。 如果用户已经准备了可用于训练的数据，直接上传到OBS即可，无需使用数据管理功能。	**创建数据集** **标注数据** **发布数据集**
开发调试	创建Notebook	创建一个Notebook作为开发环境，用于调试训练和推理代码。 建议先在开发环境中调试完成训练代码后再创建生产训练作业。	**创建Notebook实例**
训练模型	准备算法	创建训练作业前需要先准备算法，可以订阅AI Gallery中的算法，也可以使用用户自己的算法。	**准备算法**
	创建训练作业	创建一个训练作业，选择可用的数据集版本，并使用前面编写完成的训练脚本。训练完成后，将生成模型并存储至OBS中。	**创建训练作业**
管理模型	编写推理代码和配置文件	针对您生成的模型，建议您按照ModelArts提供的模型包规范，编写推理代码和配置文件，并将推理代码和配置文件存储至训练输出位置。	**模型包规范介绍**
	创建模型	将训练完成的模型导入至ModelArts创建为模型，方便将模型部署上线。	**创建模型**
部署模型	部署服务	ModelArts支持将模型部署为在线服务、批量服务和边缘服务。	• **部署为在线服务** • **部署为批量服务**
	访问服务	服务部署完成后，针对在线服务和边缘服务，您可以访问并使用服务，针对批量服务，您可以查看其预测结果。	• **访问在线服务** • **查看批量服务预测结果**

2 ModelArts Standard 准备工作

2.1 配置 ModelArts Standard 访问授权

2.1.1 快速配置 ModelArts 委托授权

场景描述

为了完成AI计算的各种操作，AI平台ModelArts在任务执行过程中需要访问用户的其他服务，典型的就是训练过程中，需要访问OBS读取用户的训练数据。在这个过程中，就出现了ModelArts“代表”用户去访问其他云服务的情形。从安全角度出发，ModelArts代表用户访问任何云服务之前，均需要先获得用户的授权，而这个动作就是一个“委托”的过程。用户授权ModelArts再代表自己访问特定的云服务，以完成其在ModelArts平台上执行的AI计算任务。

ModelArts提供了一键式自动授权功能，用户可以在ModelArts的权限管理功能中，快速完成委托授权，由ModelArts为用户自动创建委托并配置到ModelArts服务中。

一键式自动授权方式为保证使用业务过程中有足够的权限，基于依赖服务的预置系统策略指定授权范围，创建的委托的权限比较大，基本覆盖了依赖服务的全部权限。如果您需要对委托授权的权限范围进行精确控制，请使用定制化委托授权。更多权限控制的内容请参见**权限管理**章节。

本章节主要介绍一键式自动授权方式。一键式自动授权方式支持给IAM子用户、联邦用户（虚拟IAM用户）、委托用户和所有用户授权。

约束与限制

- 华为云账号
  - 只有华为云账号可以使用委托授权，可以为当前账号授权，也可以为当前账号下的所有IAM用户授权。
  - 多个IAM用户或账号，可使用同一个委托。

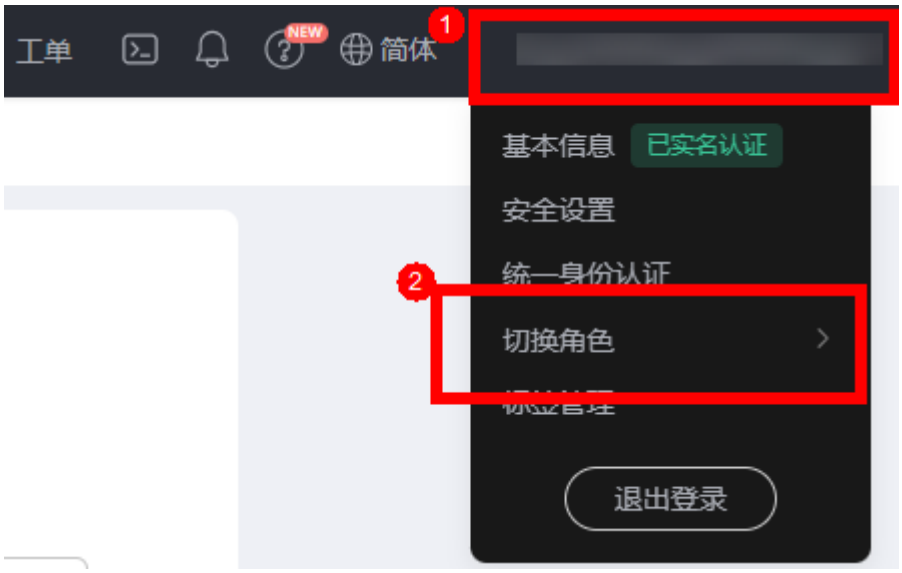
  - 一个账号下，最多可创建50个委托。
  - 对于首次使用ModelArts的新用户，请直接新增委托即可。一般用户新增普通用户权限即可满足使用要求。如果有精细化权限管理的需求，可以自定义权限按需设置。
- IAM用户
  - 如果已获得委托授权，则可以在权限管理页面中查看到已获得的委托授权信息。
  - 如果未获得委托授权，当打开“访问授权”页面时，ModelArts会提醒您当前用户未配置授权，需联系此IAM用户的管理员账号进行委托授权。

添加授权

1. 登录ModelArts管理控制台，在左侧导航栏选择“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”，进入“访问授权”配置页面，根据参数说明进行配置。

表 2-1 参数说明

参数	说明
“授权对象类型”	包括IAM子用户、联邦用户、委托用户和所有用户。 • IAM子用户：由主账号在IAM中创建的用户，是服务的使用人员，具有独立的身份凭证（密码和访问密钥），根据账号授予的权限使用资源。IAM子用户相关介绍请参见**IAM用户介绍**。 • 联邦用户：又称企业虚拟用户。联邦用户相关介绍请参见**联邦身份认证**。 • 委托用户：IAM中创建的一个委托。IAM创建委托相关介绍请参见**创建委托**。 • 所有用户：该选项表示会将委托的权限授权到当前账号下的所有子账号、包括未来创建的子账号，授权范围较大，需谨慎使用。个人用户选择“所有用户”即可。

参数	说明
“授权对象”	“授权对象类型”选择“所有用户”时不涉及此参数。 • IAM子用户：选择指定的IAM子用户，给指定的IAM子用户配置委托授权。 **图 2-1** 选择 IAM 子用户  • 联邦用户：输入联邦用户的用户名或用户ID。 **图 2-2** 选择联邦用户  • 委托用户：选择委托名称。使用账号A创建一个权限委托，在此处将该委托授权给账号B拥有的委托。在使用账号B登录控制台时，可以在控制台右上角的个人账号切换角色到账号A，使用账号A的委托权限。 **图 2-3** 委托用户切换角色 
“委托选择”	• 已有委托：列表中如果已有委托选项，则直接选择一个可用的委托为上述选择的用户授权。单击委托名称查看该委托的权限详情。 • 新增委托：如果没有委托可选，可以在新增委托中创建委托权限。对于首次使用ModelArts的用户，需要新增委托。
“新增委托 > 委托名称”	系统自动创建委托名称，用户可以手动修改。

参数	说明
“新增委托 > 授权方式”	• 角色授权：IAM最初提供的一种根据用户的工作职能定义权限的粗粒度授权机制。该机制以服务为粒度，提供有限的服务相关角色用于授权。由于华为云各服务之间存在业务依赖关系，因此给用户授予角色时，可能需要一并授予依赖的其他角色，才能正确完成业务。角色并不能满足用户对精细化授权的要求，无法完全达到企业对权限最小化的安全管控要求。 • 策略授权：IAM最新提供的一种细粒度授权的能力，可以精确到具体服务的操作、资源以及请求条件等。基于策略的授权是一种更加灵活的授权方式，能够满足企业对权限最小化的安全管控要求。 角色与策略相关介绍请参考**权限基本概念**。
“新增委托 > 权限配置 > 普通用户”	普通用户包括用户使用ModelArts完成AI开发的所有必要功能权限，如数据的访问、训练作业的创建和管理等。一般用户选择此项即可。 可以单击“查看权限列表”，查看普通用户权限。
“新增委托 > 权限配置 > 自定义”	如用户有精细化权限管理的需求，可使用自定义模式灵活按需配置ModelArts创建的委托权限。可以根据实际需要在权限列表中勾选要配置的权限。

3. 然后勾选“我已经详细阅读并同意《ModelArts服务声明》”，单击“创建”，即可完成委托配置。

查看授权的权限列表

用户可以在“权限管理”页面的授权列表中，查看已经配置的委托授权内容。单击授权内容列的“查看权限”，可以查看该授权的权限详情。

图 2-4 查看权限

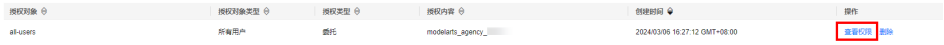
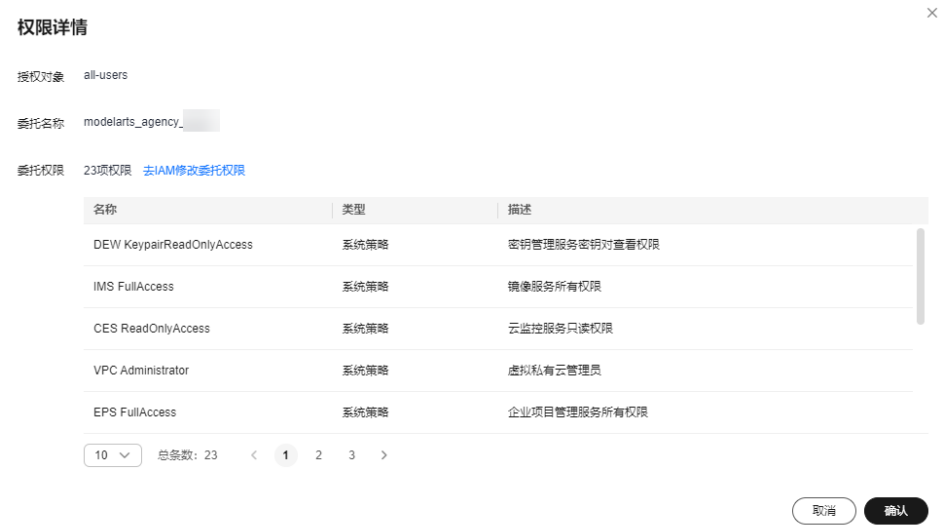


图 2-5 普通用户权限列表



修改授权权限范围

说明

请注意，修改授权权限范围会导致运行中的作业因委托缺失而运行失败，请谨慎操作。

1. 在查看授权详情时，如果想要修改授权范围，可以在权限详情页单击“去IAM修改委托权限”。

图 2-6 去 IAM 修改委托权限

2. 进入IAM控制台的委托页面。找到对应的委托信息，修改该委托的基本信息，主要是持续时间。“持续时间”可以选择永久、1天，或者自定义天数，例如 30 天。

图 2-7 手动创建的委托

3. 在授权记录页面单击“授权”，勾选要配置的策略，单击下一步设置最小授权范围，单击确定，完成授权修改。

   设置最小授权范围时，可以选择指定的区域，也可以选择所有区域，即不设置范围。

删除授权

为了更好的管理您的授权，您可以删除某一IAM用户的授权，也可批量清空所有用户的授权。

说明

请注意，删除授权操作会导致运行中的作业因缺失委托而运行失败，请谨慎操作。

- **删除某一用户的授权**

  在“权限管理”页面，展示当前账号下为其IAM用户配置的授权列表，针对某一用户，您可以单击“操作”列的“删除”，输入“DELETE”后单击“确认”，可删除此用户的授权。删除生效后，此用户将无法继续使用ModelArts的相关功能。

- **批量清空所有授权**

  在“权限管理”页面，单击授权列表上方的“清空授权”，输入“DELETE”后单击“确认”，可删除当前账号下的所有授权。删除生效后，此账号及其所有IAM子用户将无法继续使用ModelArts的相关功能。

常见问题

1. 首次使用ModelArts如何配置授权?

   直接选择“新增委托”中的“普通用户”权限即可，普通用户包括用户使用ModelArts完成AI开发的所有必要功能权限，如数据的访问、训练作业的创建和管理等。一般用户选择此项即可。

2. 如何获取访问密钥AK/SK?

   如果在其他功能（例如PyCharmtoolKit/VSCode登录，访问在线服务等）中使用到访问密钥AK/SK认证，获取AK/SK方式请参考**如何获取访问密钥**章节。

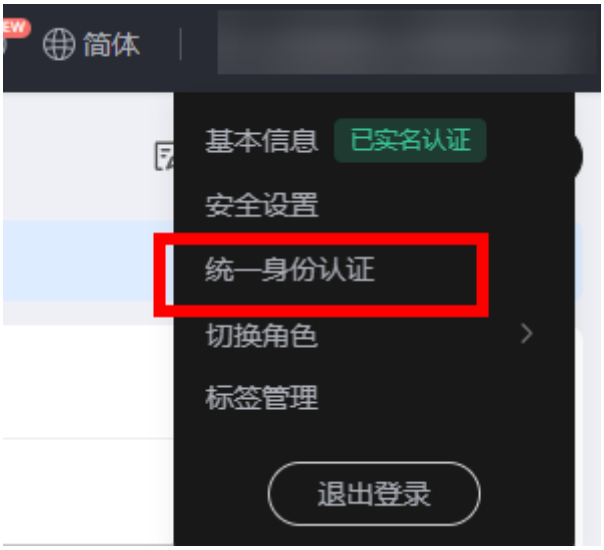
3. 如何删除已有委托列表下面的委托名称?

   图 2-8 已有委托

   需要前往统一身份认证服务IAM控制台的委托页面删除。

图 2-9 统一身份认证



4. 进入ModelArts控制台的某个页面时，为什么会提示权限不足?

   可能原因是用户委托权限配置不足或模块能力升级，需要更新授权信息。根据界面操作提示追加授权即可。

2.1.2 创建 IAM 用户并授权使用 ModelArts

快速配置ModelArts委托授权章节中介绍的一键式自动授权方式创建的委托的权限比较大，基本覆盖了依赖服务的全部权限。如果华为云账号已经能满足您的要求，不需要创建独立的IAM用户，您可以跳过本章节，不影响您使用ModelArts服务的其他功能。

ModelArts作为一个完备的AI开发平台，支持用户对其进行细粒度的权限配置，以达到精细化资源、权限管理之目的。这类特性在大型企业用户的使用场景下很常见。如果需要对委托授权的权限范围进行精确控制，参考本章节进行定制化委托授权。

本章节主要介绍如何给IAM用户下的子用户配置更细粒度的权限。

由于ModelArts的使用权限依赖OBS服务的授权，您需要为用户授予OBS的系统权限。

- 如果您需要授予用户ModelArts的操作权限以及依赖服务的操作权限，请参见**配置ModelArts Standard基础操作权限**。
- 如果您需要对用户使用ModelArts的操作权限和依赖服务的权限进行精细化管理，进行自定义策略配置，请参见**创建ModelArts Standard自定义策略**。

前提条件

给用户组授权之前，请您了解用户组可以添加的使用ModelArts及依赖服务的权限，并结合实际需求进行选择，ModelArts支持的系统权限，请参见**表2-2**。

表 2-2 服务授权列表

待授权的服务	授权说明	IAM权限设置	是否必选
ModelArts	授予子用户使用ModelArts服务的权限。 ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子用户配置此权限。	ModelArts CommonOperations	必选
	如果需要给子用户开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。	ModelArts FullAccess	可选 ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。
OBS对象存储服务	授予子用户使用OBS服务的权限。ModelArts的数据集、开发环境、训练作业、模型推理部署均需要**通过OBS进行数据中转**。	OBS OperateAccess	必选
SWR容器镜像仓库	授予子用户使用SWR服务权限。ModelArts的**自定义镜像功能**依赖镜像服务SWR FullAccess权限。	SWR OperateAccess	必选
密钥管理服务	当子用户使用ModelArts **Notebook的SSH远程功能**时，需要配置子用户密钥管理服务的使用权限。	KMS CMKFullAccess	可选
IEF智能边缘平台	授予子用户智能边缘平台使用权限，ModelArts的边缘服务依赖智能边缘平台，要求配置Tenant Administrator权限。	Tenant Administrator	可选
CES云监控	授予子用户使用CES云监控服务的权限。通过CES云监控可以查看ModelArts的在线服务和对应模型负载运行状态的整体情况，并设置监控告警。	CES FullAccess	可选

待授权的服务	授权说明	IAM权限设置	是否必选
SMN消息服务	授予子用户使用SMN消息服务的权限。SMN消息通知服务配合CES监控告警功能一起使用。	SMN FullAccess	可选
VPC虚拟私有云	子用户在创建ModelArts的专属资源池过程中，如果需要开启自定义网络配置，需要配置VPC权限。	VPC FullAccess	可选
SFS弹性文件服务	授予子用户使用SFS服务的权限，ModelArts的专属资源池中可以挂载SFS系统作为开发环境或训练的存储。	SFS Turbo FullAccess SFS FullAccess	可选

配置 ModelArts Standard 基础操作权限

步骤1 创建用户组。

登录IAM管理控制台，单击“用户组>创建用户组”。在“创建用户组”界面，输入“用户组名称”单击“确定”。

步骤2 配置用户组权限。

在用户组列表中，单击步骤1新建的用户组右侧的“授权”，在用户组“授权”页面，您需要配置的权限如下：

1. 配置ModelArts使用权限。在搜索框搜索ModelArts。ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。

   选择说明如下：

   - ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子用户配置此权限。
   - 如果需要给子用户开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。

图 2-10 配置 ModelArts 使用权限

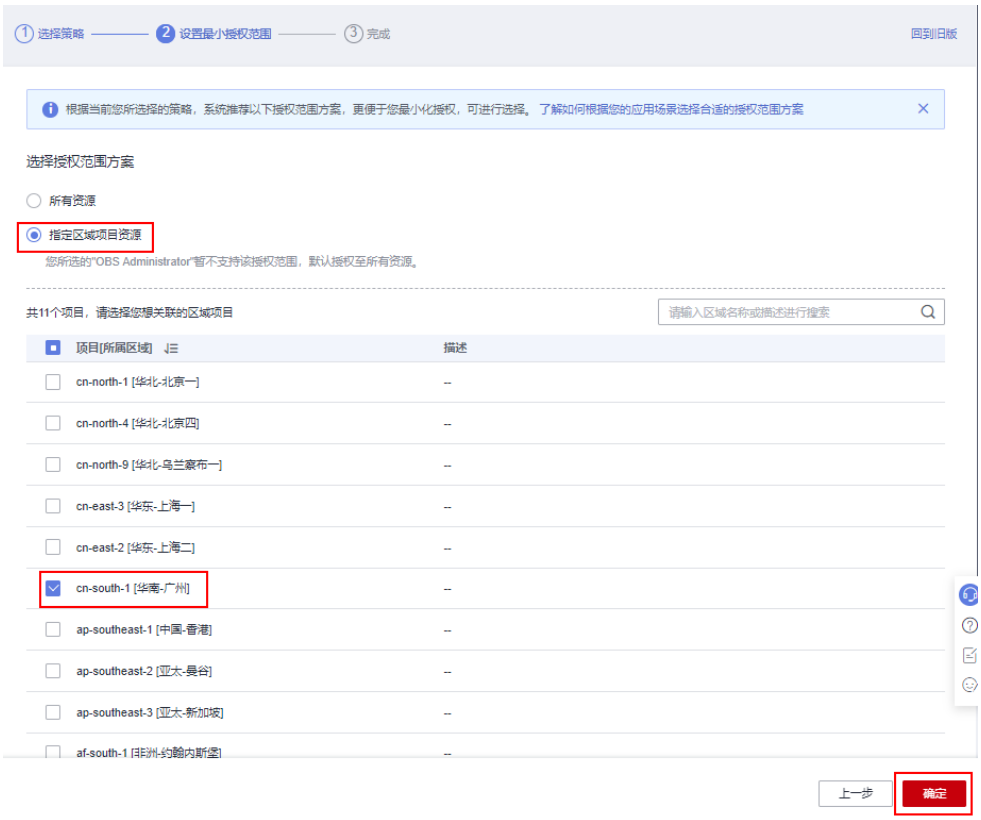
2. 配置其他依赖云服务的使用权限，此处以OBS为例，搜索OBS，勾选“OBS OperateAccess”。ModelArts训练作业中需要依赖OBS作为数据中转站，需要配置OBS的使用权限。

   更多需要配置的云服务权限请参见表2-2，比如SWR等，重复操作此步骤。

3. 再单击“下一步”，设置最小授权范围。单击“指定区域项目资源”，勾选待授权使用的区域，单击“确定”。

   本示例以只允许用户使用“中国-香港”Region的服务为例。

   图 2-11 选择授权的区域范围



4. 提示授权成功，查看授权信息，单击“完成”。此处的授权生效需要15-30分钟。

步骤3 创建子用户账号。在IAM左侧菜单栏中，选择“用户”，单击右上角“创建用户”，在“创建用户”页面中，添加多个用户。请根据界面提示，填写必选参数，然后单击“下一步”。

步骤4 将子用户子账号加入用户组。在“加入用户组”步骤中，选择“用户组”，然后单击“创建用户”。系统将前面设置的多个用户加入用户组中。

步骤5 用户登录并验证权限。

新创建的用户登录控制台，切换至授权区域，验证权限：

- 在“服务列表”中选择ModelArts，进入ModelArts主界面，选择不同类型的专属资源池，在页面单击“创建”，如果无法进行创建（当前权限仅包含ModelArts CommonOperations），表示“ModelArts CommonOperations”已生效。

- 在“服务列表”中选择除ModelArts外（假设当前策略仅包含ModelArts CommonOperations）的任一服务，如果提示权限不足，表示“ModelArts CommonOperations”已生效。
- 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“资产管理>数据集>创建数据”，如果可以成功访问对应的OBS路径，表示OBS权限已生效。
- 参考**表2-2**依次验证其他可选权限。

----结束

创建 ModelArts Standard 自定义策略

如果系统预置的ModelArts权限不满足您的授权要求，或者您需要管理用户更细一步的操作权限，比如操作OBS的操作权限，可以创建自定义策略。

目前IAM支持可视化视图创建自定义策略和JSON视图创建自定义策略，本章节将使用JSON视图方式的策略，以为ModelArts用户授予开发环境的使用权限并且配置ModelArts用户OBS相关的最小化权限项为例，指导您进行自定义策略配置。

涉及到其他更多功能和依赖服务的自定义策略内容参见**ModelArts Standard策略权限管理**章节。

更多关于创建自定义策略操作和参数说明请参见**创建自定义策略**。

说明

如果一个自定义策略中包含多个服务的授权语句，这些服务必须是同一属性，即都是全局级服务或者项目级服务。

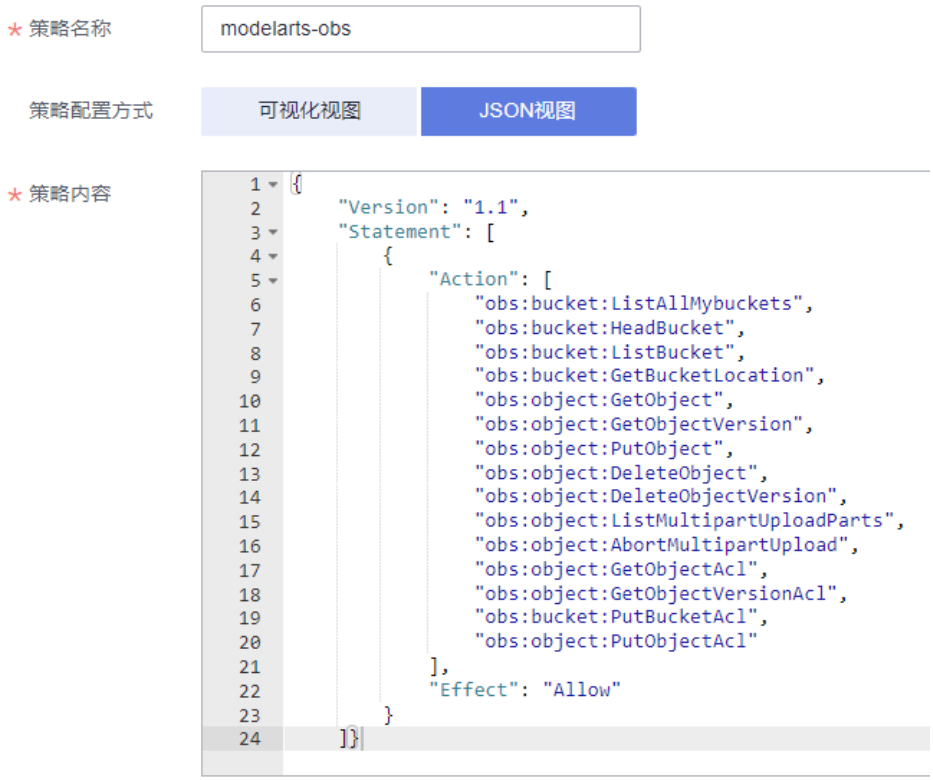
由于OBS为全局服务，ModelArts为项目级服务，所以需要创建两条“作用范围”别为“全局级服务”以及“项目级服务”的自定义策略，然后将两条策略同时授予用户。

1. 创建ModelArts相关OBS的最小化权限的自定义策略。

   登录IAM控制台，在“权限管理>权限”页面，单击“创建自定义策略”。参数配置说明如下：

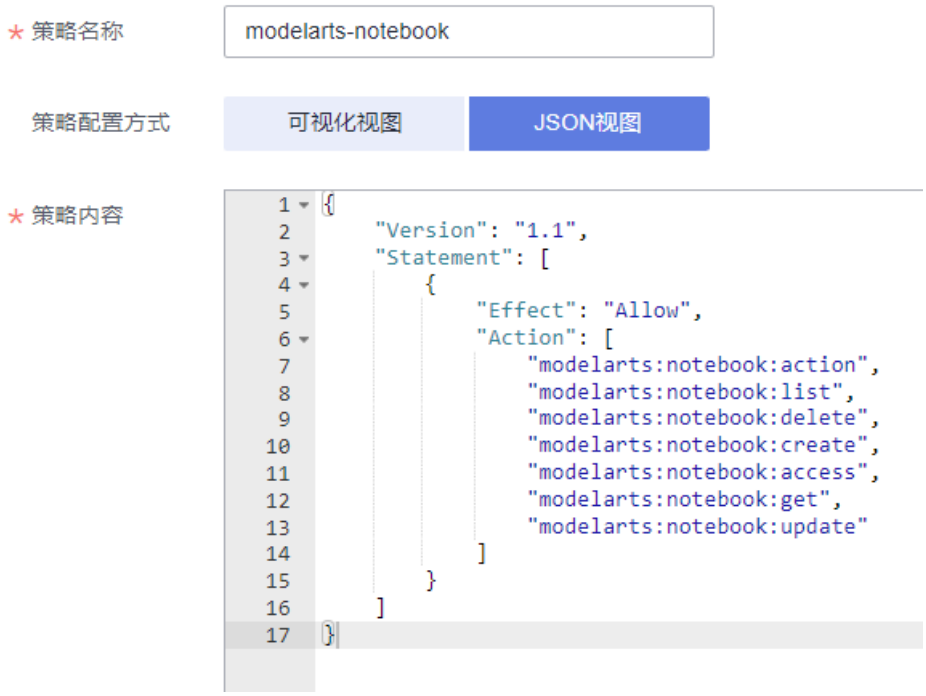
   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts依赖的OBS权限自定义策略样例**。

图 2-12 OBS 相关的最小化权限



2. 创建ModelArts开发环境的使用权限的自定义策略，如图2-13所示。参数配置说明如下：
   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts开发环境使用权限的自定义策略样例**，ModelArts自定义策略中可以添加的授权项（Action）请参见**《ModelArts API参考》>权限策略和授权项**。

图 2-13 开发环境的使用权限



  - 如果您需要对除ModelArts和OBS之外的其他服务授权，IAM支持服务的所有策略请参见**ModelArts Standard策略权限管理**章节。
3. 在IAM控制台创建用户组之后，将步骤**1**中创建的自定义策略授权给该用户组。
4. 在IAM控制台创建用户，并将其加入**3**中创建的用户组。
5. 新创建的用户登录控制台，切换至授权区域，验证权限：
   - 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“资产管理 > 数据集”，如果无法进行创建（当前仅包含开发环境的使用权限），表示仅为ModelArts用户授予开发环境的使用权限已生效。
   - 在“服务列表”中选择除ModelArts，进入ModelArts主界面，单击“开发空间>Notebook>创建”，如果可以成功访问“存储配置”项对应的OBS路径，表示为用户配置的OBS相关权限已生效。

ModelArts 依赖的 OBS 权限自定义策略样例

如下示例为ModelArts依赖OBS服务的最小化权限项，包含OBS桶和OBS对象的权限。授予示例中的权限您可以通过ModelArts正常访问OBS不受限制。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "obs:bucket:ListAllMybuckets",
                "obs:bucket:HeadBucket",
                "obs:bucket:ListBucket",
                "obs:bucket:GetBucketLocation",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:object:PutObject",
                "obs:object:DeleteObject",
                "obs:object:DeleteObjectVersion",
```

```
            "obs:object:ListMultipartUploadParts",
            "obs:object:AbortMultipartUpload",
            "obs:object:GetObjectAcl",
            "obs:object:GetObjectVersionAcl",
            "obs:bucket:PutBucketAcl",
            "obs:object:PutObjectAcl"
        ],
        "Effect": "Allow"
    }
  ]
}
```

ModelArts 开发环境使用权限的自定义策略样例

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "modelarts:notebook:list",
        "modelarts:notebook:create" ,
        "modelarts:notebook:get" ,
        "modelarts:notebook:update" ,
        "modelarts:notebook:delete" ,
        "modelarts:notebook:action" ,
        "modelarts:notebook:access"
      ]
    }
  ]
}
```

相关案例

更多权限配置案例如下，根据实际需要参考。

- **查看所有子账号的Notebook实例**
- **管理员和开发者权限分离**
- **限制用户使用公共资源池**
- **给子用户配置文件夹级的SFS Turbo访问权限**

2.2 创建并管理工作空间

说明

工作空间是白名单功能，如果有试用需求，请提工单申请权限。

背景信息

ModelArts的用户需要为不同的业务目标开发算法、管理和部署模型，此时可以创建多个工作空间，把不同应用开发过程的输出内容划分到不同工作空间中，便于管理和使用。

基于工作空间可以实现资源逻辑隔离、资源配额管理、细粒度鉴权和资源清理能力。工作空间组件可以将ModelArts各类资源整合，以工作空间体现给企业项目管理服务。

工作空间支持3种访问控制：

- PUBLIC：租户（主账号和所有子账号）内部公开访问。

- PRIVATE：仅创建者和主账号可访问。
- INTERNAL：创建者、主账号、指定IAM子账号可访问当授权类型为INTERNAL时需要指定可访问的子账号的账号名，可选择多个。

每个账号每个IAM项目都会分配1个默认工作空间，默认工作空间的访问控制为PUBLIC。

通过工作空间的访问控制能力，可限制仅允许部分人访问对应的工作空间。通过此功能可实现类似如下场景：

- **教育场景**：老师可给每个学生分配1个INTERNAL的工作空间并且限制该工作空间被指定学生访问，这样可使得学生可独立完成在ModelArts上的实验。
- **企业场景**：管理者可创建用于生产任务的工作空间并限制仅让运维人员使用，用于日常调试的工作空间并限制仅让开发人员使用。通过这种方式让不同的企业角色只能在指定工作空间下使用资源。

前提条件

已开通工作空间白名单，并配置了ModelArts基本使用权限，具体请参见**配置ModelArts基本使用权限**。

创建工作空间

1. 登录ModelArts管理控制台。
2. 在左侧导航栏中，选择“工作空间”进入工作空间列表。
3. 单击“创建工作空间”，进入创建页面。

表 2-3 创建工作空间

参数名称	说明
工作空间名称	必填，工作空间的名称。 支持4~64位可见字符，名称可以包含字母、中文、数字、中划线（-）或下划线（_）。
描述	工作空间的简介。支持0~256位字符。
企业项目	必填，选择绑定的企业项目。当没有合适的企业项目时，可以单击“新建企业项目”跳转到企业项目管理页面，创建新的企业项目再绑定。 企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。

参数名称	说明
授权类型	必填，选择工作空间的访问权限。 ● “PUBLIC”：租户（主账号和所有子账号）内部公开访问。 ● “PRIVATE”：仅创建者和主账号可访问。 ● “INTERNAL”：创建者、主账号、指定的子账号可访问。当授权类型为INTERNAL时，需要配置“授权对象类型”和“授权对象”指定可访问的子账号。当“授权对象类型”选择“IAM子用户”时，“授权对象”选择指定的IAM子用户，可选择多个。 当“授权对象类型”选择“联邦用户”时，“授权对象”输入联邦用户的用户名或用户ID，支持配置多个。 当“授权对象类型”选择“委托用户”时，“授权对象”选择委托名称，可选择多个。

4. 工作空间参数配置完成后，单击“立即创建”完成创建任务。

管理工作空间配额

工作空间创建成功后，可以查看配额信息或修改配额值。

1. 在ModelArts管理控制台的左侧导航栏中，选择“工作空间”进入工作空间列表。
2. 在工作空间列表，单击操作列的“配额管理”进入工作空间详情页。
3. 在配额信息页面可以查看工作空间设置的配额值、已用的配额、最后修改时间等配额信息。
4. 单击配额信息右侧的“修改配额”可以修改配额值。配置值的配置说明请参见**表2-4**。

表 2-4 配额信息

配额名称	配额值说明	单位
自动学习（预测分析）训练时长	默认无限制，支持设置1~60000。	分钟
自动学习（图像分类、物体检测、声音分类）训练时长	默认无限制，支持设置1~60000。	分钟
训练作业GPU规格训练时长（单张Pnt1单节点为统计基础单元）	默认无限制，支持设置1~60000。	分钟
训练作业CPU规格训练时长（单核单节点为统计基础单元）	默认无限制，支持设置1~60000。	分钟
可视化作业使用时长	默认无限制，支持设置1~60000。	分钟
开发环境CPU规格使用时长（单核为统计基础单元）	默认无限制，支持设置1~60000。	分钟

配额名称	配额值说明	单位
开发环境GPU规格使用时长（单张Pnt1为统计基础单元）	默认无限制，支持设置1~60000。	分钟
推理服务CPU规格使用时长（单节点为统计基础单元）	默认无限制，支持设置1~60000。	分钟
推理服务GPU规格使用时长（单节点为统计基础单元）	默认无限制，支持设置1~60000。	分钟
训练作业CPU规格训练核数	默认无限制，支持设置1~10000。	核
训练作业GPU规格训练卡数	默认无限制，支持设置1~1000。	卡
训练作业RAM规格训练内存大小	默认无限制，支持设置1~100000。	GB
智能标注GPU规格使用时长	默认无限制，支持设置1~60000。	分钟

5. 工作空间的配额值修改完成后，单击“提交修改”，当“配额值”数据刷新表示修改成功。

修改工作空间

工作空间创建成功后，支持修改信息。

1. 在ModelArts管理控制台的左侧导航栏中，选择“工作空间”进入工作空间列表。
2. 在工作空间列表，单击操作列的“修改”进入修改工作空间页面。

   支持修改工作空间的名称、描述、企业项目和授权类型，各参数说明请参见**表2-4**。
3. 参数修改完成后，单击“立即修改”完成工作空间的修改。

删除工作空间

当不需要使用工作空间时，支持删除工作空间。工作空间删除后会默认清理该工作空间下所有资源。其中，默认工作空间“default”不支持删除。

说明

请注意，删除工作空间将删除该空间下的所有资源，包括已创建的Notebook、训练作业和部署服务，且无法恢复，请谨慎操作。

1. 在ModelArts管理控制台的左侧导航栏中，选择“工作空间”进入工作空间列表。
2. 在工作空间列表，单击操作列的“删除”，在删除工作空间弹窗中确认待删除的工作空间信息以及该工作空间下将被一起删除的资源，确认无误后，输入“DELETE”，单击“确定”，工作空间的状态变为“删除中”，待资源清理完成，该工作空间会从列表删除。

2.3 创建 OBS 桶用于 ModelArts 存储数据

由于ModelArts本身没有数据存储的功能，ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。

AI开发过程中的输入数据、输出数据、中间缓存数据都可以在OBS桶中进行存储、读取。因此，建议您在使用ModelArts之前先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

图 2-14 ModelArts 与 OBS 交互示意

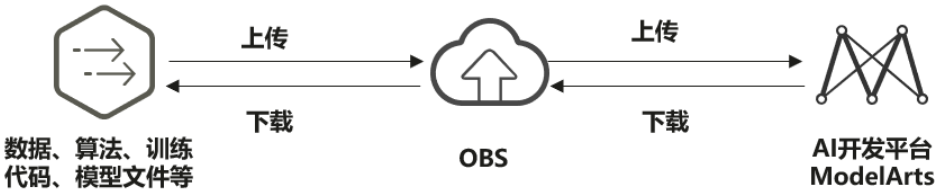


表 2-5 ModelArts 各模块与 OBS 的关系

功能	子任务	ModelArts与OBS的关系
Standard自动学习 Standard Workflow	数据标注	ModelArts标注的数据存储在OBS中。
	自动训练	训练作业结束后，其生成的模型存储在OBS中。
	部署上线	ModelArts将存储在OBS中的模型部署上线为在线服务。
Standard AI全流程开发	数据管理	• 数据集存储在OBS中。 • 数据集的标注信息存储在OBS中。 • 支持从OBS中导入数据。
	开发环境	Notebook实例中的数据或代码文件可以存储在OBS中。
	训练模型	训练作业使用的数据集、算法、运行脚本、训练输出产物、训练过程日志均可以存储在OBS中。
	推理部署	训练作业结束后，其生成的模型可以存储在OBS中，创建模型时，从OBS中导入已有的模型文件。

创建 OBS 操作步骤

1. 登录**OBS管理控制台**，在桶列表页面右上角单击“创建桶”，创建OBS桶。

图 2-15 创建桶

说明

创建桶的区域需要与ModelArts所在的区域一致。例如：当前ModelArts在中国-香港区域，在对象存储服务创建桶时，请选择中国-香港。

如何查看OBS桶与ModelArts的所处区域，请参见**查看OBS桶与ModelArts是否在同一区域**。

请勿开启桶加密，ModelArts不支持加密的OBS桶，会导致ModelArts读取OBS中的数据失败。

2. 在桶列表页面，单击桶名称，进入该桶的概览页面。

图 2-16 桶列表

3. 单击左侧导航的“对象”，在对象页面单击新建文件夹，创建OBS文件夹。例如，在已创建的OBS桶“c-flowers”中新建一个文件夹“flowers”。

图 2-17 新建文件夹

在OBS桶中创建完文件夹，即可以上传文件，上传文件操作请参见**OBS上传操作**。

常见问题

- 在ModelArts中选择OBS路径时，找不到已创建的OBS桶?
- 如何查看ModelArts与OBS桶是否在同一区域?
- 在对OBS桶操作时，出现Error: stat:403错误

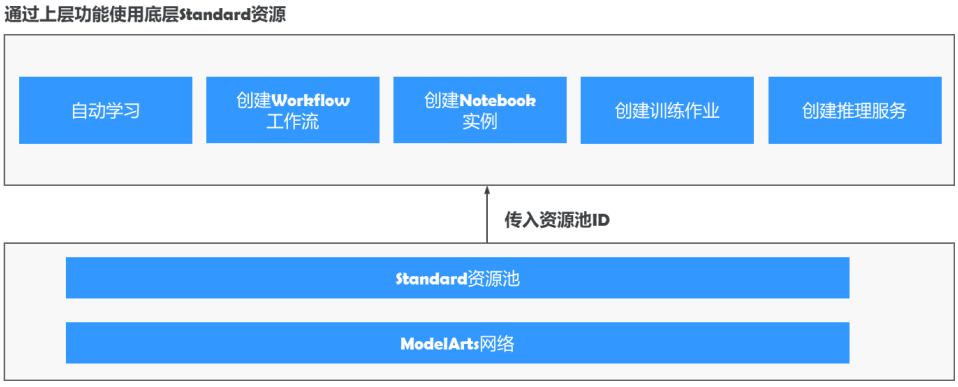
出现以上问题或其他OBS路径错误时，请参考**ModelArts中提示OBS路径错误**解决。

3 ModelArts Standard 资源管理

3.1 Standard 资源池功能介绍

ModelArts Standard资源池，提供了在使用ModelArts进行AI开发（包括自动学习、创建Workflow工作流、创建Notebook实例、创建训练作业和创建推理服务）所需的计算资源，您可根据需要购买使用Standard资源池。

图 3-1 在 AI 开发时使用 Standard 资源池



ModelArts Standard 资源池说明

在使用ModelArts进行AI开发时，您可以选择使用如下两种资源池：

- **专属资源池**：专属资源池不与其他用户共享，资源更可控。在使用专属资源池之前，您需要先创建一个专属资源池，然后在AI开发过程中选择此专属资源池。
- **公共资源池**：公共资源池提供公共的大规模计算集群，根据用户作业参数分配使用，资源按作业隔离。 用户下发训练作业、部署模型、使用开发环境实例等，均可以使用ModelArts提供的公共资源池完成，按照使用量计费，方便快捷。

专属资源池和公共资源池的能力主要差异如下：

- 专属资源池为用户提供独立的计算集群、网络，不同用户间的专属资源池物理隔离，公共资源池仅提供逻辑隔离，专属资源池的隔离性、安全性要高于公共资源池。
- 专属资源池用户资源独享，在资源充足的情况下，作业是不会排队的；而公共资源池使用共享资源，在任何时候都有可能排队。
- 专属资源池支持打通用户的网络，在该专属资源池中运行的作业可以访问打通网络中的存储和资源。例如，在创建训练作业时选择打通了网络的专属资源池，训练作业创建成功后，支持在训练时访问SFS中的数据。
- 专属资源池支持自定义物理节点运行环境相关的能力，例如GPU/Ascend驱动的自助升级，而公共资源池暂不支持。

专属资源池使用说明

- 如果您是初次使用专属资源池，建议您可从本章节开始，了解ModelArts提供的资源池详细说明。
- 在对专属资源池有一定了解后，如果您需要创建一个自己的专属资源池，您可参考**创建Standard专属资源池**来进行创建。
- 专属资源池创建成功后，可在**查看Standard专属资源池详情**中查看专属资源池的详细信息。
- 如果专属资源池的规格与您的业务不符，可通过**扩缩容Standard专属资源池**来调整专属资源池的规格。
- 每个用户对集群的驱动要求不同，在专属资源池列表页中，可自行选择加速卡驱动，并根据业务需要进行立即变更或平滑升级。ModelArts提供了自助升级专属资源池GPU/Ascend驱动的能力，可参考**升级Standard专属资源池驱动**进行升级。
- 专属资源池提供了故障节点修复的功能，可参考**修复Standard专属资源池故障节点**修复故障节点。
- 专属资源池提供了动态设置作业类型的功能，可参考**修改Standard专属资源池支持的作业类型**更新作业类型。
- 专属资源池提供了工作空间功能，管理员可以根据工作空间，隔离不同子用户操作工作空间内资源的权限，您可通过**迁移Standard专属资源池和网络至其他工作空间**将资源池移动到对应的工作空间下。
- 专属资源池可通过标签来进行管理，具体可参见**使用TMS标签实现资源分组管理**管理专属资源池标签。
- 当不再需要使用专属资源池时，您可参考**释放Standard专属资源池和删除网络**删除专属资源池。

3.2 创建 Standard 专属资源池

本章节主要介绍创建Standard专属资源池的详细操作。

前提条件

- 已经创建虚拟私有云。
- 已经创建子网。

步骤一：创建网络

ModelArts网络是承载ModelArts资源池节点的网络连接，基于华为云的VPC进行封装，对用户仅提供网络名称以及CIDR网段的选择项，为了防止在打通VPC的时候有网段的冲突，因此提供了多个CIDR网段的选项，用户可以根据自己的实际情况进行选择。虚拟私有云VPC是一套为实例构建的逻辑隔离的、由用户自主配置和管理的虚拟网络环境。为云服务器、云容器、云数据库等资源构建隔离的、用户自主配置和管理的虚拟网络环境，提升用户资源的安全性，简化用户的网络部署。

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群Cluster”，默认进入“Standard资源池”页面。
2. 切换到“网络”页签，单击“创建”，弹出“创建网络”页面。
3. 在“创建网络”弹窗中填写网络信息。
   - 网络名称：创建网络时默认生成网络名称，也可自行修改。
   - 网段类型：可选“预置”和“自定义”。自定义网络建议使用网段：10.0.0.0/8~24、172.16.0.0/12~24、192.168.0.0/16~24，子网掩码可选范围8-28。

图 3-2 创建网络

说明

- 单用户最多可创建15个网络。
- 网段设置以后不能修改，避免与将要打通的VPC网段冲突。可能冲突的网段包括：
  - 用户的VPC网段
  - 容器网段（固定是172.16.0.0/16）
  - 服务网段（固定是10.247.0.0/16）

4. 确认无误后，单击“确定”。

步骤二：打通 VPC（可选）

通过打通VPC，可以方便用户跨VPC使用资源，提升资源利用率。

1. 在“网络”页签，单击网络列表中某个网络操作列的“打通VPC”。

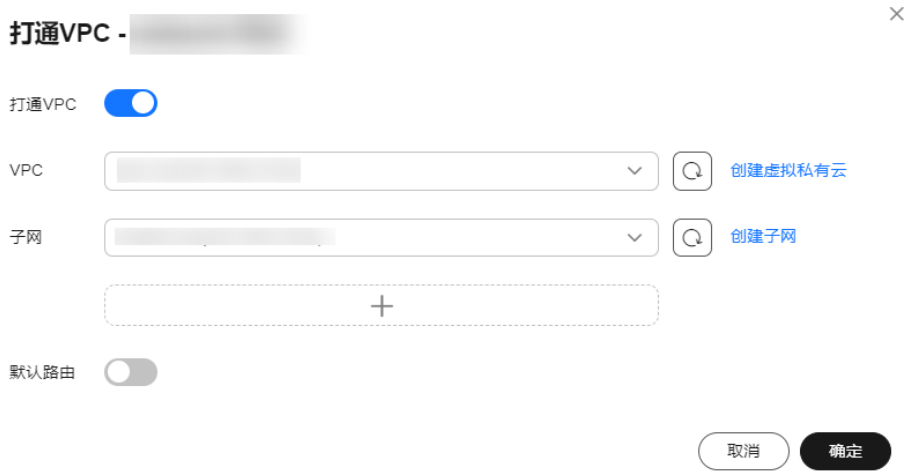
图 3-3 打通 VPC

2. 在打通VPC弹框中，打开“打通VPC”开关，在下拉框中选择可用的VPC和子网。

   说明

   需要打通的对端网络不能和当前网段重叠。

图 3-4 打通 VPC 参数选择



- 如果没有VPC可选，可以单击右侧的“创建虚拟私有云”，跳转到网络控制台，申请创建虚拟私有云。
- 如果没有子网可选，可以单击右侧的“创建子网”，跳转到网络控制台，创建可用的子网。
- 支持1个VPC下多个子网的打通，如果VPC下有多个子网，会显示“+”，单击“+”即可添加子网（上限10个）。
- 如果需要使用打通VPC的方式实现专属资源池访问公网，由于要访问的公网地址不确定，一般是建议用户在VPC中创建SNAT。此场景下，在打通VPC后，专属资源池中作业访问公网地址，默认不能转发到用户VPC的SNAT，需要提交工单联系技术支持在专属资源池VPC的路由中添加指向对等连接的缺省路由。当您开启默认路由后，在打通VPC时，会将ModelArts网络0.0.0.0/0路由作为默认路由，此时无需提交工单添加缺省路由即可完成网络配置。

步骤三：创建 Standard 专属资源池

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“AI专属资源池 > 弹性集群Cluster”。
2. 在“弹性集群Cluster”页签，单击“购买AI专属集群”，进入购买AI专属集群界面，参见下表填写参数。

表 3-1 AI 专属集群的参数说明

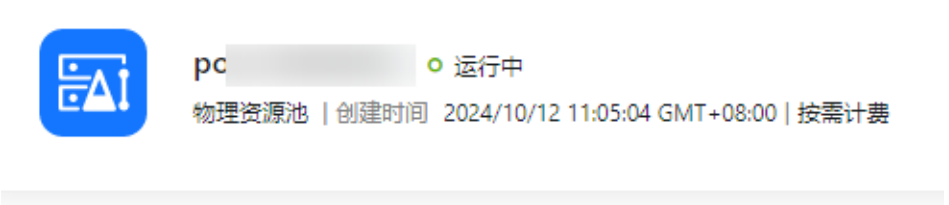
参数名称	子参数	说明
名称	-	专属资源池的名称。 只能以小写字母开头，由小写字母、数字、中划线（-）组成，不能以中划线结尾。
描述	-	专属资源池的简要说明。
计费模式	-	选择计费模式，“按需计费”。
资源池类型	-	可选物理资源池和逻辑资源池。逻辑资源池与规格有关，如果无逻辑规格则不显示逻辑资源池。
作业类型	-	根据业务需要，选择该资源池支持的作业类型。 ● 物理资源池：支持“开发环境”、“训练作业”和“推理服务”的作业类型。 ● 逻辑资源池：仅支持“训练作业”的作业类型。
网络	-	表示服务实例运行在指定的网络中，可以与该网络中的其他云服务资源实例互通。 在下拉框中选择，如果没有可用网络，单击右侧的“创建”，创建一个可用的网络。创建网络相关可以参考**步骤一：创建网络**章节。
规格管理	规格类型	请根据界面提示选择需要使用的规格。平台分配的资源规格包含了一定的系统损耗，实际可用的资源量小于规格标称的资源。实际可用的资源量可在专属资源池创建成功后，在详情页的“节点”页签中查看。 当前部分规格为受限购买（如Ascend规格），需要提前联系客户经理申请开通资源规格，预计1~3个工作日内开通（如果无客户经理可提交工单反馈）。
	可用区	您可以根据实际情况选择“随机分配”或“指定AZ”。可用区是在同一区域下，电力、网络隔离的物理区域。可用区之间内网互通，不同可用区之间物理隔离。 ● 随机分配：系统自动分配可用区。 ● 指定AZ：指定资源池节点在哪个可用区域。考虑系统容灾时，推荐指定节点在同一个可用区。可设置可用区的节点数。

参数名称	子参数	说明
	节点数量	选择专属资源池的节点数，选择的节点数越多，计算性能越强。 当“可用区”选择“指定AZ”时，节点数量会根据可用区的数据自动计算，此处无须再次设置。 **说明** 单次创建时，节点数建议不大于30，否则可能触发限流导致创建失败。 部分规格支持整柜购买，此时节点数量会显示为“数量*整柜”，购买的节点总数为两者的乘积。整柜购买可实现不同任务间的物理隔离，避免通信冲突，在任务规模增大的同时保证计算性能线性度不下降。整柜下的节点生命周期需保持一致，需要一起创建、一起删除。 **图 3-5** 整柜购买 
	高级选项	开启后，可设置容器引擎空间大小。 容器引擎空间大小仅支持整数， 默认值与最小值为50G，不同规格的最大值不同，数值有效范围请参考界面提示。自定义设置容器引擎空间大小不会造成额外费用增加。
自定义驱动	-	默认关闭。部分GPU和Ascend规格资源池允许自定义安装驱动。集群中默认会安装驱动，无需用户操作。只有需要指定驱动版本时，需要开启。
GPU驱动/Ascend驱动	-	打开“自定义驱动”开关，显示此参数，选择GPU/Ascend驱动。如果规格类型为GPU则显示“GPU驱动”，如果规格类型为Ascend则显示“Ascend驱动”。
购买时长	-	选择购买时长。只有选择“包年/包月”计费模式时才需填写。
自动续费	-	是否自动续费。只有选择“包年/包月”计费模式时才需填写。 ● 按月购买：自动续费周期为1个月。 ● 按年购买：自动续费周期为1年。
高级选项	-	选中“现在配置”，可配置标签信息、网段、控制节点分布。

参数名称	子参数	说明
标签	-	ModelArts支持对接标签管理服务TMS，在ModelArts中创建资源消耗性任务（例如：创建Notebook、训练作业、推理在线服务）时，可以为这些任务配置标签，通过标签实现资源的多维分组管理。 标签详细用法请参见**ModelArts如何通过标签实现资源分组管理**。 **说明** 可以在标签输入框下拉选择TMS预定义标签，也可以自己输入自定义标签。预定义标签对所有支持标签功能的服务资源可见。租户自定义标签只对自己服务可见。
网段	-	可选默认和自定义。 ● 默认：系统随机分配一个不冲突的网段供用户使用，因后续不支持修改建议商用场景选择手动分配，确保网段符合用户诉求。 ● 自定义：需要自定义K8S容器网段和K8S服务网段。 – K8S容器网段：集群下容器使用的网段，决定了集群下容器的数量上限。创建后不可修改。 – K8S服务网段：同一集群下容器互相访问时使用的Service资源的网段。决定了Service资源的上限。 创建后不可修改。
控制节点分布	-	控制节点的分布位置，可选择随机分配和自定义。 ● 随机分配：随机分配控制节点可用区，尽可能将控制节点随机分布在不同可用区以提高容灾能力 。如果某可用区资源不足，将分配至资源充足的可用区，优先保障集群创建成功，可能无法保障可用区级容灾。 ● 自定义：自定义选择控制节点的可用区。 控制节点推荐尽可能随机分布在不同可用区以提高容灾能力。

3. 单击“下一步”确认规格。规格确认无误后，单击“提交”，即可创建专属资源池。
   – 当资源池创建成功后，资源池的状态会变成“运行中”，当“节点个数”中的“可用”和“总数”值大于0时，资源池才能下发任务。

图 3-6 查看资源池



- 可以将鼠标放在“创建中”字样上，查看当前创建过程详情。如果单击查看详情，可跳转到“操作记录”中。

图 3-7 创建中状态

- 可以在资源池列表右上角“操作记录”中查看资源池的任务记录。

图 3-8 操作记录

图 3-9 查看资源池状态

常见问题

创建专属资源池时，能选到规格但最终创建时发生报错，提示无可用资源？

由于专属资源的可选规格是动态监测的，因此在极少数情况下会出现，规格在购买界面可以被选择，但由于没有及时支付和创建资源池，导致该规格售罄创建失败。

建议您在创建界面更换规格重新创建资源池。

为什么无法使用资源池节点上的全部CPU资源？

由于资源池节点上会安装系统、插件等内容，因此不能完全使用所有资源。例如：资源池节点是8U，节点分配给系统组件部分CPU，可用的资源会小于8U。

建议您在启动任务前，在该资源池的详情页中，单击“节点”页签，查看实际可用的CPU资源。

3.3 管理 Standard 专属资源池

3.3.1 查看 Standard 专属资源池详情

资源池详情页介绍

- 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群 Cluster”，进入“Standard资源池”列表。
- 在“Standard资源池”列表页的搜索框中，支持根据资源池的名称、资源池ID、资源池的状态、节点状态、资源池类型、创建时间搜索。
- 在资源池列表中，单击某一资源池名称，进入资源池详情页，查看资源池的基本信息和其他扩展信息。
  - 对于Standard资源池，当创建了多个资源池时，可在详情页单击左上角▾，可切换资源池。
  - 对于按需计费的Standard资源池，在详情页中，单击右上角“更多”，可进行扩缩容、删除、转包周期、设置作业类型等操作，不同资源池可进行的操作不一致，具体以控制台显示为准。
  - 对于包年/包月的Standard资源池，在详情页中，单击右上角“更多”，可进行扩容、退订、续费、开通自动续费或修改自动续费、驱动升级、设置作业类型等操作，不同资源池可进行的操作不一致，具体以控制台显示为准。
  - 在“基本信息”的“网络”中，可单击关联的资源池中的数字，查看关联的资源池。可以查看该网络中可用的IP数量。
  - 在扩展信息中可以查看监控、作业、节点、规格、事件、标签，详细介绍见下文。

查看资源池中的作业

在资源池详情页，切换到“作业”页签。您可以查看该资源池中运行的所有作业，如果当前有作业正在排队，可以查看作业在资源池排队的位置。

说明

当前仅支持查看训练作业。

查看资源池事件

在资源池详情页，切换到“事件”页签。您可以查看资源从创建到添加节点的各个阶段的事件。产生事件的原因主要有“资源池状态变化”和“资源节点状态变化”。

在事件列表中，可单击“事件类型”列的 筛选查看。

- 当资源池开始创建或者出现异常时，因资源池状态变化，会将此变化信息记录到事件中。
- 当节点的可用、异常、创建中、删除中的数量发生变化时，因资源池节点状态变化，会将此变化信息记录到事件中。

图 3-10 查看资源池事件

查看资源池节点

在资源池详情页，切换到“节点”页签。您可以查看资源池中所有的节点，并且能查看每个节点资源占用的情况。当把鼠标放在节点名称上方时，会显示节点名称和资源ID，资源ID可用于查询账单或者在费用中心查询包周期资源的计费信息。

由于集群组件会占用一部分资源，所以列表中CPU（可用/总数）呈现的资源数量不代表该节点物理资源数量，仅表示可被业务使用到的资源量。其中，CPU核数为微核，1000微核=1物理核。

如下图所示，支持对多节点批量进行删除、退订、重启、重置、开启/关闭高可用冗余操作，具体介绍请参见**修复Standard专属资源池故障节点**。还支持对节点批量添加、编辑、删除资源标签操作，“包年/包月”的节点支持批量续费、批量开通/修改自动续费功能。

图 3-11 节点批量操作

如下图所示，在单个节点的操作列，支持对单个节点进行删除、替换、修复、重置、重启、授权、运行作业列表、开启高可用冗余、关闭高可用冗余等操作，具体介绍请参见**修复Standard专属资源池故障节点**。还支持编辑资源标签操作。

图 3-12 单个节点操作

在节点的搜索栏，支持通过节点的名称、节点状态、高可用冗余、批次、驱动版本、驱动状态、IP地址、资源标签等关键字搜索节点。

支持导出Standard资源池的节点信息到Excel表格中，方便查阅。勾选节点名称，在节点列表上方单击“导出 > 导出全部数据到XLSX”或者“导出 > 导出部分数据到XLSX”，在浏览器的下载记录 中查看导出的Excel表格。

在节点列表页面中，单击设置图标 ，支持对节点列表中显示的信息进行自定义。

查看资源池规格

在资源池详情页，切换到“规格”页签。您可以查看该资源池使用的资源规格以及该规格对应的数量，并可以调整容器引擎空间大小。

图 3-13 查看资源池规格（如果创建资源池时未设置容器引擎大小，则显示默认值）

查看资源池监控

在资源池详情页，切换到“监控”页签。展示了CPU使用量、内存利用率、磁盘可用容量等使用情况，均以资源池的维度呈现。当资源池中有AI加速卡时，还会显示GPU、NPU的相关监控信息。

图 3-14 查看资源视图

表 3-2 监控指标

名称	指标含义	单位	取值范围
CPU使用率	该指标用于统计测量对象的CPU使用率。	百分比（Percent）	0～100%
内存利用率	该指标用于统计测量对象已使用内存占申请物理内存总量的百分比。	百分比（Percent）	0～100%
GPU显卡使用率	该指标用于统计测量对象已使用的显卡占显卡容量的百分比。	百分比（Percent）	0～100%
GPU显存使用率	该指标用于统计测量对象已使用的显存占显存容量的百分比。	百分比（Percent）	0～100%
NPU显卡使用率	该指标用于统计测量对象已使用的显卡占显卡容量的百分比。	百分比（Percent）	0～100%
NPU显存使用率	该指标用于统计测量对象已使用的显存占显存容量的百分比。	百分比（Percent）	0～100%
磁盘可用容量	该指标用于统计测量对象可用的磁盘容量。	MB	≥0
磁盘容量	该指标用于统计测量对象磁盘总容量。	MB	≥0
磁盘利用率	该指标用于统计测量对象的磁盘使用率。	百分比（Percent）	0～100%

名称	指标含义	单位	取值范围
GPU/NPU碎片数	由于资源调度产生碎片，导致某些卡虽然空闲，但无法被多卡任务所使用。不同卡数的任务，根据已占用卡的分布不同，实际会有不同的碎片情况，且随时间变化，表格中仅表示当前时间的状态。	/	/

管理资源池标签

通过给资源池添加标签，可以标识云资源，便于快速搜索资源池。

在资源池详情页，切换到“标签”页签。您可以查看、搜索、添加、修改、删除资源池的标签信息。

说明

最多支持添加20个标签。

查看资源池的磁盘规格

在资源池详情页的右上角，单击“更多>扩缩容”，在资源池扩缩容页面可以查看该资源规格中携带的系统盘、容器盘、数据盘的磁盘类型、大小、数量和写入模式、容器引擎空间大小、挂载路径磁盘配置等参数。

3.3.2 扩缩容 Standard 专属资源池

场景介绍

当专属资源池创建完成，使用一段时间后，由于用户AI开发业务的变化，对于资源池资源量的需求可能会产生变化，面对这种场景，ModelArts Standard专属资源池提供了扩缩容功能，用户可以根据自己的需求动态调整。

- 使用扩容功能时，可以增加资源池已有规格的实例数量。
- 使用缩容功能时，可以减少资源池已有规格的实例数量。

说明

缩容操作可能影响到正在运行的业务，建议用户在业务空窗期进行缩容，或进入资源池详情页面，在指定空闲的节点上进行删除来实现缩容。

约束限制

- 只支持对状态为“运行中”的专属资源池进行扩缩容。
- 专属资源池不能缩容到0。

扩缩容专属资源池

资源池扩缩容有以下类型，分别为：

- 对已有规格增减目标总实例数
- 修改容器引擎空间大小

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“AI专属资源池 > 弹性集群 Cluster”，进入“Standard资源池”页签，查看资源池列表。

   说明

   在旧版资源池迁移到新版资源池的过程中，资源池状态显示为“受限”。此时，资源池无法进行扩缩容和退订。

2. 单击某个资源池操作列右侧的“扩缩容”，进入“专属资源池扩缩容”页面，对资源池进行扩缩容操作。对于包周期资源池，此按钮为“扩容”，如果需要缩容，请进入到包周期资源池详情页对节点进行退订操作。

3. 在“专属资源池扩缩容”页面，可通过增减“目标总实例数”实现扩缩容，请用户根据本身业务诉求进行调整。增加目标实例数量即表示扩容，减少目标实例数量即表示缩容。

   如果购买资源池时，节点数量采用整柜方式购买（部分规格支持），则在扩缩容时为整柜方式扩缩容，目标实例总数等于“数量*整柜”。“整柜”参数为创建资源池时选择，扩缩容时不可修改。用户通过增减“数量”来改变“目标总实例数”。

   说明

   用户增加实例数量时，可以通过指定节点计费模式，为资源池新创建的节点设置不同于资源池的计费模式，例如用户可以在包周期的资源池中创建按需的节点。如果用户不指定该参数，创建的节点计费模式和资源池保持一致。

4. 在“专属资源池扩缩容”页面，设置“资源配置 > 可用区”，可用区可选择随机分配和指定AZ。
   - 选择随机分配时，扩缩容完成后，节点的可用区分布由系统后台随机选择。
   - 选择指定AZ时，可指定扩缩容完成后节点的可用区分布。

   图 3-15 资源配置（单节点方式）

5. 修改容器引擎空间大小

   扩容资源池时，可以设置新建节点的容器引擎空间大小。此操作会导致资源池内该规格下节点的dockerBaseSize不一致，可能会使得部分任务在不同节点的运行情况不一致，请谨慎操作。存量节点不支持修改容器引擎空间大小。

6. 修改操作系统。在“操作系统”下拉列表中指定操作系统版本。

7. 指定节点计费模式。用户增加节点数量时，可以打开“节点计费模式”开关，为资源池新扩容的节点设置不同于资源池的计费模式、购买时长和开启自动续费功能。例如用户可以在包周期的资源池中创建按需的节点。若用户不指定该参数，则新扩容的节点计费模式和资源池保持一致。

8. 设置完成后，单击“提交”，在弹出的确认框中单击“确定”完成扩缩容。

3.3.3 升级 Standard 专属资源池驱动

场景介绍

当专属资源池中的节点含有GPU/Ascend资源时，用户基于自己的业务，可能会有自定义GPU/Ascend驱动的需求，ModelArts面向此类客户提供了自助升级专属资源池GPU/Ascend驱动的能力。

驱动升级有两种升级方式：安全升级、强制升级。

说明

- 安全升级：不影响正在运行的业务，开始升级后会先将节点进行隔离（不能再下发新的作业），待节点上的存量作业运行完成后再进行升级，因需要等待作业完成，故升级周期可能比较长。
- 强制升级：忽略资源池中正在运行的作业，直接进行驱动升级，可能会导致运行中作业失败，需谨慎选择。

约束限制

- 专属资源池状态处于运行中，且专属池中的节点需要含有GPU/Ascend资源。
- 对于逻辑资源池，需要开启节点绑定后才能进行驱动升级，请提交工单联系华为工程师开启节点绑定。

驱动升级操作

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群Cluster”，进入“Standard资源池”页面。
2. 在资源池列表中，选择需要进行驱动升级的资源池，在右侧的操作列，单击“ ··· > 驱动升级”。
3. 在“驱动升级”弹窗中，会显示当前专属资源池的驱动类型、实例数量、当前版本、目标版本、升级方式、升级范围和开启滚动开关。
   - 目标版本：在目标版本下拉框中，选择一个目标驱动版本。对于资源池新增加的节点，可能会与资源池原有节点驱动不一致，为了保持驱动一致，目标版本可选择当前驱动版本，升级完成后所有节点驱动会升级为统一版本。
   - 升级方式：可选择安全升级或强制升级。
     - 安全升级：待节点上没有作业时再升级，升级周期可能比较长。
     - 强制升级：忽略运行中作业，直接升级，可能会导致运行中作业失败。
   - 开启滚动：开启开关后，支持滚动升级的方式升级驱动。当前支持“按节点比例”和“按实例数量”两种滚动方式。
     - 按节点比例：每批次驱动升级的实例数量为“节点比例*资源池实例总数”。
     - 按实例数量：每批次驱动升级的实例数量为设置的实例数量。

     对于不同的升级方式，滚动升级选择节点的策略会不同：
     - 如果升级方式为安全升级，则根据滚动实例数量选择无业务的节点，隔离节点并滚动升级。

- 如果升级方式为强制升级，则根据滚动实例数量随机选择节点，隔离节点并滚动升级。

说明

- 无业务节点定义：在资源池详情“节点”页签下，如果GPU/Ascend的可用数等于总数，则为无业务节点。
- 滚动驱动升级时，驱动异常的节点对升级无影响，会和驱动正常的节点一起升级。

图 3-16 驱动升级

4. 设置完成后，单击“确定”开始驱动升级。

3.3.4 修复 Standard 专属资源池故障节点

Standard专属资源池支持对故障节点进行修复操作，目前提供了替换节点、高可用冗余节点、重置节点和重启节点等方式。华为云技术支持在故障定位和性能诊断时，部分运维操作需要用户授权才可进行，本章节同时也介绍了如何进行授权操作。

故障节点处理方式

- 替换节点：替换节点后，节点名称会发生变化。原有节点会被释放掉。在资源池详情页的“节点”页签中提供了对单个节点替换的功能。可单击节点页签操作列的“替换”，即可实现对单个节点的替换。替换节点操作不会收取费用。

  单击“操作记录”可查看当前资源池替换节点的操作记录。“运行中”表示节点在替换中。替换成功后，节点列表中会显示新的节点名称。

  替换最长时间为24小时，超时后仍然未找到合适的资源，状态会变为“失败”。

  可将鼠标悬浮在  图标上，查看具体失败原因。

  说明

  - 每天累计替换的次数不超过资源池节点总数的20%，同时替换的节点数不超过资源池节点总数的5%。
  - 替换节点时需确保有空闲节点资源，否则替换可能失败。
  - 当操作记录里有节点处于重置中时，该资源池无法进行替换节点操作。

- 高可用冗余节点

  高可用冗余节点作为专属资源池内的备用节点，能够在普通节点故障时自动进行切换，可以提升资源池整体的SLA，有效避免单个节点故障造成的业务受损。用户可以根据自身业务的可靠性要求设置池内的高可用节点数量。

  须知

  高可用冗余节点不能用于业务运行，将影响资源池的实际可用节点数量。资源池下发任务时，请注意选择实际可用的节点数量，当选择的节点数未剔除资源池的高可用冗余节点数时，会导致任务持续等待。

  高可用冗余节点的运行机制：

  - 高可用冗余节点将被隔离，默认设置为不可调度，工作负载无法调度到节点上。
  - 高可用冗余节点会作为备用节点与节点的故障检测配合使用，为资源池提供故障节点自动切换能力，高可用冗余节点能够在普通节点故障时自动进行切换，切换耗时通常在分钟内。切换后，原“高可用冗余节点”与“故障节点”交换高可用冗余标签，原“高可用冗余节点”自动解隔离成为普通节点，“故障节点”则成为“高可用冗余节点”，由于高可用冗余节点仅是对故障节点的切换，此时仍需对切换后的故障节点进行维修，维修后才能用于后续的自动切换。故障节点修复后，“高可用冗余”标签维持不变，修复好的节点变为新的高可用冗余节点。

  相比于其他故障修复方式，高可用冗余节点可使用户免于关注节点状态，减少运维成本。但由于用户需要付费购买备用节点作为高可用冗余节点，因此资源成本会提高。

  如何设置高可用节点：当前支持从资源池角度批量设置多个高可用冗余节点，也支持设置单个节点为高可用冗余节点。

  - 资源池批量设置多个高可用冗余节点
    - 方式一：在购买时设置（仅Snt9C支持）

      参数说明：

      - 开启高可用冗余：是否开启资源池的高可用冗余，超节点默认开启高可用冗余。

- 冗余节点分布策略：冗余节点的分布策略，超节点仅支持step均分：每个超节点内预留相同数量的冗余节点。
- 冗余实例数：此规格设置的高可用冗余实例数量。冗余系数指的是冗余节点分布策略为step均分时，每个超节点内预留的冗余节点数量。

- 方式二：在资源池详情页的规格页签设置

图 3-17 规格页签设置

- 方式三：在扩缩容页面设置

– 设置单节点为高可用冗余节点

- 开启高可用冗余

  挑选无业务节点作为高可用冗余节点使用，在资源池详情页，“节点”页签下，在想要开启高可用冗余的节点操作列，单击“更多 > 开启高可用冗余”，设置成功后，该节点高可用冗余列标签变为“启用”。

  如果想批量设置节点开启高可用冗余，可勾选多个节点后，单击列表上方的“开启高可用冗余”按钮实现批量开启。

图 3-18 开启高可用冗余能力

图 3-19 高可用冗余节点

说明

- 资源池内高可用冗余节点的建议比例： 每种资源规格建议按5%设置，如每20个节点中挑选一个作为1个高可用冗余节点。
- 无业务节点定义：在资源池详情“节点”页签下，如果GPU/Ascend的可用数等于总数，则为无业务节点。

- 关闭高可用冗余

  在资源池详情页，“节点”页签下，在想要关闭高可用冗余的节点操作列，单击“更多 > 关闭高可用冗余”，设置成功后，该节点高可用冗余列标签变为“--”。

  取消高可用冗余将会解除隔离，工作负载可正常调度到节点上，节点不再作为备用节点使用。

  如果想批量设置节点关闭高可用冗余，可勾选多个节点后，单击列表上方的“关闭高可用冗余”按钮实现批量关闭。

  图 3-20 关闭高可用冗余能力

  图 3-21 非高可用冗余

- 重置节点：需要升级节点操作系统时，可通过重置节点完成。更新节点配置时产生故障报错，也可通过重置节点修复故障。

  “节点”页签中提供节点重置的功能。单击操作列的“重置”，可实现对单个节点的重置。勾选多个节点的复选框，单击操作记录旁的“重置”按钮，可实现对多个节点的重置。

  下发重置节点任务时需要填写以下参数：

  表 3-3 重置参数说明

参数名称	说明
操作系统	选择下拉框中支持的操作系统。
配置方式	选择重置节点的配置方式。 • 按节点比例：重置任务包含多个节点时，同时被重置节点的最高比例。 • 按节点数量：重置任务包含多个节点时，同时被重置节点的最大个数。

单击“操作记录”可查看当前资源池重置节点的操作记录。重置中节点状态为“重置中”，重置成功后，节点状态变为“可用”。重置节点操作不会收取费用。

图 3-22 重置节点

说明

- 重置节点将影响相关业务的运行，请谨慎操作。
- 节点状态为“可用”的节点才能进行重置。
- 同一时间单个节点只能处于一个重置任务中，无法对同一个节点同时下发多个重置任务。
- 当操作记录里有节点处于替换中时，该资源池无法进行重置节点操作。
- 当资源池处于驱动升级状态时，该资源池无法进行重置节点操作。
- GPU和NPU规格，重置节点完成后，节点可能会出现驱动升级的现象，请耐心等待。

图 3-23 操作记录

- 重启节点

  资源池详情页的“节点”页签中提供节点重启的功能。单击操作列的“重启”，可实现对单个节点的重启。勾选多个节点的复选框，单击操作记录旁的“重启”按钮，可实现对多个节点的重启。

下发重启节点任务时需要选择对应节点，重启节点将影响相关业务的运行，请谨慎操作。

单击“操作记录”可查看当前资源池节点的操作记录。重启中节点状态为“重启中”，重启成功后，节点状态变为“可用”。重启节点操作不会收取费用。

图 3-24 重启节点

图 3-25 操作记录

说明

- 重启节点将影响相关业务的运行，请谨慎操作。
- 节点状态为“可用”、“不可用”的节点才能进行重启。
- 同一时间单个节点只能处于一个重启任务中，无法对同一个节点同时下发多个重启任务。
- 当操作记录里某节点处于替换中、重置中或删除中时，无法对该节点进行重启节点操作。
- 当资源池处于驱动升级状态时，该资源池无法进行重启节点操作。
- 节点重启成功后，可能出现短暂不可用现象，是正在拉起业务服务及健康检查，请耐心等待。

- 删除/退订节点：
  - 如果是“按需计费”的资源池，您可单击操作列的“删除”，即可实现对单个节点的资源释放。

    如果想批量删除节点，勾选待删除节点名称前的复选框，然后单击名称上方的“删除”，即可实现对多个节点的资源释放。
  - 如果是“包年/包月”且资源未到期的资源池，您可单击操作列的“退订”，即可实现对单个节点的资源释放。
  - 如果是“包年/包月”且资源到期的资源池（处于宽限期），您可单击操作列的“释放”，即可实现对单个节点的资源释放。

部分“包年/包月”节点会出现“删除”按钮，原因是该节点为存量节点，单击“删除”即可实现节点的资源释放。

说明

- 删除/退订/释放节点可能导致该节点上运行的作业失败，请保证该节点无任务运行时再进行操作。
- 当资源池中存在异常节点时，可通过删除/退订/释放操作，将资源池中指定的异常节点移除，再通过扩容专属资源池获得和之前相同的总节点个数。
- 仅有一个节点时，无法进行删除/退订/释放操作。

授权技术支持定位故障

华为云技术支持在故障定位和性能诊断时，部分运维操作需要用户授权才可进行。您可在资源池详情页的节点页签下，找到对应节点，在操作列单击“更多 > 授权”，在弹出的提示框中单击“确认”即可完成授权。

说明

正常情况下，该授权按钮为置灰状态。当华为云技术支持发起运维申请后，按钮会变为可点状态。

在完成运维操作后，华为云技术支持会主动关闭已获得授权，无需您额外操作。

3.3.5 修改 Standard 专属资源池支持的作业类型

场景介绍

ModelArts含有许多“作业”类型（作业为统称，并非单指训练作业），其中有一部分是可以运行在Standard专属资源池上的，包括“训练”、“推理”服务及“Notebook”开发环境。

专属资源池提供了动态设置作业类型的功能，您可以在创建资源池时、创建完成后，对资源池支持的作业类型进行编辑（新增或减少）。当前支持的“作业类型”有“训练作业”、“推理服务”和“开发环境”，用户可按需自行选择。

设置某一作业类型后，即可在此专属资源池中下发此种类型的作业，没有设置的作业类型不能下发。

注意

为了支持不同的作业类型，后台需要在专属资源池上进行不同的初始化操作，例如安装插件、设置网络环境等。其中部分操作需要占据资源池的资源，导致用户实际可用资源减少。因此建议用户按需设置，避免不必要的资源浪费。

约束限制

专属资源池状态处于“运行中”。

操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群 Cluster”，进入“Standard资源池”页面。

2. 在资源池列表中，选择某个资源池右侧操作列的“ ··· > 设置作业类型”。
3. 在“设置作业类型”弹窗中，选择需要设置的作业类型。

图 3-26 设置作业类型

4. 设置完成后，单击“确定”，启用作业类型。

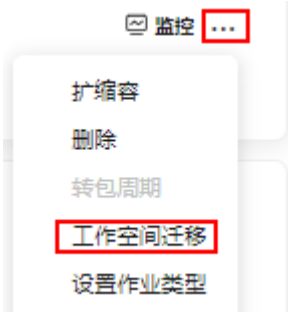
3.3.6 迁移 Standard 专属资源池和网络至其他工作空间

背景信息

专属资源池的工作空间关联了企业项目，企业项目涉及到账单归集。为隔离不同子用户操作资源的权限，ModelArts提供了工作空间功能，管理员可以根据工作空间，隔离不同子用户操作工作空间内资源的权限。工作空间迁移包括资源池迁移和网络迁移，具体方法可见下文说明。

资源池工作空间迁移

1. 登录ModelArts管理控制台，选择“AI专属资源池 > 弹性集群Cluster”，进入“Standard资源池”页面。
2. 在资源池列表中，选择目标资源池右侧操作列的“ ··· > 工作空间迁移”。



3. 在弹出的“迁移专属资源池”中，选择要迁移的“目标工作空间”，单击“确定”。

图 3-27 工作空间迁移

说明

子用户仅限于对自己创建的工作空间下的资源池进行迁移操作。

网络工作空间迁移

1. 登录ModelArts管理控制台，选择“AI专属资源池 > 弹性集群Cluster”，切换到“网络”页签。
2. 在网络列表中，选择目标网络“操作 > 更多 > 工作空间迁移”。
3. 在弹出的“迁移网络”中，选择要迁移的“目标工作空间”，单击“确定”。

图 3-28 工作空间迁移

说明

子用户仅限于对自己创建的工作空间下的网络进行迁移操作。

3.3.7 配置 Standard 专属资源池可访问公网

场景介绍

当您使用专属资源池创建作业时（如训练作业），如果需要作业运行过程中需要专属资源池访问外网，可打通VPC的方式，使得专属资源池和已绑定EIP的弹性云服务器处于同一VPC内，实现专属资源池访问外网。

前提条件

- 已拥有需要部署SNAT的弹性云服务器。
- 待部署SNAT的弹性云服务器操作系统为Linux操作系统。
- 待部署SNAT的弹性云服务器网卡已配置为单网卡。

步骤一：打通 VPC

通过打通VPC，可以方便用户跨VPC使用资源，提升资源利用率。

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群 Cluster”，在“网络”页签，单击网络列表中某个网络操作列的“打通VPC”。

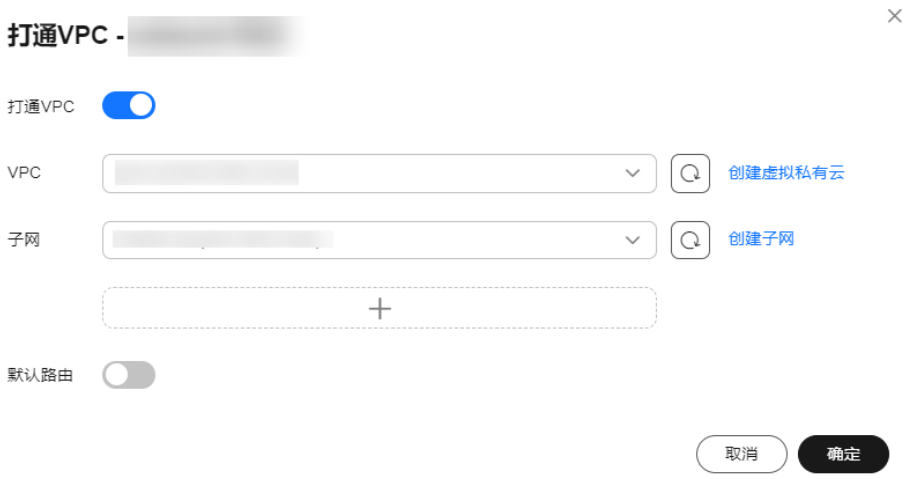
图 3-29 打通 VPC

2. 在打通VPC弹框中，打开“打通VPC”开关，在下拉框中选择可用的VPC和子网。

说明

需要打通的对端网络不能和当前网段重叠。

图 3-30 打通 VPC 参数选择



- 如果没有VPC可选，可以单击右侧的“创建虚拟私有云”，跳转到网络控制台，申请创建虚拟私有云。
- 如果没有子网可选，可以单击右侧的“创建子网”，跳转到网络控制台，创建可用的子网。
- 支持1个VPC下多个子网的打通，如果VPC下有多个子网，会显示“+”，您可单击“+”即可添加子网（上限10个）。
- 如果需要使用打通VPC的方式实现专属资源池访问公网，由于要访问的公网地址不确定，一般是建议用户在VPC中创建SNAT。此场景下，在打通VPC后，专属资源池中作业访问公网地址，默认不能转发到用户VPC的SNAT，需要提交工单联系技术支持在专属资源池VPC的路由中添加指向对等连接的默认路由。当您开启默认路由后，在打通VPC时，会给ModelArts网络0.0.0.0/0路由作为默认路由，此时无需提交工单添加默认路由即可完成网络配置。

步骤二：配置 SNAT

参考**通过公网NAT网关的SNAT规则访问公网**章节，配置并验证SNAT。

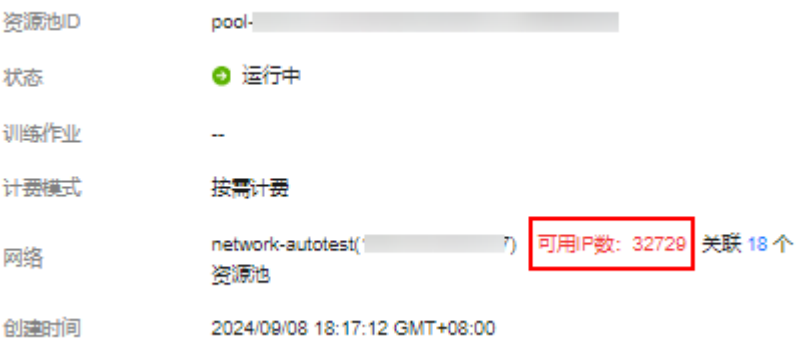
查看可用 IP 数量（可选）

登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群Cluster”，进入“网络”页签，单击网络列表中某个网络操作列的“更多 > 查看可用IP数量”，可以看到该网络所在的网段中可以使用的IP地址数量。

图 3-31 查看可用 IP 数量

在单个资源池的详情页中，也可以查看该资源池绑定网络的可用IP数量。

图 3-32 查看可用 IP 数量



3.3.8 使用 TMS 标签实现资源分组管理

ModelArts支持对接标签管理服务TMS，在ModelArts中创建资源消耗性任务时，可以为这些任务配置标签，通过标签实现资源的多维分组管理。

ModelArts支持配置标签的任务有：创建训练作业任务、创建Notebook、创建推理在线服务、创建ModelArts Standard专属资源池。

使用流程

1. **Step1 在TMS上创建预定义标签**。
2. **Step2 在ModelArts任务中添加标签**。

3. **Step3 在TMS中根据资源类型查询ModelArts任务**。

Step1 在 TMS 上创建预定义标签

登录TMS控制台，在预定义标签页面创建标签。此处创建的标签是全局标签，在华为云所有Region可见。

Step2 在 ModelArts 任务中添加标签

在ModelArts中创建Notebook、创建训练作业、创建推理在线服务时，对这些任务配置标签。

- 在ModelArts的Notebook中添加标签。

  可以在创建Notebook页面添加标签，也可以在已经创建完成的Notebook详情页面的“标签”页签中添加标签。
- 在ModelArts的训练作业中添加标签。

  可以在创建训练作业页面添加标签，也可以在已经创建完成的训练作业详情页面的“标签”页签中添加标签。
- 在ModelArts的在线服务中添加标签。

  可以在创建在线服务页面添加标签，也可以在已经创建完成的在线服务详情页面的“标签”页签中添加标签。
- 在ModelArts的专属资源池中添加标签。

  可以在创建ModelArts Standard专属资源池页面添加标签，也可以在已经创建的Standard专属资源池详情页面的“标签”页签中添加标签。

图 3-33 添加标签

说明

用户也可以在ModelArts任务中添加标签时，创建新的标签，直接输入标签键和标签值即可。此处创建的标签仅当前的项目Project可见。不同的项目中查看不到。

Step3 在 TMS 中根据资源类型查询 ModelArts 资源使用情况

登录TMS控制台，在资源标签页面根据资源类型和资源标签查询指定区域的资源任务。

- 区域：使用华为云的具体Region，区域概念请参见**什么是区域、可用区？**。
- 资源类型：ModelArts支持查询的资源类型如**表3-4**所示。
- 资源标签：不填写标签时，表示查询所有资源，无论此资源是否有配置标签。选择相应标签查询资源，用户可以通过多个标签组合查询资源使用情况。

表 3-4 ModelArts 的资源类型

资源类型	说明
ModelArts-Notebook	ModelArts的开发环境Notebook对应的资源类型。
ModelArts-TrainingJob	ModelArts的训练作业对应的资源类型。
ModelArts-RealtimeService	ModelArts的推理在线服务对应的资源类型。
ModelArts-ResourcePool	ModelArts的专属资源池对应的资源类型。

说明

如您的组织已经设定ModelArts的相关标签策略，则需按照标签策略规则为资源添加标签。标签如果不符合标签策略的规则，则可能会导致资源创建失败，请联系组织管理员了解标签策略详情。

3.3.9 管理 Standard 专属资源池的游离节点

如果资源中存在游离节点，即没有被纳管到资源池中的节点，可在“AI专属资源池 > 弹性集群Cluster >节点”下查看此类节点的相关信息。

系统支持对游离节点进行续费、退订、开通/修改自动续费、添加/编辑资源标签、删除资源标签、搜索等操作。

续费/开通自动续费/修改自动续费

对于包年/包月的节点，在“节点管理”页签中提供了续费、开通自动续费和修改自动续费功能，并支持对多个节点进行批量操作。

添加/编辑/删除资源标签

资源标签用于方便管理资源的计费账单。

勾选节点名称，选择节点列表上方的“添加/编辑资源标签”或“删除资源标签”，操作单个节点或批量操作节点资源标签。

查找搜索节点

在节点管理页面的搜索栏中，支持通过节点名称、IP地址、资源标签等关键字搜索节点。

设置节点列表显示信息

在节点页面中，单击右上角的设置图标 ，支持对节点列表中显示的信息进行自定义。

删除/退订/释放节点

具体操作请参见**释放游离节点**章节。

3.3.10 释放 Standard 专属资源池和删除网络

删除资源池

当AI业务开发不再需要使用专属资源池时，您可以删除专属资源池，释放资源。

说明

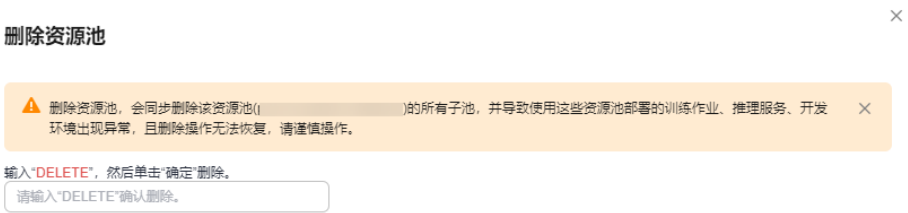
专属资源池删除后，将导致使用此资源的开发环境、训练作业和推理服务等不可用，且删除后不可恢复，请谨慎操作。

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群Cluster”，进入“Standard资源池”页面。
2. 在专属资源池列表中，在需要删除的资源池的右侧操作列选择“ ··· > 删除”。

3. 在“删除资源池”页面，需在文本框中输入“DELETE”，单击“确定”，删除资源池。

   可切换“训练作业”、“推理服务”、“开发环境”页签查看资源池上创建的训练作业、部署的推理服务、创建的Notebook实例。

图 3-34 删除资源池



释放游离节点

如果您的资源中存在游离节点（即没有被纳管到资源池中的节点），您可在“AI专属资源池 > 弹性集群Cluster >节点”下查看此类节点的相关信息。

针对游离节点，可以通过以下方式释放节点资源：

- 如果是“包年/包月”且资源未到期的节点，您可单击操作列的“退订”，即可实现对单个节点的资源释放。支持批量退订节点。
- 如果是“包年/包月”且资源到期的节点（处于宽限期），您可单击操作列的“释放”，即可实现对单个节点的资源释放。不支持批量释放处于宽限期的节点。

说明

退订/释放操作无法恢复，请谨慎操作。

删除网络

当AI业务开发不再需要使用网络时，您可以删除网络。

说明

请注意，删除网络会导致使用该网络的资源池网络不可用，请谨慎操作。

1. 在“网络”页签，单击某个网络操作列的“更多 > 删除”。
2. 确认删除，单击“确定”即可。

4 使用自动学习实现零代码 AI 开发

4.1 自动学习简介

自动学习功能介绍

ModelArts自动学习是帮助人们实现模型的低门槛、高灵活、零代码的定制化模型开发工具。自动学习功能根据标注数据自动设计模型、自动调参、自动训练、自动压缩和部署模型。开发者无需专业的开发基础和编码能力，只需上传数据，通过自动学习界面引导和简单操作即可完成模型训练和部署。

当前自动学习支持快速创建图像分类、物体检测、预测分析、声音分类和文本分类模型的定制化开发。可广泛应用在工业、零售安防等领域。

- 图像分类：识别图片中物体的类别。
- 物体检测：识别出图片中每个物体的位置和类别。
- 预测分析：对结构化数据做出分类或数值预测。
- 声音分类：对环境中不同声音进行分类识别。
- 文本分类：识别一段文本的类别。

自动学习流程介绍

使用ModelArts自动学习开发AI模型无需编写代码，您只需上传数据、创建项目、完成数据标注、发布训练、然后将训练的模型部署上线。具体流程请参见**图4-1**。新版自动学习中，该流程可完全由Workflow进行承载，如**图4-2**。开发者可以通过Workflow进

行有向无环图（Directed Acyclic Graph，DAG）的开发，整个DAG的执行就是有序的任务执行模板，依次执行从数据标注、数据集版本发布、模型训练、模型注册到服务部署环节。如果想了解更多关于Workflow您可以参考**Workflow简介**。

图 4-1 自动学习操作流程

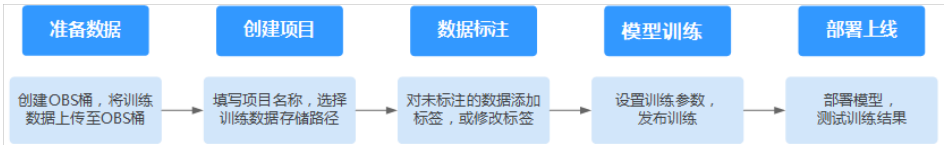


图 4-2 Workflow 运行流程

项目类型介绍

- **图像分类**

  图像分类项目，是对图像进行分类。需要添加图片并对图像进行分类标注，完成图片标注后开始模型训练，即可快速生成图像分类模型。可应用于商品的自动分类、运输车辆种类识别和残次品的自动分类等。例如质量检查的场景，则可以上传产品图片，将图片标注“合格”、“不合格”，通过训练部署模型，实现产品的质检。

- **物体检测**

  物体检测项目，是检测图片中物体的类别与位置。需要添加图片，用合适的框标注物体作为训练集，进行训练输出模型。适用于一张图片中要识别多个物体或者物体的计数等。可应用于园区人员穿戴规范检测和物品摆放的无人巡检。

- **预测分析**

  预测分析项目，是一种针对结构化数据的模型自动训练应用，能够对结构化数据进行分类或者数据预测。可用于用户画像分析，实现精确营销。也可应用于制造设备预测性维护，根据设备实时数据的分析，进行故障识别。

- **声音分类**

  声音分类项目，是识别一段音频中是否包含某种声音。可应用于生产或安防场景的异常声音监控。

- **文本分类**

  文本分类项目，识别一段文本的类别。可应用于情感分析或新闻分类等场景。

4.2 使用自动学习实现图像分类

4.2.1 准备图像分类数据

使用ModelArts自动学习构建模型时，您需要将数据上传至对象存储服务（OBS）中。OBS桶需要与ModelArts在同一区域。

数据集要求

- 保证图片质量：不能有损坏的图片，目前支持的格式包括jpg、jpeg、bmp、png。
- 不要把明显不同的多个任务数据放在同一个数据集内。
- 每一类数据尽量多，尽量均衡。期望获得良好效果，图像分类项目中，至少有两种以上的分类，每种分类的样本不少于20张。
- 为了保证模型的预测准确度，训练样本跟真实使用场景尽量相似。
- 为保证模型的泛化能力，数据集尽量覆盖可能出现的各种场景。

数据上传至 OBS

在本文档中，采用通过OBS管理控制台将数据上传至OBS桶。

上传OBS的文件规范：

- 文件名规范：不能有+、空格、制表符。
- 如不需要提前上传训练数据，请创建一个空文件夹用于存放工程后期生成的文件。如：“/bucketName/data-cat”。
- 如需要提前上传待标注的图片，请创建一个空文件夹，然后将图片文件保存在该文件夹下，图片的目录结构如：“/bucketName/data-cat/cat.jpg”。
- 如您将已标注好的图片上传至OBS桶，请按照如下规范上传。
  - 图像分类数据集要求将标注对象和标注文件存储在同一目录，并且一一对应，例如标注对象文件名为“10.jpg”，那么标注文件的文件名应为“10.txt”。

    数据文件存储示例：

    ```
    ├─<dataset-import-path>
    │       10.jpg
    │       10.txt
    │       11.jpg
    │       11.txt
    │       12.jpg
    │       12.txt
    ```

  - 只支持JPG、JPEG、PNG、BMP格式的图片。在OBS管理控制台上传时，单张图片的大小不能超过5MB，单次上传的图片总大小不能超过8MB，数据量大时推荐使用OBS Browser+上传 。
  - 标签名是由中文、大小写字母、数字、中划线或下划线组成，且不超过32位的字符串。
  - 图像分类标签“.txt”规范如下。

    一行一个标签：

    ```
    flower
    book
    ...
    ```

上传OBS操作步骤：

执行如下操作，将数据上传到OBS中，以便用于模型训练和构建。

1. 登录OBS管理控制台，在ModelArts同一区域内**创建桶**。如果已存在可用的桶，需确保OBS桶与ModelArts在同一区域。
2. 参考**上传文件**，将本地数据上传至OBS桶中。如果您的数据较多，推荐OBS Browser+上传数据或上传文件夹。上传的数据需满足此类型自动学习项目的数据集要求。

说明

在上传数据时，请选择非加密桶进行上传，否则会由于加密桶无法解密导致后期的训练失败。

创建数据集

数据准备完成后，需要创建相应项目支持的类型的数据集，具体操作请参考**创建ModelArts数据集**。

4.2.2 创建图像分类项目

ModelArts自动学习，包括图像分类、物体检测、预测分析、声音分类和文本分类项目。您可以根据业务需求选择创建合适的项目。您需要执行如下操作来创建自动学习项目。

创建项目

1. 登录ModelArts管理控制台，在左侧导航栏选择“开发空间 > 自动学习”，进入自动学习页面。
2. 在您需要的自动学习项目列表中，单击“创建项目”，进入创建自动学习项目界面。
3. 在创建自动学习项目页面，参考**表4-1**填写相应参数。

表 4-1 参数说明

参数	说明
“名称”	项目的名称。 ● 名称只能包含数字、字母、下划线和中划线，长度不能超过64位且不能为空。 ● 名称请以字母开头。 ● 名称不允许重复。
“描述”	对项目的简要描述。
“数据集”	可在右侧下拉框选择已有数据集，或单击“创建数据集”前往新建数据集。 ● 已有数据集：在“数据集”右侧的下拉框中选择，仅展示同类型的数据集供选择。 ● 创建数据集：前往创建数据集页面创建一个新的数据集。具体操作请参考**创建ModelArts数据集**。
“输出路径”	选择自动学习数据输出的统一OBS路径。 **说明** “输出路径”是存储自动学习在运行过程中所有产物的路径。
“训练规格”	选择自动学习训练节点所使用的资源规格，以实际界面显示为准，将会根据不同的规格计费。 **说明** ● 如果您购买了套餐包，可优先选择您对应规格的套餐包，在“配置费用”处会显示您的套餐余量，以及超出的部分如何计费，请您关注，避免造成不必要的资源浪费。

4. 单击“创建项目”，图像分类项目创建成功后页面自动跳转到“自动学习工作流”。
5. 图像分类项目的工作流，将依次运行如下节点：
   a. 数据标注：对您的数据标注情况进行确认。
   b. 数据集版本发布：将已完成标注的数据进行版本发布。
   c. 数据校验：对您的数据集的数据进行校验，是否存在数据异常。
   d. 图像分类：将发布好的数据集版本进行训练，生成对应的模型。
   e. 模型注册：将训练后的结果注册到模型管理中。
   f. 服务部署：将生成的模型部署为在线服务。

快速查找创建好的项目

在自动学习总览页，您可以通过搜索框，根据自动学习的属性类型（项目名称）快速搜索过滤到相应的工作流，可节省您的时间。

1. 登录ModelArts管理控制台，在左侧导航栏选择“开发空间>自动学习”，进入自动学习总览页面。
2. 在自动学习列表上方的搜索框中，根据您需要的属性类型，例如，名称、状态、项目类型、当前节点、标签等，过滤出相应的工作流。
3. 单击搜索框右侧的  按钮，可选择自动学习的基础设置，需要的显示列。

   表格内容折行：默认为关闭状态，启用此能力可让表格内容自动折行，禁用此功能可截断文本。

   操作列：默认为关闭状态，启用此能力可让操作列固定在最后一列永久可见。

   自定义显示列：默认所有显示项全部勾选，您可以根据实际需要定义您的显示列。

图 4-3 表格显示设置

4. 单击“确定”即可按照设置好的显示列进行显示。
5. 同时可支持对自动学习项目显示页进行排序，单击表头中的箭头，就可对该列进行排序。

4.2.3 标注图像分类数据

由于模型训练过程需要大量有标签的图片数据，因此在模型训练之前需对没有标签的图片添加标签。通过ModelArts您可对图片进行一键式批量添加标签，快速完成对图片的标注操作，也可以对已标注图片修改或删除标签进行重新标注。

说明

请确保数据集中已标注的图片不低于100张，否则会导致数据集校验环节不通过，影响您的模型训练。

项目创建完成后，将会自动跳转至自动学习页面，并开始运行。单击“数据标注”节点，当状态变为“等待操作”时，需要手动进行确认数据集中的数据标注情况，也可以对数据集中的数据进行标签的修改，数据的增加或删减。

图 4-4 数据标注节点状态

图片标注

1. 在新版自动学习页面的数据标注节点单击“实例详情”按钮，前往数据标注页面。

图 4-5 单击实例详情

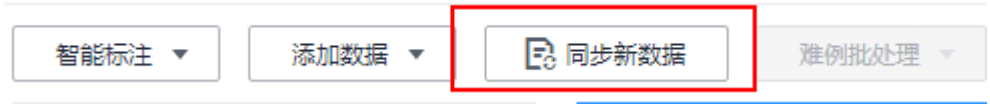
2. 依次勾选待标注的图片，或勾选“选择当前页”选中该页面所有图片，在页面右侧进行图片标注。
3. 选中图片后，在页面右侧“添加标签”，输入“标签名”或从下拉列表中选择已添加的标签。单击“确定”，完成选中图片的标注操作。例如，您可以选择多张图片，按照花朵种类将图片标注为“tulips”。同样选择其他未标注分类图片，将其标注为“sunflowers”、“roses”等。标注完成后，图片将存储至“已标注”页签下。
   a. 图片标注不支持多标签，即一张图片不可以添加多个标签。
   b. 标签名是由中文、大小写字母、数字、中划线或下划线组成。
4. 当图片目录中所有图片都完成标注后，您可以在“已标注”页签下查看已完成标注的图片，或者通过右侧的“全部标签”列表，了解当前已完成的标签名称和标签数量。

同步或添加图片

在“数据标注”节点单击“实例详情”进入数据标注页面，数据标注的图片来源有两种，通过本地添加图片和同步OBS中的图片数据。

图 4-6 添加本地图片

图 4-7 同步 OBS 图片数据



- **添加数据**：您可以将本地图片快速添加到ModelArts，同时自动上传至创建项目时所选择的OBS路径中。单击“添加数据”，根据弹出的对话框的引导，输入正确的数据并添加。
- **同步新数据**：将图片数据上传至创建项目时指定的OBS目录，然后单击“同步新数据”，快速将原OBS目录中的新数据添加到ModelArts数据集。
- **删除图片**：您可以依次单击选中图片进行删除，也可以勾选“选择当前页”对该页面所有图片进行删除。

说明

所有的删除操作均不可恢复，请谨慎操作。

修改标注

当数据完成标注后，您还可以进入已标注页签，对已标注的数据进行修改。

- **基于图片修改**

  在数据标注页面，单击“已标注”页签，然后在图片列表中选中待修改的图片（选择一个或多个）。在右侧标签信息区域中对图片信息进行修改。

  - 添加标签：在“标签名”右侧文本框中，选择已有标签或输入新的标签名，然后单击，为选中图片增加标签。
  - 修改标签：在“选中文件标签”区域中，单击操作列的编辑图标，然后在文本框中输入正确的标签名，然后单击确定图标完成修改。

    图 4-8 编辑标签

  - 删除标签：在“选中文件标签”区域中，单击操作列的删除该标签。

- **基于标签修改**

  在数据标注概览页，单击右侧的“标签管理”，即可显示全部标签的信息。

  图 4-9 全部标签的信息

  - 修改标签：在需要修改的标签的“操作”列，单击“修改”，输入修改后的标签，单击“确定”即可。
  - 删除标签：选择对应的标签，单击操作列的“删除”，在弹出的“删除标签”对话框中单击“确定”即可删除对应的标签。

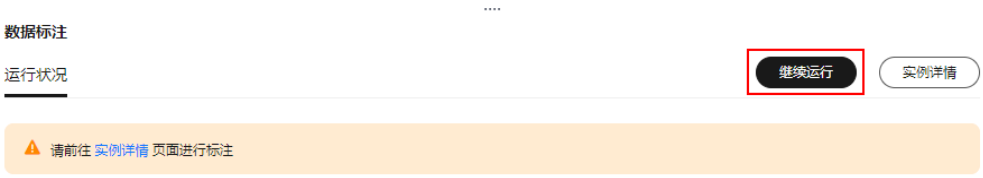
    说明

    删除后无法再恢复，请谨慎操作。

继续运行

完成数据的确认之后，返回自动学习的页面，在数据标注节点单击“继续运行”，工作流将会继续依次运行直到所有节点运行成功。

图 4-10 继续运行



4.2.4 训练图像分类模型

完成图片标注后，可进行模型的训练。模型训练的目的是得到满足需求的图像分类模型。请参考**前提条件**确保已标注的图片符合要求，否则数据集校验将会不通过。

前提条件

1. 请确保您的数据集中的已标注的图片不低于100张。
2. 请确保您的数据集中至少存在2种以上的图片分类，且每种分类的图片不少于5张。

操作步骤

1. 参考**标注图像分类数据**章节，确保您的数据已全部完成标注。
2. 在新版自动学习页面，单击数据标注节点的“继续运行”按钮，然后等待工作流按顺序进入训练节点即可。
3. 模型将会自动进入训练，无需人工介入，训练时间相对较长，建议您耐心等待。如果关闭或退出此页面，系统仍然在执行训练操作。
4. 在“图像分类”节点中，待训练状态由“运行中”变为“运行成功”，即完成了模型的自动训练。
5. 训练完成后，您可以单击“图像分类”节点上方的  按钮，查看相关指标信息，如“准确率”、“评估结果”等。评估结果参数说明请参见**表4-2**。

表 4-2 评估结果参数说明

参数名称	参数含义	说明
recall	召回率	被用户标注为某个分类的所有样本中，模型正确预测为该分类的样本比率，反映模型对正样本的识别能力。
precision	精确率	被模型预测为某个分类的所有样本中，模型正确预测的样本比率，反映模型对负样本的区分能力。
accuracy	准确率	所有样本中，模型正确预测的样本比率，反映模型对样本整体的识别能力。
f1	F1值	F1值是模型精确率和召回率的加权调和平均，用于评价模型的好坏，当F1较高时说明模型效果较好。

说明

同一个自动学习项目可以训练多次，每次训练会注册一个新的模型版本。如第一次训练版本号为“0.0.1”，下一个版本为“0.0.2”。基于训练版本可以对训练模型进行管理。当训练的模型达到目标后，再执行模型部署的操作。

4.2.5 部署图像分类服务

模型部署

模型部署操作即将模型部署为在线服务，并且提供在线的测试UI与监控能力。完成模型训练后，可选择准确率理想且训练状态为“运行成功”的版本部署上线。具体操作步骤如下。

1. 在“运行节点”页面中，待服务部署节点的状态变为“等待输入”时，双击“服务部署”进入配置详情页，完成资源的参数配置操作。
2. 在服务部署页面，选择模型部署使用的资源规格。
   - 模型来源：默认为生成的模型。
   - 选择模型及版本：自动匹配当前使用的模型版本，支持选择版本。
   - 资源池：默认公共资源池。
   - 分流：默认为100，输入值必须是0-100之间。
   - 计算节点规格：请根据界面显示的列表，选择可用的规格，置灰的规格表示当前环境无法使用。如果公共资源池下规格为空数据，表示当前环境无公共资源。建议使用专属资源池，或者联系系统管理员创建公共资源池。
   - 计算节点个数：默认为1，输入值必须是1-5之间的整数。
   - 是否自动停止：启用该参数并设置时间后，服务将在指定时间后自动停止。如果不启用此参数，在线服务将一直运行，同时一直收费，自动停止功能可以帮您避免产生不必要的费用。默认开启自动停止功能，且默认值为“1小时后”。

     目前支持设置为“1小时后”、“2小时后”、“4小时后”、“6小时后”、“自定义”。如果选择“自定义”的模式，可在右侧输入框中输入1~24范围内的任意整数。

     说明

     如果您购买了套餐包，计算节点规格可选择您的套餐包，同时在“配置费用”页签还可查看您的套餐包余量以及超出部分的计费方式，请您务必关注，避免造成不必要的资源浪费。
3. 完成资源配置后，单击“继续运行”，服务部署节点将继续运行，直至状态变为“运行成功”，至此，已将模型部署为在线服务。

服务测试

服务部署节点运行成功后，单击“实例详情”可跳转至对应的在线服务详情页面。单击“预测”页签，进行服务测试。

图 4-11 服务测试

下面的测试，是您在自动学习图像分类项目页面将模型部署上线之后进行服务测试的操作步骤。

1. 模型部署完成后，“在服务部署”节点，单击“实例详情”按钮，进入服务预测界面，在“预测”页签单击“上传”，选择本地图片进行测试。
2. 单击“预测”进行测试，预测完成后，右侧“预测结果”区域输出标签名称“sunflowers”和检测的评分。如模型准确率不满足预期，可在“数据标注”页签中添加图片并进行标注，重新进行模型训练及模型部署。预测结果中的参数说明请参见表4-3。如果您对模型预测结果满意，可根据界面提示调用接口访问在线服务。

   目前只支持jpg、jpeg、bmp、png格式的图片。

图 4-12 预测结果

表 4-3 预测结果中的参数说明

参数	说明
predicted_label	表示图片预测的标签。
scores	表示Top5标签的预测置信度。

说明

- 由于“运行中”的在线服务将持续耗费资源，如果不需再使用此在线服务，建议在“在线服务”的操作列单击“更多>停止”，避免产生不必要的费用。如果需要继续使用此服务，可单击“启动”恢复。
- 如果您启用了自动停止功能，服务将在指定时间后自动停止，不再产生费用。

4.3 使用自动学习实现物体检测

4.3.1 准备物体检测数据

使用ModelArts自动学习构建模型时，您需要将数据上传至对象存储服务（OBS）中。OBS桶需要与ModelArts在同一区域。

数据集要求

- 保证图片质量：不能有损坏的图片；目前支持的格式包括jpg、jpeg、bmp、png。
- 不要把明显不同的多个任务数据放在同一个数据集内。
- 为了保证模型的预测准确度，训练样本跟真实使用场景尽量相似。
- 为保证模型的泛化能力，数据集尽量覆盖可能出现的各种场景。
- 物体检测数据集中，如果标注框坐标超过图片，将无法识别该图片为已标注图片。

数据上传至 OBS

在本文档中，采用通过OBS管理控制台将数据上传至OBS桶。

上传OBS的文件规范：

- 文件名规范，不能有中文，不能有+、空格、制表符。
- 如不需要提前上传训练数据，请创建一个空文件夹用于存放工程后期生成的文件。如：“/bucketName/data-cat”。
- 如需要提前上传待标注的图片，请创建一个空文件夹，然后将图片文件保存在该文件夹下，图片的目录结构如：“/bucketName/data-cat/cat.jpg”。
- 如您将已标注好的图片上传至OBS桶，请按照如下规范上传。
  - 物体检测数据集要求用户将标注对象和标注文件存储在同一目录，并且一一对应。例如标注对象文件名为“IMG_20180919_114745.jpg”，那么标注文件的文件名应为“IMG_20180919_114745.xml”。

    物体检测的标注文件需要满足PASCAL VOC格式，格式详细说明请参见 **表 4-4**。

    数据存储示例：

    ```
    ├─<dataset-import-path>
    │       IMG_20180919_114732.jpg
    │       IMG_20180919_114732.xml
    │       IMG_20180919_114745.jpg
    │       IMG_20180919_114745.xml
    │       IMG_20180919_114945.jpg
    │       IMG_20180919_114945.xml
    ```

  - 只支持JPG、JPEG、PNG、BMP格式的图片，在OBS管理控制台上传时，单张图片的大小不能超过5MB，单次上传的图片总大小不能超过8MB，数据量大时推荐使用OBS Browser+上传 。
  - 标签名是由中文、大小写字母、数字、中划线或下划线组成，且不超过32位的字符串。

表 4-4 PASCAL VOC 格式说明

字段	是否必选	说明
folder	是	表示数据源所在目录。
filename	是	被标注文件的文件名。

字段	是否必选	说明
size	是	表示图像的像素信息。 ● width：必选字段，图片的宽度。 ● height：必选字段，图片的高度。 ● depth：必选字段，图片的通道数。
segmented	是	表示是否用于分割。
object	是	表示物体检测信息，多个物体标注会有多个object体。 ● name：必选字段，标注内容的类别。 ● pose：必选字段，标注内容的拍摄角度。 ● truncated：必选字段，标注内容是否被截断（0表示完整）。 ● occluded：必选字段，标注内容是否被遮挡（0表示未遮挡） ● difficult：必选字段，标注目标是否难以识别（0表示容易识别）。 ● confidence：可选字段，标注目标的置信度，取值范围0-1之间。 ● bndbox：必选字段，标注框的类型，标注信息请参见 表4-5 。

表 4-5 标注框类型描述

type	形状	标注信息
bndbox	矩形框	左上和右下两个点坐标。 <xmin>100<xmin> <ymin>100<ymin> <xmax>200<xmax> <ymax>200<ymax>

标注文件示例：

```
<annotation>
  <folder>test_data</folder>
  <filename>260730932.jpg</filename>
  <size>
     <width>767</width>
     <height>959</height>
     <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
     <name>bag</name>
     <pose>Unspecified</pose>
```

```
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <bndbox>
      <xmin>108</xmin>
      <ymin>101</ymin>
      <xmax>251</xmax>
      <ymax>238</ymax>
    </bndbox>
  </object>
</annotation>
```

上传OBS的操作步骤：

执行如下操作，将数据导入到您的数据集中，以便用于模型训练和构建。

1. 登录OBS管理控制台，在ModelArts同一区域内**创建桶**。如果已存在可用的桶，需确保OBS桶与ModelArts在同一区域。
2. 参考**上传文件**，将本地数据上传至OBS桶中。如果您的数据较多，推荐OBS Browser+上传数据或上传文件夹。上传的数据需满足此类型自动学习项目的数据集要求。

说明

- 在上传数据时，请选择非加密桶进行上传，否则会由于加密桶无法解密导致后期的训练失败。
- 用于训练的图片，至少有1种以上的分类，每种分类的图片数不少50张。

创建数据集

数据准备完成后，需要创建相应项目支持的类型的数据集，具体操作请参考**创建ModelArts数据集**。

4.3.2 创建物体检测项目

ModelArts自动学习，包括图像分类、物体检测、预测分析、声音分类和文本分类项目。您可以根据业务需求选择创建合适的项目。您需要执行如下操作来创建自动学习项目。

创建项目

1. 登录ModelArts管理控制台，在左侧导航栏单击“开发空间>自动学习”，进入新版自动学习页面。
2. 在您需要的自动学习项目列表中，单击“创建项目”，进入创建自动学习项目界面。
3. 在创建自动学习项目页面，参考**表4-6**填写相应参数。

表 4-6 参数说明

参数	说明
“名称”	项目的名称。 ● 名称只能包含数字、字母、下划线和中划线，长度不能超过64位且不能为空。 ● 名称请以字母开头。 ● 名称不允许重复。
“描述”	对项目的简要描述。
“数据集”	可在右侧下拉框选择已有数据集，或单击“创建数据集”前往新建数据集。 ● 已有数据集：在“数据集”右侧的下拉框中选择，仅展示同类型的数据集供选择。 ● 创建数据集：前往创建数据集页面创建一个新的数据集。具体操作请参考**创建ModelArts数据集**。
“输出路径”	选择自动学习数据输出的统一OBS路径。 **说明** “输出路径”是存储自动学习在运行过程中所有产物的路径。
“训练规格”	选择自动学习训练节点所使用的资源规格，以实际界面显示为准，将会根据不同的规格计费。 **说明** ● 如果您购买了套餐包，可优先选择您对应规格的套餐包，在“配置费用”处会显示您的套餐余量，以及超出的部分如何计费，请您关注，避免造成不必要的资源浪费。

4. 单击“创建项目”，物体检测项目创建成功后页面自动跳转到“自动学习工作流”。
5. 物体检测项目的工作流，将依次运行如下节点：
   a. 数据标注：对您的数据进行标注情况确认。
   b. 数据集版本发布：将已完成标注的数据进行版本发布。
   c. 数据校验：对您的数据集的数据进行校验，是否存在数据异常。
   d. 物体检测：将发布好的数据集版本进行训练，生成对应的模型。
   e. 模型注册：将训练后的结果注册到模型管理中。
   f. 服务部署：将生成的模型部署为在线服务。

快速查找创建好的项目

在自动学习总览页，您可以通过搜索框，根据自动学习的属性类型（项目名称）快速搜索过滤到相应的工作流，可节省您的时间。

1. 登录ModelArts管理控制台，在左侧导航栏选择“开发空间>自动学习”，进入自动学习总览页面。
2. 在自动学习列表上方的搜索框中，根据您需要的属性类型，例如，名称、状态、项目类型、当前节点、标签等，过滤出相应的工作流。

3. 单击搜索框右侧的  按钮，可选择自动学习的基础设置，需要的显示列。

   表格内容折行：默认为关闭状态，启用此能力可让表格内容自动折行，禁用此功能可截断文本。

   操作列：默认为关闭状态，启用此能力可让操作列固定在最后一列永久可见。

   自定义显示列：默认所有显示项全部勾选，您可以根据实际需要定义您的显示列。

   图 4-13 表格显示设置

4. 单击“确定”即可按照设置好的显示列进行显示。

5. 同时可支持对自动学习项目显示页进行排序，单击表头中的箭头  ，就可对该列进行排序。

4.3.3 标注物体检测数据

物体检测之前，首先需考虑如何设计标签，标签设计需要对应所检测图片的明显特征，并且选择的标签比较容易识别（画面主体物与背景区分度较高），每个标签就是对所检测图片期望识别的全部结果。物体的标签设计完成之后，基于设计好的标签准备该图片的数据，每种需识别出的标签，建议应在所有图片个数相加超过100张，如果某些图片的标签具有相似性，则需要更多的图片。用于训练的图片，至少有1种以上的分类，每种分类的图片数不少50张。

- 标注时，类内方差尽量要小。即相同类别的标注，尽量近似；不同类别的标注，尽量保持差距较大。
- 标记的每个标签尽量和背景有较大的区分度。
- 物体检测标注，需要保证目标框内物体的完整性；针对图片中存在多个物体的情形，做到不重标、不漏标。

项目创建完成后，将会自动跳转至新版自动学习页面，并开始运行，当数据标注节点的状态变为“等待操作”时，需要手动进行确认数据集中的数据标注情况，也可以对数据集中的数据进行标签的修改，数据的增加或删减。

图 4-14 数据标注节点状态

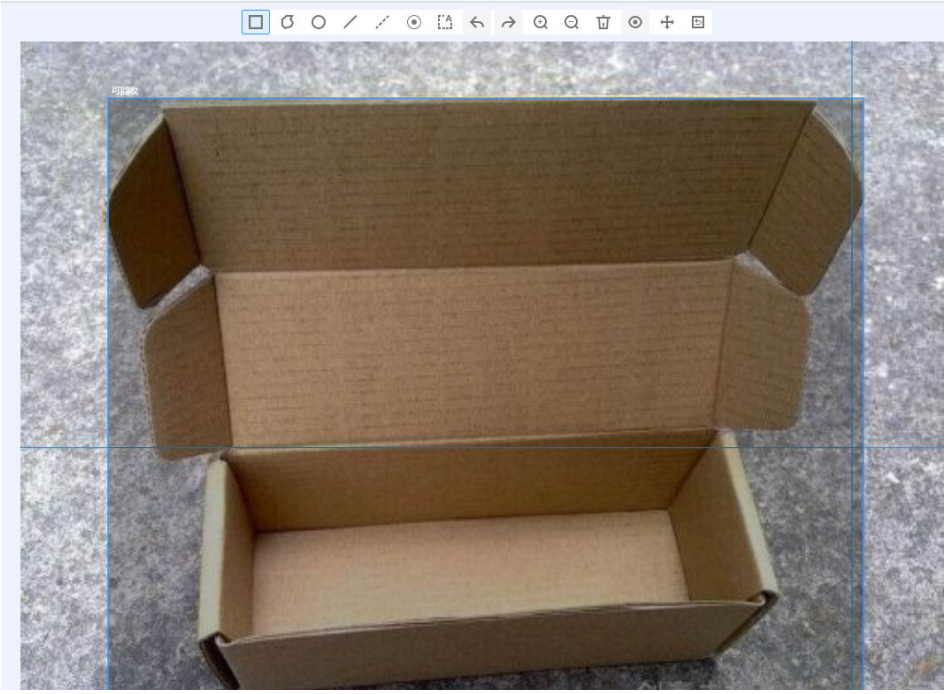
图片标注

1. 在新版自动学习页面单击“实例详情”按钮，前往数据标注页面。单击任意一张图片，进入图片标注界面。
2. 用鼠标框选图片中的物体所在区域，然后在弹出的对话框中选择标签颜色，输入标签名称，例如此示例中的“yunbao”，按“Enter”键完成此标签的添加。标注完成后，左侧图片目录中此图片的状态将显示为“已标注”。

   数据标注的更多说明：

   - 您可以在图片上方或下方单击左右切换键，或者按键盘的左右方向键，选择其他图片，重复上述操作继续进行图片标注。如果一张图片有多个物体，您可以标注多处。
   - 同一个物体检测自动学习项目内，可以增加多个标签，且标签可选择不同颜色，方便识别。使用鼠标完成物体框选后，在弹出的对话框中，选择新的颜色，输入新的标签名称，即可添加一个新的标签。
   - 自动学习项目中，物体检测仅支持矩形标注框。在“资产管理 > 数据集”功能中，物体检测类型的数据集，支持更多类型的标注框。
   - 在标注窗口中，您可以滚动鼠标，放大或缩小图片，方便您快速定位到物体位置。

图 4-15 物体检测图片标注



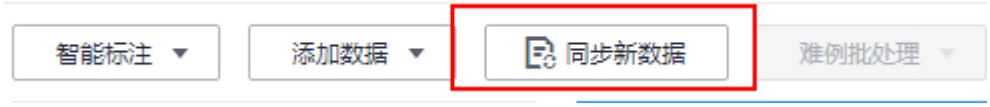
3. 当图片目录中所有图片都完成标注后，返回“自动学习工作流”页面，单击“继续运行”按钮，工作流将会自动发布数据标注版本，并进行下一步训练步骤。

同步或添加图片

在“数据标注”节点单击“实例详情”进入数据标注页面，数据标注的图片来源有两种，通过本地添加图片和同步OBS中的图片数据。

图 4-16 添加本地图片

图 4-17 同步 OBS 图片数据



- **添加数据**：您可以将本地图片快速添加到ModelArts，同时自动上传至创建项目时所选择的OBS路径中。单击“添加数据”，根据弹出的对话框的引导，输入正确的数据并添加。
- **同步新数据**：将图片数据上传至创建项目时指定的OBS目录，然后单击“同步新数据”，快速将原OBS目录中的新数据添加到ModelArts数据集。
- **删除图片**：您可以依次单击选中图片进行删除，也可以勾选“选择当前页”对该页面所有图片进行删除。

  说明

  所有的删除操作均不可恢复，请谨慎操作。

修改标注

当数据完成标注后，您还可以进入已标注页签，对已标注的数据进行修改。

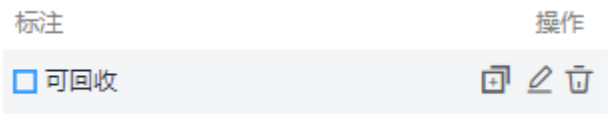
- **基于图片修改**

  在数据集详情页面，单击“已标注”页签，然后在图片列表中选中待修改的图片，在右侧“标注”区域中对图片信息进行修改。

  - 修改标签：“标注”区域中，单击编辑按钮，在文本框中输入正确的标签名，然后单击确定按钮完成修改。标签颜色不支持修改。
  - 删除标签：在“标注”区域中，单击删除按钮，即可删除此图片中的标签。

    标签删除后，单击页面左上角的项目名称离开标注页面。该图片会重新回到“未标注”页签。

    图 4-18 编辑物体检测标签

- **基于标签修改**

  在数据标注作业概览页，单击右侧的“标签管理”，进入标签管理页面，标签管理页展示所有标签信息。

  图 4-19 标签管理页

  - 修改标签：单击操作列的“修改”按钮，然后在弹出的对话框中输入修改后的标签名，然后单击“确定”完成修改。修改后，之前添加了此标签的图片，都将被标注为新的标签名称。
  - 删除标签：单击操作列的“删除”按钮，在弹出的对话框中，根据界面提示选择删除对象，然后单击“确定”。

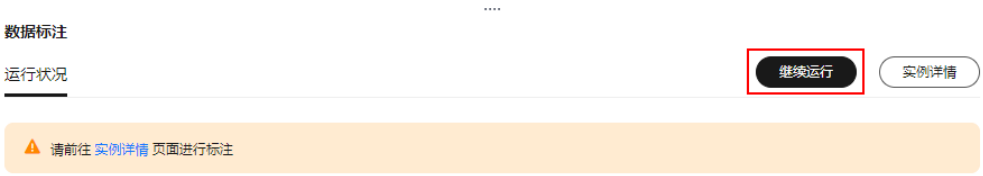
    说明

    删除后的标签无法再恢复，请谨慎操作。

继续运行

完成数据的确认之后，返回新版自动学习的页面，在数据标注节点单击“继续运行”，工作流将会继续依次运行直到所有节点运行成功。

图 4-20 继续运行



4.3.4 训练物体检测模型

自动学习物体检测项目，在图片标注完成后，通过模型训练得到合适的模型版本。

操作步骤

1. 在新版自动学习页面，单击项目名称进入运行总览页面，单击“数据标注”节点的“实例详情”进入数据标注页面，完成数据标注。

   图 4-21 完成数据标注

2. 返回新版自动学习页面，单击数据标注节点的“继续运行”，然后等待工作流按顺序进入训练节点。
3. 模型将会自动进入训练，无需人工介入，训练时间相对较长，建议您耐心等待。如果关闭或退出此页面，系统仍然在执行训练操作。
4. 在“物体检测”节点中，待训练状态由“运行中”变为“运行成功”，即完成模型的自动训练。
5. 训练完成后，您可以单击物体检测节点上方的  按钮，查看相关指标信息，如“准确率”、“评估结果”等。评估结果参数说明请参见表4-7。

   表 4-7 评估结果参数说明

参数	说明
recall：召回率	被用户标注为某个分类的所有样本中，模型正确预测为该分类的样本比率，反映模型对正样本的识别能力。
precision：精确率	被模型预测为某个分类的所有样本中，模型正确预测的样本比率，反映模型对负样本的区分能力。

参数	说明
accuracy：准确率	所有样本中，模型正确预测的样本比率，反映模型对样本整体的识别能力。
f1：F1值	F1值是模型精确率和召回率的加权调和平均，用于评价模型的好坏，当F1较高时说明模型效果较好。

说明

同一个自动学习项目可以训练多次，每次训练会注册一个新的模型一个版本。如第一次训练版本号为“0.0.1”，下一个版本为“0.0.2”。基于训练版本可以对训练模型进行管理。当训练的模型达到目标后，再执行模型部署的操作。

4.3.5 部署物体检测服务

模型部署

模型部署操作即将模型部署为在线服务，并且提供在线的测试UI与监控能力。完成模型训练后，可选择准确率理想且训练状态为“运行成功”的版本部署上线。具体操作步骤如下。

1. 在“运行节点”页面中，待服务部署节点的状态变为“等待输入”时，双击“服务部署”进入配置详情页，完成资源的参数配置操作。
2. 在服务部署页面，选择模型部署使用的资源规格。
   - 模型来源：默认为生成的模型。
   - 选择模型及版本：自动匹配当前使用的模型版本，支持选择版本。
   - 资源池：默认公共资源池。
   - 分流：默认为100，输入值必须是0-100之间。
   - 计算节点规格：请根据界面显示的列表，选择可用的规格，置灰的规格表示当前环境无法使用。如果公共资源池下规格为空数据，表示当前环境无公共资源。建议使用专属资源池，或者联系系统管理员创建公共资源池。
   - 计算节点个数：默认为1，输入值必须是1-5之间的整数。
   - 是否自动停止：启用该参数并设置时间后，服务将在指定时间后自动停止。如果不启用此参数，在线服务将一直运行，同时一直收费，自动停止功能可以帮您避免产生不必要的费用。默认开启自动停止功能，且默认值为“1小时后”。

     目前支持设置为“1小时后”、“2小时后”、“4小时后”、“6小时后”、“自定义”。如果选择“自定义”的模式，可在右侧输入框中输入1~24范围内的任意整数。

     说明

     如果您购买了套餐包，计算节点规格可选择您的套餐包，同时在“配置费用”页签还可查看您的套餐包余量以及超出部分的计费方式，请您务必关注，避免造成不必要的资源浪费。
3. 完成资源配置后，单击“继续运行”，服务部署节点将继续运行，直至状态变为“运行成功”，至此，已将模型部署为在线服务。

服务测试

服务部署节点运行成功后，单击“实例详情”可跳转至对应的在线服务详情页面。单击“预测”页签，进行服务测试。

图 4-22 服务测试

下面的测试，是您在自动学习物体检测项目页面将模型部署上线之后进行服务测试的操作步骤。

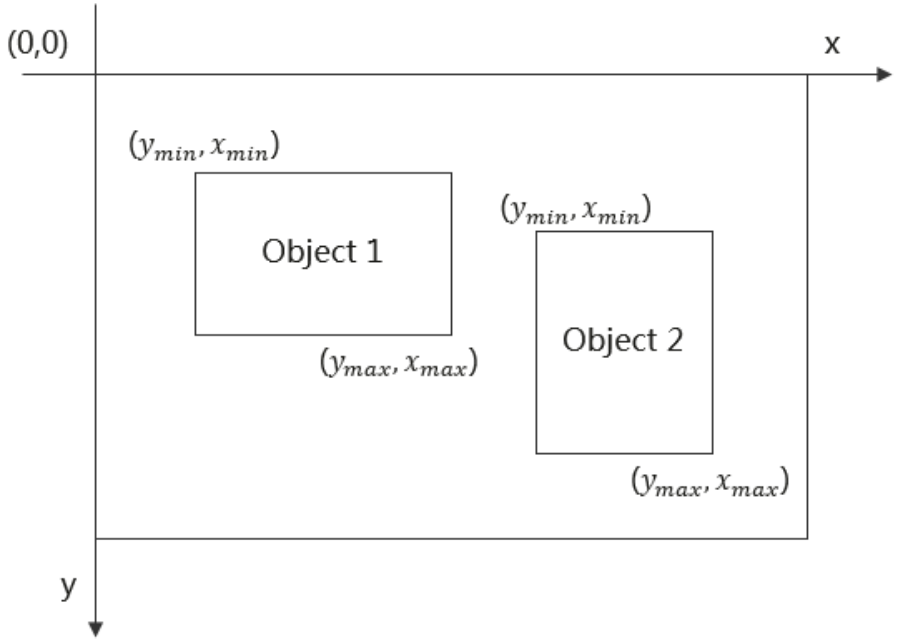
1. 模型部署完成后，“服务部署”节点，单击“实例详情”按钮，进入服务预测界面，在“预测”页签单击“上传”，选择本地图片进行测试。
2. 单击“预测”进行测试，预测完成后，右侧“预测结果”区域输出结果。如模型准确率不满足预期，可在“数据标注”页签中添加图片并进行标注，重新进行模型训练及模型部署。预测结果中的参数说明请参见**表4-8**。如果您对模型预测结果满意，可根据界面提示调用接口访问在线服务。

   目前只支持jpg、jpeg、bmp、png格式的图片。

   表 4-8 预测结果中的参数说明

参数	说明
detection_classes	每个检测框的标签。
detection_boxes	每个检测框的四点坐标（y_min,x_min,y_max,x_max），如**图4-23**所示。
detection_scores	每个检测框的置信度。

图 4-23 检测框的四点坐标示意图



说明

- 由于“运行中”的在线服务将持续耗费资源，如果不需再使用此在线服务，建议在版本管理区域，单击“停止”，即可停止在线服务的部署，避免产生不必要的费用。如果需要继续使用此服务，可单击“启动”恢复。
- 如果您启用了自动停止功能，服务将在指定时间后自动停止，不再产生费用。

4.4 使用自动学习实现预测分析

4.4.1 准备预测分析数据

使用ModelArts自动学习构建预测分析模型时，您需要将数据上传至对象存储服务（OBS）中。OBS桶需要与ModelArts在同一区域，例如OBS桶区域为“北京四”时，必须保证ModelArts管理控制台区域也在“北京四”区域，否则会导致无法获取到相关数据。

数据集要求

预测分析项目中需要使用到的数据集为表格数据集，数据格式支持csv格式。表格数据集的具体介绍请参见**表格数据集**。

说明

将原始.xlsx格式的数据转换为.csv格式的数据的方法如下：

将原始表格数据（.xlsx）另存。单击“文件>另存为”，选择本地地址后，下拉选择“保存类型”为“CSV (逗号分隔)(*.csv)”单击“保存”，在弹窗中，单击“确定”后就可以将.xlsx格式数据集转换为.csv格式。

表格数据集对训练数据的要求：

- 训练数据列数一致，总数据量不少于100条不同数据（有一个特征取值不同，即视为不同数据）。
- 训练数据列内容不能有时间戳格式（如：yy-mm-dd、yyyy-mm-dd等）的数据。
- 如果某一列的取值只有一种，会被视为无效列。请确保标签列的取值至少有两个且无数据缺失。

  说明

  标签列指的是在训练作业中被指定为训练目标的列，即最终通过该数据集训练得到模型时的输出（预测项）。
- 除标签列外数据集中至少还应包含两个有效特征列（列的取值至少有两个且数据缺失比例低于10%）。
- 当前由于特征筛选算法限制，预测数据列建议放在数据集最后一列，否则可能导致训练失败。

表格数据集示例：

以银行存款预测数据集为例：根据预测人的年龄、工作类型、婚姻状况、文化程度、是否有房贷和是否有个人贷款。

表 4-9 数据源的具体字段及意义

字段名	含义	类型	描述
attr_1	年龄	Int	表示客户的年龄。
attr_2	职业	String	表示客户所从事的职业。
attr_3	婚姻情况	String	表示客户是否结婚或已离异。
attr_4	教育情况	String	表示客户受教育的程度。
attr_5	房产情况	String	表示客户名下是否有房产。
attr_6	贷款情况	String	表示客户名下是否有贷款。
attr_7	存款情况	String	表示客户名下是否有存款。

表 4-10 数据集样本数据

attr_1	attr_2	attr_3	attr_4	attr_5	attr_6	attr_7
31	blue-collar	married	secondary	yes	no	no
41	management	married	tertiary	yes	yes	no
38	technician	single	secondary	yes	no	no
39	technician	single	secondary	yes	no	yes

attr_1	attr_2	attr_3	attr_4	attr_5	attr_6	attr_7
39	blue-collar	married	secondary	yes	no	no
39	services	single	unknown	yes	no	no

数据上传至 OBS

在本文档中，采用通过OBS管理控制台将数据上传至OBS桶。

上传OBS的文件规范：

预测分析项目的OBS数据路径需符合以下规则：

- 输入数据的OBS路径应指向数据文件，且文件不能直接放在OBS桶的根目录下，应该存放在OBS桶的文件夹内。如：“/obs-xxx/data/input.csv”。
- 输入数据的格式必须为csv格式，有效数据行数必须大于100行。列数必须小于200列，数据总大小不能超过100MB。

上传OBS操作步骤：

执行如下操作，将数据导入到您的数据集中，以便用于模型训练和构建。

1. 登录OBS管理控制台，在ModelArts同一区域内**创建桶**。如果已存在可用的桶，需确保OBS桶与ModelArts在同一区域。
2. 参考**上传文件**，将本地数据上传至OBS桶中。如果您的数据较多，推荐OBS Browser+上传数据或上传文件夹。上传的数据需满足此类型自动学习项目的数据集要求。

说明

在上传数据时，请选择非加密桶进行上传，否则会由于加密桶无法解密导致后期的训练失败。

创建数据集

数据准备完成后，需要创建相应项目支持的类型的数据集，具体操作请参考**创建ModelArts数据集**。

常见问题

使用从OBS选择的数据创建表格数据集如何处理Schema信息?

Schema信息表示表格的列名和对应类型，需要跟导入数据的列数保持一致。

- 如果您的原始表格中已包含表头，需要开启“导入是否包含表头”开关，系统会导入文件的第一行（表头）作为列名，无需再手动修改Schema信息。
- 如果您的原始表格中没有表头，需关闭“导入是否包含表头”开关，从OBS选择数据后，Schema信息的列名默认为表格中的第一行数据，请更改Schema信息中的“列名”为attr_1、attr_2、……、attr_n，其中attr_n为最后一列，代表预测列。

4.4.2 创建预测分析项目

ModelArts自动学习，包括图像分类、物体检测、预测分析、声音分类和文本分类项目。您可以根据业务需求选择创建合适的项目。您需要执行如下操作来创建自动学习项目。

创建项目

1. 登录ModelArts管理控制台，在左侧导航栏单击“开发空间>自动学习”，进入新版自动学习页面。
2. 在您需要的自动学习项目列表中。例如选择预测分析项目，单击“创建项目”，进入创建自动学习项目界面。
3. 在创建自动学习项目页面，计费模式默认“按需计费”，参考**表4-11**填写相应参数。

表 4-11 参数说明

参数	说明
“名称”	项目的名称。 ● 名称只能包含数字、字母、下划线和中划线，长度不能超过64位且不能为空。 ● 名称请以字母开头。 ● 名称不允许重复。
“描述”	对项目的简要描述。
“数据集”	可在右侧下拉框选择已有数据集，或单击“创建数据集”前往新建数据集。 ● 已有数据集：在“数据集”右侧的下拉框中选择，仅展示同类型的数据集供选择。 ● 创建数据集：前往创建数据集页面创建一个新的数据集。具体操作请参考**创建ModelArts数据集**。
“标签列”	可自行选择您需要预测的列名。 标签列是预测模型的输出。模型训练步骤将使用全部信息训练预测模型，该模型以其他列的数据为输入，以标签列的预测值为输出。模型部署步骤将使用预测模型发布在线预测服务。
“输出路径”	选择自动学习数据输出的统一OBS路径。 **说明** “输出路径”是存储自动学习在运行过程中所有产物的路径。
“训练规格”	选择自动学习训练节点所使用的资源规格，以实际界面显示为准，将会根据不同的规格计费。 **说明** ● 如果您购买了套餐包，可优先选择您对应规格的套餐包，在“配置费用”处会显示您的套餐余量，以及超出的部分如何计费，请您关注，避免造成不必要的资源浪费。

4. 单击“创建项目”，预测分析项目创建成功后页面自动跳转到“自动学习工作流”。

5. 预测分析项目的工作流，将依次运行如下节点：
   a. 数据集版本发布：将已完成确认的数据进行版本发布。
   b. 数据校验：对您的数据集的数据进行校验，是否存在数据异常。
   c. 预测分析：将发布好的数据集版本进行训练，生成对应的模型。
   d. 模型注册：将训练后的结果注册到模型管理中。
   e. 服务部署：将生成的模型部署为在线服务。

快速查找创建好的项目

在自动学习总览页，您可以通过搜索框，根据自动学习的属性类型（项目名称）快速搜索过滤到相应的工作流，可节省您的时间。

1. 登录ModelArts管理控制台，在左侧导航栏选择“开发空间>自动学习”，进入自动学习总览页面。
2. 在自动学习列表上方的搜索框中，根据您需要的属性类型，例如，名称、状态、项目类型、当前节点、标签等，过滤出相应的工作流。
3. 单击搜索框右侧的  按钮，可选择自动学习的基础设置，需要的显示列。

   表格内容折行：默认为关闭状态，启用此能力可让表格内容自动折行，禁用此功能可截断文本。

   操作列：默认为关闭状态，启用此能力可让操作列固定在最后一列永久可见。

   自定义显示列：默认所有显示项全部勾选，您可以根据实际需要定义您的显示列。

   图 4-24 表格显示设置

4. 单击“确定”即可按照设置好的显示列进行显示。
5. 同时可支持对自动学习项目显示页进行排序，单击表头中的箭头  ，就可对该列进行排序。

4.4.3 训练预测分析模型

创建自动学习后，将会进行模型的训练，得到预测分析的模型。模型部署步骤将使用预测模型发布在线预测服务。

操作步骤

1. 在新版自动学习页面，单击创建成功的项目名称，查看当前工作流的执行情况。
2. 在“预测分析”节点中，待节点状态由“运行中”变为“运行成功”，即完成了模型的自动训练。
3. 训练完成后，您可以在预测分析节点中单击  查看训练详情，如“标签列”和“标签列数据类型”、“准确率”、“评估结果”等。

   该示例为二分类的离散型数值，评估效果参数说明请参见**表4-12**。

   不同类型标签列数据产生的评估结果说明请参见**评估结果说明**。

   说明

   同一个自动学习项目可以训练多次，每次训练会注册一个新的模型一个版本。如第一次训练版本号为“0.0.1”，下一个版本为“0.0.2”。基于训练版本可以对训练模型进行管理。当训练的模型达到目标后，再执行模型部署的操作。

评估结果说明

根据训练数据类的不同评估结果会包含不同的指标。

- 离散值评估结果

  包含评估指标为召回率（Recall）、精确率（Precision）、准确率（Accuracy）与F1值（F1 Score）。下表为具体说明：

  表 4-12 离散值评估结果包含指标说明

参数	说明
recall：召回率	被用户标注为某个分类的所有样本中，模型正确预测为该分类的样本比率，反映模型对正样本的识别能力。
precision：精确率	被模型预测为某个分类的所有样本中，模型正确预测的样本比率，反映模型对负样本的区分能力。
accuracy：准确率	所有样本中，模型正确预测的样本比率，反映模型对样本整体的识别能力。
f1：F1值	F1值是模型精确率和召回率的加权调和平均，用于评价模型的好坏，当F1较高时说明模型效果较好。

- 连续数值评估结果

  包含评估指标为平均绝对误差（Mean Absolute Error）、均方误差（Mean Squared Error）与均方根误差（Root Mean Squared Error）。三个误差值能够

表征真实值和预测值之间的差距。在多次建模的过程中，每一次建模结果都会产生一组误差值，评判一个模型好坏的方法就是看这三个误差值是否变小或者变大，误差值越小表示模型越好。

4.4.4 部署预测分析服务

模型部署

模型部署操作即将模型部署为在线服务，并且提供在线的测试UI与监控能力。完成模型训练后，可选择准确率理想且训练状态为“运行成功”的版本部署上线。具体操作步骤如下。

1. 在“运行节点”页面中，待训练状态变为“等待输入”，双击“服务部署”节点，完成相关参数配置。
2. 在服务部署页面，选择模型部署使用的资源规格。
   - 模型来源：默认为生成的模型。
   - 选择模型及版本：自动匹配当前使用的模型版本，支持选择版本。
   - 资源池：默认公共资源池。
   - 分流：默认为100，输入值必须是0-100之间。
   - 计算节点规格：请根据界面显示的列表，选择可用的规格，置灰的规格表示当前环境无法使用。如果公共资源池下规格为空数据，表示当前环境无公共资源。建议使用专属资源池，或者联系系统管理员创建公共资源池。
   - 计算节点个数：默认为1，输入值必须是1-5之间的整数。
   - 是否自动停止：启用该参数并设置时间后，服务将在指定时间后自动停止。如果不启用此参数，在线服务将一直运行，同时一直收费，自动停止功能可以帮您避免产生不必要的费用。默认开启自动停止功能，且默认值为“1小时后”。

     目前支持设置为“1小时后”、“2小时后”、“4小时后”、“6小时后”、“自定义”。如果选择“自定义”的模式，可在右侧输入框中输入1~24范围内的任意整数。

     说明

     如果您购买了套餐包，计算节点规格可选择您的套餐包，同时在“配置费用”页签还可查看您的套餐包余量以及超出部分的计费方式，请您务必关注，避免造成不必要的资源浪费。
3. 完成资源配置后，单击“继续运行”，在弹框中确认继续运行后，服务部署节点将继续运行，直至状态变为“运行成功”，至此，已将模型部署为在线服务。

服务测试

服务部署节点运行成功后，单击“实例详情”可跳转至对应的在线服务详情页面。单击“预测”页签，进行服务测试。

图 4-25 服务测试

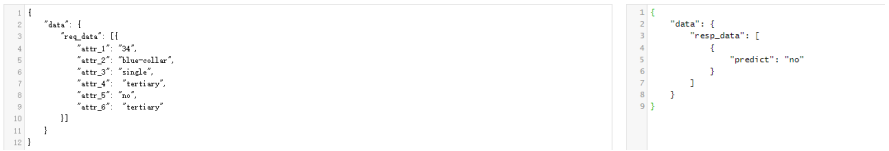
下面的测试，是您在自动学习预测分析项目页面将模型部署上线之后进行服务测试的操作步骤。

1. 模型部署完成后，您可输入代码进行测试。在“自动学习”页面，在服务部署节点，单击“实例详情”进入“在线服务”界面，在“预测”页签的“预测代码”区域，输入调试代码。
2. 单击“预测”进行测试，预测完成后，右侧“返回结果”区域输出测试结果。如模型准确率不满足预期，可在“数据标注”页签，重新进行模型训练及模型部署。如果您对模型预测结果满意，可根据界面提示调用接口访问在线服务。
   - 输入代码：其中预测分析要求数据集中数据的预测列名称为class，否则会导致预测失败。

```
{
   "data": {
      "req_data": [{
         "attr_1": "34",
         "attr_2": "blue-collar",
         "attr_3": "single",
         "attr_4":  "tertiary",
         "attr_5": "no",
         "attr_6":  "tertiary"
      }]
   }
}
```

   - 返回结果如图4-26所示：predict为目标列的预测结果。

图 4-26 预测结果



说明

- 由于“运行中”的在线服务将持续耗费资源，如果不需再使用此在线服务，建议在“在线服务”的操作列单击“更多>停止”，即可停止在线服务的部署，避免产生不必要的费用。如果需要继续使用此服务，可单击“启动”恢复。
- 如果您启用了自动停止功能，服务将在指定时间后自动停止，不再产生费用。

4.5 使用自动学习实现声音分类

4.5.1 准备声音分类数据

使用ModelArts自动学习构建模型时，您需要将数据上传至对象存储服务（OBS）中。OBS桶需要与ModelArts在同一区域。

声音分类的数据要求

- 音频只支持16bit的WAV格式。支持WAV的所有子格式。
- 单条音频时长应大于1s，大小不能超过4MB。
- 适当增加训练数据，会提升模型的精度。声音分类建议每类音频至少20条，每类音频总时长至少5分钟。
- 建议训练数据和真实识别场景的声音保持一致并且每类的音频尽量覆盖真实环境的所有场景。

- 训练集的数据质量对于模型的精度有很大影响，建议训练集音频的采样率和采样精度保持一致。
- 标注质量对于最终的模型精度有极大的影响，标注过程中尽量不要出现误标情况。
- 音频标注涉及到的标注标签和声音内容只支持中文和英文，不支持小语种。

数据上传至 OBS

在本文档中，采用通过OBS管理控制台将数据上传至OBS桶。

上传OBS的文件规范：

- 如不需要提前上传训练数据，请创建一个空文件夹用于存放工程后期生成的文件。如：“/bucketName/data-cat”。
- 如需要提前上传待标注的音频，请创建一个空文件夹，然后将音频文件保存在该文件夹下，音频的目录结构如：“/bucketName/data-cat/cat.wav”。

上传OBS的操作步骤：

执行如下操作，将数据导入到您的数据集中，以便用于模型训练和构建。

1. 登录OBS管理控制台，在ModelArts同一区域内**创建桶**。如果已存在可用的桶，需确保OBS桶与ModelArts在同一区域。
2. 参考**上传文件**，将本地数据上传至OBS桶中。如果您的数据较多，推荐OBS Browser+上传数据或上传文件夹。上传的数据需满足此类型自动学习项目的数据集要求。

说明

- 在上传数据时，请选择非加密桶进行上传，否则会由于加密桶无法解密导致后期的训练失败。
- 用于训练的音频，至少有2种以上的分类，每种分类的音频数据数不少20条。

创建数据集

数据准备完成后，需要创建相应项目支持的类型的数据集，具体操作请参考**创建ModelArts数据集**。

4.5.2 创建声音分类项目

ModelArts自动学习，包括图像分类、物体检测、预测分析、声音分类和文本分类项目。您可以根据业务需求选择创建合适的项目。您需要执行如下操作来创建自动学习项目。

创建项目

1. 登录ModelArts管理控制台，在左侧导航栏单击“开发空间>自动学习”，进入新版自动学习页面。
2. 在您需要的自动学习项目列表中，单击“创建项目”，进入创建自动学习项目界面。
3. 在创建自动学习项目页面，计费模式默认“按需计费”，参考**表4-13**填写相应参数。

表 4-13 参数说明

参数	说明
“名称”	项目的名称。 ● 名称只能包含数字、字母、下划线和中划线，长度不能超过64位且不能为空。 ● 名称请以字母开头。 ● 名称不允许重复。
“描述”	对项目的简要描述。
“数据集”	可在右侧下拉框选择已有数据集，或单击“创建数据集”前往新建数据集。 ● 已有数据集：在“数据集”右侧的下拉框中选择，仅展示同类型的数据集供选择。 ● 创建数据集：前往创建数据集页面创建一个新的数据集。具体操作请参考**创建ModelArts数据集**。
“输出路径”	选择自动学习数据输出的统一OBS路径。 **说明** “输出路径”是存储自动学习在运行过程中所有产物的路径。
“训练规格”	选择自动学习训练节点所使用的资源规格，以实际界面显示为准，将会根据不同的规格计费。 **说明** ● 如果您购买了套餐包，可优先选择您对应规格的套餐包，在“配置费用”处会显示您的套餐余量，以及超出的部分如何计费，请您关注，避免造成不必要的资源浪费。

4. 单击“创建项目”，声音分类项目创建成功后页面自动跳转到“自动学习工作流”。
5. 声音分类项目的工作流，将依次运行如下节点：
   a. 数据标注：对您的数据进行标注情况确认。
   b. 数据集版本发布：将已完成确认的数据进行版本发布。
   c. 数据校验：对您的数据集的数据进行校验，是否存在数据异常。
   d. 声音分类：将发布好的数据集版本进行训练，生成对应的模型。
   e. 模型注册：将训练后的结果注册到模型管理中。
   f. 服务部署：将生成的模型部署为在线服务。

快速查找创建好的项目

在自动学习总览页，您可以通过搜索框，根据自动学习的属性类型（项目名称）快速搜索过滤到相应的工作流，可节省您的时间。

1. 登录ModelArts管理控制台，在左侧导航栏选择“开发空间>自动学习”，进入自动学习总览页面。
2. 在自动学习列表上方的搜索框中，根据您需要的属性类型，例如，名称、状态、项目类型、当前节点、标签等，过滤出相应的工作流。

3. 单击搜索框右侧的  按钮，可选择自动学习的基础设置，需要的显示列。

   表格内容折行：默认为关闭状态，启用此能力可让表格内容自动折行，禁用此功能可截断文本。

   操作列：默认为关闭状态，启用此能力可让操作列固定在最后一列永久可见。

   自定义显示列：默认所有显示项全部勾选，您可以根据实际需要定义您的显示列。

   图 4-27 表格显示设置

4. 单击“确定”即可按照设置好的显示列进行显示。

5. 同时可支持对自动学习项目显示页进行排序，单击表头中的箭头  ，就可对该列进行排序。

4.5.3 标注声音分类数据

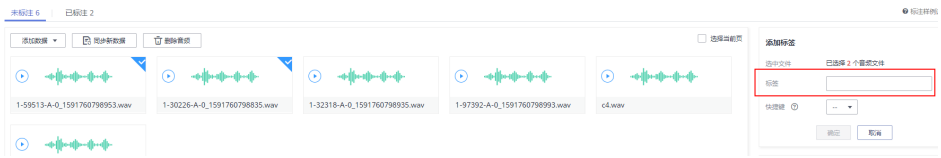
项目创建完成后，将会自动跳转至新版自动学习页面，并开始运行，当数据标注节点的状态变为“等待操作”时，需要手动进行确认数据集中的数据标注情况，也可以对数据集中的数据进行标签的修改，数据的增加或删减。

图 4-28 数据标注节点状态

音频标注

1. 在新版自动学习页面单击“实例详情”按钮，前往数据标注页面。单击任意一张图片，进入音频标注页面。
2. 在“音频标注”页面单击“未标注”页签，此页面展示所有未标注的音频数据。依次单击选中待标注的音频，或勾选“选择当前页”选中该页面所有音频，在页面右侧进行标注。

   图 4-29 音频标注

3. 添加标注。先对音频进行播放识别，然后选中音频文件，在右侧“标签”区域，输入“标签名”或从下拉列表中选择已添加的标签，同时可在下拉菜单中选择标签“快捷键”。单击“确定”，完成选中音频的标注操作。
4. 当目录中所有音频都完成标注后，您可以在“已标注”页签下查看已完成标注的音频，或者通过右侧的“全部标签”列表，了解当前已完成的标签名称和标签数量。

同步或添加音频

在“数据标注”节点单击“实例详情”进入“音频标注”页面。声音分类项目创建时，音频来源有两种，通过本地添加或同步OBS中的数据。

- **添加音频**：您可以将本地音频快速添加到ModelArts，同时自动上传至创建项目时所选择的OBS路径中。单击“添加数据”，在弹出的对话框中输入正确的数据并添加。

  说明

  仅支持16bit WAV格式音频文件，单个音频文件不能超过4MB，且单次上传的音频文件总大小不能超过8MB。

- **数据源同步**：为了快速获取用户OBS桶中最新音频，单击“数据源同步”，快速将通过OBS上传的音频数据添加到ModelArts。

- **删除音频**：您可以依次单击选中音频，或勾选“选择当前页”选中该页面所有音频进行删除操作。

  说明

  所有的删除操作均不可恢复，请谨慎操作。

修改标注

当数据完成标注后，您还可以进入“已标注”页签，对已标注的数据进行修改。

- **基于音频修改**

  在数据集详情页，单击“已标注”页签，然后在音频列表中选中待修改的音频（选择一个或多个）。在右侧标签信息区域中对标签进行修改。

  - 修改标签：在“选中文件标签”区域中，单击操作列的编辑图标，然后在文本框中输入正确的标签名，然后单击确定图标完成修改。
  - 删除标签：在“选中文件标签”区域中，单击操作列的删除图标，在弹出的对话框中单击“确定”删除该标签。

- **基于标签修改**

  在数据标注页面，单击右侧的“标签管理”，在标签管理页，显示全部标签的信息。

  - 修改标签：单击操作列的“修改”按钮，在弹出的对话框中输入修改后的标签名、选择修改后的快捷键，然后单击“确定”完成修改。修改后，之前添加了此标签的音频，都将被标注为新的标签名称。
  - 删除标签：单击操作列的“删除”按钮，在弹出的对话框中，根据提示选择删除对象，然后单击“确定”。

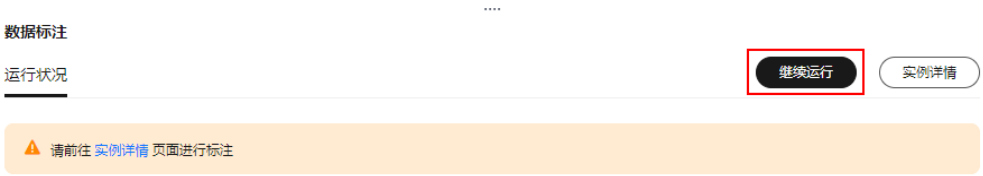
    说明

    删除后的标签无法恢复，请谨慎操作。

继续运行

完成数据的确认之后，返回新版自动学习的页面，在数据标注节点单击“继续运行”，工作流将会继续依次运行直到所有节点运行成功。

图 4-30 继续运行



4.5.4 训练声音分类模型

完成音频标注后，可以进行模型的训练。模型训练的目的是得到满足需求的声音分类模型。由于用于训练的音频，至少有2种以上的分类，每种分类的音频数不少于5个。

操作步骤

在开始训练之前，需要完成数据标注，然后再开始模型的自动训练。

1. 在新版自动学习页面，单击项目名称进入运行总览页面，单击数据标注节点的“实例详情”进入数据标注页面，完成数据标注。
2. 返回新版自动学习页面，单击数据标注节点的“继续运行”，然后等待工作流按顺序进入训练节点。
3. 模型将会自动进入训练，无需人工介入，训练时间相对较长，建议您耐心等待。如果关闭或退出此页面，系统仍然在执行训练操作。
4. 在“声音分类”节点中，待训练状态由“运行中”变为“运行成功”，即完成模型的自动训练。
5. 训练完成后，您可以单击声音分类节点上方的  按钮，查看相关指标信息，如“准确率”、“评估结果”等。

表 4-14 评估结果参数说明

参数	说明
recall：召回率	被用户标注为某个分类的所有样本中，模型正确预测为该分类的样本比率，反映模型对正样本的识别能力。
precision：精确率	被模型预测为某个分类的所有样本中，模型正确预测的样本比率，反映模型对负样本的区分能力。
accuracy：准确率	所有样本中，模型正确预测的样本比率，反映模型对样本整体的识别能力。
f1：F1值	F1值是模型精确率和召回率的加权调和平均，用于评价模型的好坏，当F1较高时说明模型效果较好。

说明

同一个自动学习项目可以训练多次，每次训练会注册一个新的模型版本。如第一次训练版本号为“0.0.1”，下一个版本为“0.0.2”。基于训练版本可以对训练模型进行管理。当训练的模型达到目标后，再执行模型部署的操作。

4.5.5 部署声音分类服务

模型部署

模型部署操作即将模型部署为在线服务，并且提供在线的测试UI与监控能力。完成模型训练后，可选择准确率理想且训练状态为“运行成功”的版本部署上线。具体操作步骤如下。

1. 在“运行总览”页面中，待服务部署节点的状态变为“等待输入”时，双击“服务部署”进入配置详情页，完成资源的参数配置操作。
2. 在服务部署页面，选择模型部署使用的资源规格。
   - 模型来源：默认为生成的模型。
   - 选择模型及版本：自动匹配当前使用的模型版本，支持选择版本。
   - 资源池：默认公共资源池。
   - 分流：默认为100，输入值必须是0-100之间。

- 计算节点规格：请根据界面显示的列表，选择可用的规格，置灰的规格表示当前环境无法使用。如果公共资源池下规格为空数据，表示当前环境无公共资源。建议使用专属资源池，或者联系系统管理员创建公共资源池。
- 计算节点个数：默认为1，输入值必须是1-5之间的整数。
- 是否自动停止：启用该参数并设置时间后，服务将在指定时间后自动停止。如果不启用此参数，在线服务将一直运行，同时一直收费，自动停止功能可以帮您避免产生不必要的费用。默认开启自动停止功能，且默认值为“1小时后”。

  目前支持设置为“1小时后”、“2小时后”、“4小时后”、“6小时后”、“自定义”。如果选择“自定义”的模式，可在右侧输入框中输入1~24范围内的任意整数。

  说明

  如果您购买了套餐包，计算节点规格可选择您的套餐包，同时在“配置费用”页签还可查看您的套餐包余量以及超出部分的计费方式，请您务必关注，避免造成不必要的资源浪费。

3. 完成资源配置后，单击“继续运行”，在弹框中确认继续运行后，服务部署节点将继续运行，直至状态变为“运行成功”，至此，已将模型部署为在线服务。

服务测试

服务部署节点运行成功后，单击“实例详情”可跳转至对应的在线服务详情页面。单击“预测”页签，进行服务测试。

图 4-31 服务测试

下面的测试，是您在自动学习声音分类项目页面将模型部署之后进行服务测试的操作步骤。

1. 模型部署完成后，您可添加音频文件进行测试。在“自动学习”页面，选择服务部署节点，单击实例详情，进入“模型部署”界面，选择状态为“运行中”的服务版本，在“服务测试”区域单击“上传”，选择本地音频进行测试。
2. 单击“预测”进行测试，预测完成后，右侧“预测结果”区域输出测试结果。如模型准确率不满足预期，可在“数据标注”页签中添加音频并进行标注，重新进行模型训练及模型部署。预测结果中的参数说明请参见表4-15。如果您对模型预测结果满意，可根据界面提示调用接口访问在线服务。

表 4-15 预测结果中的参数说明

参数	说明
predicted_label	该段音频的预测类别。
score	预测为此类别的置信度。

> **说明**
>
> 由于“运行中”的在线服务将持续耗费资源，如果不需再使用此在线服务，建议在版本管理区域，单击“停止”，即可停止在线服务的部署，避免产生不必要的费用。如果需要继续使用此服务，可单击“启动”恢复。
>
> 如果您启用了自动停止功能，服务将在指定时间后自动停止，不再产生费用。

4.6 使用自动学习实现文本分类

4.6.1 准备文本分类数据

使用ModelArts自动学习构建模型时，您需要将数据上传至对象存储服务（OBS）中。OBS桶需要与ModelArts在同一区域。

数据集要求

- 文件格式要求为txt或者csv，文件大小不能超过8MB。
- 以换行符作为分隔符，每行数据代表一个标注对象。
- 文本分类目前只支持中文。

数据上传至 OBS

在本文档中，采用通过OBS管理控制台将数据上传至OBS桶。

OBS上传文件的规范：

- 如不需要提前上传训练数据，请创建一个空文件夹用于存放工程后期生成的文件。
- 如需要提前上传待标注的文件，请创建一个空文件夹，然后将文本文件保存在该文件夹下，文本文件的目录结构如：“/bucketName/data/text.csv”。
- 标签名是由中文、大小写字母、数字、中划线或下划线组成，且不超过32位的字符串。
- 如您将已标注好的文本文件上传至OBS桶，请按照如下规范上传。
  - 要求将标注对象和标注文件存储在同一目录，并且一一对应，如标注对象文件名为“COMMENTS_114745.txt”，那么标注文件名为“COMMENTS_114745_result.txt”。

    数据文件存储示例：

    ```
    ├─<dataset-import-path>
    │       COMMENTS_114732.txt
    │       COMMENTS_114732_result.txt
    │       COMMENTS_114745.txt
    │       COMMENTS_114745_result.txt
    │       COMMENTS_114945.txt
    │       COMMENTS_114945_result.txt
    ```

  - 文本分类的标注对象和标注文件均为文本文件，并且以行数进行对应。

OBS上传操作步骤：

执行如下操作，将数据导入到您的数据集中，以便用于模型训练和构建。

1. 登录OBS管理控制台，在ModelArts同一区域内**创建桶**。如果已存在可用的桶，需确保OBS桶与ModelArts在同一区域。

2. 参考**上传文件**，将本地数据上传至OBS桶中。如果您的数据较多，推荐OBS Browser+上传数据或上传文件夹。上传的数据需满足此类型自动学习项目的数据集要求。

说明

- 在上传数据时，请选择非加密桶进行上传，否则会由于加密桶无法解密导致后期的训练失败。
- 用于训练的文本，至少有2种以上的分类，每种分类样本数据数不少20行。

创建数据集

数据准备完成后，需要创建相应项目支持的类型的数据集，具体操作请参考**创建ModelArts数据集**。

4.6.2 创建文本分类项目

ModelArts自动学习，包括图像分类、物体检测、预测分析、声音分类和文本分类项目。您可以根据业务需求选择创建合适的项目。您需要执行如下操作来创建自动学习项目。

创建项目

1. 登录ModelArts管理控制台，在左侧导航栏单击“开发空间>自动学习”，进入新版自动学习页面。
2. 在您需要的自动学习项目列表中，单击“创建项目”，进入创建自动学习项目界面。
3. 在创建自动学习项目页面，计费模式默认“按需计费”，参考**表4-16**填写相应参数。

表 4-16 参数说明

参数	说明
“名称”	项目的名称。 ● 名称只能包含数字、字母、下划线和中划线，长度不能超过64位且不能为空。 ● 名称请以字母开头。 ● 名称不允许重复。
“描述”	对项目的简要描述。
“数据集”	可在右侧下拉框选择已有数据集，或单击“创建数据集”前往新建数据集。 ● 已有数据集：在“数据集”右侧的下拉框中选择，仅展示同类型的数据集供选择。 ● 创建数据集：前往创建数据集页面创建一个新的数据集。具体操作请参考**创建ModelArts数据集**。
“输出路径”	选择自动学习数据输出的统一OBS路径。 **说明** “输出路径”是存储自动学习在运行过程中所有产物的路径。

参数	说明
“训练规格”	选择自动学习训练节点所使用的资源规格，以实际界面显示为准，将会根据不同的规格计费。 **说明** • 如果您购买了套餐包，可优先选择您对应规格的套餐包，在“配置费用”处会显示您的套餐余量，以及超出的部分如何计费，请您关注，避免造成不必要的资源浪费。

4. 单击“创建项目”，文本分类项目创建成功后页面自动跳转到“自动学习工作流”。
5. 文本分类项目的工作流，将依次运行如下节点：
   a. 数据标注：对您的数据进行标注情况确认。
   b. 数据集版本发布：将已完成确认的数据进行版本发布。
   c. 数据校验：对您的数据集的数据进行校验，是否存在数据异常。
   d. 文本分类：将发布好的数据集版本进行训练，生成对应的模型。
   e. 模型注册：将训练后的结果注册到模型管理中。
   f. 服务部署：将生成的模型部署为在线服务。

快速查找创建好的项目

在自动学习总览页，您可以通过搜索框，根据自动学习的属性类型（项目名称）快速搜索过滤到相应的工作流，可节省您的时间。

1. 登录ModelArts管理控制台，在左侧导航栏选择“开发空间>自动学习”，进入自动学习总览页面。
2. 在自动学习列表上方的搜索框中，根据您需要的属性类型，例如，名称、状态、项目类型、当前节点、标签等，过滤出相应的工作流。
3. 单击搜索框右侧的  按钮，可选择自动学习的基础设置，需要的显示列。
   表格内容折行：默认为关闭状态，启用此能力可让表格内容自动折行，禁用此功能可截断文本。
   操作列：默认为关闭状态，启用此能力可让操作列固定在最后一列永久可见。
   自定义显示列：默认所有显示项全部勾选，您可以根据实际需要定义您的显示列。

图 4-32 表格显示设置

4. 单击“确定”即可按照设置好的显示列进行显示。
5. 同时可支持对自动学习项目显示页进行排序，单击表头中的箭头  ，就可对该列进行排序。

4.6.3 标注文本分类数据

项目创建完成后，将会自动跳转至新版自动学习页面，并开始运行，当数据标注节点的状态变为“等待操作”时，需要手动进行确认数据集中的数据标注情况，也可以对数据集中的数据进行标签的修改，数据的增加或删减。

图 4-33 数据标注节点状态

双击“数据标注”节点，单击**实例详情**按钮，打开数据标注页面。

文本分类的数据标注

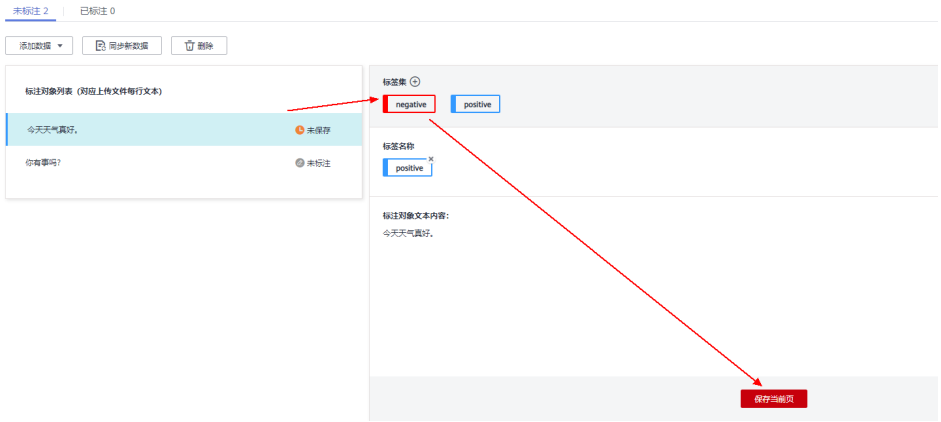
1. 在“标注对象列表”中选中待标注文本，然后单击“标签集”区域中不同标签进行标注。

   一个文本对象只能添加一个标签。

2. 确认文件标签后，单击右下方“保存当前页”，完成标注。

   当“标注对象列表”内容较多时，其区域下方将呈现翻页，请务必在本页完成标注后，单击“保存当前页”保存后再翻页。如果您未完成保存即翻页，将导致前一页的标注信息丢失，需重新标注。

图 4-34 数据标注-文本分类



添加或删除数据

自动学习项目中，数据来源为数据集中输入位置对应的OBS目录，当目录下的数据无法满足现有业务时，您可以在ModelArts自动学习页面中，添加或删除数据。

- **添加文件**

  在“未标注”页签下，可单击页面左上角的“添加数据”，您可以在弹出对话框中，选择本地文件上传。

  上传文件格式需满足文本分类型的**数据集要求**。

- **删除文本对象**

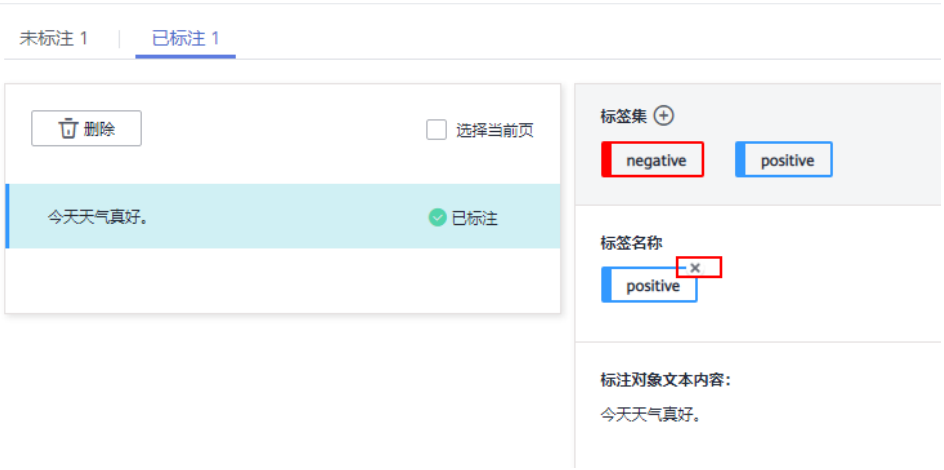
  在“已标注”页签或“未标注”页签下，选中需要删除的文本对象，单击页面左上角的“删除”，在弹出的对话框中，确认删除信息后，单击“确定”。

  在“已标注”页签下，您还可以勾选“选择当前页”，单击“删除”，即可删除当前页下所有的文本对象及其标注信息。

修改已标注的数据

针对“已标注”的文本数据，仅支持删除此文本对象的标签。在“已标注”页签下，在标签名称区域单击标签右上角的叉号，即可删除此文本对象的标签。标签删除后，此文本对象将被呈现至“未标注”页签下。

图 4-35 删除已标注文本的标签



修改标签

针对文本分类的自动学习项目，项目创建成功后，您可以根据业务变化，修改用于标注的标签。支持添加、修改和删除标签。

- 添加标签

  在“未标注”页签下，单击“标签集”右侧的加号，在弹出“新增标签”对话框中，设置“标签名称”和“标签颜色”，然后单击“确定”完成标签添加。
- 修改标签

  在“已标注”页签中“全部标签”的下方操作列，选择需要修改的标签，单击操作列的编辑图标，在弹出“修改标签”对话框中，修改“标签名称”或“标签颜色”，然后单击“确定”完成标签修改。
- 删除标签

  在“已标注”页签中“全部标签”的下方，选择需要删除的标签，单击操作列的删除图标，在弹出“删除”对话框中，选择“仅删除标签”或“删除标签及仅包含此标签的标注对象”，然后单击“确定”完成标签删除。

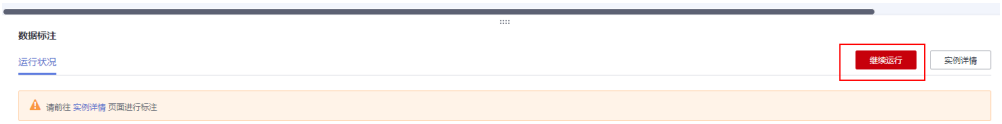
  说明

  所有的删除操作均不可恢复，请谨慎操作。

继续运行

完成数据的确认之后，返回新版自动学习的页面，在数据标注节点单击“继续运行”，工作流将会继续依次运行直到所有节点运行成功。

图 4-36 继续运行



4.6.4 训练文本分类模型

完成数据标注后，可进行模型的训练。模型训练的目的是得到满足需求的文本分类模型。由于用于训练的文本，至少有2种以上的分类（即2种以上的标签），每种分类的

文本数不少于20个。因此在单击“继续运行”按钮之前，请确保已标注的文本符合要求。

操作步骤

1. 在新版自动学习页面，单击项目名称进入运行总览，单击“数据标注”节点的“实例详情”进入“数据标注”页面，完成数据标注。

   图 4-37 完成数据标注

2. 返回新版自动学习页面，单击数据标注节点的“继续运行”，然后等待工作流按顺序进入训练节点。
3. 模型将会自动进入训练，无需人工介入，训练时间相对较长，建议您耐心等待。如果关闭或退出此页面，系统仍然在执行训练操作。
4. 在“文本分类”节点中，待训练状态由“运行中”变为“运行成功”，即完成模型的自动训练。
5. 训练完成后，您可以单击文本分类节点上方的按钮，查看相关指标信息，如“准确率”、“评估结果”等。评估结果参数说明请参见表4-17。

   表 4-17 评估结果参数说明

参数	说明
recall：召回率	被用户标注为某个分类的所有样本中，模型正确预测为该分类的样本比率，反映模型对正样本的识别能力。
precision：精确率	被模型预测为某个分类的所有样本中，模型正确预测的样本比率，反映模型对负样本的区分能力。
accuracy：准确率	所有样本中，模型正确预测的样本比率，反映模型对样本整体的识别能力。
f1：F1值	F1值是模型精确率和召回率的加权调和平均，用于评价模型的好坏，当F1较高时说明模型效果较好。

说明

同一个自动学习项目可以训练多次，每次训练生成一个版本。如第一次训练版本号为“0.0.1”，下一个版本为“0.0.2”。基于训练版本可以对训练模型进行管理。当训练的模型达到目标后，再执行模型部署的操作。

4.6.5 部署文本分类服务

模型部署

模型部署操作即将模型部署为在线服务，并且提供在线的测试UI与监控能力。完成模型训练后，可选择准确率理想且训练状态为“运行成功”的版本部署上线。具体操作步骤如下。

1. 在“运行总览”页面中，待服务部署节点的状态变为“等待输入”，双击“服务部署”节点，进入配置详情页，完成资源的参数配置操作。
2. 在服务部署页面，选择模型部署使用的资源规格。
   - 模型来源：默认为生成的模型。
   - 选择模型版本：自动匹配当前使用的模型版本，支持选择版本。
   - 资源池：默认公共资源池。
   - 分流：默认为100，输入值必须是0-100之间。
   - 计算节点规格：请根据界面显示的列表，选择可用的规格，置灰的规格表示当前环境无法使用。如果公共资源池下规格为空数据，表示当前环境无公共资源。建议使用专属资源池，或者联系系统管理员创建公共资源池。
   - 计算节点个数：默认为1，输入值必须是1-5之间的整数。
   - 是否自动停止：启用该参数并设置时间后，服务将在指定时间后自动停止。如果不启用此参数，在线服务将一直运行，同时一直收费，自动停止功能可以帮您避免产生不必要的费用。默认开启自动停止功能，且默认值为“1小时后”。

     目前支持设置为“1小时后”、“2小时后”、“4小时后”、“6小时后”、“自定义”。如果选择“自定义”的模式，可在右侧输入框中输入1~24范围内的任意整数。

     说明

     如果您购买了套餐包，计算节点规格可选择您的套餐包，同时在“配置费用”页签还可查看您的套餐包余量以及超出部分的计费方式，请您务必关注，避免造成不必要的资源浪费。
3. 完成资源配置后，单击“继续运行”，在弹框中确认继续运行后，服务部署节点将继续运行，直至状态变为“运行成功”，至此，已将模型部署为在线服务。

服务测试

服务部署节点运行成功后，单击“实例详情”可跳转至对应的在线服务详情页面。单击“预测”页签，进行服务测试。

图 4-38 服务测试

下面的测试，是您在自动学习文本分类项目页面将模型部署上线之后进行服务测试的操作步骤。

1. 模型部署完成后，您可添加文本进行测试。在“自动学习”页面，选择目标项目，进入“模型部署”界面，选择状态为“运行中”的服务版本，在“服务测试”区域的文本框中，输入需测试的文本。
2. 单击“预测”进行测试，预测完成后，右侧“预测结果”区域输出测试结果。如模型准确率不满足预期，可在“数据标注”页签中添加数据并进行标注，重新进行模型训练及模型部署。预测结果中的参数说明请参见表4-18。如果您对模型预测结果满意，可根据界面提示调用接口访问在线服务。

表 4-18 预测结果中的参数说明

参数	说明
predicted_label	该段文本的预测类别。
score	预测为此类别的置信度。

说明

由于“运行中”的在线服务将持续耗费资源，如果不需再使用此在线服务，建议在版本管理区域，单击“停止”，即可停止在线服务的部署，避免产生不必要的费用。如果需要继续使用此服务，可单击“启动”恢复。

如果您启用了自动停止功能，服务将在指定时间后自动停止，不再产生费用。

4.7 使用窍门

4.7.1 创建项目时，如何快速创建 OBS 桶及文件夹?

在创建项目时需要选择训练数据路径，本章节将指导您如何在选择训练数据路径时，快速创建OBS桶和OBS文件夹。

1. 在创建自动学习项目页面，单击数据集输入位置右侧的“”按钮，进入“数据集输入位置”对话框。
2. 单击“新建对象存储服务（OBS）桶”，进入创建桶页面，具体请参见《对象存储服务控制台指南》中的创建桶章节。

图 4-39 快速创建 OBS 桶

3. 桶创建完成后，选择对应桶名称，单击“新建文件夹”，在“新建文件夹”对话框中，填写文件夹“名称”，单击“确定”完成创建，选择创建的文件夹。

- 文件夹名称不能包含以下字符：\/:*?"<>|。
- 文件夹名称不能以英文句号（.）或斜杠（/）开头或结尾。
- 文件夹的绝对路径总长度不能超过1023字符。
- 任何单个斜杠（/）表示分隔并创建多层级的文件夹。

图 4-40 新建文件夹

4.7.2 自动学习生成的模型，存储在哪里？支持哪些其他操作？

模型统一管理

针对自动学习项目，当模型训练完成后，其生成的模型，将自动进入“模型管理”页面，如下图所示。模型名称由系统自动命名，前缀与自动学习项目的名称一致，方便辨识。

注意

自动学习生成的模型，不支持下载使用。

图 4-41 自动学习生成的模型

自动学习生成的模型，支持哪些其他操作

- **支持部署为在线服务、批量服务或边缘服务。**

  在自动学习页面中，仅支持部署为在线服务，如需部署为批量服务或边缘服务，可在“模型管理 > 模型”页面中直接部署。
- **支持创建新版本**

创建新版本，仅支持从ModelArts训练作业、OBS、模型模板、或自定义镜像中选择元模型。无法从原自动学习项目中，创建新版本。

- **支持删除模型或其模型版本**

5 使用 Workflow 实现低代码 AI 开发

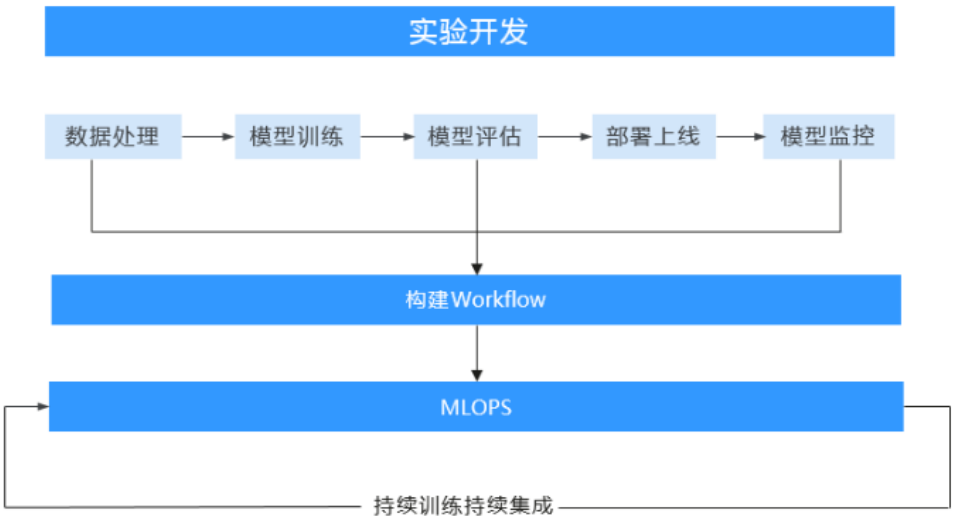
5.1 什么是 Workflow

MLOps 简介

在介绍Workflow之前，先了解MLOps的概念。

MLOps(Machine Learning Operation)是“机器学习”（Machine Learning）和“DevOps”（Development and Operations）的组合实践。机器学习开发流程主要可以定义为四个步骤：项目设计、数据工程、模型构建、部署落地。AI开发并不是一个单向的流水线作业，在开发的过程中，会根据数据和模型结果进行多轮的实验迭代。算法工程师会根据数据特征以及数据的标签做多样化的数据处理以及多种模型优化，以获得在已有的数据集上更好的模型效果。传统的模型交付会直接在实验迭代结束后以输出的模型为终点。当应用上线后，随着时间的推移，会出现模型漂移的问题。新的数据和新的特征在已有的模型上表现会越来越差。在MLOps中，实验迭代的产物将会是一条固化下来的流水线，这条流水线将会包含数据工程、模型算法、训练配置等。用户将会使用这条流水线在持续产生的数据中持续迭代训练，确保这条流水线生产出来的模型始终维持在一个较好的状态。

图 5-1 MLOps



MLOps的整条链路需要有一个工具去承载，MLOps打通了算法开发到交付运维的全流程。和以往的开发交付不同，以往的开发与交付过程是分离的，算法工程师开发完的模型，一般都需要交付给下游系统工程师。MLOps和以往的开发交付不同，在这个过程中，算法工程师参与度还是非常高的。企业内部一般都是有一个交付配合的机制。从项目管理角度上需要增加一个AI项目的工作流程机制管理，流程管理不是一个简单的流水线构建管理，它是一个任务管理体系。

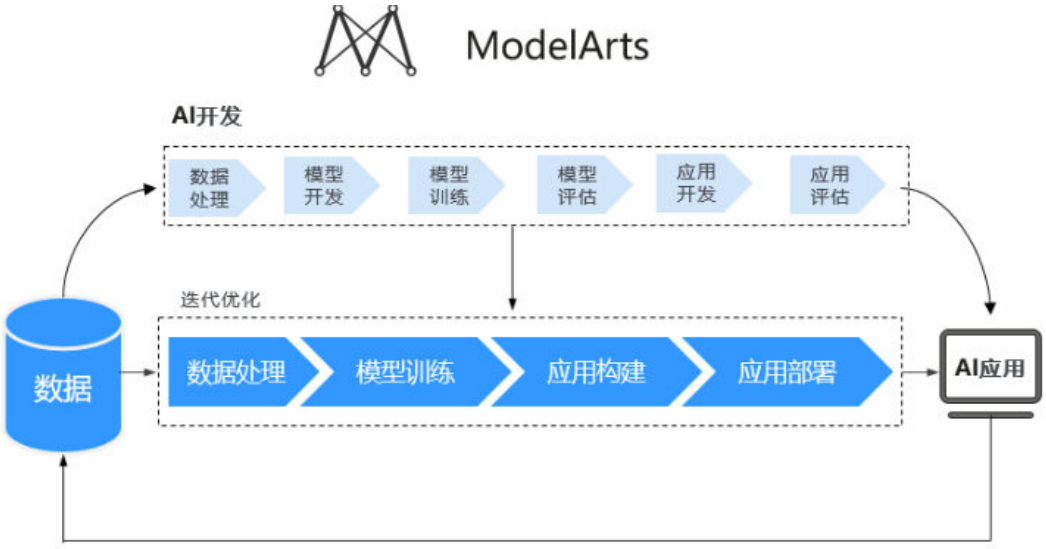
这个工具需要具备以下的能力：

- 流程分析：沉淀行业样例流水线，帮助用户能快速进行AI项目的参考设计，启动快速的AI项目流程设计。
- 流程定义与重定义：以流水线作为承载项，用户能快速定义AI项目，实现训练+推理上线的工作流设计。
- 资源分配：支持账号管理机制给流水线中的参与人员（包含开发者和运维人员）分配相应的资源配额与权限，并查看相应的资源使用情况等。
- 时间安排：围绕子流水线配置相应的子任务安排，并加以通知机制，实现流程执行过程之间配合的运转高效管理。
- 流程质量与效率测评：提供流水线的任务执行过程视图，增加不同的检查点，如数据评估、模型评估、性能评估等，让AI项目管理者能很方便的查看流水线执行过程的质量与效率。
- 流程优化：围绕流水线每一次迭代，用户可以自定义输出相关的核心指标，并获取相应的问题数据与原因等，从而基于这些指标，快速决定下一轮迭代的执行优化。

Workflow 介绍

Workflow（也称工作流，下文中均可使用工作流进行描述）本质是开发者基于实际业务场景开发用于部署模型或应用的流水线工具。在机器学习的场景中，流水线可能会覆盖数据标注、数据处理、模型开发/训练、模型评估、应用开发、应用评估等步骤。

图 5-2 Workflow



区别于传统的机器学习模型构建，开发者可以使用Workflow开发生产流水线。基于MLOps的概念，Workflow会提供运行记录、监控、持续运行等功能。根据角色的分工与概念，产品上将工作流的开发和持续迭代分开。

一条流水线由多个节点组成，Workflow SDK提供了流水线需要覆盖的功能以及功能需要的参数描述。用户在开发流水线的时候，使用SDK对节点以及节点之间串联的关系进行描述。对流水线的开发操作在Workflow中统称为Workflow的开发态。当确定好整条流水线后，开发者可以将流水线固化下来，提供给其他人使用。使用者无需关注流水线中包含什么算法，也不需要关注流水线是如何实现的。使用者只需要关注流水线生产出来的模型或者应用是否符合上线要求，如果不符合，是否需要调整数据和参数重新迭代。这种使用固化下来的流水线的状态，在Workflow中统称为运行态。

总的来说，Workflow有两种形态。

- 开发态：使用Workflow的Python SDK开发和调测流水线。
- 运行态：可视化配置运行生产好的流水线。

Workflow基于对当前ModelArts已有能力的编排，基于DevOps原则和实践，应用于AI开发过程中，提升了模型开发与落地效率，更快地进行模型实验和开发，并更快地将模型部署到生产环境。

工作流的开发态和运行态分别实现了不同的功能。

开发态-开发工作流

开发者结合实际业务的需求，通过Workflow提供的Python SDK，将ModelArts的能力封装成流水线中的一个个步骤。对于AI开发者来说是非常熟悉的开发模式，而且灵活度极高。Python SDK主要提供以下能力。

- 开发构建：使用python代码灵活编排构建工作流。
- 调测：支持debug以及run两种模式，其中run模式支持节点部分运行、全部运行。
- 发布：支持将调试后的工作流进行固化，发布至运行态，支持配置运行。
- 实验记录：实验的持久化及管理。

- 共享：支持将工作流作为资产发布至AI Gallery，分享给其他用户使用。

运行态-运行工作流

Workflow提供了可视化的工作流运行方式。使用者不需要了解工作流的内部细节，只需要关注一些简单的参数配置即可启动运行工作流。运行态的工作流来源主要为：通过开发态发布或者从gallery订阅。

运行态工作流的来源为：通过开发态发布，或者通过AI Gallery订阅。

运行态主要提供以下能力。

- 统一配置管理：管理工作流需要配置的参数及使用的资源等。
- 操作工作流：启动、停止、重试、复制、删除工作流。
- 查看运行记录：查看工作流历史运行的参数以及状态记录。

Workflow 的构成

工作流是对一个有向无环图的描述。开发者可以通过Workflow进行有向无环图（Directed Acyclic Graph，DAG）的开发。一个DAG是由节点和节点之间的关系描述组成的。开发者通过定义节点的执行内容和节点的执行顺序定义DAG。绿色的矩形表示为一个节点，节点与节点之间的连线则是节点的关系描述。整个DAG的执行其实就是有序的任务执行模板。

Workflow 提供的样例

ModelArts提供了丰富的基于场景的工作流样例，用户可以**前往AI Gallery进行订阅**。

5.2 管理 Workflow

5.2.1 查找 Workflow 工作流

查找 Workflow

在Workflow列表页，您可以通过搜索框，根据工作流的属性类型快速搜索过滤到相应的工作流，可节省您的时间。

1. 登录ModelArts管理控制台，在左侧导航栏选择“开发空间>Workflow”，进入Workflow总览页面。
2. 在工作流列表上方的搜索框中，根据您需要的属性类型，例如名称、状态、当前节点、启动时间、运行时长或标签等，过滤出相应的工作流。

图 5-3 属性类型

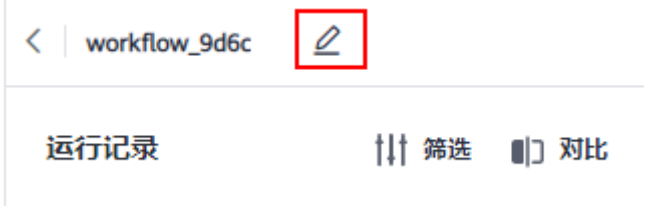
3. 单击搜索框右侧的按钮，可设置Workflow列表页需要展示的内容和展示效果。
   - 表格内容折行：默认为关闭状态。启用此功能可以让Workflow列表页中的内容在显示时自动换行。禁用此功能可截断文本，Workflow列表页中仅显示部分内容。
   - 操作列：默认为开启状态，启用此能力可让操作列固定在最后一列永久可见。
   - 自定义显示列：默认所有显示项全部勾选，您可以根据实际需要定义您的显示列。
4. 设置完成后，单击“确定”即可。
5. 同时可支持对Workflow显示列进行排序，单击表头中的箭头，就可对该列进行排序。

编辑 Workflow 名称和标签

通过修改Workflow的名称和标签，方便快速查找Workflow。

1. 在ModelArts管理控制台，左侧菜单栏单击“开发空间>Workflow”。进入Workflow列表页。
2. 在列表页单击工作流名称，进入工作流详情页。
3. 在工作流详情页，单击左上角编辑按钮。

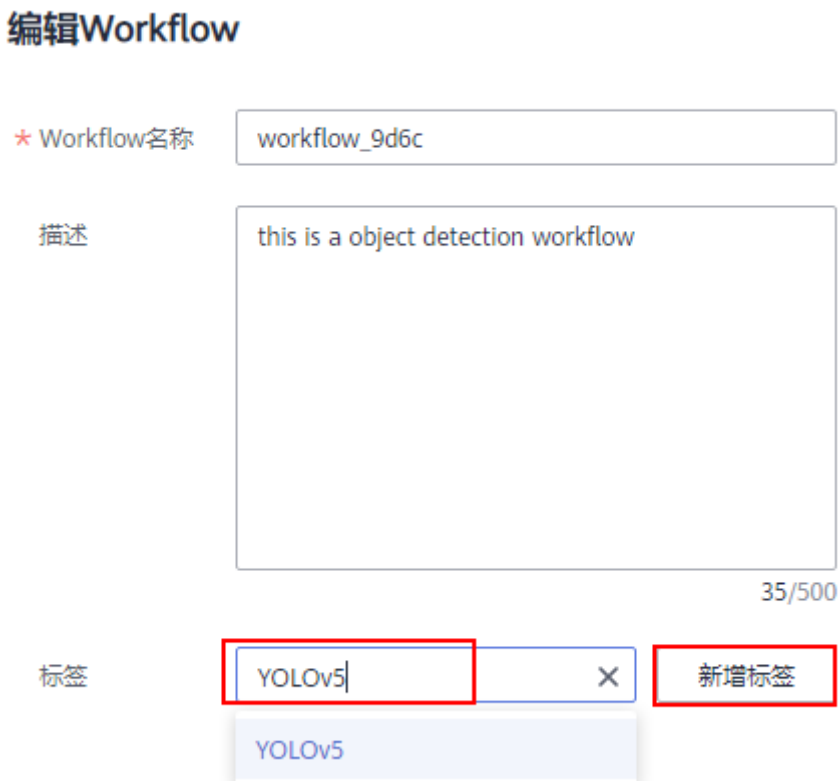
图 5-4 编辑 Workflow



4. 在弹出的编辑Workflow弹窗中，可以修改Workflow名称和标签。

在标签框中输入相应的标签后，单击“新增标签”，新生成的标签会展示在标签行的下方，您可以同时增加多个标签。标签增加完成后，单击“确定”，标签即可生成。

图 5-5 新增标签



5. 生成了标签的Workflow，支持在搜索框中按照标签筛选对应的Workflow。

5.2.2 查看 Workflow 工作流运行记录

运行记录是展示某条工作流所有运行状态数据的地方。

1. 在Workflow列表页，单击某条工作流的名称，进入该工作流的详情页面。
2. 在工作流的详情页，左侧区域即为该条工作流的所有运行记录。

图 5-6 查看运行记录

3. 您可以对当前工作流的所有运行记录，进行删除、编辑以及重新运行的操作。
   - 删除：如果该条运行记录不再需要，您可以单击“删除”，在弹出的确认框中单击“确定”即可完成运行记录的删除。
   - 编辑：如果您想对您当前的工作流下的所有运行记录进行区分，您可以单击“编辑”，对每一条运行记录添加相应的标签予以区分。
   - 重新运行：可以单击“重新运行”直接在某条记录上运行该工作流。
4. 您可以对该条工作流的所有运行记录进行筛选和对比。
   - 筛选：该功能支持您对所有运行记录按照“运行状态”和“运行标签”进行筛选。

     图 5-7 筛选

   - 对比：针对某条工作流的所有运行记录，按照状态、运行记录、启动时间、运行时长、参数等进行对比。

     图 5-8 对比

当单击“启动”运行工作流时，运行记录列表会自动刷新，并更新至最新一条的执行记录数据，且与DAG图和总览数据面板双向联动更新数据。每次启动后都会新增一条运行记录。

用户可以单击Workflow详情页中任一节点查询节点运行状况。包括节点的属性（节点的运行状态、启动时间以及运行时长）、输入位置与输出位置以及参数（数据集的标注任务名称）。

5.2.3 管理 Workflow 工作流

启动 Workflow

登录ModelArts管理控制台，在左侧导航栏选择“开发空间>Workflow”，进入Workflow总览页面。

有3种操作方式运行工作流。

- 工作流列表页：单击操作栏的“启动”按钮，出现启动Workflow询问弹窗，单击“确定”。
- 工作流运行页面：单击右上角的“启动”按钮，出现启动Workflow询问弹窗，单击“确定”。
- 工作流参数配置页面：单击右上角的“启动”按钮，出现启动Workflow询问弹窗，单击“确定”。

说明

启动Workflow后，运行过程中将会按需收费，请关注实例状态，完成后的工作流请及时停止，避免产生不必要的费用。

停止 Workflow

登录ModelArts管理控制台，在左侧导航栏选择“开发空间>Workflow”，进入Workflow总览页面。

可以通过“停止”按钮，主动停止正在运行的工作流，有2种操作方式：

- 工作流列表页：

  当工作流处于“运行中”时，操作栏会出现“停止”按钮。单击“停止”，出现停止Workflow询问弹窗，单击“确定”。
- 进入某条运行中的工作流，单击右上角的“停止”按钮，出现停止Workflow询问弹窗，单击确定。

说明

只有处于“运行中”状态的工作流，才会出现“停止”按钮。

停止Workflow后，关联的训练作业和在线服务也会停止。

复制 Workflow

某条工作流，目前只能存在一个正在运行的实例，如果用户想要使同一个工作流同时运行多次，可以使用复制工作流的功能。单击列表页的操作栏“更多”，选择“复制”，出现复制Workflow弹窗，新名称会自动生成（生成规则：原工作流名称 + '_copy'）。

用户也可以自行修改新工作流名称，但会有校验规则验证新名称是否符合要求。

说明

新的Workflow名称，必须为1~64位只包含英文、数字、下划线（_）和中划线（-）且以英文开头的名称。

删除 Workflow

登录ModelArts管理控制台，在左侧导航栏选择“开发空间>Workflow”，进入Workflow总览页面。

删除工作流有2种方式

- 工作流列表页
  a. 单击操作栏的“更多”，选择“删除”，出现删除Workflow询问弹窗。
  b. 输入“delete”，单击“确定”，删除Workflow。

- 工作流运行页面

  单击界面右上角的“删除”，出现删除Workflow弹窗，输入“DELETE”，单击“确定”，删除Workflow。

  说明

  - 删除后的Workflow无法恢复，请谨慎操作。
  - 删除Workflow后，对应的训练作业和在线服务不会随之被删除，需要分别在“模型训练>训练作业”和“模型部署>在线服务”页面中手动删除任务。

5.2.4 重试/停止/运行 Workflow 节点

重试/停止/继续运行 Workflow 节点

- 重试

  当单个节点运行失败时，用户可以通过重试按钮重新执行当前节点，无需重新启动工作流。在当前节点的运行状况页面，单击“重试”。在重试之前您也可以前往权限管理页面修改配置，节点重试启动后新修改的配置信息可以在当前执行中立即生效。

- 停止

  单击指定节点查看详情，可以对运行中的节点进行停止操作。

- 继续运行

  对于单个节点中设置了需要运行中配置的参数时，节点运行会处于“等待操作”状态，用户完成相关数据的配置后，可单击“继续运行”按钮并确认继续执行当前节点。

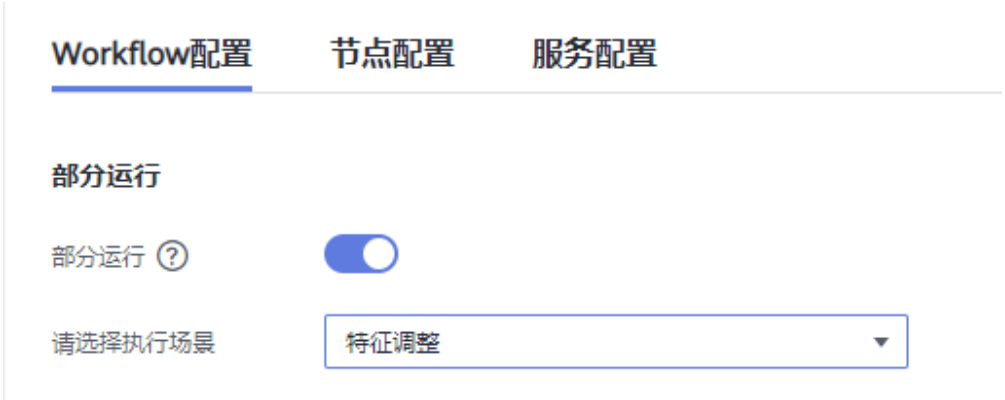
部分运行 Workflow 节点

针对大型、复杂的Workflow，为节省重复运行消耗的时间，在运行业务场景时，用户可以选择其中的部分节点作为业务场景运行，工作流在执行时将会按顺序执行部分运行节点。

部分运行Workflow节点，首先在新开发Workflow时，需要预先定义好部分运行场景。具体流程如下：

1. 通过SDK创建工作流时，预先定义好部分运行场景，具体可参考**在Workflow中指定仅运行部分节点**。
2. 在配置工作流时，打开“部分运行”开关，选择需要执行的部分运行场景，并填写完善相关节点的参数。

图 5-9 部分运行



3. 保存上一步的配置后，单击“启动”按钮即可启动部分运行场景。

5.3 开发 Workflow 命令参考

5.3.1 开发 Workflow 的核心概念介绍

Workflow

Workflow是一个有向无环图（Directed Acyclic Graph，DAG），由节点和节点之间的关系描述组成。

节点与节点之间的依赖关系由单箭头的线段来表示，依赖关系决定了节点的执行顺序，示例中的工作流在启动后将从左往右顺序执行。DAG也支持多分支结构，用户可根据实际场景进行灵活设计，在多分支场景下，并行分支的节点支持并行运行，具体请参考**配置多分支节点数据**章节。

表 5-1 Workflow

属性	描述	是否必填	数据类型
name	工作流的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64位字符	是	str
desc	工作流的描述信息	是	str
steps	工作流包含的节点列表	是	list[Step]
storages	统一存储对象列表	否	Storage或者list[Storage]
policy	工作流的配置策略，主要用于部分运行场景	否	Policy

Step

Step是组成Workflow的最小单元，体现在DAG中就是一个一个的节点，不同的Step类型承载了不同的服务能力，主要构成如下。

表 5-2 Step

属性	描述	是否必填	数据类型
name	节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符	是	str

属性	描述	是否必填	数据类型
title	节点的标题信息，主要用于在DAG中的展示，如果该字段未填写，则默认使用name进行展示	否	str
step_type	节点的类型，决定了节点的功能	是	enum
inputs	节点的输入列表	否	AbstractInput或者list[AbstractInput]
outputs	节点的输出列表	否	AbstractOutput或者list[AbstractOutput]
properties	节点的属性信息	否	dict
policy	节点的执行策略，主要包含节点调度运行的时间间隔、节点执行的超时时间、以及节点执行是否跳过的相关配置	否	StepPolicy
depend_steps	依赖节点的列表，该字段决定了DAG的结构，也决定了节点执行的顺序	否	Step或者list[Step]

表 5-3 StepPolicy

属性	描述	是否必填	数据类型
poll_interval_seconds	节点调度时间周期，默认为1秒	是	str
max_execution_minutes	节点运行超时时间，默认为10080分钟，即7天	是	str
skip_conditions	节点是否跳过的条件列表	否	Condition或者Condition列表

Step是节点的超类，主要用于概念上的承载，用户不直接使用。根据功能的不同，构建了不同类型的节点，主要包括**CreateDatasetStep**、**LabelingStep**、**DatasetImportStep**、**ReleaseDatasetStep**、**JobStep**、**ModelStep**、**ServiceStep**、**ConditionStep**等，详情请见**创建Workflow节点**。

Data

数据对象用于节点的输入，主要可分为以下三种类型：

- 真实的数据对象，在工作流构建时直接指定：

  - Dataset：用于定义已有的数据集，常用于数据标注，模型训练等场景
  - LabelTask: 用于定义已有的标注任务，常用于数据标注，数据集版本发布等场景
  - OBSPath：用于定义指定的OBS路径，常用于模型训练，数据集导入，模型导入等场景
  - ServiceData：用于定义一个已有的服务，只用于服务更新的场景
  - SWRImage：用于定义已有的SWR路径，常用于模型注册场景
  - GalleryModel：用于定义从gallery订阅的模型，常用于模型注册场景
- 占位符式的数据对象，在工作流运行时指定：
  - DatasetPlaceholder：用于定义在运行时需要确定的数据集，对应Dataset对象，常用于数据标注，模型训练等场景
  - LabelTaskPlaceholder：用于定义在运行时需要确定的标注任务，对应LabelTask对象，常用于数据标注，数据集版本发布等场景
  - OBSPlaceholder：用于定义在运行时需要确定的OBS路径，对应OBSPath对象，常用于模型训练，数据集导入，模型导入等场景
  - ServiceUpdatePlaceholder：用于定义在运行时需要确定的已有服务，对应ServiceData对象，只用于服务更新的场景
  - SWRImagePlaceholder：用于定义在运行时需要确定的SWR路径，对应SWRImage对象，常用于模型注册场景
  - ServiceInputPlaceholder：用于定义在运行时需要确定服务部署所需的模型相关信息，只用于服务部署及服务更新场景
  - DataSelector：支持多种数据类型的选择，当前仅支持在JobStep节点中使用（仅支持选择OBS或者数据集）
- 数据选择对象：

  DataConsumptionSelector：用于在多个依赖节点的输出中选择一个有效输出作为数据输入，常用于存在条件分支的场景中（在构建工作流时未能确定数据输入来源为哪个依赖节点的输出，需根据依赖节点的实际执行情况进行自动选择）

表 5-4 Dataset

属性	描述	是否必填	数据类型
dataset_name	数据集名称	是	str
version_name	数据集版本名称	否	str

示例：

```
example = Dataset(dataset_name = "**", version_name = "**")
# 通过ModelArts的数据集，获取对应的数据集名称及相应的版本名称。
```

说明

当Dataset对象作为节点的输入时，需根据业务需要自行决定是否填写version_name字段（比如LabelingStep、ReleaseDatasetStep不需要填写，JobStep必须填写）。

表 5-5 LabelTask

属性	描述	是否必填	数据类型
dataset_name	数据集名称	是	str
task_name	标注任务名称	是	str

示例：

```
example = LabelTask(dataset_name = "**", task_name = "**")
# 通过ModelArts的新版数据集，获取对应的数据集名称及相应的标注任务名称
```

表 5-6 OBSPath

属性	描述	是否必填	数据类型
obs_path	OBS路径	是	str，Storage

示例：

```
example = OBSPath(obs_path = "**")
# 通过对象存储服务，获取已存在的OBS路径值
```

表 5-7 ServiceData

属性	描述	是否必填	数据类型
service_id	服务的ID	是	str

示例：

```
example = ServiceData(service_id = "**")
# 通过ModelArts的在线服务，获取对应服务的服务ID，描述指定的在线服务。用于服务更新的场景。
```

表 5-8 SWRImage

属性	描述	是否必填	数据类型
swr_path	容器镜像的SWR路径	是	str

示例：

```
example = SWRImage(swr_path = "**")
# 容器镜像地址，用于模型注册节点的输入
```

表 5-9 GalleryModel

属性	描述	是否必填	数据类型
subscription_id	订阅模型的订阅ID	是	str
version_num	订阅模型的版本号	是	str

示例：

```
example = GalleryModel(subscription_id="**", version_num="**")
# 订阅的模型对象，用于模型注册节点的输入
```

表 5-10 DatasetPlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str
data_type	数据类型	否	DataTypeEnum
delay	标志数据对象是否在节点运行时配置，默认为False	否	bool
default	数据对象的默认值	否	Dataset

示例：

```
example = DatasetPlaceholder(name = "**", data_type = DataTypeEnum.IMAGE_CLASSIFICATION)
# 数据集对象的占位符形式，可以通过指定data_type限制数据集的数据类型
```

表 5-11 OBSPlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str
object_type	表示OBS对象类型，仅支持"file"或者"directory"	是	str
delay	标志数据对象是否在节点运行时配置，默认为False	否	bool
default	数据对象的默认值	否	OBSPath

示例：

```
example = OBSPlaceholder(name = "**", object_type = "directory" )
# OBS对象的占位符形式，object_type只支持两种类型， "file" 以及 "directory"
```

表 5-12 LabelTaskPlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str
task_type	表示标注任务的类型	否	LabelTaskTypeEnum
delay	标志数据对象是否在节点运行时配置，默认为False	否	bool

示例：

```
example = LabelTaskPlaceholder(name = "**")
# LabelTask对象的占位符形式
```

表 5-13 ServiceUpdatePlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str
delay	标志数据对象是否在节点运行时配置，默认为False	否	bool

示例：

```
example = ServiceUpdatePlaceholder(name = "**")
# ServiceData对象的占位符形式，用于服务更新节点的输入
```

表 5-14 SWRImagePlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str
delay	标志数据对象是否在节点运行时配置，默认为False	否	bool

示例：

```
example = SWRImagePlaceholder(name = "**" )
# SWRImage对象的占位符形式，用于模型注册节点的输入
```

表 5-15 ServiceInputPlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str
model_name	模型名称	是	str或者Placeholder
model_version	模型版本	否	str
envs	环境变量	否	dict
delay	服务部署相关信息是否在节点运行时配置，默认为True	否	bool

示例：

```
example = ServiceInputPlaceholder(name = "**" , model_name = "model_name")
# 用于服务部署或者服务更新节点的输入
```

表 5-16 DataSelector

属性	描述	是否必填	数据类型
name	名称	是	str
data_type_list	支持的数据类型列表，当前仅支持obs、dataset	是	list
delay	标志数据对象是否在节点运行时配置，默认为False	否	bool

示例：

```
example = DataSelector(name = "**" ,data_type_list=["obs", "dataset"])
# 用于作业类型节点的输入
```

表 5-17 DataConsumptionSelector

属性	描述	是否必填	数据类型
data_list	依赖节点的输出数据对象列表	是	list

示例：

```
example = DataConsumptionSelector(data_list=[step1.outputs["step1_output_name"].as_input(),
step2.outputs["step2_output_name"].as_input()])
```

```
# 从step1以及step2中选择有效输出作为输入，当step1跳过无输出，step2执行有输出时，将step2的有效输出作为输入（需保证data_list中同时只有一个有效输出）
```

5.3.2 配置 Workflow 参数

功能介绍

参数相关的配置使用Placeholder对象来表示，以占位符的形式实现用户数据运行时配置的能力，当前支持的数据类型包括：int、str、bool、float、Enum、dict、list。开发者可根据场景需要，将节点中的相关字段（如算法超参）通过Placeholder的形式透出，支持设置默认值，供用户修改配置使用。

属性总览（Placeholder）

属性	描述	是否必填	数据类型
name	参数名称，需要保证全局唯一。	是	str
placeholder_type	参数类型，与真实数据类型的映射关系如下： PlaceholderType.INT -> int PlaceholderType.STR -> str PlaceholderType.BOOL -> bool PlaceholderType.FLOAT -> float PlaceholderType.ENUM -> Enum PlaceholderType.JSON -> dict PlaceholderType.LIST -> list • 当类型为PlaceholderType.ENUM时，enum_list字段不能为空。 • 当类型为PlaceholderType.LIST时，placeholder_format字段不能为空，且只能填写str/int/float/bool，用来表示list中的数据类型。	是	PlaceholderType
default	参数默认值，数据类型需要与placeholder_type一致。	否	Any
placeholder_format	支持的format格式数据，当前支持obs、flavor、train_flavor、swr、pacific。	否	str
delay	参数是否运行时输入，默认为“False”，在工作流启动运行前进行配置。设置为“True”，则在使用的相应节点运行时卡点配置。	否	bool
description	参数描述信息。	否	str
enum_list	参数枚举值列表，只有当参数类型为PlaceholderType.ENUM时才需要填写。	否	list

属性	描述	是否必填	数据类型
constraint	参数相关的约束配置，当前该字段仅支持训练规格的约束，且用户不感知。	否	dict
required	参数是否必填标记。 ● 默认required=True。 ● Delay参数不能设required=False。 运行时前端可以不填此参数。	否	bool

使用案例

- int类型参数
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_int", placeholder_type=wf.PlaceholderType.INT, default=1, description="这是一个int类型的参数")
```
- str类型参数
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_str", placeholder_type=wf.PlaceholderType.STR, default="default_value", description="这是一个str类型的参数")
```
- bool类型参数
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_bool", placeholder_type=wf.PlaceholderType.BOOL, default=True, description="这是一个bool类型的参数")
```
- float类型参数
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_float", placeholder_type=wf.PlaceholderType.FLOAT, default=0.1, description="这是一个float类型的参数")
```
- Enum类型参数
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_enum", placeholder_type=wf.PlaceholderType.ENUM, default="a", enum_list=["a", "b"], description="这是一个enum类型的参数")
```
- dict类型参数
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_dict", placeholder_type=wf.PlaceholderType.JSON, default={"key": "value"}, description="这是一个dict类型的参数")
```
- list类型参数
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_list", placeholder_type=wf.PlaceholderType.LIST, default=[1, 2], placeholder_format="int", description="这是一个list类型的参数，并且value类型为int")
```

5.3.3 配置 Workflow 的输入输出目录

功能介绍

统一存储主要用于工作流的目录管理，帮助用户统一管理一个工作流中的所有存储路径，主要分为以下两个功能：

- 输入目录管理：开发者在编辑开发工作流时可以对所有数据的存储路径做统一管理，规定用户按照自己的目录规划来存放数据，而存储的根目录可以根据用户自己的需求自行配置。该方式只做目录的编排，不会自动创建新的目录。
- 输出目录管理：开发者在编辑开发工作流时可以对所有的输出路径做统一管理，用户无需手动创建输出目录，只需要在工作流运行前配置存储根路径，并且可以

根据开发者的目录编排规则在指定目录下查看输出的数据信息。此外同一个工作流的多次运行支持输出到不同的目录下，对不同的执行做了很好的数据隔离。

常用方式

- **InputStorage**（路径拼接）

  该对象主要用于帮助用户统一管理输入的目录，使用示例如下：

  ```
  import modelarts.workflow as wf
  storage = wf.data.InputStorage(name="storage_name", title="title_info",
  description="description_info") # name字段必填，title, description可选填
  input_data = wf.data.OBSPath(obs_path = storage.join("directory_path")) # 注意，如果是目录则最后需要加"/"，例如：storage.join("/input/data/")
  ```

  工作流运行时，如果storage对象配置的根路径为"/root/"，则最后得到的路径为"**/root/directory_path**"

- **OutputStorage**（目录创建）

  该对象主要用于帮助用户统一管理输出的目录，保证工作流每次执行输出到新目录，使用示例如下：

  ```
  import modelarts.workflow as wf
  storage = wf.data.OutputStorage(name="storage_name", title="title_info",
  description="description_info") # name字段必填，title, description可选填
  output_path = wf.data.OBSOutputConfig(obs_path = storage.join("directory_path")) # 注意，只能创建目录，不能创建文件 。
  ```

  工作流运行时，如果storage对象配置的根路径为"/root/"，则系统自动创建相对目录，最后得到的路径为"**/root/执行ID/directory_path**"

进阶用法

Storage

该对象是InputStorage和OutputStorage的基类，包含了两者的所有能力，可以供用户灵活使用。

属性	描述	是否必填	数据类型
name	名称。	是	str
title	不填默认使用name的值。	否	str
description	描述信息。	否	str
create_dir	表示是否自动创建目录，默认为“False”。	否	bool
with_execution_id	表示创建目录时是否拼接execution_id，默认为“False”。该字段只有在create_dir为True时才支持设置为True。	否	bool

使用示例如下：

- 实现**InputStorage**相同的能力

  ```
  import modelarts.workflow as wf
  # 构建一个Storage对象, with_execution_id=False, create_dir=False
  storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info",
  ```

```
with_execution_id=False, create_dir=False)
input_data = wf.data.OBSPath(obs_path = storage.join("directory_path")) # 注意，如果是目录则最后需要加"/"，例如：storage.join("/input/data/")

工作流运行时，如果storage对象配置的根路径为"/root/"，则最后得到的路径为"/root/directory_path"
```

- 实现**OutputStorage**相同的能力

```
import modelarts.workflow as wf
# 构建一个Storage对象, with_execution_id=True, create_dir=True
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info", with_execution_id=True, create_dir=True)
output_path = wf.data.OBSOutputConfig(obs_path = storage.join("directory_path")) # 注意，只能创建目录，不能创建文件

工作流运行时，如果storage对象配置的根路径为"/root/"，则系统自动创建相对目录，最后得到的路径为"/root/执行ID/directory_path"
```

- 通过**join**方法的参数实现同一个Storage的不同用法

```
import modelarts.workflow as wf
# 构建一个Storage对象, 并且假设Storage配置的根目录为"/root/"
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info", with_execution_id=False, create_dir=False)
input_data1 = wf.data.OBSPath(obs_path = storage) # 得到的路径为：/root/
input_data2 = wf.data.OBSPath(obs_path = storage.join("directory_path")) # 得到的路径为：/root/directory_path，需要用户自行保证该路径存在
output_path1 = wf.data.OBSOutputConfig(obs_path = storage.join(directory="directory_path", with_execution_id=False, create_dir=True)) # 系统自动创建目录，得到的路径为：/root/directory_path
output_path2 = wf.data.OBSOutputConfig(obs_path = storage.join(directory="directory_path", with_execution_id=True, create_dir=True)) # 系统自动创建目录，得到的路径为：/root/执行ID/directory_path
```

Storage可实现链式调用

使用示例如下：

```
import modelarts.workflow as wf
# 构建一个基类Storage对象, 并且假设Storage配置的根目录为"/root/"
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info", with_execution_id=False, create_dir=Fals)
input_storage = storage.join("directory_path_1") # 得到的路径为：/root/directory_path_1
input_storage_next = input_storage.join("directory_path_2") # 得到的路径为： /root/directory_path_1/directory_path_2
```

使用案例

统一存储主要用于JobStep中，下面代码示例全部以单训练节点为例。

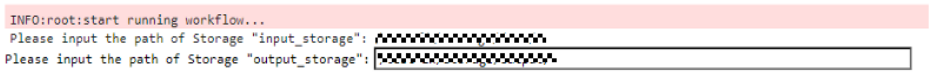
```
from modelarts import workflow as wf

# 构建一个InputStorage对象, 并且假设配置的根目录为"/root/input-data/"
input_storage = wf.data.InputStorage(name="input_storage_name", title="title_info", description="description_info") # name字段必填，title, description可选填

# 构建一个OutputStorage对象, 并且假设配置的根目录为"/root/output/"
output_storage = wf.data.OutputStorage(name="output_storage_name", title="title_info", description="description_info") # name字段必填，title, description可选填

# 通过JobStep来定义一个训练节点，输入数据来源为OBS，并将训练结果输出到OBS中
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(subscription_id="subscription_ID", item_version_id="item_version_ID"), # 训练使用的算法对象，示例中使用AIGallery订阅的算法
    inputs=[
        wf.steps.JobInput(name="data_url_1", data=wf.data.OBSPath(obs_path = input_storage.join("/dataset1/new.manifest"))),  # 获得的路径为：/root/input-data/dataset1/new.manifest
        wf.steps.JobInput(name="data_url_2", data=wf.data.OBSPath(obs_path = input_storage.join("/dataset2/new.manifest")))  # 获得的路径为：/root/input-data/dataset2/new.manifest
    ],
```

```
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/model/"))), # 训练输出的路径为：/root/output/执行ID/model/
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
                flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")
        ),
        log_export_path=wf.steps.job_step.LogExportPath(obs_url=output_storage.join("/logs/")) # 日志输出的路径为：/root/output/执行ID/logs/
    )# 训练资源规格信息
)

# 定义一个只包含job_step的工作流
workflow = wf.Workflow(
    name="test-workflow",
    desc="this is a test workflow",
    steps=[job_step],
    storages=[input_storage, output_storage] # 注意在整个工作流中使用到的Storage对象需要在这里添加
)
```

开发态配置

调用工作流对象的run方法，在开始运行时展示输入框，等待用户输入，如下所示：

图 5-10 等待用户输入



要求用户输入已存在的路径，否则会报错，路径格式要求为：/桶名称/文件夹路径/。

运行态配置

调用工作流对象的release方法将工作流发布到运行态，在ModelArts管理控制台，单击Workflow找到相应的工作流进行根路径配置，如下所示：

图 5-11 根目录配置

5.3.4 创建 Workflow 节点

5.3.4.1 创建 Workflow 数据集节点

功能介绍

通过对ModelArts数据集能力进行封装，实现新版数据集的创建功能。主要用于通过创建数据集对已有数据（已标注/未标注）进行统一管理的场景，后续常见数据集导入节点或者数据集标注节点。

属性总览

您可以使用**CreateDatasetStep**来构建数据集创建节点，**CreateDatasetStep**及相关对象结构如下。

表 5-18 CreateDatasetStep

属性	描述	是否必填	数据类型
name	数据集创建节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复。	是	str
inputs	数据集创建节点的输入列表。	是	CreateDatasetInput或者CreateDatasetInput的列表
outputs	数据集创建节点的输出列表。	是	CreateDatasetOutput或者CreateDatasetOutput的列表
properties	数据集创建相关的配置信息。	是	DatasetProperties
title	title信息，主要用于前端的名称展示。	否	str
description	数据集创建节点的描述信息。	否	str
policy	节点执行的policy。	否	StepPolicy
depend_steps	依赖的节点列表。	否	Step或者Step的列表

表 5-19 CreateDatasetInput

属性	描述	是否必填	数据类型
name	数据集创建节点的输入名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输入名称不能重复。	是	str

属性	描述	是否必填	数据类型
data	数据集创建节点的输入数据对象。	是	OBS相关对象，当前仅支持OBSPath、OBSConsumption、OBSPlaceholder、DataConsumptionSelector

表 5-20 CreateDatasetOutput

属性	描述	是否必填	数据类型
name	数据集创建节点的输出名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输出名称不能重复。	是	str
config	数据集创建节点的输出相关配置。	是	当前仅支持OBSOutputConfig

表 5-21 DatasetProperties

属性	描述	是否必填	数据类型
dataset_name	数据集的名称，只能是中文、字母、数字、下划线或中划线组成的合法字符串，长度为1-100位。	是	str、Placeholder
dataset_format	数据集格式，默认为0，表示文件类型。	否	0：文件类型 1：表格类型
data_type	数据类型，默认为FREE_FORMAT。	否	DataTypeEnum
description	描述信息。	否	str
import_data	是否要导入数据，当前只支持表格数据，默认为False。	否	bool
work_path_type	数据集输出路径类型，当前仅支持OBS，默认为0。	否	int

属性	描述	是否必填	数据类型
import_config	标签导入的相关配置，默认为None，当基于已标注的数据创建数据集时，可指定该字段导入相关标注信息。	否	ImportConfig

表 5-22 Importconfig

属性	描述	是否必填	数据类型
import_annotations	是否自动导入输入目录下的标注信息，支持检测/图像分类/文本分类。可选值如下： ● true：导入输入目录下的标注信息（默认值） ● false：不导入输入目录下的标注信息	否	str、Placeholder
import_type	导入方式。可选值如下： ● dir：目录导入 ● manifest：按manifest文件导入	否	0：文件类型 ImportTypeEnum
annotation_format_config	导入的标注格式的配置参数。	否	DAnnotationFormaTypeEtConumfig的列表

表 5-23 AnnotationFormatConfig

属性	描述	是否必填	数据类型
format_name	标注格式的名称。	否	AnnotationFormatEnum
scene	标注场景，可选参数。	否	LabelTaskTypeEnum

枚举类型	枚举值
ImportTypeEnum	DIR MANIFEST

枚举类型	枚举值
DataTypeEnum	IMAGE TEXT AUDIO TABULAR VIDEO FREE_FORMAT
AnnotationFormatEnum	MA_IMAGE_CLASSIFICATION_V1 MA_IMAGENET_V1 MA_PASCAL_VOC_V1 YOLO MA_IMAGE_SEGMENTATION_V1 MA_TEXT_CLASSIFICATION_COMBINE_V1 MA_TEXT_CLASSIFICATION_V1 MA_AUDIO_CLASSIFICATION_DIR_V1

使用案例

主要包含两种场景的用例。

- 基于未标注数据创建数据集
- 基于已标注的数据创建数据集，并自动导入标注信息

基于未标注数据创建数据集

数据准备：存储在OBS文件夹中的未标注的数据。

```
from modelarts import workflow as wf
# 通过CreateDatasetStep将存储在OBS中的数据创建成一个新版数据集

# 定义数据集输出路径参数
dataset_output_path = wf.Placeholder(name="dataset_output_path",
placeholder_type=wf.PlaceholderType.STR, placeholder_format="obs")

# 定义数据集名称参数
dataset_name = wf.Placeholder(name="dataset_name", placeholder_type=wf.PlaceholderType.STR)

create_dataset = wf.steps.CreateDatasetStep(
    name="create_dataset",# 数据集创建节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="数据集创建", # 标题信息，不填默认使用name值
    inputs=wf.steps.CreateDatasetInput(name="input_name",
data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")),# CreateDatasetStep的输入，数据在运行时进行配置；data字段也可使用wf.data.OBSPath(obs_path="fake_obs_path")对象表示
    outputs=wf.steps.CreateDatasetOutput(name="output_name",
config=wf.data.OBSOutputConfig(obs_path=dataset_output_path)),# CreateDatasetStep的输出
    properties=wf.steps.DatasetProperties(
        dataset_name=dataset_name, # 该名称对应的数据集如果不存在，则创建新的数据集;如果已存在，则直接使用该名称对应的数据集
        data_type=wf.data.DataTypeEnum.IMAGE, # 数据集对应的数据类型, 示例为图像
    )
```

```
)
# 注意dataset_name这个参数配置的数据集名称需要用户自行确认在该账号下未被他人使用，否则会导致期望的数据集未被创建，而后续节点错误使用了他人创建的数据集

workflow = wf.Workflow(
    name="create-dataset-demo",
    desc="this is a demo workflow",
    steps=[create_dataset]
)
```

基于已标注数据创建数据集，并导入标注信息

数据准备：存储在OBS文件夹中的已标注数据。

OBS目录导入已标注数据的规范：可参见**OBS目录导入数据规范说明**。

```
from modelarts import workflow as wf
# 通过CreateDatasetStep将存储在OBS中的数据创建成一个新版数据集

# 定义数据集输出路径参数
dataset_output_path = wf.Placeholder(name="dataset_placeholder_name",
placeholder_type=wf.PlaceholderType.STR, placeholder_format="obs")

# 定义数据集名称参数
dataset_name = wf.Placeholder(name="dataset_placeholder_name",
placeholder_type=wf.PlaceholderType.STR)

create_dataset = wf.steps.CreateDatasetStep(
    name="create_dataset",# 数据集创建节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="数据集创建", # 标题信息，不填默认使用name值
    inputs=wf.steps.CreateDatasetInput(name="input_name",
data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")),#
CreateDatasetStep的输入，数据在运行时进行配置；data字段也可使用
wf.data.OBSPath(obs_path="fake_obs_path")对象表示
    outputs=wf.steps.CreateDatasetOutput(name="output_name",
config=wf.data.OBSOutputConfig(obs_path=dataset_output_path)),# CreateDatasetStep的输出
    properties=wf.steps.DatasetProperties(
        dataset_name=dataset_name, # 该名称对应的数据集如果不存在，则创建新的数据集;如果已存在，则直接使用该名称对应的数据集
        data_type=wf.data.DataTypeEnum.IMAGE, # 数据集对应的数据类型, 示例为图像
        import_config=wf.steps.ImportConfig(
            annotation_format_config=[
                wf.steps.AnnotationFormatConfig(
                    format_name=wf.steps.AnnotationFormatEnum.MA_IMAGE_CLASSIFICATION_V1, # 已标注数据的标注格式
                    scene=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION) # 标注的场景类型
            ]
        )
    )
)
# 注意dataset_name这个参数配置的数据集名称需要用户自行确认在该账号下未被他人使用，否则会导致期望的数据集未被创建，而后续节点错误使用了他人创建的数据集

workflow = wf.Workflow(
    name="create-dataset-demo",
    desc="this is a demo workflow",
    steps=[create_dataset]
)
```

5.3.4.2 创建 Workflow 数据集标注节点

功能介绍

通过对ModelArts数据集能力进行封装，实现数据集的标注功能。数据集标注节点主要用于创建标注任务或对已有的标注任务进行卡点标注，主要用于需要对数据进行人工标注的场景。

属性总览

您可以使用**LabelingStep**来构建数据集标注节点，**LabelingStep**结构如下：

表 5-24 LabelingStep

属性	描述	是否必填	数据类型
name	数据集标注节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复	是	str
inputs	数据集标注节点的输入列表	是	LabelingInput或者LabelingInput的列表
outputs	数据集标注节点的输出列表	是	LabelingOutput或者LabelingOutput的列表
properties	数据集标注相关的配置信息	是	LabelTaskProperties
title	title信息，主要用于前端的名称展示	否	str
description	数据集标注节点的描述信息	否	str
policy	节点执行的policy	否	StepPolicy
depend_steps	依赖的节点列表	否	Step或者Step的列表

表 5-25 LabelingInput

属性	描述	是否必填	数据类型
name	数据集标注节点的输入名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输入名称不能重复	是	str
data	数据集标注节点的输入数据对象	是	数据集或标注任务相关对象，当前仅支持Dataset，DatasetConsumption，DatasetPlaceholder，LabelTask，LabelTaskPlaceholder，LabelTaskConsumption，DataConsumptionSelector

表 5-26 LabelingOutput

属性	描述	是否必填	数据类型
name	数据集标注节点的输出名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输出名称不能重复	是	str

表 5-27 LabelTaskProperties

属性	描述	是否必填	数据类型
task_type	标注任务类型，返回指定标注任务类型的任务列表。	是	LabelTaskTypeEnum
task_name	标注任务名称，名称只能包含中文、字母、数字、中划线和下划线，长度为1-100位。 当输入是数据集对象时，该参数必填	否	str、Placeholder
labels	待创建的标签列表	否	Label
properties	标注任务的属性，可扩展字段，可以记录自定义信息。	否	dict
auto_sync_dataset	标注任务的标注结果是否自动同步至数据集。可选值如下： ● true：标注任务的标注结果自动同步至数据集（默认值） ● false：标注任务的标注结果不自动同步至数据集	否	bool
content_labeling	语音分割标注任务是否开启内容标注，默认开启。	否	bool
description	标注任务描述信息，长度为0-256位，不能包含^!<>=&"'特殊字符。	否	str

表 5-28 Label

属性	描述	是否必填	数据类型
name	标签名称	否	str
property	标签基本属性键值对，如颜色、快捷键等	否	str、dic、Placeholder

属性	描述	是否必填	数据类型
type	标签类型	否	LabelTypeEnum

枚举类型	枚举值
LabelTaskTypeEnum	IMAGE_CLASSIFICATION OBJECT_DETECTION IMAGE_SEGMENTATION TEXT_CLASSIFICATION NAMED_ENTITY_RECOGNITION TEXT_TRIPLE AUDIO_CLASSIFICATION SPEECH_CONTENT SPEECH_SEGMENTATION DATASET_TABULAR VIDEO_ANNOTATION FREE_FORMAT

Workflow 数据集标注节点代码样例

主要包含三种场景的用例：

- 场景一：基于用户指定的数据集创建标注任务，并等待用户标注完成。

  使用场景：

  - 用户只创建了一个未标注完成的数据集，需要在工作流运行时对数据进行人工标注。
  - 可以放在数据集导入节点之后，对导入的新数据进行人工标注。

  数据准备：提前在ModelArts管理控制台创建一个数据集。

```
from modelarts import workflow as wf
# 通过LabelingStep给输入的数据集对象创建新的标注任务，并等待用户标注完成

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 定义标注任务的名称参数
task_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

labeling = wf.steps.LabelingStep(
    name="labeling", # 数据集标注节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="数据集标注", # 标题信息，不填默认使用name值
    properties=wf.steps.LabelTaskProperties(
        task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,  # 标注任务的类型，以图像分类为例
        task_name=task_name  # 该名称对应的标注任务如果不存在则创建，如果存在则直接使用该任务
    ),
    inputs=wf.steps.LabelingInput(name="input_name", data=dataset), # LabelingStep的输入，数据集对
```



```
象在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="fake_dataset_name")表示
    outputs=wf.steps.LabelingOutput(name="output_name"), # LabelingStep的输出
)

workflow = wf.Workflow(
    name="labeling-step-demo",
    desc="this is a demo workflow",
    steps=[labeling]
)
```

- 场景二：基于指定的标注任务进行标注。

  使用场景：

  - 用户基于数据集自主创建了一个标注任务，需要在工作流运行时对数据进行人工标注。
  - 可以放在数据集导入节点之后，对导入的新数据进行人工标注。

  数据准备：提前在ModelArts管理控制台，基于使用的数据集创建一个标注任务。

```
from modelarts import workflow as wf
# 用户输入标注任务，等待用户标注完成

# 定义数据集的标注任务对象
label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

labeling = wf.steps.LabelingStep(
    name="labeling", # 数据集标注节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="数据集标注", # 标题信息，不填默认使用name值
    inputs=wf.steps.LabelingInput(name="input_name", data=label_task), # LabelingStep的输入，标注任务对象在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")来表示
    outputs=wf.steps.LabelingOutput(name="output_name"), # LabelingStep的输出
)

workflow = wf.Workflow(
    name="labeling-step-demo",
    desc="this is a demo workflow",
    steps=[labeling]
)
```

- 场景三：基于数据集创建节点的输出创建标注任务。

  使用场景：数据集创建节点的输出作为数据集数据标注节点的输入。

```
from modelarts import workflow as wf

# 定义数据集输出路径参数
dataset_output_path = wf.Placeholder(name="dataset_output_path",
placeholder_type=wf.PlaceholderType.STR, placeholder_format="obs")

# 定义数据集名称参数
dataset_name = wf.Placeholder(name="dataset_name", placeholder_type=wf.PlaceholderType.STR)

create_dataset = wf.steps.CreateDatasetStep(
    name="create_dataset",# 数据集创建节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="数据集创建", # 标题信息，不填默认使用name值
    inputs=wf.steps.CreateDatasetInput(name="input_name",
data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")),# CreateDatasetStep的输入，数据在运行时进行配置；data字段也可使用wf.data.OBSPath(obs_path="fake_obs_path")对象表示
    outputs=wf.steps.CreateDatasetOutput(name="create_dataset_output",
config=wf.data.OBSOutputConfig(obs_path=dataset_output_path)),# CreateDatasetStep的输出
    properties=wf.steps.DatasetProperties(
        dataset_name=dataset_name, # 该名称对应的数据集如果不存在，则创建新的数据集;如果已存在，则直接使用该名称对应的数据集
        data_type=wf.data.DataTypeEnum.IMAGE, # 数据集对应的数据类型, 示例为图像
    )
)
```



```
# 定义标注任务的名称参数
task_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

labeling = wf.steps.LabelingStep(
    name="labeling", # 数据集标注节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="数据集标注", # 标题信息，不填默认使用name值
    properties=wf.steps.LabelTaskProperties(
        task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,  # 标注任务的类型，以图像分类为例
        task_name=task_name  # 该名称对应的标注任务如果不存在则创建，如果存在则直接使用该任务
    ),
    inputs=wf.steps.LabelingInput(name="input_name", data=create_dataset.outputs["create_dataset_output"].as_input()), # LabelingStep的输入，data数据来源为数据集创建节点的输出
    outputs=wf.steps.LabelingOutput(name="output_name"), # LabelingStep的输出
    depend_steps=create_dataset # 依赖的数据集创建节点对象
)
# create_dataset是 wf.steps.CreateDatasetStep的一个实例，create_dataset_output是wf.steps.CreateDatasetOutput的name字段值

workflow = wf.Workflow(
    name="labeling-step-demo",
    desc="this is a demo workflow",
    steps=[create_dataset, labeling]
)
```

5.3.4.3 创建 Workflow 数据集导入节点

功能介绍

通过对ModelArts数据集能力进行封装，实现数据集的数据导入功能。数据集导入节点主要用于将指定路径下的数据导入到数据集或者标注任务中，主要应用场景如下：

- 适用于数据不断迭代的场景，可以将一些新增的原始数据或者已标注数据导入到标注任务中，并通过后续的数据集标注节点进行标注。
- 对于一些已标注好的原始数据，可以直接导入到数据集或者标注任务中，并通过后续的数据集版本发布节点获取带有版本信息的数据集对象。

属性总览

您可以使用**DatasetImportStep**来构建数据集导入节点，**DatasetImportStep**结构如下。

表 5-29 DatasetImportStep

属性	描述	是否必填	数据类型
name	数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复。	是	str
inputs	数据集导入节点的输入列表。	是	DatasetImportInput或者DatasetImportInput的列表
outputs	数据集导入节点的输出列表。	是	DatasetImportOutput或者DatasetImportOutput的列表
properties	数据集导入相关的配置信息。	是	ImportDataInfo
title	title信息，主要用于前端的名称展示。	否	str
description	数据集导入节点的描述信息。	否	str
policy	节点执行的policy。	否	StepPolicy
depend_steps	依赖的节点列表。	否	Step或者Step的列表

表 5-30 DatasetImportInput

属性	描述	是否必填	数据类型
name	数据集导入节点的输入名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输入名称不能重复。	是	str

属性	描述	是否必填	数据类型
data	数据集导入节点的输入数据对象。	是	数据集、OBS或标注任务相关对象，当前仅支持Dataset，DatasetConsumption，DatasetPlaceholder，OBSPath，OBSConsumption，OBSPlaceholder，LabelTask，LabelTaskPlaceholder，LabelTaskConsumption，DataConsumptionSelector

表 5-31 DatasetImportOutput

属性	描述	是否必填	数据类型
name	数据集导入节点的输出名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输出名称不能重复。	是	str

表 5-32 ImportDataInfo

属性	描述	是否必填	数据类型
annotation_format_config	导入的标注格式的配置参数。	否	AnnotationFormatConfig
excluded_labels	不导入包含指定标签的样本。	否	Label的列表

属性	描述	是否必填	数据类型
import_annotated	用于导入智能标注结果的任务，是否导入原数据集中已标注的样本到待确认，默认值为"false"即不导入原数据集中已标注的样本到待确认。可选值如下： ● true：导入原数据集中已标注的样本到待确认 ● false：不导入原数据集中已标注的样本到待确认	否	bool
import_annotations	是否导入标签。可选值如下： ● true：导入标签（默认值） ● false：不导入标签	否	bool
import_samples	是否导入样本。可选值如下： ● true：导入样本（默认值） ● false：不导入样本	否	bool
import_type	导入方式。可选值如下： ● dir：目录导入 ● manifest：按manifest文件导入	否	ImportTypeEnum
included_labels	导入包含指定标签的样本。	否	Label的列表
label_format	标签格式，此参数仅文本类数据集使用。	否	LabelFormat

表 5-33 AnnotationFormatConfig

属性	描述	是否必填	数据类型
format_name	标注格式的名称。	否	AnnotationFormatEnum
parameters	标注格式的高级参数。	否	AnnotationFormatParameters
scene	标注场景，可选参数。	否	LabelTaskTypeEnum

表 5-34 AnnotationFormatParameters

属性	描述	是否必填	数据类型
difficult_only	是否只导入难例。可选值如下： ● true：只导入难例样本 ● false：导入全部样本（默认值）	否	bool
included_labels	导入包含指定标签的样本。	否	Label的列表
label_separator	标签与标签之间的分隔符，默认为逗号分隔，分隔符需转义。分隔符仅支持一个字符，必须为大小写字母，数字和“!@#$ %^&*_=|?/':;,”其中的某一字符。	否	str
sample_label_separator	文本与标签之间的分隔符，默认为Tab键分隔，分隔符需转义。分隔符仅支持一个字符，必须为大小写字母，数字和“!@#$ %^&*_=|?/':;,”其中的某一字符。	否	str

使用案例

主要包含三种场景的用例：

- 场景一：将指定存储路径下的数据导入到目标数据集中。适用于需要对数据集进行数据更新的操作。
  - 用户将指定路径下已标注的数据导入到数据集中（同时导入标签信息），后续可增加数据集版本发布节点进行版本发布。

    数据准备：提前在ModelArts管理控制台，创建数据集，并将已标注的数据上传至OBS中。

```
from modelarts import workflow as wf
# 通过DatasetImportStep将指定路径下的数据导入到数据集中，输出数据集对象

# 定义数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 定义OBS数据对象
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

dataset_import = wf.steps.DatasetImportStep(
    name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="数据集导入", # 标题信息，不填默认使用name值
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=dataset), # 目标数据集在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="dataset_name")表示
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
    ],# DatasetImportStep的输入
    outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep的输出
    properties=wf.steps.ImportDataInfo(
        annotation_format_config=[
            wf.steps.AnnotationFormatConfig(
                format_name=wf.steps.AnnotationFormatEnum.MA_IMAGE_CLASSIFICATION_V1, # 已标注数据的标注格式，示例为图像分类
                scene=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION # 标注的场景类型
            )
        ]
    )
)

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

- 用户将指定路径下未标注的数据导入到数据集中，后续可增加数据集标注节点对新增数据进行标注。

  数据准备：提前在ModelArts界面创建数据集，并将未标注的数据上传至OBS中。

```
from modelarts import workflow as wf
# 通过DatasetImportStep将指定路径下的数据导入到数据集中，输出数据集对象

# 定义数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 定义OBS数据对象
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

dataset_import = wf.steps.DatasetImportStep(
    name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="数据集导入", # 标题信息，不填默认使用name值
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=dataset), # 目标数据集在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="dataset_name")表示
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
    ],# DatasetImportStep的输入
    outputs=wf.steps.DatasetImportOutput(name="output_name") # DatasetImportStep的输出
)

workflow = wf.Workflow(
```



```
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

- 场景二：将指定存储路径下的数据导入到指定标注任务中。适用于需要对标注任务进行数据更新的操作。
  - 用户将指定路径下已标注的数据导入到标注任务中（同时导入标签信息），后续可增加数据集版本发布节点进行版本发布。

    数据准备：基于使用的数据集，提前创建标注任务，并将已标注的数据上传至OBS中。

    ```
    from modelarts import workflow as wf
    # 通过DatasetImportStep将指定路径下的数据导入到标注任务中，输出标注任务对象

    # 定义标注任务对象
    label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

    # 定义OBS数据对象
    obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

    dataset_import = wf.steps.DatasetImportStep(
        name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
        title="数据集导入", # 标题信息，不填默认使用name值
        inputs=[
            wf.steps.DatasetImportInput(name="input_name_1", data=label_task), # 目标标注任务对象，在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")表示
            wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
        ],# DatasetImportStep的输入
        outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep的输出
        properties=wf.steps.ImportDataInfo(
            annotation_format_config=[
                wf.steps.AnnotationFormatConfig(
                    format_name=wf.steps.AnnotationFormatEnum.MA_IMAGE_CLASSIFICATION_V1, # 已标注数据的标注格式，示例为图像分类
                    scene=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION # 标注的场景类型
                )
            ]
        )
    )

    workflow = wf.Workflow(
        name="dataset-import-demo",
        desc="this is a demo workflow",
        steps=[dataset_import]
    )
    ```

  - 用户将指定路径下未标注的数据导入到标注任务中，后续可增加数据集标注节点对新增数据进行标注。

    数据准备：基于使用的数据集，提前创建标注任务，并将未标注的数据上传至OBS中。

    ```
    from modelarts import workflow as wf
    # 通过DatasetImportStep将指定路径下的数据导入到标注任务中，输出标注任务对象

    # 定义标注任务对象
    label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

    # 定义OBS数据对象
    obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

    dataset_import = wf.steps.DatasetImportStep(
    ```

```
    name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="数据集导入", # 标题信息，不填默认使用name值
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=label_task), # 目标标注任务对象，在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")表示
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
    ],# DatasetImportStep的输入
    outputs=wf.steps.DatasetImportOutput(name="output_name") # DatasetImportStep的输出
)

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

- 场景三：基于数据集创建节点构建数据集导入节点。数据集创建节点的输出作为数据集导入节点的输入。

```
from modelarts import workflow as wf
# 通过DatasetImportStep将指定路径下的数据导入到数据集中，输出数据集对象

# 定义OBS数据对象
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

dataset_import = wf.steps.DatasetImportStep(
    name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="数据集导入", # 标题信息，不填默认使用name值
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=create_dataset.outputs["create_dataset_output"].as_input()), # 数据集创建节点的输出作为导入节点的输入
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
    ],# DatasetImportStep的输入
    outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep的输出
    depend_steps=create_dataset # 依赖的数据集创建节点对象
)
# create_dataset是 wf.steps.CreateDatasetStep的一个实例，create_dataset_output是wf.steps.CreateDatasetOutput的name字段值

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

5.3.4.4 创建 Workflow 数据集版本发布节点

功能介绍

通过对ModelArts数据集能力进行封装，实现数据集的版本自动发布的功能。数据集版本发布节点主要用于将已存在的数据集或者标注任务进行版本发布，每个版本相当于数据的一个快照，可用于后续的数据溯源。主要应用场景如下：

- 对于数据标注这种操作，可以在标注完成后自动帮助用户发布新的数据集版本，结合as_input的能力提供给后续节点使用。
- 当模型训练需要更新数据时，可以使用数据集导入节点先导入新的数据，然后再通过该节点发布新的版本供后续节点使用。

属性总览

您可以使用**ReleaseDatasetStep**来构建数据集版本发布节点，**ReleaseDatasetStep**结构如下：

表 5-35 ReleaseDatasetStep

属性	描述	是否必填	数据类型
name	数据集版本发布节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复	是	str
inputs	数据集版本发布节点的输入列表	是	ReleaseDatasetInput或者ReleaseDatasetInput的列表
outputs	数据集版本发布节点的输出列表	是	ReleaseDatasetOutput或者ReleaseDatasetOutput的列表
title	title信息，主要用于前端的名称展示	否	str
description	数据集版本发布节点的描述信息	否	str
policy	节点执行的policy	否	StepPolicy
depend_steps	依赖的节点列表	否	Step或者Step的列表

表 5-36 ReleaseDatasetInput

属性	描述	是否必填	数据类型
name	数据集版本发布节点的输入名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输入名称不能重复	是	str

属性	描述	是否必填	数据类型
data	数据集版本发布节点的输入数据对象	是	数据集或标注任务相关对象，当前仅支持Dataset，DatasetConsumption，DatasetPlaceholder，LabelTask，LabelTaskPlaceholder，LabelTaskConsumption，DataConsumptionSelector

表 5-37 ReleaseDatasetOutput

属性	描述	是否必填	数据类型
name	数据集版本发布节点的输出名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输出名称不能重复	是	str
dataset_version_config	数据集版本发布相关配置信息	是	DatasetVersionConfig

表4 DatasetVersionConfig

属性	描述	是否必填	数据类型
version_name	数据集版本名称，推荐使用类似V001的格式，不填则默认从V001往上递增。	否	str或者Placeholder
version_format	版本格式，默认为"Default"，也可支持"CarbonData"。	否	str
train_evaluate_sample_ratio	训练-验证集比例，默认值为"1.00"。取值范围为0-1.00， 例如"0.8"表示训练集比例为80%，验证集比例为20%。	否	str或者Placeholder
clear_hard_property	是否清除难例，默认为“True”。	否	bool或者Placeholder

属性	描述	是否必填	数据类型
remove_sample_usage	是否清除数据集已有的usage信息，默认为“True”。	否	bool或者Placeholder
label_task_type	标注任务的类型。当输入是数据集时，该字段必填，用来指定数据集版本的标注场景。输入是标注任务时该字段不用填写。	否	LabelTaskTypeEnum 支持以下几种类型： • IMAGE_CLASSIFICATION（图像分类） • OBJECT_DETECTION = 1（物体检测） • IMAGE_SEGMENTATION（图像分割） • TEXT_CLASSIFICATION（文本分类） • NAMED_ENTITY_RECOGNITION（命名实体） • TEXT_TRIPLE（文本三元组） • AUDIO_CLASSIFICATION（声音分类） • SPEECH_CONTENT（语音内容）SPEECH_SEGMENTATION（语音分割） • TABLE（表格数据） • VIDEO_ANNOTATION（视频标注）
description	版本描述信息。	否	str

说明

如果您没有特殊需求，则可直接使用内置的默认值，例如example = DatasetVersionConfig()

使用案例

场景一：基于数据集发布版本

使用场景：当数据集更新了数据时，可以通过该节点发布新的数据集版本供后续的节点使用。

```
from modelarts import workflow as wf
# 通过ReleaseDatasetStep将输入的数据集对象发布新的版本，输出带有版本信息的数据集对象

# 定义数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 定义训练验证切分比参数
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR,
default="0.8")

release_version = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # 数据集发布节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="数据集版本发布", # 标题信息，不填默认使用name值
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=dataset), # ReleaseDatasetStep的输入，数据集对象在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="dataset_name")表示
    outputs=wf.steps.ReleaseDatasetOutput(
        name="output_name",
        dataset_version_config=wf.data.DatasetVersionConfig(
            label_task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,  # 数据集发布版本时需要指定标注任务的类型
            train_evaluate_sample_ratio=train_ration # 数据集的训练验证切分比
            )
    ) # ReleaseDatasetStep的输出
)

workflow = wf.Workflow(
    name="dataset-release-demo",
    desc="this is a demo workflow",
    steps=[release_version]
)
```

场景二：基于标注任务发布版本

当标注任务更新了数据或者标注信息时，可以通过该节点发布新的数据集版本供后续的节点使用。

```
from modelarts import workflow as wf
# 通过ReleaseDatasetStep将输入的标注任务对象发布新的版本，输出带有版本信息的数据集对象

# 定义标注任务对象
label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

# 定义训练验证切分比参数
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR,
default="0.8")

release_version = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # 数据集发布节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="数据集版本发布", # 标题信息，不填默认使用name值
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=label_task), # ReleaseDatasetStep的输入，
标注任务对象在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")表示
    outputs=wf.steps.ReleaseDatasetOutput(name="output_name", dataset_version_config=wf.data.DatasetVersionConfig(train_evaluate_sample_ratio=train_ration)), # 数据集的训练验证切分比
)

workflow = wf.Workflow(
    name="dataset-release-demo",
    desc="this is a demo workflow",
    steps=[release_version]
)
```


场景三：基于数据集标注节点，构建数据集版本发布节点

使用场景：数据集标注节点的输出作为数据集版本发布节点的输入。

```
from modelarts import workflow as wf
# 通过ReleaseDatasetStep将输入的标注任务对象发布新的版本，输出带有版本信息的数据集对象

# 定义训练验证切分比参数
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR, default="0.8")

release_version = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # 数据集发布节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="数据集版本发布", # 标题信息，不填默认使用name值
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=labeling_step.outputs["output_name"].as_input()), # ReleaseDatasetStep的输入，标注任务对象在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")表示
    outputs=wf.steps.ReleaseDatasetOutput(name="output_name", dataset_version_config=wf.data.DatasetVersionConfig(train_evaluate_sample_ratio=train_ration)), # 数据集的训练验证切分比
    depend_steps = [labeling_step] # 依赖的数据集标注节点对象
)
# labeling_step是wf.steps.LabelingStep的实例对象，output_name是wf.steps.LabelingOutput的name字段值

workflow = wf.Workflow(
    name="dataset-release-demo",
    desc="this is a demo workflow",
    steps=[release_version]
)
```

5.3.4.5 创建 Workflow 训练作业节点

功能介绍

该节点通过对算法、输入、输出的定义，实现ModelArts作业管理的能力。主要用于数据处理、模型训练、模型评估等场景。主要应用场景如下：

- 当需要对图像进行增强，对语音进行除噪等操作时，可以使用该节点进行数据的预处理。
- 对于一些物体检测，图像分类等模型场景，可以根据已有的数据使用该节点进行模型的训练。

属性总览

您可以使用**JobStep**来构建作业类型节点，**JobStep**结构如下

表 5-38 JobStep

属性	描述	是否必填	数据类型
name	作业节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复	是	str
algorithm	算法对象	是	• BaseAlgorithm • Algorithm • AIGalleryAlgorithm
spec	作业使用的资源规格相关配置	是	JobSpec
inputs	作业节点的输入列表	是	JobInput或者JobInput的列表
outputs	作业节点的输出列表	是	JobOutput或者JobOutput的列表
title	title信息，主要用于前端的名称展示	否	str
description	作业节点的描述信息	否	str
policy	节点执行的policy	否	StepPolicy
depend_steps	依赖的节点列表	否	Step或者Step的列表

表 5-39 JobInput

属性	描述	是否必填	数据类型
name	作业类型节点的输入名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输入名称不能重复	是	str
data	作业类型节点的输入数据对象	是	数据集或OBS相关对象，当前仅支持Dataset、DatasetPlaceholder、DatasetConsumption、OBSPath、OBSConsumption、OBSPlaceholder、DataConsumptionSelector

表 5-40 JobOutput

属性	描述	是否必填	数据类型
name	作业类型节点的输出名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输出名称不能重复	是	str
obs_config	输出的OBS相关配置	否	OBSOutputConfig
model_config	输出的模型相关配置	否	ModelConfig
metrics_config	metrics相关配置	否	MetricsConfig

表 5-41 OBSOutputConfig

属性	描述	是否必填	数据类型
obs_path	已存在的OBS目录	是	str、Placeholder、Storage
metric_file	存储metric信息的文件名称	否	str、Placeholder

表 5-42 BaseAlgorithm

属性	描述	是否必填	数据类型
id	算法管理的算法ID	否	str
subscription_id	订阅算法的订阅ID	否	str
item_version_id	订阅算法的版本号	否	str
code_dir	代码目录	否	str、Placeholder、Storage
boot_file	启动文件	否	str、Placeholder、Storage
command	启动命令	否	str、Placeholder
parameters	算法超参	否	AlgorithmParameters的列表
engine	作业使用的镜像信息	否	JobEngine
environments	环境变量	否	dict

表 5-43 Algorithm

属性	描述	是否必填	数据类型
algorithm_id	算法管理的算法ID	是	str
parameters	算法超参	否	Algorithm Parameters的列表

表 5-44 AIGalleryAlgorithm

属性	描述	是否必填	数据类型
subscription_id	订阅算法的订阅ID	是	str
item_version_id	订阅算法的版本号	是	str
parameters	算法超参	否	Algorithm Parameters的列表

表 5-45 AlgorithmParameters

属性	描述	是否必填	数据类型
name	算法超参的名称	是	str
value	算法超参的值	是	int、bool、float、str、Placeholder、Storage

表 5-46 JobEngine

属性	描述	是否必填	数据类型
engine_id	镜像ID	否	str、Placeholder
engine_name	镜像名称	否	str、Placeholder
engine_version	镜像版本	否	str、Placeholder
image_url	镜像url	否	str、Placeholder

表 5-47 JobSpec

属性	描述	是否必填	数据类型
resource	资源信息	是	JobResource
log_export_path	日志输出路径	否	LogExportPath
schedule_policy	作业调度配置策略	否	SchedulePolicy
volumes	作业挂载的文件系统信息	否	list[Volume]

表 5-48 JobResource

属性	描述	是否必填	数据类型
flavor	资源规格	是	Placeholder
node_count	节点个数，默认为1，多节点表示支持分布式	否	int、Placeholder

表 5-49 SchedulePolicy

属性	描述	是否必填	数据类型
priority	作业调度的优先级，仅支持配置为1、2、3，分别对应低、中、高三种优先级	是	int、Placeholder

表 5-50 Volume

属性	描述	是否必填	数据类型
nfs	NFS文件系统对象，在一个Volume对象中，nfs、pacific、pfs同时只能配置一个	否	NFS
pacific	pacific文件系统对象，在一个Volume对象中，nfs、pacific、pfs同时只能配置一个	否	Placeholder
pfs	OBS并行文件系统对象，在一个Volume对象中，nfs、pacific、pfs同时只能配置一个	否	PFS、Placeholder

表 5-51 NFS

属性	描述	是否必填	数据类型
nfs_server_path	NFS文件系统的服务地址	是	str、Placeholder

属性	描述	是否必填	数据类型
local_path	挂载到容器里面的路径	是	str、Placeholder
read_only	是否只读的方式挂载	否	bool、Placeholder

表 5-52 PFS

属性	描述	是否必填	数据类型
pfs_path	并行文件系统的路径	是	str、Placeholder
local_path	挂载到容器里面的路径	是	str、Placeholder

资源规格查询

您在创建作业类型节点之前可以通过以下操作来获取该账号所支持的训练资源规格列表以及引擎规格列表：

- 导包

```
from modelarts.session import Session
from modelarts.estimatorV2 import TrainingJob
from modelarts.workflow.client.job_client import JobClient
```

- session初始化

```
# 如果您在本地IDEA环境中开发工作流，则Session初始化使用如下方式
# 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
# 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_SDK_SK"]
# 如果进行了加密还需要进行解密操作
session = Session(
    access_key=__AK, # 账号的AK信息
    secret_key=__SK, # 账号的SK信息
    region_name="***", # 账号所属的region
    project_id="***" ,# 账号的项目ID
)

# 如果您在Notebook环境中开发工作流，则Session初始化使用如下方式
session = Session()
```

- 公共池查询

```
# 公共资源池规格列表查询
spec_list = TrainingJob(session).get_train_instance_types(session) # 返回的类型为list,可按需打印查看
print(spec_list)
```

- 专属池查询

```
# 运行中的专属资源池列表查询
pool_list = JobClient(session).get_pool_list() # 返回专属资源池的详情列表
pool_id_list = JobClient(session).get_pool_id_list() # 返回专属资源池ID列表
专属资源池规格ID列表如下，根据所选资源池的实际规格自行选择：
    1. modelarts.pool.visual.xlarge 对应1卡
    2. modelarts.pool.visual.2xlarge 对应2卡
```

```
    3. modelarts.pool.visual.4xlarge 对应4卡
    4. modelarts.pool.visual.8xlarge 对应8卡
```

- 引擎规格查询

```
# 引擎规格查询
engine_dict = TrainingJob(session).get_engine_list(session) # 返回的类型为dict,可按需打印查看
print(engine_dict)
```

使用案例

主要包含七种场景的用例：

- 使用订阅自AI Gallery的算法
- 使用算法管理中的算法
- 使用自定义算法（代码目录+启动文件+官方镜像）
- 使用自定义算法（代码目录+脚本命令+自定义镜像）
- 基于数据集版本发布节点构建作业类型节点
- 作业类型节点结合可视化能力
- 输入使用DataSelector对象，支持选择OBS或者数据集

使用订阅自AI Gallery的算法

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") # name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID，也可直接填写版本号
        item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR, default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值

    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表示
    outputs=wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格")

        )
    )# 训练资源规格信息
)

workflow = wf.Workflow(
```



```
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

使用算法管理中的算法

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") #
name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），
并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.Algorithm(
        algorithm_id="algorithm_id", # 算法ID
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中的算法来源于算法管理；部分算法超参的值如果无需修改，则在parameters
字段中可以不填写，系统自动填充相关超参值

    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段也可使
用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表示
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    )# 训练资源规格信息
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

使用自定义算法（代码目录+启动文件+官方镜像）

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") #
name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），
并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
```



```
    algorithm=wf.BaseAlgorithm(
        code_dir="fake_code_dir", # 代码目录存储的路径
        boot_file="fake_boot_file", # 启动文件存储路径，需要在代码目录下
        engine=wf.steps.JobEngine(engine_name="fake_engine_name",
engine_version="fake_engine_version"), # 官方镜像的名称以及版本信息

        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 自定义算法使用代码目录+启动文件+官方镜像的方式实现


    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段也可使
用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表示
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    )# 训练资源规格信息
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

使用自定义算法（代码目录+脚本命令+自定义镜像）

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") #
name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），
并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.BaseAlgorithm(
        code_dir="fake_code_dir", # 代码目录存储的路径
        command="fake_command", # 执行的脚本命令
        engine=wf.steps.JobEngine(image_url="fake_image_url"), # 自定义镜像的url，格式为：组织名/镜像名
称:版本号，不需要携带相应的域名地址；如果image_url需要设置为运行态可配置，则使用如下方式：
image_url=wf.Placeholder(name="image_url", placeholder_type=wf.PlaceholderType.STR,
placeholder_format="swr", description="自定义镜像")
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), 自定义算法使用代码目录+脚本命令+自定义镜像的方式实现

    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段也可使
用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表示
```

```
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    )# 训练资源规格信息
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

说明

上述四种方式使用数据集对象作为输入，如果您需要使用OBS路径作为输入时，只需将JobInput中的data数据替换为**data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")**或者**data=wf.data.OBSPath(obs_path="fake_obs_path")**即可。

此外，在构建工作流时就指定好数据集对象或者OBS路径的方式可以减少配置操作，方便您在开发态进行调试。但是对于发布到运行态或者gallery的工作流，更推荐的方式是采用数据占位符的方式进行编写，您可以在工作流启动之前对参数进行配置，自由度更高。

基于数据集版本发布节点构建作业类型节点

使用场景：数据集版本发布节点的输出作为作业类型节点的输入。

```
from modelarts import workflow as wf

# 定义数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 定义训练验证切分比参数
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR,
default="0.8")

release_version_step = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # 数据集发布节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="数据集版本发布", # 标题信息，不填默认使用name值
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=dataset), # ReleaseDatasetStep的输入，数据集对象在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="dataset_name")表示
    outputs=wf.steps.ReleaseDatasetOutput(
        name="output_name",
        dataset_version_config=wf.data.DatasetVersionConfig(
            label_task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,  # 数据集发布版本时需要指定标注任务的类型
            train_evaluate_sample_ratio=train_ration # 数据集的训练验证切分比
            )
    ) # ReleaseDatasetStep的输出
)

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") # name字段必填，title, description可选填

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID
        parameters=[
```



```
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中使用AI Gallery订阅的算法；部分算法超参的值如果无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值


    inputs=wf.steps.JobInput(name="data_url",
data=release_version_step.outputs["output_name"].as_input()), # 使用数据集版本发布节点的输出作为JobStep
的输入
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    ), # 训练资源规格信息
    depend_steps=release_version_step # 依赖的数据集版本发布节点对象
)
# release_version_step是wf.steps.ReleaseDatasetStep的实例对象，output_name是
wf.steps.ReleaseDatasetOutput的name字段值

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[release_version_step, job_step],
    storages=[storage]
)
```

作业类型节点结合可视化能力

节点可视化特性将用户在使用Workflow时产生的一些衡量指标进行一个可视化的展示，支持数据的实时可视化，并且允许独立呈现可视化外挂节点。形态上基于作业类型节点原有的使用方式，新增一个针对metrics信息展示的输出，通过MetricsConfig对象进行配置。

表 5-53 MetricsConfig

属性	描述	是否必填	数据类型
metric_files	metrics输出文件列表	是	list，列表内元素支持（str、Placeholder、Storage）
realtime_visualization	输出的metrics信息是否需要实时展示	否	bool，默认为False
visualization	是否呈现独立的可视化节点	否	bool，默认为True

对于输出的metrics文件，数据内容必须为标准的json数据，大小限制为1M，并且与当前支持的几种数据格式保持一致：

- 键值对类型的数据

```
[
  {
```



```
    "key": "loss",
    "title": "loss",
    "type": "float",
    "data": {
      "value": 1.2
    }
  },
  {
    "key": "accuracy",
    "title": "accuracy",
    "type": "float",
    "data": {
      "value": 1.6
    }
  }
]
```

- 折线图数据(type是line chart)

```
[
  {
    "key": "metric",
    "title": "metric",
    "type": "line chart",
    "data": {
      "x_axis": [
        {
          "title": "step/epoch",
          "value": [
            1,
            2,
            3
          ]
        }
      ],
      "y_axis": [
        {
          "title": "value",
          "value": [
            0.5,
            0.4,
            0.3
          ]
        }
      ]
    }
  }
]
```

- 柱状图数据(type是histogram)

```
[
  {
    "key": "metric",
    "title": "metric",
    "type": "histogram",
    "data": {
      "x_axis": [
        {
          "title": "step/epoch",
          "value": [
            1,
            2,
            3
          ]
        }
      ],
      "y_axis": [
        {
          "title": "value",
          "value": [
            0.5,
```

```
                0.4,
                0.3
            ]
        }
    ]
  }
 }
]
```

- 混淆矩阵

```
[
  {
    "key": "confusion_matrix",
    "title": "confusion_matrix",
    "type": "table",
    "data": {
      "cell_value": [
        [
          1,
          2
        ],
        [
          2,
          3
        ]
      ],
      "col_labels": {
        "title": "labels",
        "value": [
          "daisy",
          "dandelion"
        ]
      },
      "row_labels": {
        "title": "predictions",
        "value": [
          "daisy",
          "dandelion"
        ]
      }
    }
  }
]
```

- 一维表格

```
[
  {
    "key": "Application Evaluation Results",
    "title": "Application Evaluation Results",
    "type": "one-dimensional-table",
    "data": {
      "cell_value": [
        [
          10,
          2,
          0.5
        ]
      ],
      "labels": [
        "samples",
        "maxResTine",
        "p99"
      ]
    }
  }
]
```

使用案例：

```
from modelarts import workflow as wf

# 构建一个Storage对象，对训练输出目录做统一管理
```



```
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info",
with_execution_id=True, create_dir=True) # name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线
（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]

    ), # 训练使用的算法对象，示例中使用AI Gallery订阅的算法；部分算法超参的值如果无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值


    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段
也可使用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表
示
    outputs=[
        wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),# JobStep的输出
        wf.steps.JobOutput(name="metrics_output",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # 相关metrics信息由作业的脚本代码自行输出到配置的路径下
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    )# 训练资源规格信息
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

说明

Workflow不会自动获取训练输出的指标信息，要求用户自行在算法代码中获取指标信息并且按照指定的数据格式构造出metrics.json文件，自行上传到MetricsConfig中配置的OBS路径下，Workflow只进行数据的读取以及渲染展示。

输入使用DataSelector对象，支持选择OBS或者数据集

该方式主要用于输入支持可选择的场景，使用DataSelector对象作为输入时，用户在页面配置时可自由选择数据集对象或者OBS对象作为训练的输入，代码示例如下：

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") #
name字段必填，title, description可选填
```



```
# 定义DataSelector对象
data_selector = wf.data.DataSelector(name="input_data", data_type_list=["dataset", "obs"])

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID，也可直接填写版本号
        item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR, default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值

    inputs=wf.steps.JobInput(name="data_url", data=data_selector), # JobStep的输入在运行时配置,可自由选择OBS或者数据集作为输入
    outputs=wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格")

        )
    )# 训练资源规格信息
)

workflow = wf.Workflow(
    name="job-step-demo",
    desc="this is a demo workflow",
    steps=[job_step],
    storages=[storage]
)
```

说明

使用DataSelector作为输入时，需要用户自行保证算法的输入同时支持数据集或者OBS。

5.3.4.6 创建 Workflow 模型注册节点

功能介绍

通过对ModelArts模型管理的能力进行封装，实现将训练后的结果注册到模型管理中，便于后续服务部署、更新等步骤的执行。主要应用场景如下：

- 注册ModelArts训练作业中训练完成的模型。
- 注册自定义镜像中的模型。

属性总览

您可以使用**ModelStep**来构建模型注册节点，**ModelStep**结构如下：

表 5-54 ModelStep

属性	描述	是否必填	数据类型
name	模型注册节点的名称。只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符，一个Workflow里的两个step名称不能重复	是	str
inputs	模型注册节点的输入列表	否	ModelInput或者ModelInput的列表
outputs	模型注册节点的输出列表	是	ModelOutput或者ModelOutput的列表
title	title信息，主要用于前端的名称展示	否	str
description	模型注册节点的描述信息	否	str
policy	节点执行的policy	否	StepPolicy
depend_steps	依赖的节点列表	否	Step或者Step的列表

表 5-55 ModelInput

属性	描述	是否必填	数据类型
name	模型注册节点的输入名称，只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符。同一个Step的输入名称不能重复	是	str
data	模型注册节点的输入数据对象	是	OBS、SWR或订阅模型相关对象，当前仅支持OBSPath、SWRImage、OBSConsumption、OBSPlaceholder、SWRImagePlaceholder、DataConsumptionSelector、GalleryModel

表 5-56 ModelOutput

属性	描述	是否必填	数据类型
name	模型注册节点的输出名称，只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符。同一个Step的输出名称不能重复	是	str
model_config	模型注册相关配置信息	是	ModelConfig

表 5-57 ModelConfig

属性	描述	是否必填	数据类型
model_type	模型的类型，支持的格式有（"TensorFlow", "MXNet", "Caffe", "Spark_MLlib", "Scikit_Learn", "XGBoost", "Image", "PyTorch", "Template"，"Custom"）默认为TensorFlow。	是	str
model_name	模型的名称，支持1-64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线。	否	str、Placeholder
model_version	模型的版本，格式需为“数值.数值.数值”，其中数值为1-2位正整数。该字段不填时，版本号自动增加。 **注意** 版本不可以出现例如01.01.01等以0开头的版本号形式。	否	str、Placeholder
runtime	模型运行时环境，runtime可选值与model_type相同。	否	str、Placeholder
description	模型备注信息，1-100位长度，不能包含&!'"<>=	否	str
execution_code	执行代码存放的OBS地址，默认值为空，名称固定为“customize_service.py”。 推理代码文件需存放在模型“model”目录。该字段不需要填，系统也能自动识别出model目录下的推理代码。	否	str
dependencies	推理代码及模型需安装的包，默认为空。从配置文件读取。	否	str

属性	描述	是否必填	数据类型
model_metrics	模型精度信息，从配置文件读取。	否	str
apis	模型所有的apis入参出参信息（选填），从配置文件中解析出来。	否	str
initial_config	模型配置相关数据。	否	dict
template	模板的相关配置项，使用模板导入模型(即model_type为Template)时必选	否	Template
dynamic_load_mode	动态加载模式，当前仅支持"Single"	否	str、Placeholder
prebuild	模型是否提前构建，默认为False	否	bool、Placeholder
install_type	模型的安装类型，支持"real_time", "edge", "batch"，该字段不填时默认均支持	否	list[str]

表 5-58 Template

属性	描述	是否必填	数据类型
template_id	所使用的模板ID，模板中会内置一个输入输出模式	是	str、Placeholder
infer_format	输入输出模式ID，提供时覆盖模板中的内置输入输出模式	否	str、Placeholder
template_inputs	模板输入项配置，即配置模型的源路径	是	list of TemplateInputs object

表 5-59 TemplateInputs

属性	描述	是否必填	数据类型
input_id	输入项ID，从模板详情中获取	是	str、Placeholder
input	模板输入路径，可以是OBS文件路径或OBS目录路径。使用多输入项的模板创建模型时，如果模板定义的目标路径input_properties是一样的，则此处输入的obs目录或者obs文件不能重名，否则会覆盖。	是	str、Placeholder、Storage

使用案例

主要包含六种场景的用例：

- 基于JobStep的输出注册模型
- 基于OBS数据注册模型
- 使用模板方式注册模型
- 使用自定义镜像注册模型
- 使用自定义镜像+OBS的方式注册模型
- 使用订阅模型+OBS的方式注册模型

从训练作业中注册模型（模型输入来源 JobStep 的输出）

```
import modelarts.workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info", description="description_info") #
name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），
并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID，也可直接填写版本号
        item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值

    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段也可使
用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表示
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    )# 训练资源规格信息
)

# 通过ModelStep来定义一个模型注册节点，输入来源于JobStep的输出

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中
划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="模型注册", # 标题信息
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()), #
ModelStep的输入来源于依赖的JobStep的输出
```

```
outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=mo
del_name, model_type="TensorFlow")), # ModelStep的输出
    depend_steps=job_step # 依赖的作业类型节点对象
)
# job_step是wf.steps.JobStep的 实例对象，train_url是wf.steps.JobOutput的name字段值

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[job_step, model_registration],
    storages=[storage]
)
```

从训练作业中注册模型（模型输入来源 OBS 路径，训练完成的模型已存储到 OBS 路径）

```
import modelarts.workflow as wf
# 通过ModelStep来定义一个模型注册节点，输入来源于OBS中

# 定义OBS数据对象
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必
须是file或者directory

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中
划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="模型注册", # 标题信息
    inputs=wf.steps.ModelInput(name='model_input', data=obs), # ModelStep的输入在运行时配置；data字段
的值也可使用wf.data.OBSPath(obs_path="fake_obs_path")表示

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=mo
del_name, model_type="TensorFlow"))# ModelStep的输出
)

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

使用模板的方式注册模型

```
import modelarts.workflow as wf
# 通过ModelStep来定义一个模型注册节点，并通过预置模板进行注册

# 定义预置模板对象，Template对象中的字段可使用Placeholder表示
template = wf.steps.Template(
    template_id="fake_template_id",
    infer_format="fake_infer_format",
    template_inputs=[
        wf.steps.TemplateInputs(
            input_id="fake_input_id",
            input="fake_input_file"
        )
    ]
)

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中
划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="模型注册", # 标题信息
```

```
    outputs=wf.steps.ModelOutput(
        name='model_output',
        model_config=wf.steps.ModelConfig(
            model_name=model_name,
            model_type="Template",
            template=template
        )
    )# ModelStep的输出
)

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

从自定义镜像中注册模型

```
import modelarts.workflow as wf
# 通过ModelStep来定义一个模型注册节点，输入来源于自定义镜像地址

# 定义镜像数据
swr = wf.data.SWRImagePlaceholder(name="placeholder_name")

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="模型注册", # 标题信息
    inputs=wf.steps.ModelInput(name="input",data=swr), # ModelStep的输入在运行时配置；data字段的值也可使用wf.data.SWRImage(swr_path="fake_path")表示

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow"))# ModelStep的输出
)

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

使用自定义镜像+OBS 的方式注册模型

```
import modelarts.workflow as wf
# 通过ModelStep来定义一个模型注册节点，输入来源于自定义镜像地址

# 定义镜像数据
swr = wf.data.SWRImagePlaceholder(name="placeholder_name")

# 定义OBS模型数据
model_obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="模型注册", # 标题信息
    inputs=[
        wf.steps.ModelInput(name="input",data=swr), # ModelStep的输入在运行时配置；data字段的值也可使用wf.data.SWRImage(swr_path="fake_path")表示
        wf.steps.ModelInput(name="input",data=model_obs) # ModelStep的输入在运行时配置；data字段的值也可使用wf.data.OBSPath(obs_path="fake_obs_path")表示
    ],
```

```
    outputs=wf.steps.ModelOutput(
        name='model_output',
        model_config=wf.steps.ModelConfig(
            model_name=model_name,
            model_type="Custom",
            dynamic_load_mode="Single"
        )
    )# ModelStep的输出
)

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo orkflow",
    steps=[model_registration]
)
```

使用订阅模型+OBS 的方式注册模型

该方式本质上与自定义镜像+OBS的方式没有区别，只是自定义镜像变成从订阅模型中获取。

具体使用案例：

```
import modelarts.workflow as wf

# 定义订阅的模型对象
base_model = wf.data.GalleryModel(subscription_id="fake_subscription_id", version_num="fake_version") # 从gallery订阅的模型，一般由开发者自行创建发布

# 定义OBS模型数据
model_obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="模型注册", # 标题信息
    inputs=[
        wf.steps.ModelInput(name="input",data=base_model) # ModelStep的输入使用订阅的模型
        wf.steps.ModelInput(name="input",data=model_obs) # ModelStep的输入在运行时配置；data字段的值也可使用wf.data.OBSPath(obs_path="fake_obs_path")表示
    ],
    outputs=wf.steps.ModelOutput(
        name='model_output',
        model_config=wf.steps.ModelConfig(
            model_name=model_name,
            model_type="Custom",
            dynamic_load_mode="Single"
        )
    )# ModelStep的输出
)

workflow = wf.Workflow(
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

上述案例中，系统会自动获取订阅模型中的自定义镜像，然后结合输入的OBS模型路径，注册生成一个新的模型，其中model_obs可以替换成JobStep的动态输出。

说明

model_type支持的类型有："**TensorFlow**"、"**MXNet**"、"**Caffe**"、 "**Spark_MLlib**"、"**Scikit_Learn**"、"**XGBoost**"、 "**Image**"、"**PyTorch**"、"**Template**"、"**Custom**"。

在wf.steps.ModelConfig对象中，如果model_type字段未填写，则表示默认使用"**TensorFlow**"。

- 如果您构建的工作流对注册的模型类型没有修改的需求，则按照上述示例使用即可。
- 如果您构建的工作流需要多次运行可以修改模型类型，则可使用占位符参数的方式进行编写：

  model_type = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.ENUM, default="TensorFlow", enum_list=["TensorFlow", "MXNet", "Caffe", "Spark_MLlib", "Scikit_Learn", "XGBoost", "Image", "PyTorch", "Template", "Custom"], description="模型类型")

5.3.4.7 创建 Workflow 服务部署节点

功能介绍

通过对ModelArts服务管理能力的封装，实现Workflow新增服务和更新服务的能力。主要应用场景如下：

- 将模型部署为一个Web Service。
- 更新已有服务，支持灰度更新等能力。

属性总览

您可以使用**ServiceStep**来构建服务部署节点，**ServiceStep**结构如下

表 5-60 ServiceStep

属性	描述	是否必填	数据类型
name	服务部署节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复	是	str
inputs	服务部署节点的输入列表	否	ServiceInput或者ServiceInput的列表
outputs	服务部署节点的输出列表	是	ServiceOutput或者ServiceOutput的列表
title	title信息，主要用于前端的名称展示	否	str

属性	描述	是否必填	数据类型
description	服务部署节点的描述信息	否	str
policy	节点执行的policy	否	StepPolicy
depend_steps	依赖的节点列表	否	Step或者Step的列表

表 5-61 ServiceInput

属性	描述	是否必填	数据类型
name	服务部署节点的输入名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输入名称不能重复	是	str
data	服务部署节点的输入数据对象	是	模型列表或服务相关对象，当前仅支持ServiceInputPlaceholder，ServiceData，ServiceUpdatePlaceholder

表 5-62 ServiceOutput

属性	描述	是否必填	数据类型
name	服务部署节点的输出名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)。同一个Step的输出名称不能重复	是	str
service_config	服务部署相关配置信息	是	ServiceConfig

表4 ServiceConfig

属性	描述	是否必填	数据类型
infer_type	推理方式：取值可为real-time/batch/edge。默认为real-time。 ● real-time代表在线服务，将模型部署为一个Web Service。 ● batch为批量服务，批量服务可对批量数据进行推理，完成数据处理后自动停止。 ● edge表示边缘服务，通过华为云智能边缘平台，在边缘节点将模型部署为一个Web Service，需提前在IEF（智能边缘服务）创建好节点。	是	str
service_name	服务名称，支持1-64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线。 **说明** 该字段不填时默认为自动生成的服务名称。	否	str、Placeholder
description	服务备注，默认为空，不超过100个字符。	否	str
vpc_id	在线服务实例部署的虚拟私有云ID，默认为空，此时ModelArts会为每个用户分配一个专属的VPC，用户之间隔离。如需要在服务实例中访问名下VPC内的其他服务组件，则可配置此参数为对应VPC的ID。VPC一旦配置，不支持修改。当vpc_id与cluster_id一同配置时，只有专属资源池参数生效。	否	str
subnet_network_id	子网的网络ID，默认为空，当配置了vpc_id则此参数必填。需填写虚拟私有云控制台子网详情中显示的“网络ID”。通过子网可提供与其他网络隔离的、可以独享的网络资源。	否	str
security_group_id	安全组，默认为空，当配置了vpc_id则此参数必填。安全组起着虚拟防火墙的作用，为服务实例提供安全的网络访问控制策略。安全组须包含至少一条入方向规则，对协议为TCP、源地址为0.0.0.0/0、端口为8080的请求放行。	否	str

属性	描述	是否必填	数据类型
cluster_id	专属资源池ID，默认为空，不使用专属资源池。使用专属资源池部署服务时需确保集群状态正常；配置此参数后，则使用集群的网络配置，vpc_id参数不生效；与下方real-time config中的cluster_id同时配置时，优先使用real-time config中的cluster_id参数。	否	str
additional_properties	附加的相关配置信息。	否	dict
apps	服务部署支持APP认证。支持填入多个app name。	否	str、Placeholder、list
envs	环境变量	否	dict

示例：

```
example = ServiceConfig()
# 主要在服务部署节点的输出中使用
```

如果您没有特殊需求，可直接使用内置的默认值。

使用案例

主要包含三种场景的用例：

- 新增在线服务
- 更新在线服务
- 服务部署输出推理地址

新增在线服务

```
import modelarts.workflow as wf
# 通过ServiceStep来定义一个服务部署节点，输入指定的模型进行服务部署

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

service_step = wf.steps.ServiceStep(
    name="service_step", # 服务部署节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="新增服务", # 标题信息
    inputs=wf.steps.ServiceInput(name="si_service_ph",
data=wf.data.ServiceInputPlaceholder(name="si_placeholder1",
                                                    # 模型名称的限制/约束,在运行态只能选择该模型名称；一般与模型注册节点中的model_name使用同一个参数对象
                                                    model_name=model_name)),# ServiceStep的输入列表
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep的输出
)

workflow = wf.Workflow(
    name="service-step-demo",
```



```
    desc="this is a demo workflow",
    steps=[service_step]
)
```

更新在线服务

使用场景：使用新版本的模型对已有的服务进行更新，需要保证新版本的模型与已部署服务的模型名称一致。

```
import modelarts.workflow as wf
# 通过ServiceStep来定义一个服务部署节点，输入指定的模型对已部署的服务进行更新

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

# 定义服务对象
service = wf.data.ServiceUpdatePlaceholder(name="placeholder_name")

service_step  = wf.steps.ServiceStep(
    name="service_step", # 服务部署节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="服务更新", # 标题信息
    inputs=[wf.steps.ServiceInput(name="si2",
data=wf.data.ServiceInputPlaceholder(name="si_placeholder2",
                                                                    # 模型名称的限制/约束,在运行态只能选择该模型名称
                                                                    model_name=model_name)),
            wf.steps.ServiceInput(name="si_service_data", data=service) # 已部署的服务在运行时配置；data也可使用wf.data.ServiceData(service_id="fake_service")表示
    ], # ServiceStep的输入列表
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep的输出
)

workflow = wf.Workflow(
    name="service-step-demo",
    desc="this is a demo workflow",
    steps=[service_step]
)
```

服务部署输出推理地址

服务部署节点支持输出推理地址，通过**get_output_variable("access_address")**方法获取输出值，并在后续节点中使用。

- 针对部署在公共资源池的服务，可以通过**access_address**属性从输出中获取注册在公网的推理地址。
- 针对部署在专属资源池的服务，除了可以获取注册在公网的推理地址，还能通过**cluster_inner_access_address**属性从输出中获取内部使用的推理地址，并且该地址只能在其他推理服务中进行访问。

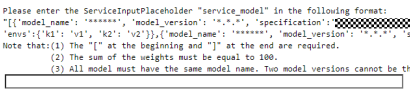
  ```
  import modelarts.workflow as wf

  # 定义模型名称参数
  sub_model_name = wf.Placeholder(name="si_placeholder1",
  placeholder_type=wf.PlaceholderType.STR)

  sub_service_step = wf.steps.ServiceStep(
      name="sub_service_step", # 服务部署节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
      title="子服务", # 标题信息
      inputs=wf.steps.ServiceInput(
          name="si_service_ph",
          data=wf.data.ServiceInputPlaceholder(name="si_placeholder1", model_name=sub_model_name)
      ),# ServiceStep的输入列表
      outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep的输出
  )
  ```



```
main_model_name = wf.Placeholder(name="si_placeholder2",
placeholder_type=wf.PlaceholderType.STR)

# 获取子服务输出的推理地址，并通过envs传递给主服务
main_service_config = wf.steps.ServiceConfig(
    infer_type="real-time",
    envs={"infer_address":
sub_service_step.outputs["service_output"].get_output_variable("access_address")} # 获取子服务输出的
推理地址，并通过envs传递到主服务中
)

main_service_step = wf.steps.ServiceStep(
    name="main_service_step", # 服务部署节点的名称，命名规范(只能包含英文字母、数字、下划线
（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不
能重复
    title="主服务", # 标题信息
    inputs=wf.steps.ServiceInput(
        name="si_service_ph",
        data=wf.data.ServiceInputPlaceholder(name="si_placeholder2",
model_name=main_model_name)
    ),# ServiceStep的输入列表
    outputs=wf.steps.ServiceOutput(name="service_output", service_config=main_service_config), #
ServiceStep的输出
    depend_steps=sub_service_step
)

workflow = wf.Workflow(
    name="service-step-demo",
    desc="this is a demo workflow",
    steps=[sub_service_step, main_service_step]
)
```

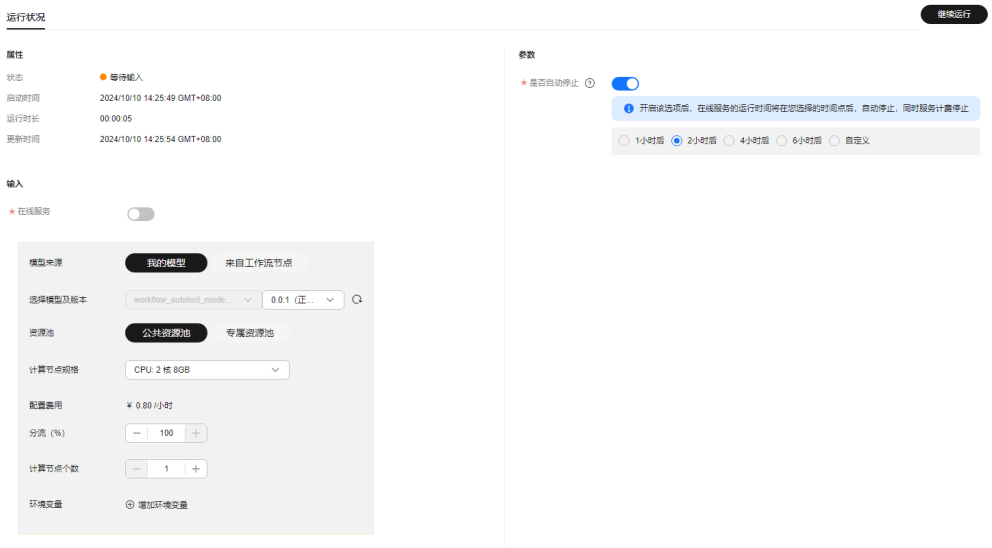
同步推理服务部署相关信息配置操作

在开发态中（一般指Notebook），节点启动运行后，用户根据日志打印的输入格式进行配置，如下所示：



1. 在ModelArts管理控制台，左侧菜单栏选择“开发空间>Workflow”进入Workflow页面。
2. 在服务部署节点启动之后会等待用户设置相关配置信息，配置完成后直接单击“继续运行”即可。

   如果想要跳过手动配置的步骤，直接自动运行部署节点，则按照需要在代码中提前配置ServiceInputConfig或ServiceConfig参数，如：service_config=wf.steps.ServiceConfig(cluster_id="XX")。当不缺少参数时，服务部署节点将会自动启动。



异步推理服务部署相关信息配置操作

1. 在ModelArts管理控制台，左侧菜单栏选择“Workflow”进入Workflow页面。
2. 在服务部署节点启动之后会等待用户设置相关配置信息，选择模型及版本为异步推理模型，设置**服务启动参数**，配置完成后直接单击**继续运行**即可。

说明

其中**服务启动参数**与您选择的异步推理模型相关，选择了需要的模型及版本后，系统会自动匹配响应的**服务启动参数**。

5.3.5 构建 Workflow 多分支运行场景

5.3.5.1 Workflow 多分支运行介绍

当前支持两种方式实现多分支的能力，条件节点只支持双分支的选择执行，局限性较大，推荐使用**配置节点参数控制分支执行**的方式，可以在不添加新节点的情况下完全覆盖ConditionStep的能力，使用上更灵活。

构建条件节点控制分支执行主要用于执行流程的条件分支选择，可以简单的进行数值比较来控制执行流程，也可以根据节点输出的metric相关信息决定后续的执行流程。

配置节点参数控制分支执行与ConditionStep的使用场景类似，但功能更加强大。主要用于存在多分支选择执行的复杂场景，在每次启动执行后需要根据相关配置信息决定哪些分支需要执行，哪些分支需要跳过，达到分支部分执行的目的。

5.3.5.2 构建条件节点控制分支执行

功能介绍

主要用于执行流程的条件分支选择，可以简单的进行数值比较来控制执行流程，也可以根据节点输出的metric相关信息决定后续的执行流程。主要应用场景如下：

可以用于需要根据不同的输入值来决定后续执行流程的场景。例如：需要根据训练节点输出的精度信息来决定是重新训练还是进行模型的注册操作时可以使用该节点来实现流程的控制。

属性总览

您可以使用**ConditionStep**来构建条件节点，**ConditionStep**结构如下：

表 5-63 ConditionStep

属性	描述	是否必填	数据类型
name	条件节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复	是	str
conditions	条件列表，列表中的多个Condition执行“逻辑与”操作	是	Condition或者Condition的列表
if_then_steps	条件表达式计算结果为True时，执行的step列表	否	str或者str列表
else_then_steps	条件表达式计算结果为False时，执行的step列表	否	str或者str列表
title	title信息，主要用于前端节点的名称展示	否	str
description	条件节点的描述信息	否	str
depend_steps	依赖的节点列表	否	Step或者Step的列表

表 5-64 Condition

属性	描述	是否必填	数据类型
condition_type	条件类型，支持"=="、">"、">="、"in"、"<"、"<="、"!="、"or"操作符	是	ConditionTypeEnum
left	条件表达式的左值	是	int、float、str、bool、Placeholder、Sequence、Condition、MetricInfo

属性	描述	是否必填	数据类型
right	条件表达式的右值	是	int、float、str、bool、Placeholder、Sequence、Condition、MetricInfo

表 5-65 MetricInfo

属性	描述	是否必填	数据类型
input_data	metric文件的存储对象，当前仅支持JobStep节点的输出	是	JobStep的输出
json_key	需要获取的metric信息对应的key值	是	str

结构内容详解：

- **Condition对象（由三部分组成：条件类型，左值以及右值）**
  - 条件类型使用ConditionTypeEnum来获取，支持"=="、">"、">="、"in"、"<"、"<="、"!="、"or"操作符，具体映射关系如下表所示。

枚举类型	操作符
ConditionTypeEnum.EQ	==
ConditionTypeEnum.GT	>
ConditionTypeEnum.GTE	>=
ConditionTypeEnum.IN	in
ConditionTypeEnum.LT	<
ConditionTypeEnum.LTE	<=
ConditionTypeEnum.NOT	!=
ConditionTypeEnum.OR	or

  - 左右值支持的类型有：int、float、str、bool、Placeholder、Sequence、Condition、MetricInfo。
  - 一个ConditionStep支持多个Condition对象，使用list表示，多个Condition之间进行&&操作。
- **if_then_steps和else_then_steps**。
  - if_then_steps表示的是当Condition比较的结果为true时允许执行的节点列表，存储的是节点名称；此时else_then_steps中的step跳过不执行。

- else_then_step表示的是当Condition比较的结果为false时允许执行的节点列表，存储的是节点名称；此时if_then_steps中的step跳过不执行。

使用案例

根据需求参考简单示例或进阶示例。

简单示例

- 通过参数配置实现

```
import modelarts.workflow as wf

left_value = wf.Placeholder(name="left_value", placeholder_type=wf.PlaceholderType.BOOL, default=True)

# 条件对象
condition = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=left_value, right=True) # 条件对象，包含类型以及左右值

# 条件节点
condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # 条件节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    conditions=condition, # 条件对象,允许多个条件，条件之间的关系为&&
    if_then_steps="job_step_1", # 当condition结果为true时，名称为job_step_1的节点允许执行，名称为job_step_2的节点跳过不执行
    else_then_steps="job_step_2" # 当condition结果为false时，名称为job_step_2的节点允许执行，名称为job_step_1的节点跳过不执行
)

# 该节点仅作为示例使用，其他字段需自行补充
job_step_1 = wf.steps.JobStep(
    name="job_step_1",
    depend_steps=condition_step
)

# 该节点仅作为示例使用，其他字段需自行补充
model_step_1 = wf.steps.ModelStep(
    name="model_step_1",
    depend_steps=job_step_1
)

# 该节点仅作为示例使用，其他字段需自行补充
job_step_2 = wf.steps.JobStep(
    name="job_step_2",
    depend_steps=condition_step
)

# 该节点仅作为示例使用，其他字段需自行补充
model_step_2 = wf.steps.ModelStep(
    name="model_step_2",
    depend_steps=job_step_2
)

workflow = wf.Workflow(
    name="condition-demo",
    desc="this is a demo workflow",
    steps=[condition_step, job_step_1, job_step_2, model_step_1, model_step_2]
)
```

说明

场景说明：job_step_1和job_step_2表示两个训练节点，并且均直接依赖于condition_step。condition_step通过参数配置决定后继节点的执行行为。

执行情况分析：

  - 参数left_value默认值为True，则condition逻辑表达式计算结果为True：job_step_1执行，job_step_2跳过，并且以job_step_2为唯一根节点的分支所包含的所有节点也将跳过，即model_step_2会跳过，因此最终执行的节点有condition_step、job_step_1、model_step_1。
  - 如果设置left_value的值为Fasle，则condition逻辑表达式计算结果为False：job_step_2执行，job_step_1跳过，并且以job_step_1为唯一根节点的分支所包含的所有节点也将跳过，即model_step_1会跳过，因此最终执行的节点有condition_step、job_step_2、model_step_2。

- 通过获取JobStep输出的相关metric指标信息实现

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.Storage(name="storage_name", title="title_info", with_execution_id=True,
create_dir=True, description="description_info")  # name字段必填，title, description可选填

# 定义输入的OBS对象
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# 通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job",  # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练",  # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # 算法订阅ID
        item_version_id="item_version_id",  # 算法订阅版本ID，也可直接填写版本号
        parameters=[]

    ),  # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
        wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # 指定metric的输出路径，相关指标信息由作业脚本代码根据指定的数据格式自行输出（示例中需要将metric信息输出到训练输出目录下的metrics.json文件中）
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")
        )
    ) # 训练资源规格信息
)

# 定义条件对象
condition_lt = wf.steps.Condition(
    condition_type=wf.steps.ConditionTypeEnum.LT,
    left=wf.steps.MetricInfo(job_step.outputs["metrics"].as_input(), "accuracy"),
    right=0.5
)

condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # 条件节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    conditions=condition_lt, # 条件对象,允许多个条件，条件之间的关系为&&
    if_then_steps="training_job_retrain", # 当condition结果为true时，名称为training_job_retrain的节点允许执行，名称为model_registration的节点跳过不执行
    else_then_steps="model_registration", # 当condition结果为false时，名称为model_registration的节点允许执行，名称为training_job_retrain的节点跳过不执行
    depend_steps=job_step
)
```



```
# 通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step_retrain = wf.steps.JobStep(
    name="training_job_retrain", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类重新训练训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
        parameters=[]

    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_retrain"))),
        wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path_retrain/metrics.json", create_dir=False))) # 指定metric的输出路径，相关指标信息由作业脚本代码根据指定的数据格式自行输出（示例中需要将metric信息输出到训练输出目录下的metrics.json文件中）
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor_retrain",
placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格")
        )
    ), # 训练资源规格信息
    depend_steps=condition_step
)

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="模型注册", # 标题信息
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()), # job_step的输出作为输入
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")), # ModelStep的输出
    depend_steps=condition_step,
)

workflow = wf.Workflow(
    name="condition-demo",
    desc="this is a demo workflow",
    steps=[job_step, condition_step, job_step_retrain, model_step],
    storages=storage
)
```

案例中ConditionStep节点通过获取job_step输出的accuracy指标信息与预置的值进行比较，决定重新训练还是模型注册。当job_step输出的accuracy指标数据小于阈值0.5时，condition_lt的计算结果为True，此时job_step_retrain运行，model_step跳过；反之job_step_retrain跳过，model_step执行。

说明

job_step输出的metric文件格式要求可参考**创建Workflow训练作业节点**部分，并且在Condition中只支持使用type为float类型的指标数据作为输入。

此案例中metrics.json的内容示例如下：

```
[
    {
        "key": "loss",
```

```
        "title": "loss",
        "type": "float",
        "data": {
            "value": 1.2
        }
    },
    {
        "key": "accuracy",
        "title": "accuracy",
        "type": "float",
        "data": {
            "value": 0.8
        }
    }
]
```

进阶示例

```
import modelarts.workflow as wf

left_value = wf.Placeholder(name="left_value", placeholder_type=wf.PlaceholderType.BOOL, default=True)
condition1 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=left_value, right=True)

internal_condition_1 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.GT, left=10, right=9)
internal_condition_2 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.LT, left=10, right=9)

# condition2的结果为internal_condition_1 || internal_condition_2
condition2 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.OR, left=internal_condition_1,
right=internal_condition_2)

condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # 条件节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线
（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    conditions=[condition1, condition2], # 条件对象,允许多个条件，条件之间的关系为&&
    if_then_steps=["job_step_1"], # 当condition结果为true时，名称为job_step_1的节点允许执行，名称为
job_step_2的节点跳过不执行
    else_then_steps=["job_step_2"] # 当condition结果为false时，名称为job_step_2的节点允许执行，名称为
job_step_1的节点跳过不执行
)

# 该节点仅作为示例使用，其他字段需自行补充
job_step_1 = wf.steps.JobStep(
    name="job_step_1",
    depend_steps=condition_step
)

# 该节点仅作为示例使用，其他字段需自行补充
job_step_2 = wf.steps.JobStep(
    name="job_step_2",
    depend_steps=condition_step
)

workflow = wf.Workflow(
    name="condition-demo",
    desc="this is a demo workflow",
    steps=[condition_step, job_step_1, job_step_2],
)
```

ConditionStep支持多条件节点的嵌套使用，用户可以基于不同的场景灵活设计。

说明

条件节点只支持双分支的选择执行，局限性较大，推荐您使用新的分支功能，可以在不添加新节点的情况下完全覆盖ConditionStep的能力，详情请参见**配置节点参数控制分支执行**章节。

5.3.5.3 配置节点参数控制分支执行

功能介绍

支持单节点通过参数配置或者获取训练输出的metric指标信息来决定执行是否跳过，同时可以基于此能力完成对执行流程的控制。

应用场景

主要用于存在多分支选择执行的复杂场景，在每次启动执行后需要根据相关配置信息决定哪些分支需要执行，哪些分支需要跳过，达到分支部分执行的目的，与ConditionStep的使用场景类似，但功能更加强大。当前该能力适用于数据集创建节点、数据集标注节点、数据集导入节点、数据集版本发布节点、作业类型节点、模型注册节点以及服务部署节点。

控制单节点的执行

- 通过参数配置实现

```
from modelarts import workflow as wf

condition_equal = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
                                description="description_info")  # name字段必填，title, description可选填

# 定义输入的OBS对象
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# 通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job",  # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练",  # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # 算法订阅ID
        item_version_id="item_version_id",  # 算法订阅版本ID，也可直接填写版本号
        parameters=[]

    ),  # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值

    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    # JobStep的输入在运行时配置；data字段也可使用data=wf.data.OBSPath(obs_path="fake_obs_path")表示
    outputs=wf.steps.JobOutput(name="train_url",
                               obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
    # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    ), # 训练资源规格信息
    policy=wf.steps.StepPolicy(
        skip_conditions=[condition_equal] # 通过skip_conditions中的计算结果决定job_step是否跳过
    )
)

workflow = wf.Workflow(
    name="new-condition-demo",
    desc="this is a demo workflow",
```

```
    steps=[job_step],
    storages=storage
)
```

案例中job_step配置了相关的跳过策略，并且通过一个bool类型的参数进行控制。当name为is_skip的Placeholder参数配置为True时，condition_equal的计算结果为True，此时job_step会被置为跳过，反之job_step正常执行，其中Condition对象详情可参考**构建条件节点控制分支执行**。

- 通过获取JobStep输出的相关metric指标信息实现

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.Storage(name="storage_name", title="title_info", with_execution_id=True,
create_dir=True, description="description_info")  # name字段必填，title, description可选填

# 定义输入的OBS对象
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# 通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job",  # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练",  # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # 算法订阅ID
        item_version_id="item_version_id",  # 算法订阅版本ID，也可直接填写版本号
        parameters=[]

    ),  # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
        wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # 指定metric的输出路径，相关指标信息由作业脚本代码根据指定的数据格式自行输出（示例中需要将metric信息输出到训练输出目录下的metrics.json文件中）
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")
        )
    ) # 训练资源规格信息
)

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

# 定义条件对象
condition_lt = wf.steps.Condition(
    condition_type=wf.steps.ConditionTypeEnum.LT,
    left=wf.steps.MetricInfo(job_step.outputs["metrics"].as_input(), "accuracy"),
    right=0.5
)

model_step = wf.steps.ModelStep(
    name="model_registration",  # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="模型注册",  # 标题信息
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()),  # job_step的输出作为输入
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),  # ModelStep的输出
```

```
    depend_steps=[job_step], # 依赖的作业类型节点对象
    policy=wf.steps.StepPolicy(skip_conditions=condition_lt) # 通过skip_conditions中的计算结果决定
model_step是否跳过
)

workflow = wf.Workflow(
    name="new-condition-demo",
    desc="this is a demo workflow",
    steps=[job_step, model_step],
    storages=storage
)
```

案例中model_step配置了相关的跳过策略，并且通过获取job_step输出的accuracy指标信息与预置的值进行比较，决定是否需要进行模型注册。当job_step输出的accuracy指标数据小于阈值0.5时，condition_lt的计算结果为True，此时model_step会被置为跳过，反之model_step正常执行。

说明

job_step输出的metric文件格式要求可参考**创建Workflow训练作业节点**部分，并且在Condition中只支持使用type为float类型的指标数据作为输入。

此案例中metrics.json的内容示例如下：

```
[
    {
        "key": "loss", // 指标数据名称，不支持特殊字符，长度限制为64字符
        "title": "loss", // 指标数据标题，长度限制为64字符
        "type": "float", // 指标数据类型，支持以下类型：浮点：float、折线图：line chart、柱状图：
histogram、矩阵：table、一维表格：one-dimensional-table
        "data": {
            "value": 1.2 // 指标数据值，不同类型的使用示例可以参考创建Workflow训练作业节点
        }
    },
    {
        "key": "accuracy",
        "title": "accuracy",
        "type": "float",
        "data": {
            "value": 0.8
        }
    }
]
```

控制多分支的部分执行

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.Storage(name="storage_name", title="title_info", with_execution_id=True, create_dir=True,
description="description_info")  # name字段必填，title, description可选填

# 定义输入的OBS对象
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

condition_equal_a = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="job_step_a_is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)

# 通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step_a = wf.steps.JobStep(
    name="training_job_a", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线
（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
        parameters=[]

    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在
```

```
parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_a")))],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    ), # 训练资源规格信息
    policy=wf.steps.StepPolicy(skip_conditions=condition_equal_a)
)

condition_equal_b = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="job_step_b_is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)

# 通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step_b = wf.steps.JobStep(
    name="training_job_b", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线
（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
        parameters=[]

    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_b")))],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    ), # 训练资源规格信息
    policy=wf.steps.StepPolicy(skip_conditions=condition_equal_b)
)

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中
划线（-），并且只能以英文字母开头，长度限制为64字符），一个Workflow里的两个step名称不能重复
    title="模型注册", # 标题信息
    inputs=wf.steps.ModelInput(name='model_input',
data=wf.data.DataConsumptionSelector(data_list=[job_step_a.outputs["train_url"].as_input(),
job_step_b.outputs["train_url"].as_input()])), # 选择job_step_a或者job_step_b的输出作为输入
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")), # ModelStep
的输出
    depend_steps=[job_step_a, job_step_b], # 依赖的作业类型节点对象
)

workflow = wf.Workflow(
    name="new-condition-demo",
    desc="this is a demo workflow",
    steps=[job_step_a, job_step_b, model_step],
    storages=storage
)
```

案例中job_step_a和job_step_b均配置了跳过策略，并且都使用参数进行控制。当参数值配置不同时，model_step的执行可以分为以下几种情况（model_step没有配置跳过策略，因此会遵循默认规则）：

job_step_a_is_skip参数值	job_step_b_is_skip参数值	model_step是否执行
True	True	跳过
	False	执行
False	True	执行
	False	执行

⚠ 注意

默认规则：当某个节点依赖的所有节点状态均为跳过时，该节点自动跳过，否则正常执行，此判断逻辑可扩展至任意节点。

在上述案例的基础上，如果需要打破默认规则，在job_step_a以及job_step_b跳过时，model_step也允许执行，则只需要在model_step中也配置跳过策略即可（跳过策略的优先级高于默认规则）。

5.3.5.4 配置多分支节点数据

功能介绍

仅用于存在多分支执行的场景，在编写构建工作流节点时，节点的数据输入来源暂不确定，可能是多个依赖节点中任意一个节点的输出。只有当依赖节点全部执行完成后，才会根据实际执行情况自动获取有效输出作为输入。

使用案例

```
from modelarts import workflow as wf

condition_equal = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="is_true", placeholder_type=wf.PlaceholderType.BOOL), right=True)
condition_step = wf.steps.ConditionStep(
    name="condition_step",
    conditions=[condition_equal],
    if_then_steps=["training_job_1"],
    else_then_steps=["training_job_2"],
)

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
                                description="description_info")  # name字段必填，title, description可选填

# 定义输入的OBS对象
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# 通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step_1 = wf.steps.JobStep(
    name="training_job_1",  # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练",  # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # 算法订阅ID
        item_version_id="item_version_id",  # 算法订阅版本ID，也可直接填写版本号
        parameters=[]
```

```
    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值

    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    # JobStep的输入在运行时配置；data字段也可使用data=wf.data.OBSPath(obs_path="fake_obs_path")表示
    outputs=wf.steps.JobOutput(name="train_url",
                               obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
    # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    ), # 训练资源规格信息
    depend_steps=[condition_step]
)

# 通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step_2 = wf.steps.JobStep(
    name="training_job_2", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线
（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
        parameters=[]

    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值

    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    # JobStep的输入在运行时配置；data字段也可使用data=wf.data.OBSPath(obs_path="fake_obs_path")表示
    outputs=wf.steps.JobOutput(name="train_url",
                               obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
    # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    ), # 训练资源规格信息
    depend_steps=[condition_step]
)

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中
划线（-），并且只能以英文字母开头，长度限制为64字符)，一个Workflow里的两个step名称不能重复
    title="模型注册", # 标题信息
    inputs=wf.steps.ModelInput(name='model_input',
data=wf.data.DataConsumptionSelector(data_list=[job_step_1.outputs["train_url"].as_input(),
job_step_2.outputs["train_url"].as_input()])), # 选择job_step_1或者job_step_2的输出作为输入
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")), # ModelStep
的输出
    depend_steps=[job_step_1, job_step_2] # 依赖的作业类型节点对象
)# job_step是wf.steps.JobStep的 实例对象，train_url是wf.steps.JobOutput的name字段值


workflow = wf.Workflow(name="data-select-demo",
                       desc="this is a test workflow",
                       steps=[condition_step, job_step_1, job_step_2, model_step],
                       storages=storage
                       )
```


说明

案例中的Workflow存在两个并行分支，并且同时只有一条分支会执行，由condition_step的相关配置决定。model_step的输入来源为job_step_1或者job_step_2的输出，当job_step_1节点所在分支执行，job_step_2节点所在分支跳过时，model_step节点执行时自动获取job_step_1的输出作为输入，反之自动获取job_step_2的输出作为输入。

5.3.6 编排 Workflow

Workflow的编排主要在于每个节点的定义，您可以参考**创建Workflow节点**章节，按照自己的场景需求选择相应的代码示例模板进行修改。编排过程主要分为以下几个步骤。

1. 梳理场景，了解预置Step的功能，确定最终的DAG结构。
2. 单节点功能，如训练、推理等在ModelArts相应服务中调试通过。
3. 根据节点功能选择相应的代码模板，进行内容的补充。
4. 根据DAG结构编排节点，完成Workflow的编写。

导入 Workflow Data 包

在编写Workflow过程中，相关对象都通过Workflow包进行导入，梳理如下：

```
from modelarts import workflow as wf
```

Data包相关内容导入：

```
wf.data.DatasetTypeEnum
wf.data.Dataset
wf.data.DatasetVersionConfig
wf.data.DatasetPlaceholder
wf.data.ServiceInputPlaceholder
wf.data.ServiceData
wf.data.ServiceUpdatePlaceholder
wf.data.DataTypeEnum
wf.data.ModelData
wf.data.GalleryModel
wf.data.OBSPath
wf.data.OBSOutputConfig
wf.data.OBSPlaceholder
wf.data.SWRImage
wf.data.SWRImagePlaceholder
wf.data.Storage
wf.data.InputStorage
wf.data.OutputStorage
wf.data.LabelTask
wf.data.LabelTaskPlaceholder
wf.data.LabelTaskConfig
wf.data.LabelTaskTypeEnum
wf.data.MetricsConfig
wf.data.TripartiteServiceConfig
wf.data.DataConsumptionSelector
```

policy包相关内容导入：

```
wf.policy.Policy
wf.policy.Scene
```

steps包相关内容导入：

```
wf.steps.MetricInfo
wf.steps.Condition
wf.steps.ConditionTypeEnum
wf.steps.ConditionStep
```

```
wf.steps.LabelingStep
wf.steps.LabelingInput
wf.steps.LabelingOutput
wf.steps.LabelTaskProperties
wf.steps.ImportDataInfo
wf.steps.DataOriginTypeEnum
wf.steps.DatasetImportStep
wf.steps.DatasetImportInput
wf.steps.DatasetImportOutput
wf.steps.AnnotationFormatConfig
wf.steps.AnnotationFormatParameters
wf.steps.AnnotationFormatEnum
wf.steps.Label
wf.steps.ImportTypeEnum
wf.steps.LabelFormat
wf.steps.LabelTypeEnum
wf.steps.ReleaseDatasetStep
wf.steps.ReleaseDatasetInput
wf.steps.ReleaseDatasetOutput
wf.steps.CreateDatasetStep
wf.steps.CreateDatasetInput
wf.steps.CreateDatasetOutput
wf.steps.DatasetProperties
wf.steps.SchemaField
wf.steps.ImportConfig
wf.steps.JobStep
wf.steps.JobMetadata
wf.steps.JobSpec
wf.steps.JobResource
wf.steps.JobTypeEnum
wf.steps.JobEngine
wf.steps.JobInput
wf.steps.JobOutput
wf.steps.LogExportPath
wf.steps.MrsJobStep
wf.steps.MrsJobInput
wf.steps.MrsJobOutput
wf.steps.MrsJobAlgorithm
wf.steps.ModelStep
wf.steps.ModelInput
wf.steps.ModelOutput
wf.steps.ModelConfig
wf.steps.Template
wf.steps.TemplateInputs
wf.steps.ServiceStep
wf.steps.ServiceInput
wf.steps.ServiceOutput
wf.steps.ServiceConfig
wf.steps.StepPolicy
```

Workflow包相关内容导入：

```
wf.workflow
wf.Subgraph
wf.Placeholder
wf.PlaceholderType
wf.AlgorithmParameters
wf.BaseAlgorithm
wf.Algorithm
wf.AIGalleryAlgorithm
wf.resource
wf.SystemEnv
wf.add_whitelist_users
wf.delete_whitelist_users
```

5.3.7 发布 Workflow

5.3.7.1 发布 Workflow 到 ModelArts

发布Workflow到ModelArts有两种方式，这两种方式的区别在**发布Workflow至运行态**后，需要在Workflow页面配置输入输出等参数；而**发布Workflow至运行态并运行**通过对代码进行改造，用户直接在SDK侧发布并运行工作流，节省了前往控制台进行配置运行的操作。

发布 Workflow 至运行态

工作流编写完成后，可以进行固化保存，调用Workflow对象的release()方法发布到运行态进行配置执行（在管理控制台Workflow页面配置）。

执行如下命令：

```
workflow.release()
```

上述命令执行完成后，如果日志打印显示发布成功，则可前往ModelArts的Workflow页面中查看新发布的工作流，进入Workflow详情，单击“配置”进行参数配置。

基于release()方法，提供了release_and_run()方法，支持用户在开发态发布并运行工作流，节省了前往console配置执行的操作。

⚠ 注意

使用该方法时需要注意以下几个事项：

- Workflow中所有出现占位符相关的配置对象时，均需要设置默认值，或者直接使用固定的数据对象
- 方法的执行依赖于Workflow对象的名称：当该名称的工作流不存在时，则创建新工作流并创建新执行；当该名称的工作流已存在时，则更新存在的工作流并基于新的工作流结构创建新的执行

  ```
  workflow.release_and_run()
  ```

发布 Workflow 至运行态并运行

该方式支持用户直接在SDK侧发布并运行工作流，节省了前往控制台进行配置运行的操作，对Workflow代码改造如下。

```
from modelarts import workflow as wf

# 定义统一存储对象管理输出目录
output_storage = wf.data.OutputStorage(name="output_storage", description="输出目录统一配置",
default="**")

# 数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_data", default=wf.data.Dataset(dataset_name="**",
version_name="**"))

# 创建训练作业
job_step = wf.steps.JobStep(
    name="training_job",
    title="图像分类训练",
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="**", # 图像分类算法的订阅ID，自行前往算法管理页面进行查看，可选参数，此处以订阅
算法举例
        item_version_id="10.0.0", # 订阅算法的版本号，可选参数，此处以订阅算法举例
        parameters=[
            wf.AlgorithmParameters(name="task_type", value="image_classification_v2"),
            wf.AlgorithmParameters(name="model_name", value="resnet_v1_50"),
```

```
            wf.AlgorithmParameters(name="do_train", value="True"),
            wf.AlgorithmParameters(name="do_eval_along_train", value="True"),
            wf.AlgorithmParameters(name="variable_update", value="horovod"),
            wf.AlgorithmParameters(name="learning_rate_strategy",
value=wf.Placeholder(name="learning_rate_strategy", placeholder_type=wf.PlaceholderType.STR,
default="0.002", description="训练的学习率策略(10:0.001,20:0.0001代表0-10个epoch学习率0.001，
10-20epoch学习率0.0001),如果不指定epoch, 会根据验证精度情况自动调整学习率，并当精度没有明显提升时，
训练停止")),
            wf.AlgorithmParameters(name="batch_size", value=wf.Placeholder(name="batch_size",
placeholder_type=wf.PlaceholderType.INT, default=64, description="每步训练的图片数量（单卡）")),
            wf.AlgorithmParameters(name="eval_batch_size", value=wf.Placeholder(name="eval_batch_size",
placeholder_type=wf.PlaceholderType.INT, default=64, description="每步验证的图片数量（单卡）")),
            wf.AlgorithmParameters(name="evaluate_every_n_epochs",
value=wf.Placeholder(name="evaluate_every_n_epochs", placeholder_type=wf.PlaceholderType.FLOAT,
default=1.0, description="每训练n个epoch做一次验证")),
            wf.AlgorithmParameters(name="save_model_secs", value=wf.Placeholder(name="save_model_secs",
placeholder_type=wf.PlaceholderType.INT, default=60, description="保存模型的频率（单位：s）")),
            wf.AlgorithmParameters(name="save_summary_steps",
value=wf.Placeholder(name="save_summary_steps", placeholder_type=wf.PlaceholderType.INT, default=10,
description="保存summary的频率（单位：步）")),
            wf.AlgorithmParameters(name="log_every_n_steps",
value=wf.Placeholder(name="log_every_n_steps", placeholder_type=wf.PlaceholderType.INT, default=10,
description="打印日志的频率（单位：步）")),
            wf.AlgorithmParameters(name="do_data_cleaning",
value=wf.Placeholder(name="do_data_cleaning", placeholder_type=wf.PlaceholderType.STR, default="True",
description="是否进行数据清洗, 数据格式异常会导致训练失败，建议开启，保证训练稳定性。数据量过大时，数
据清洗可能耗时较久，可自行线下清洗（支持BMP.JPEG,PNG格式, RGB三通道）。建议用JPEG格式数据")),
            wf.AlgorithmParameters(name="use_fp16", value=wf.Placeholder(name="use_fp16",
placeholder_type=wf.PlaceholderType.STR, default="True", description="是否使用混合精度, 混合精度可以加速
训练，但是可能会造成一点精度损失，如果对精度无极严格的要求，建议开启")),
            wf.AlgorithmParameters(name="xla_compile", value=wf.Placeholder(name="xla_compile",
placeholder_type=wf.PlaceholderType.STR, default="True", description="是否开启xla编译，加速训练，默认启
用")),
            wf.AlgorithmParameters(name="data_format", value=wf.Placeholder(name="data_format",
placeholder_type=wf.PlaceholderType.ENUM, default="NCHW", enum_list=["NCHW", "NHWC"],
description="输入数据类型，NHWC表示channel在最后，NCHW表channel在最前，默认值NCHW（速度有提
升）")),
            wf.AlgorithmParameters(name="best_model", value=wf.Placeholder(name="best_model",
placeholder_type=wf.PlaceholderType.STR, default="True", description="是否在训练过程中保存并使用精度最
高的模型，而不是最新的模型。默认值True,保存最优模型。在一定误差范围内，最优模型会保存最新的高精度模
型")),
            wf.AlgorithmParameters(name="jpeg_preprocess", value=wf.Placeholder(name="jpeg_preprocess",
placeholder_type=wf.PlaceholderType.STR, default="True", description="是否使用jpeg预处理加速算子(仅支持
jpeg格式数据)，可加速数据读取，提升性能，默认启用。如果数据格式不是jpeg格式，开启数据清洗功能即可使
用")
        ]
    ),
    inputs=[wf.steps.JobInput(name="data_url", data=dataset)],
    outputs=[wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/train_output/")))],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(
                name="training_flavor",
                placeholder_type=wf.PlaceholderType.JSON,
                description="训练资源规格",
                default={"flavor_id": "**"}
            )
        )
    )
)

# 构建工作流对象
workflow = wf.Workflow(
    name="image-classification-ResNeSt",
    desc="this is a image classification workflow",
    steps=[job_step],
    storages=[output_storage]
)
```

1. 用户需要完成上述代码中**部分的配置，主要涉及以下三项。
   - 统一存储：output_storage对象的default值，需填写一个已存在的OBS路径，路径格式为：/OBS桶名称/文件夹路径/。
   - 数据集对象：填写相应的**数据集名称**以及**版本号**。
   - 训练资源规格：配置计算资源。由于举例的算法只能跑GPU，此处必须配置GPU类型的资源，可使用免费规格（modelarts.p3.large.public.free）。
2. 配置项修改完成后执行如下代码。
   ```
   workflow.release_and_run()
   ```
3. 执行完成后可前往ModelArts管理控制台，在总览页中选择Workflow，查看工作流的运行情况。

5.3.7.2 发布 Workflow 到 AI Gallery

Workflow支持发布到AI Gallery，分享给其他用户使用，执行如下代码即可完成发布。

```
workflow.release_to_gallery()
```

发布完成后可前往gallery查看相应的资产信息，资产权限默认为private，可在资产的console页面自行修改。

1. 进入AI Gallery。
2. 单击“我的Gallery>我的资产>Workflow”，进入我的Workflow页面。
3. 在“我的发布”页签中查看发布到AI Gallery的工作流。
4. 您可以单击工作流名称，查看发布的工作流详情。

其中release_to_gallery()方法包含以下入参：

参数名称	描述	是否必填	参数类型
content_id	Workflow资产ID	否	str
version	Workflow资产的版本号，格式为x.x.x	否	str
desc	Workflow资产版本的描述信息	否	str
title	Workflow资产名称，该参数未填写时默认使用Workflow的名称作为资产名称	否	str
visibility	Workflow资产可见性，支持"public"-公开、"group"-白名单、"private"-私有，仅自己可见三种，默认为"private"。	否	str
group_users	白名单列表，仅支持填写domain_id，当visibility为"group"时才需要填写该字段	否	list[str]

根据方法的入参不同，主要可分为以下两种使用场景：

- Workflow.release_to_gallery(title="资产名称")发布Workflow新资产，版本号为"1.0.0"；如果Workflow包含非gallery的算法，则自动将依赖算法发布至gallery，版本号为"1.0.0"。
- Workflow.release_to_gallery(content_id="**", title="资产名称")基于指定的Workflow资产，发布新的版本，版本号自动增加；如果Workflow包含gallery的算法，则自动将依赖的算法资产发布新版本，版本号也自动增加。

Workflow资产白名单设置：

在资产第一次发布时，可以通过release_to_gallery方法的visibility+group_users字段进行设置，后续需要对指定资产进行用户白名单添加或删除操作时，可执行如下命令：

```
from modelarts import workflow as wf

# 添加指定的白名单用户列表
wf.add_whitelist_users(content_id="**", version_num="*.*.*", user_groups=["**", "**"])

# 删除指定的白名单用户列表
wf.delete_whitelist_users(content_id="**", version_num="*.*.*", user_groups=["**", "**"])
```

说明

在给Workflow资产添加或删除指定白名单用户列表时，会自动查询该版本依赖的算法资产信息，同步对算法资产进行相应的白名单设置。

5.3.8 Workflow 高阶能力

5.3.8.1 在 Workflow 中使用大数据能力（DLI/MRS）

功能介绍

该节点通过调用MRS服务，提供大数据集群计算能力。主要用于数据批量处理、模型训练等场景。

应用场景

需要使用MRS Spark组件进行大量数据的计算时，可以根据已有数据使用该节点进行训练计算。

使用案例

在华为云MRS服务下查看自己账号下可用的MRS集群，如果没有，则需要创建，当前需要集群有Spark组件，安装时，注意勾选上。

您可以使用MrsStep来创建作业类型节点。定义MrsStep示例如下。

- **指定启动脚本与集群**

```
from modelarts import workflow as wf
# 通过MrsStep来定义一个MrsJobStep节点，
```

```
algorithm = wf.steps.MrsJobAlgorithm(
    boot_file="obs://spark-sql/wordcount.py",    #执行脚本OBS路径
    parameters=[wf.AlgorithmParameters(name="run_args", value="--master,yarn-cluster")]
)
inputs = wf.steps.MrsJobInput(name="mrs_input", data=wf.data.OBSPath(obs_path="/spark-sql/
mrs_input/")) #输入数据的OBS路径
outputs = wf.steps.MrsJobOutput(name="mrs_output",
obs_config=wf.data.OBSOutputConfig(obs_path="/spark-sql/mrs_output")) #输出的OBS路径
step = wf.steps.MrsJobStep(
        name="mrs_test",          #step名称，可自定义
        mrs_algorithm=algorithm,
        inputs=inputs,
        outputs=outputs,
        cluster_id="cluster_id_xxx"   #MRS集群ID
)
```

- **使用选取集群和启动脚本的形式**

```
from modelarts import workflow as wf
# 通过MrsJobStep来定义一个节点
run_arg_description = "程序执行参数, 作为程序运行环境参数, 默认为（ --master,yarn-cluster)"
app_arg_description = "程序执行参数, 作为启动脚本的入参, 例如（ --param_a=3,--param_b=4 ）默认为
空，非必填"
mrs_outputs_description = "数据输出路径, 可以通过从参数列表中获取--train_url参数获取"
cluster_id_description = "cluster id of MapReduce Service"

algorithm = wf.steps.MrsJobAlgorithm(
    boot_file=wf.Placeholder(name="boot_file",
                        description="程序启动脚本",
                        placeholder_type=wf.PlaceholderType.STR,
                        placeholder_format="obs"),
    parameters=[wf.AlgorithmParameters(name="run_args",
                                value=wf.Placeholder(name="run_args",
                                                     description=run_arg_description,
                                                     default="--master,yarn-cluster",
                                                     placeholder_type=wf.PlaceholderType.STR),
                                ),
                wf.AlgorithmParameters(name="app_args",
                                value=wf.Placeholder(name="app_args",
                                                    description=app_arg_description,
                                                     default="",
                                                     placeholder_type=wf.PlaceholderType.STR)
                                )
                ]
    )

inputs = wf.steps.MrsJobInput(name="data_url",
                        data=wf.data.OBSPlaceholder(name="data_url",object_type="directory"))

outputs = wf.steps.MrsJobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=wf.Placeholder(name="train_url",
placeholder_type=wf.PlaceholderType.STR,
placeholder_format="obs",description=mrs_outputs_description)))

mrs_job_step = wf.steps.MrsJobStep(
    name="mrs_job_test",
    mrs_algorithm=algorithm,
    inputs=inputs,
    outputs=outputs,
    cluster_id=wf.Placeholder(name="cluster_id", placeholder_type=wf.PlaceholderType.STR,
description=cluster_id_description, placeholder_format="cluster")
)
```

- **在控制台上如何使用MRS节点**

  Workflow发布后，在Workflow配置页，配置节点的数据输入，输出，启动脚本，集群ID等参数。

5.3.8.2 在 Workflow 中指定仅运行部分节点

Workflow通过支持预置场景的方式来实现部分运行的能力，在开发工作流时按照场景的不同对DAG进行划分，之后在运行态可选择任意场景单独运行。具体代码示例如下所示：

```
workflow =wf.Workflow(
    name="image_cls",
    desc="this is a demo workflow",
    steps=[label_step, release_data_step, training_step, model_step, service_step],
    policy=wf.policy.Policy(
        scenes=[
            wf.policy.Scene(
                scene_name="模型训练",
                scene_steps=[label_step, release_data_step, training_step]
            ),
            wf.policy.Scene(
                scene_name="服务部署",
                scene_steps=[model_step, service_step]
            ),
        ]
    )
)
```

该示例中Workflow包含了五个节点（节点相关定义已省略），在policy中定义了两个预置场景：模型训练和服务部署，工作流发布至运行态后，部分运行的开关默认关闭，节点全部运行。用户可在权限管理页面打开开关，选择指定的场景进行运行。

说明

部分运行能力支持同一个节点被定义在不同的运行场景中，但是需要用户自行保证节点之间数据依赖的正确性。另外，部分运行能力仅支持在运行态进行配置运行，不支持在开发态进行调试。

6 使用 Notebook 进行 AI 开发调试

6.1 Notebook 使用场景

ModelArts提供灵活开放的开发环境，您可以根据实际情况选择。

- ModelArts提供了云化版本的Notebook，无需关注安装配置，即开即用，具体参见**创建Notebook实例**。
- ModelArts Notebook支持以下几种使用方式，用于开发基于PyTorch、TensorFlow和MindSpore等引擎的AI模型。
  - 支持通过JupyterLab工具在线打开Notebook，具体请参见**通过JupyterLab在线使用Notebook实例进行AI开发**。
  - 支持本地IDE的方式开发模型，通过开启SSH连接，用户本地IDE可以远程连接到ModelArts的Notebook开发环境中，调试和运行代码。本地IDE方式不影响用户的编码习惯，并且可以方便快捷地使用云上的Notebook开发环境。

    本地IDE当前支持VS Code、PyCharm、SSH工具。PyCharm和VS Code还分别有专门的插件PyCharm Toolkit、VS Code Toolkit，让远程连接操作更便捷。具体参见**通过PyCharm远程使用Notebook实例**、**通过VS Code远程使用Notebook实例**、**通过SSH工具远程使用Notebook**。
- 在AI开发过程中，如何将文件方便快速地上传到Notebook几乎是每个开发者都会遇到的问题。ModelArts提供了多种文件上传方式，在文件上传过程中，可以查看上传进度和速度。

  - 将本地文件上传，请参考**支持上传本地文件**；
  - GitHub的开源仓库的文件上传，请参考**支持Clone GitHub开源仓库**；
  - 存放在OBS中的文件上传，请参考**支持上传OBS文件**；
  - 类似开源数据集这样的远端文件上传，请参考**支持上传远端文件**；
- 在Notebook的使用中，可以快速查找实例，可以在同一个Notebook实例中切换镜像，方便用户灵活调整实例的AI引擎；可以切换节点运行规格，方便用户灵活调整规格资源；可以初期存储使用量较小时选择小存储，可以在创建完成后根据需要扩充EVS容量；使用动态挂载OBS将OBS对象存储模拟成本地文件系统；还可以在Notebook异常时查看实例的事件定位等，具体参见**管理Notebook实例**。
- ModelArts CLI，集成在ModelArts开发环境Notebook中，用于连接ModelArts服务并在ModelArts资源上执行管理命令。ma-cli支持用户在ModelArts Notebook及线下虚拟机中与云端服务交互，使用ma-cli命令可以实现命令自动补全、鉴权、镜像构建、提交ModelArts训练作业、提交DLI Spark作业、OBS数据复制等，具体参见**ModelArts CLI命令参考**。
- ModelArts Notebook内置MoXing Framework模块，ModelArts mox.file提供了一套更为方便地访问OBS的API，允许用户通过一系列模仿操作本地文件系统的API来操作OBS文件。具体参见**在Notebook中使用Moxing命令**。

6.2 创建 Notebook 实例

在开始进行模型开发前，您需要创建Notebook实例，并打开Notebook进行编码。

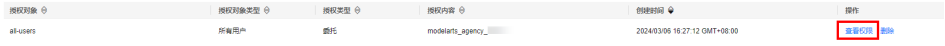
背景信息

- Notebook使用涉及到计费，具体收费项如下：
  - 处于“运行中”状态的Notebook，会消耗资源，产生费用。根据您选择的资源不同，收费标准不同，价格详情请参见**产品价格详情**。当您不需要使用Notebook时，建议停止Notebook，避免产生不必要的费用。
  - 创建Notebook时，如果选择使用云硬盘EVS存储配置，实例不删除，云硬盘EVS会一直收费，建议及时停止并删除Notebook，避免产品不必要的费用。
- 在创建Notebook时，默认会开启自动停止功能，在指定时间内停止运行Notebook，避免资源浪费。
- 只有处于“运行中”状态的Notebook，才可以执行打开、停止操作。
- 一个账户最多创建10个Notebook。

创建 Notebook 实例

1. 登录ModelArts管理控制台，在左侧导航栏中选择“权限管理”，检查是否配置了访问授权。如果未配置，请先配置访问授权。参考**使用委托授权**完成操作。

   图 6-1 查看委托配置信息

2. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”页面。
3. 单击右上角“创建”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止，详细参数请参见**表6-1**。

图 6-2 Notebook 基本信息

表 6-1 基本信息的参数描述

参数名称	说明
“名称”	Notebook的名称。系统会自动生成一个名称，您可以根据业务需求重新命名，命名规则如下：只能包含数字、大小写字母、下划线和中划线，长度不能超过128位且不能为空。
“描述”	对Notebook的简要描述。
“自动停止”	默认开启，且默认值为“1小时”，表示该Notebook实例将在运行1小时之后自动停止，即1小时后停止规格资源计费。可选择“1小时”、“2小时”、“4小时”、“6小时”或“自定义”几种模式。选择“自定义”模式时，可指定1~72小时范围内任意整数。 • 定时停止：开启定时停止功能后，该Notebook实例将在运行时长超出您所选择的时长后，自动停止。 **说明** 出于对用户任务进度的保护，在您设置的自动停止时间到达后，Notebook不会立即自动停止，可能会有2-5分钟的延迟（此过程正常计费），方便您进行续约。

b. 填写Notebook详细参数，如镜像、资源规格等，详细参数请参见表6-2。

表 6-2 Notebook 实例的详细参数说明

参数名称	说明
“镜像”	支持公共镜像和自定义镜像。 ● 公共镜像：即预置在ModelArts内部的AI引擎。 ● 自定义镜像：可以将基于公共镜像创建的实例保存下来，作为自定义镜像使用，请参考**保存Notebook实例**。也可以基于预置镜像或第三方镜像制作自定义镜像，请参考**Notebook的自定义镜像制作方法**。 一个镜像对应支持一种AI引擎，创建Notebook实例时选择好了对应AI引擎的镜像。用户可以根据需要选择镜像。在右侧搜索框中输入镜像名称关键字，可快速查找镜像。 Notebook运行停止后，可以在同一个Notebook实例中变更镜像。
“资源类型”	支持公共资源池和专属资源池。 “公共资源池”无需单独购买，即开即用，按需付费，即按您的Notebook实例运行时长进行收费。 “专属资源池”按实际情况选择已创建的专属资源池。如果没有专属资源，需要单独购买并创建。 **说明** 如果您购买的专属池是单节点的Tnt004规格：GPU: 1*tnt004 | CPU: 8核 32GiB (modelarts.vm.gpu._tnt004u8)，使用该集群创建Notebook实例时，Tnt004卡空闲但是规格显示售罄或者创建失败显示资源不足时，请联系技术支撑。
“类型”	芯片类型包括CPU、GPU类型。 不同的镜像支持的芯片类型不同，根据实际需要选择。 GPU性能更佳，但是相对CPU而言，费用更高。
“实例规格”	根据选择的芯片类型不同，可选资源规格也不同。请根据界面实际情况和需要选择。 ● CPU规格 “2核8GB”：Intel CPU通用规格，用于快速数据探索和实验 “8核32GB”：Intel CPU算力增强型，适用于密集计算场景下运算 ● GPU规格 “GPU: 1*Vnt1(32GB)|CPU: 8 核 64GB”：GPU单卡规格，32GB显存，适合深度学习场景下的算法训练和调测 “GPU: 1*Tnt004(16GB)|CPU: 8核* 32GB”： GPU单卡规格，16GB显存，推理计算最佳选择，覆盖场景包括计算机视觉、视频处理、NLP等 “GPU: 1*Pnt1(16GB)|CPU: 8核 64GB”：GPU单卡规格，16GB显存，适合深度学习场景下的算法训练和调测

参数名称	说明
“存储配置”	包括“云硬盘EVS”、“弹性文件服务SFS”、“对象存储服务OBS”和“并行文件系统PFS”。请根据界面实际情况和需要选择。 **说明** “对象存储服务OBS”、“并行文件系统PFS”是白名单功能，如果有试用需求，请提工单申请权限。 • 选择“云硬盘EVS”作为存储位置。 根据实际使用量设置磁盘规格。磁盘规格默认5GB。磁盘规格的最大值请以实际界面显示为准。 从Notebook实例创建成功开始，直至实例删除成功，磁盘每GB按照规定费用收费。 • 选择“弹性文件服务SFS”作为存储位置。 仅专属资源池支持，并需要在专属资源池对应的网络打通VPC才能生效，具体操作请参见**ModelArts网络**。 **说明** 如果需要设置SFS Turbo的文件夹权限，请参考**权限管理**文档配置。 – “弹性文件服务”：选择已创建的SFS Turbo（在**弹性文件服务控制台**创建SFS Turbo）。 – “云上挂载路径”：默认为/home/ma-user/work/。 – “子目录挂载”：选择SFS Turbo的存储位置。 – “挂载方式”：当用户配置了文件夹控制权限，则显示此参数。根据SFS Turbo存储位置的权限显示“读写”或“只读”。 • 选择“对象存储服务OBS”或“并行文件系统PFS”作为存储位置。 选择“存储位置”：设置用于存储Notebook数据的OBS路径。如果想直接使用已有的文件或数据，可将数据提前上传至对应的OBS路径下。“存储位置”不能设置为OBS桶的根目录，需设置为对应OBS桶下的具体目录。 选择“凭据”：选择已有的凭据或单击右侧的“立即创建”，跳转至数据加密控制台**创建凭据**，凭据键/值填写用户的AK、SK信息（“键”分别填写“accessKeyId”，“secretAccessKey”；“值”在控制台个人账号下“我的凭证>访问密钥”获取AK、SK）。 **图 6-3** 设置凭据值  “云硬盘EVS”、“弹性文件服务SFS”的存储路径挂载在/home/ma-user/work目录下。 Notebook实例运行中，可以通过**动态挂载OBS并行文件系统**操作来增加数据存储路径。

参数名称	说明
	停止或重启Notebook实例时，存储的内容会被保留，不丢失。 删除Notebook实例时，EVS存储会一起释放，存储的内容不保留。SFS可以重新挂载到新的Notebook，可以保留数据。
“扩展存储配置”	**说明** “扩展存储配置”功能是白名单功能，如果有试用需求，请提工单申请权限。 如果有多个数据存储路径，可以单击“增加扩展存储配置”，增加用户指定的存储挂载目录。支持增加的存储类型有“存储桶OBS”、“并行文件系统PFS”、“弹性文件服务SFS”。 约束限制： ● 每种存储类型最多支持挂载5个。 ● 扩展存储挂载目录不允许重复，不允许挂载到黑名单目录，允许嵌套挂载。不允许挂载的黑名单目录为以下前缀匹配的目录：/data/、/cache/、/dev/、/etc/、/bin/、/lib/、/sbin/、/modelarts/、/train-worker1-log/、/var/、/resource_info/、/usr/、/sys/、/run/、/tmp/、/infer/、/opt/ 添加扩展存储后，可进入Notebook实例详情页，单击“存储配置 > 扩展存储”，查看或编辑扩展存储信息。在存储个数未达到最大个数时，也可在右侧单击“添加扩展存储”。
“SSH远程开发”	● 开启此功能后，用户可以在本地开发环境中远程接入Notebook实例的开发环境。 ● 实例在停止状态时，用户可以在Notebook详情页中更新SSH的配置信息。 **说明** 开启此功能的实例中会预置VS Code插件（python、jupyter等）以及VS Code Server包，会占用约1G左右的持久化存储空间。
“密钥对”	开启“SSH远程开发”功能后，需要设置此参数。 可以选择已有密钥对。 也可以单击密钥对右侧的“立即创建”，跳转到数据加密控制台，在“密钥对管理 > 账号密钥对”页面，单击“创建密钥对”。 创建完Notebook后，可以在Notebook详情页中修改密钥对。 **注意** 创建好的密钥对，请下载并妥善保存，使用本地IDE远程连接云上Notebook开发环境时，需要用到密钥对进行鉴权认证。

参数名称	说明
“远程访问白名单”	可选，开启“SSH远程开发”功能后，可以设置此参数。 设置为允许远程接入访问这个Notebook的IP地址（例如本地PC的IP地址或者访问机器的外网IP地址，最多配置5个，用英文逗号隔开），不设置则表示无接入IP地址限制。 如果用户使用的访问机器和ModelArts服务的网络有隔离，则访问机器的外网地址需要在主流搜索引擎中搜索“IP地址查询”获取，而不是使用**ipconfig**或**ifconfig/ip**命令在本地查询。 创建完Notebook后，可以在Notebook详情页中修改白名单IP地址。

c. 可选：添加Notebook标签。在标签栏输入“标签键”和“标签值”，单击“添加”。

表 6-3 添加标签

参数名	参数说明
“标签”	ModelArts支持对接标签管理服务TMS，在ModelArts中创建资源消耗性任务（例如：创建Notebook、训练作业、推理在线服务）时，可以为这些任务配置标签，通过标签实现资源的多维分组管理。 标签详细用法请参见**ModelArts如何通过标签实现资源分组管理**。 添加标签后，可在Notebook实例详情页查看标签内容，也可进行修改、删除标签。

说明

可以在标签输入框下拉选择TMS预定义标签，也可以自己输入自定义标签。预定义标签对所有支持标签功能的服务资源可见。租户自定义标签只对自己服务可见。

4. 参数填写完成后，单击“立即创建”进行规格确认。
5. 参数确认无误后，单击“提交”，完成Notebook的创建操作。

   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。
6. 在Notebook列表，单击实例名称，进入实例详情页，查看Notebook实例配置信息。

   “SSH远程开发”功能开启时，在“白名单”右侧单击修改，可以修改允许远程访问的白名单IP地址。实例在停止状态时，在“认证”右侧单击修改，用户可以更新密钥对。

单击“存储配置”页签的“添加数据存储”，可以挂载OBS并行文件系统，方便读取数据，具体操作参见**动态挂载OBS并行文件系统**。

如果存储使用的是云硬盘EVS，单击存储容量右侧的“扩容”，可以动态扩充云硬盘EVS的容量，具体操作参见**动态扩充云硬盘EVS容量**。

打开 Notebook 实例

针对创建好的Notebook实例（即状态为“运行中”的实例），可以打开Notebook并在开发环境中启动编码。

- 在线JupyterLab访问，具体参见**通过JupyterLab在线使用Notebook实例进行AI开发**。
- 本地IDE使用PyCharm工具，远程连接访问，具体参见**通过PyCharm远程使用Notebook实例**。
- 本地IDE使用VS Code工具，远程连接访问，具体参见**通过VS Code远程使用Notebook实例**。
- 本地IDE使用SSH工具，远程连接访问，具体参见**通过SSH工具远程使用Notebook**。

ModelArts提供的Notebook实例是以ma-user启动的，用户进入实例后，工作目录默认是/home/ma-user。

```
sh-4.4$pwd
/home/ma-user
sh-4.4$
```

Notebook 容器挂载目录介绍

创建Notebook实例，存储选择EVS时，Notebook会使用/home/ma-user/work目录作为用户的工作空间持久化存储。

存放在work目录的内容，在实例停止、重新启动后依然保留，其他目录下的内容不会保留，使用开发环境时建议将需要持久化的数据放在/home/ma-user/work目录。

更多Notebook实例的目录挂载情况（以下挂载点在保存镜像的时候不会保存）如**表6-4**所示。

表 6-4 Notebook 挂载目录介绍

挂载点	是否只读	备注
/home/ma-user/work/	否	客户数据的持久化目录。
/data	否	客户PFS的挂载目录。
/cache	否	裸机规格时支持，用于挂载宿主机NVMe的硬盘。
/train-worker1-log	否	兼容训练作业调试过程。
/dev/shm	否	用于PyTorch引擎加速。

Notebook 选择存储说明

不同存储的实现方式都不同，在性能、易用性、成本的权衡中可以有不同的选择，没有一个存储可以覆盖所有场景，了解下云上开发环境中各种存储使用场景说明，更能提高使用效率。

表 6-5 云上开发环境中各种存储使用场景说明

存储类型	建议使用场景	优点	缺点
云硬盘EVS	比较适合只在开发环境中做数据、算法探索，性能较好。	块存储SSD，可以理解为一个磁盘，整体IO性能比NFS要好，可以动态扩充，最大可以到4096GB。 云硬盘EVS作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留，存储支持在线按需扩容。	只能在单个开发环境中使用 。

存储类型	建议使用场景	优点	缺点
并行文件系统PFS	**说明** • 并行文件系统PFS为白名单功能，如需使用，请联系华为技术支持开通。 • 仅支持挂载同一区域下的OBS并行文件系统（PFS）。 适合直接使用PFS桶作为持久化存储进行AI开发和探索，使用场景如下。 1. 数据集的存储。将存储在PFS桶的数据集直接挂载到Notebook进行浏览和数据处理，在训练时直接使用。直接在创建Notebook的时候选择**并行文件系统PFS**。或在实例运行后，将承载数据集的OBS并行文件系统动态挂载至Notebook中，详细操作请参考**动态挂载OBS并行文件系统**。 2. 代码的存储。在Notebook调测完成，可以直接指定对应的对象存储路径作为启动训练的代码路径，方便临时修改。 3. 训练观测。可以将训练日志等输出路径进行挂载，在Notebook中实时查看和观测，特别是利用**TensorBoard可视化**功能完成对训练输出的分析。	PFS是一种经过优化的高性能对象存储文件系统，存储成本低，吞吐量大，能够快速处理高性能计算（HPC）工作负载。在需要使用对象存储服务场景下，推荐使用PFS挂载。 **说明** 建议上传时按照128MB或者64MB打包或者切分，使用时边下载边解压后在本地存储读取，以获取更好的读写与吞吐性能。	小文件频繁读写性能较差，例如直接作为存储用于模型重型训练，大文件解压等场景慎用。 **说明** PFS挂载需要用户对当前桶授权给ModelArts完整读写权限，Notebook删除后，此权限策略不会被删除。

存储类型	建议使用场景	优点	缺点
对象存储服务OBS	**说明** • OBS对象存储为白名单功能，如需使用，请联系华为技术支持开通。 • 仅支持挂载同一区域下的OBS对象存储。 在开发环境中做大规模的数据上传下载时，可以通过OBS桶做中转。	存储成本低，吞吐量大，但是小文件读写较弱。建议上传时按照128MB或者64MB打包或者切分，使用时边下载边解压后在本地读取。	对象存储语义，和Posix语义有区别，需要进一步理解。
弹性文件服务SFS	目前只支持在专属资源池中使用；针对探索、实验等非正式生产场景，建议使用这种。开发环境和训练环境可以同时挂载一块SFS存储，省去了每次训练作业下载数据的要求，一般来说重IO读写模型，超过32卡的大规模训练不适合。	实现为NFS，可以在多个开发环境、开发环境和训练之间共享，如果不需要重型分布式训练作业，特别是启动训练作业时，不需要额外再对数据进行下载，这种存储便利性可以作为首选。	性能比EVS云硬盘块存储低。
OceanStor Pacific存储（SFS容量型2.0）	目前只支持在天工资源池中使用。 适合直接使用SFS容量型2.0提供的文件系统作为训练作业所需的存储进行AI模型的训练和探索。同时提供OBS接口，支持从云外导入训练数据。	提供高性能文件客户端，满足重型训练作业中对存储高带宽诉求，同时提供OBS访问功能，同一份训练数据通过OBS接口导入到存储之后不需要再进相关转化，即可支持模型训练。	提供对象存储语义，和Posix语义有区别，需要进一步理解。
本地存储	重型训练作业首选	运行所在虚拟机或者裸金属机器上自带的SSD高性能存储，文件读写的吞吐量大，建议对于重型训练作业先将数据准备到对应目录再启动训练。 默认在容器/cache目录下进行挂载，/cache目录可用空间请参考**开发环境中不同Notebook规格资源“/cache”目录的大小**。	存储生命周期和容器生命周期绑定，每次训练都要下载数据。

1. 在开发环境中如何使用云硬盘EVS块存储?

   例如，在**创建Notebook实例**时选择云硬盘EVS存储小容量，Notebook运行过程中如果发现存储容量不够，可以扩容，请参考**动态扩充云硬盘EVS容量**。

2. 在开发环境中如何使用OBS并行文件系统?

   例如，在Notebook中训练时，可直接使用挂载至Notebook容器中的数据集，在运行过程中可以**动态挂载OBS并行文件系统**。

6.3 通过 JupyterLab 在线使用 Notebook 实例进行 AI 开发

6.3.1 使用 JupyterLab 在线开发和调试代码

JupyterLab是一个交互式的开发环境，可以使用它编写Notebook、操作终端、编辑MarkDown文本、打开交互模式、查看csv文件及图片等功能。可以说，JupyterLab是开发者们下一阶段更主流的开发环境。

ModelArts支持通过JupyterLab工具在线打开Notebook，开发基于PyTorch、TensorFlow和MindSpore引擎的AI模型。具体操作流程如**图1 使用JupyterLab在线开发调试代码**所示。

图 6-4 使用 JupyterLab 在线开发调试代码

操作步骤

1. 创建Notebook实例。

   在ModelArts控制台创建一个Notebook实例，选择要使用的AI框架。具体参见**创建Notebook实例**。

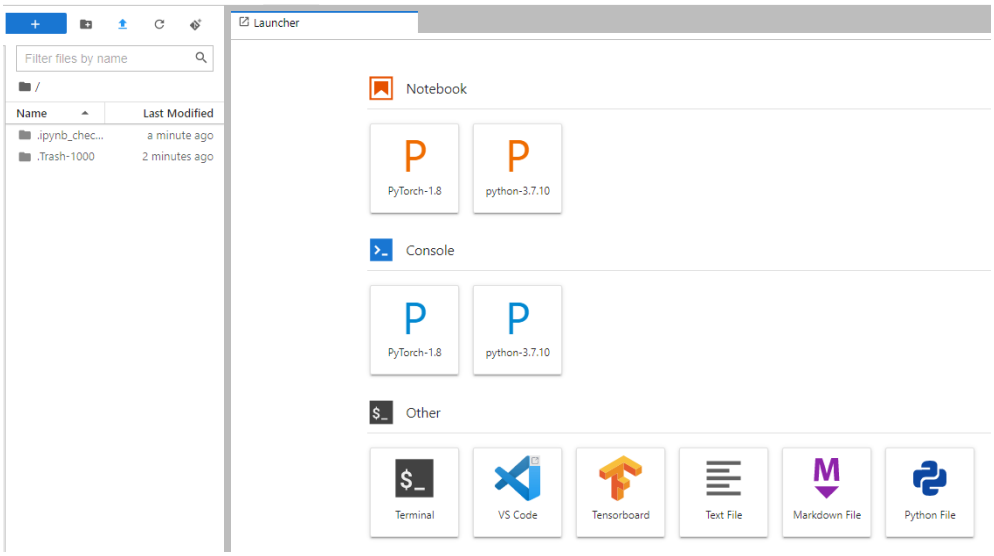
2. 创建成功后，Notebook实例的状态为“运行中”，单击操作列的“打开”，访问JupyterLab。

   图 6-5 打开 Notebook 实例

3. 进入JupyterLab页面后，自动打开Launcher页面，如下图所示。您可以使用开源支持的所有功能，详细操作指导可参见**JupyterLab官网文档**。

图 6-6 JupyterLab 主页

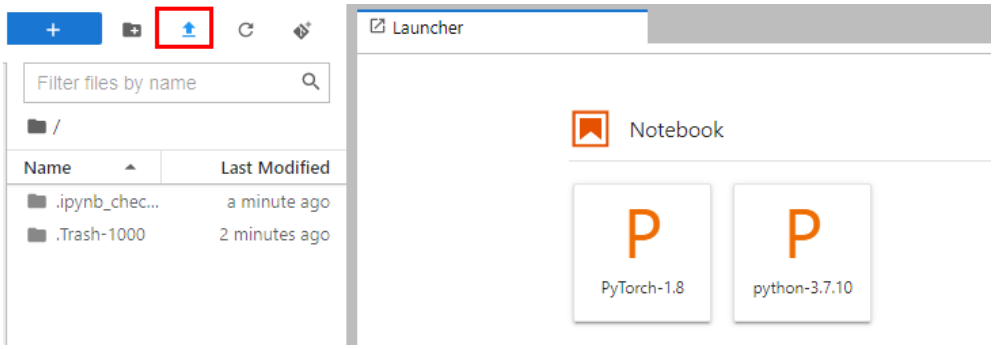


说明

不同AI引擎的Notebook，打开后Launcher页面呈现的Notebook和Console内核及版本均不同，图6-6仅作为示例，请以实际控制台为准。

4. 准备训练数据和代码文件，上传到JupyterLab中。具体参见**上传本地文件至JupyterLab**。

图 6-7 文件上传按钮

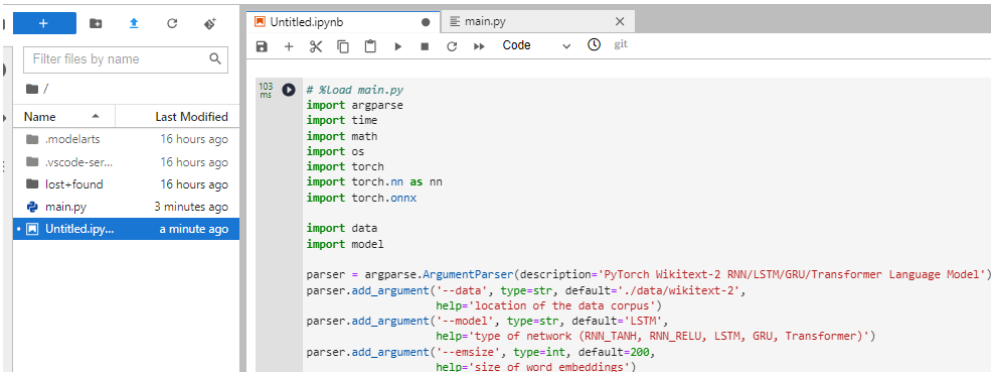


5. 在左侧导航双击打开上传的代码文件，在JupyterLab中编写代码文件，并运行调试。有关JupyterLab的使用具体参见**JupyterLab常用功能介绍**。

说明

如果您的代码文件是.py格式，请新打开一个.ipynb文件，执行**%load main.py**命令将.py文件内容加载至.ipynb文件后进行编码、调试等。

图 6-8 打开代码文件



6. 在JupyterLab中直接调用ModelArts提供的SDK，创建训练作业，上云训练。
   调用SDK创建训练作业的操作请参见**调用SDK创建训练作业**。

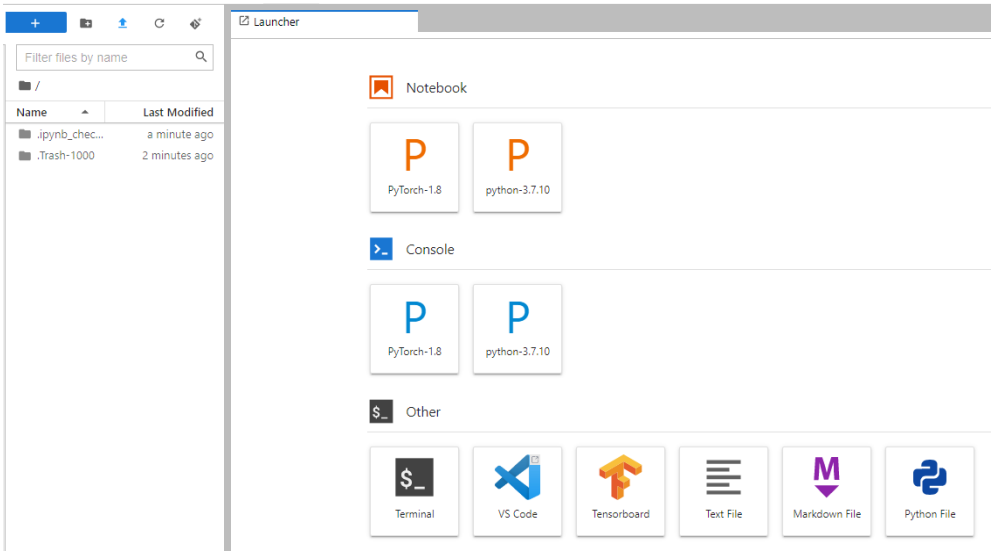
6.3.2 JupyterLab 常用功能介绍

JupyterLab 主页介绍

下面介绍如何从运行中的Notebook实例打开JupyterLab。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“开发空间 > Notebook”，进入Notebook页面。
2. 选择状态为“运行中”的Notebook实例，单击操作列的“打开”，访问JupyterLab。
3. 进入JupyterLab页面后，自动打开Launcher页面，如下图所示。您可以使用开源支持的所有功能，详细操作指导可参见**JupyterLab官网文档**。

图 6-9 JupyterLab 主页



说明

不同AI引擎的Notebook，打开后Launcher页面呈现的Notebook和Console内核及版本均不同，**图6-9**仅作为示例，请以实际控制台为准。

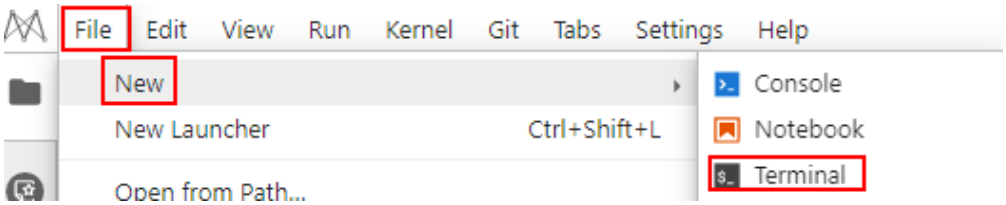
- Notebook：选择运行Notebook的一个内核，例如TensorFlow、python
- Console：可调出终端进行命令控制
- Other：可编辑其他文件

在 JupyterLab 中新建 Terminal

在Terminal中可以执行Python命令，操作终端，如下步骤详细介绍了如何打开JupyterLab的Terminal。

1. 创建Notebook实例，实例处于“运行中”，单击“操作”列的“打开”，进入“JupyterLab”开发页面。
2. 选择“Files > New > Terminal”，进入到Terminal界面。

图 6-10 进入 Terminal 界面



3. 例如，通过Terminal在“TensorFlow-1.8”的环境中使用**pip**安装Shapely。

   在代码输入栏输入以下命令，获取当前环境的kernel，并激活需要安装依赖的python环境。

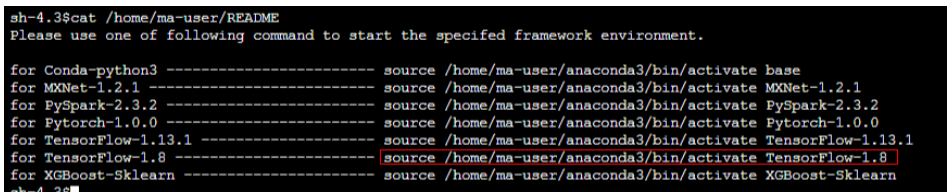
   cat /home/ma-user/README

   source /home/ma-user/anaconda3/bin/activate TensorFlow-1.8

   说明

   如果需要在其他python环境里安装，请将命令中“TensorFlow-1.8”替换为其他引擎。

图 6-11 激活环境



在代码输入栏输入以下命令安装Shapely。

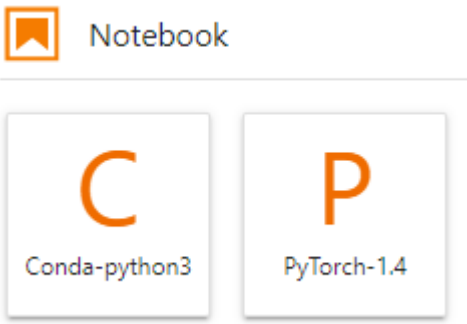
pip install Shapely

在 JupyterLab 中新建 ipynb 文件

进入JupyterLab主页后，可在“Notebook”区域下，选择适用的AI引擎，单击后将新建一个对应框架的ipynb文件。

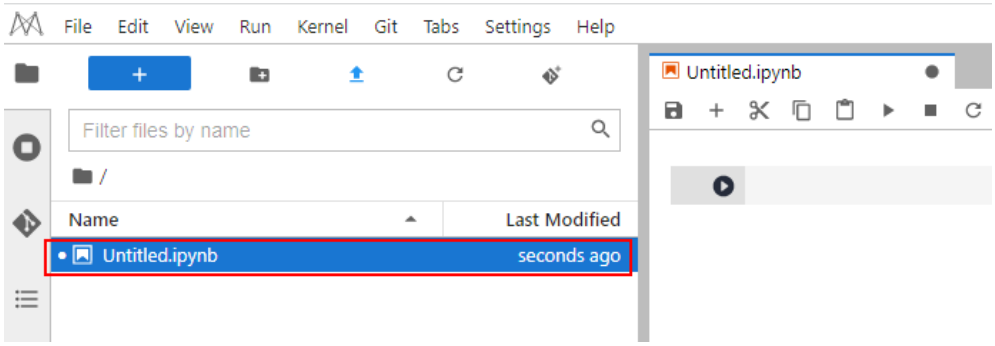
由于每个Notebook实例选择的工作环境不同，其支持的AI框架也不同，下图仅为示例，请根据实际显示界面选择AI框架。

图 6-12 选择 AI 引擎并新建一个 ipynb 文件



新建的ipynb文件将呈现在左侧菜单栏中。

图 6-13 新建文件



新建文件并打开 Console

Console的本质为Python终端，输入一条语句就会给出相应的输出，类似于Python原生的IDE。

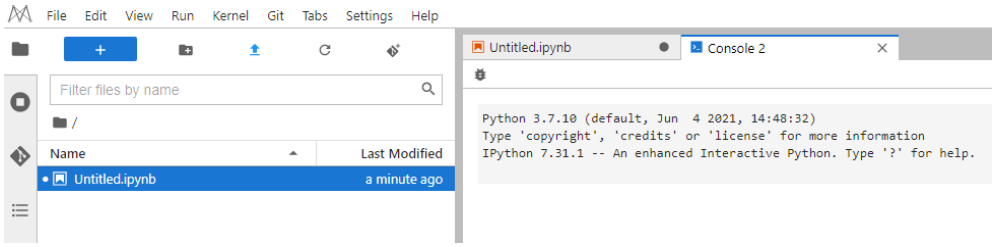
进入JupyterLab主页后，可在“Console”区域下，选择适用的AI引擎，单击后将新建一个对应框架的Notebook文件。

由于每个Notebook实例选择的工作环境不同，其支持的AI框架也不同，下图仅为示例，请根据实际显示界面选择AI框架。

图 6-14 选择 AI 引擎并新建一个 Console

文件创建成功后，将直接呈现Console页面。

图 6-15 新建文件（Console）

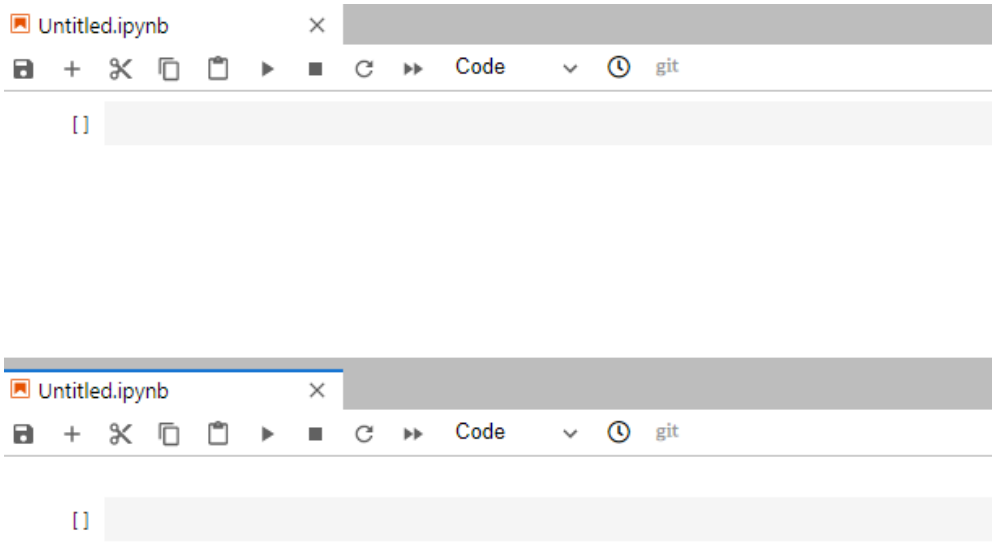


在 JupyterLab 中编辑文件

JupyterLab可以在同一个窗口同时打开几个Notebook或文件（如HTML、TXT、Markdown等），以页签形式展示。

JupyterLab的一大优点是，可以任意排版多个文件。在右侧文件展示区，您可以拖动打开文件，随意调整文件展示位置，可以同时打开多个文件。

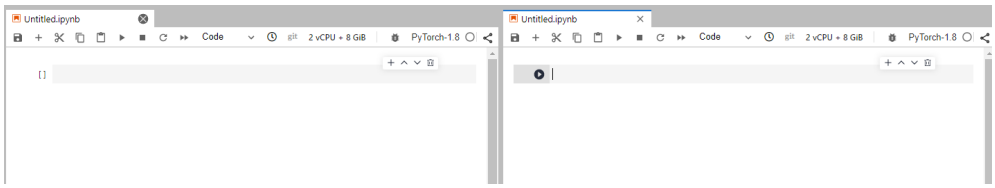
图 6-16 多文件任意编排



当在一个Notebook中写代码时，如果需要实时同步编辑文件并查看执行结果，可以新建该文件的多个视图。

打开ipynb文件，然后单击菜单栏“File > New View for Notebook”，即可打开多个视图。

图 6-17 同一个文件的多个视图



JupyterLab的ipynb文件代码栏中输入代码，需要在代码前加!符号。

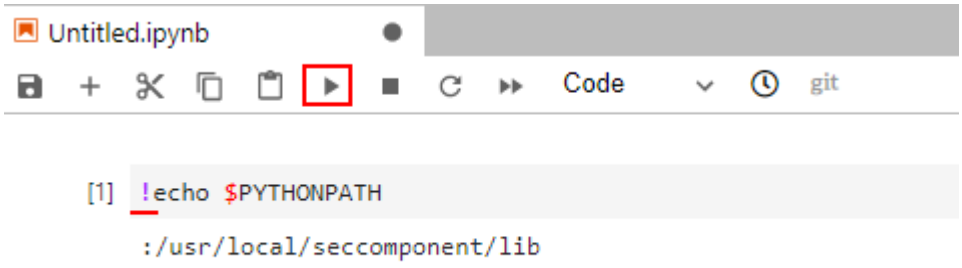
例如：安装外部库Shapely

```
!pip install Shapely
```

例如：查看PythonPath

```
!echo $PYTHONPATH
```

图 6-18 运行代码

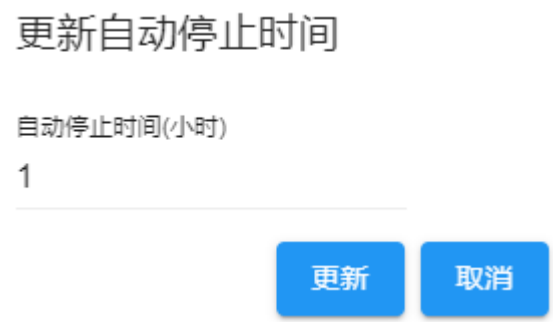


自动停止及续期

在创建或启动Notebook时，如果启用了自动停止功能，则在JupyterLab的右上角会显示当前实例停止的剩余时长，在计时结束前可以单击剩余时间进行续期。

图 6-19 自动停止

图 6-20 续期



JupyterLab 常用快捷键和插件栏

图 6-21 JupyterLab 常用快捷键和插件栏

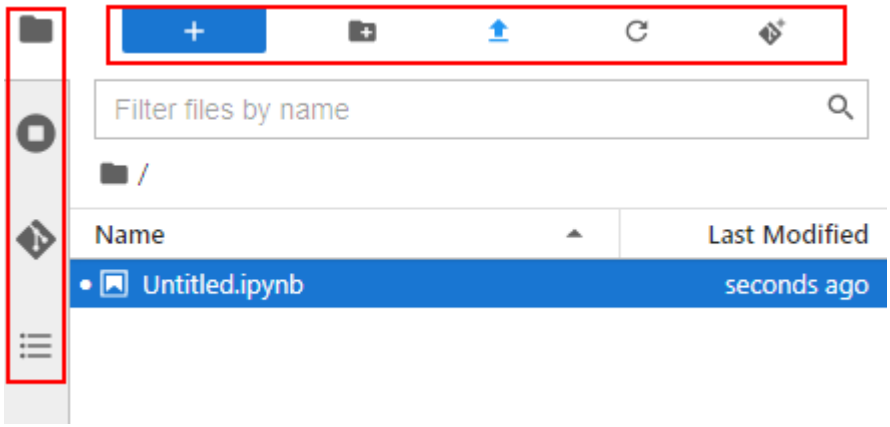


表 6-6 快捷键说明

快捷键	说明
+	快速打开Notebook、Terminal。或打开Launcher页面，可快速创建新的Notebook、Console或其他文件。
	创建文件夹。
	上传文件。
	刷新文件目录。
	Git插件，可连接此Notebook实例关联的Github代码库。

表 6-7 插件栏常用插件说明

插件	说明
	文件列表。单击此处，将展示此Notebook实例下的所有文件列表。
	当前实例中正在运行的Terminal和Kernel。
	Git插件，可以方便快捷地使用Github代码库。
	属性检查器。

插件	说明
	文档结构图。

图 6-22 导航栏按钮

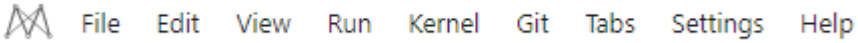


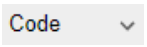
表 6-8 导航栏按钮介绍

按钮	说明
File	新建、关闭、保存、重新加载、重命名、导出、打印Notebook等功能。
Edit	编辑ipynb文件中代码块的相关操作，包括撤销、重做、剪切、复制、粘贴、选择、移动、合并、清除、查找代码块等。
View	查看视图相关操作。
Run	运行代码块相关操作，例如：运行选中代码块、一键运行所有代码块等。
Kernel	中断、重启、关闭、改变Kernel相关操作。
Git	Git插件相关操作，可以方便快捷地使用Github代码库。
Tabs	同时打开多个ipynb文件时，通过Tabs激活或选择文件。
Settings	JupyterLab工具系统设置。
Help	JupyterLab工具自带的帮助参考。

图 6-23 ipynb 文件菜单栏中的快捷键

表 6-9 ipynb 文件菜单栏中的快捷键

快捷键	说明
	保存文件。
+	添加新代码块。

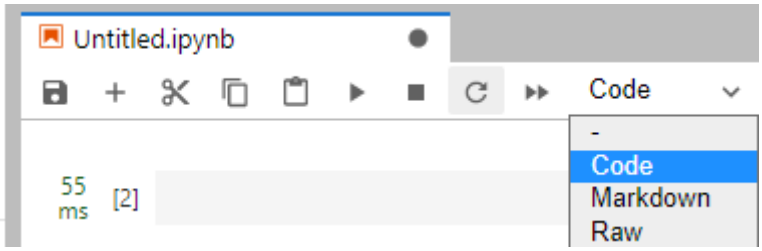
快捷键	说明
	剪切选中的代码块。
	复制选中的代码块。
	粘贴选中的代码块。
	执行选中的代码块。
	终止kernel。
	重启kernel。
	重启kernel，然后重新运行当前Notebook的所有代码。
	此处下拉框有4个选项，分别是： Code（写python代码），Markdown（写Markdown代码，通常用于注释），Raw（一个转换工具），-（不修改）。
	查看代码历史版本。
	git插件，图标显示灰色表示当前Region不支持。
2 vCPU + 4 GiB	当前的资源规格。
PyTorch-1.4	单击可以选择Kernel。
	表示代码运行状态，变为实心圆  时，表示代码在运行中。

代码化参数插件的使用

代码参数化插件可以降低Notebook案例的复杂度，用户无需感知复杂的源码，按需调整参数快速进行案例复现、模型训练等。该插件可用于定制Notebook案例，适用于比赛、教学等场景。

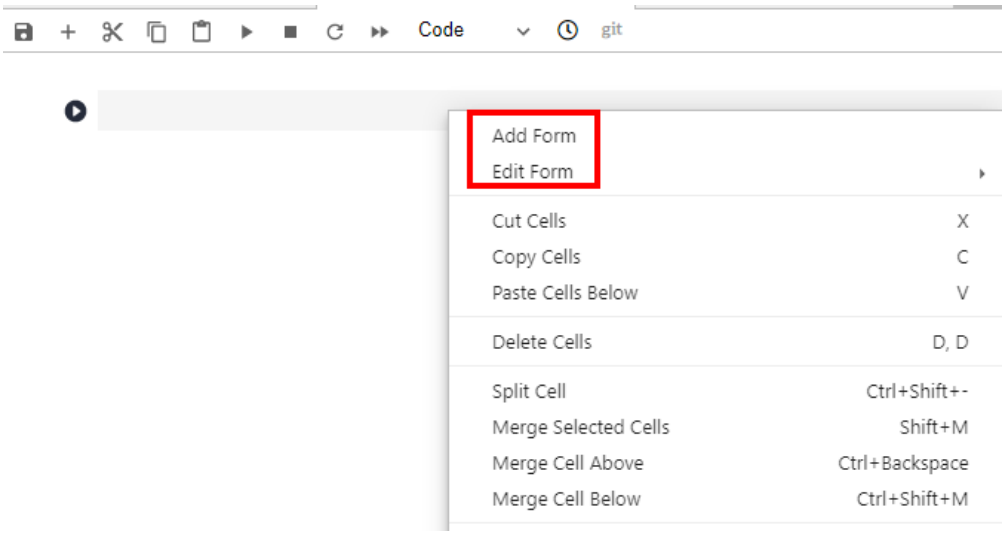
- 仅对Code cell类型新增了Edit Form和Add Form功能，如果cell类型是Markdown或者Raw类型则不支持。如下图所示：

图 6-24 查看 Code cell



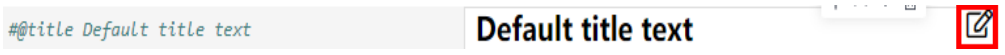
- 打开新的代码后，需先Add Form，再Edit Form。

图 6-25 Code 类型的 cell 右键选项



- “Add Form”会将Code cell水平拆分为两种编辑区域，左侧为代码区域，右侧为表单区域。单击表单右侧的“Edit”可修改默认标题。

图 6-26 两种编辑区域



- “Edit Form”按钮有四个子选项，分别是“Add new form field”、“Hide code”、“Hide form”和“show all”四个按钮，下文介绍这四个选项的功能。

表 6-10 “Edit Form”子选项介绍

“Edit Form”子选项	功能说明
Add new form field	• 支持新增“dropdown”、“input”和“slider”类型的表单。如图6-27所示。每新增一个字段，会分别在代码和表单区域中增加对应的变量，修改表单区域的值也会同时修改代码变量值。 **说明** 创建dropdown类型的表单时，“ADD Item”至少创建2项。如图6-28所示。 • 表单字段类型为“dropdown”时，支持的变量类型为“raw”和“string”。 • 表单字段类型为“input”时，支持的变量类型有“boolean”、“date”、“integer”、“number”、“raw”和“string”。 • 表单字段类型为“slider”时，支持输入滑动条的最小值、最大值和步长。
Hide code	隐藏代码区域。
Hide form	隐藏表单区域。

“Edit Form”子选项	功能说明
Show all	同时展示code和form区域。

图 6-27 “dropdown”，“input”，“slider”的表单样式

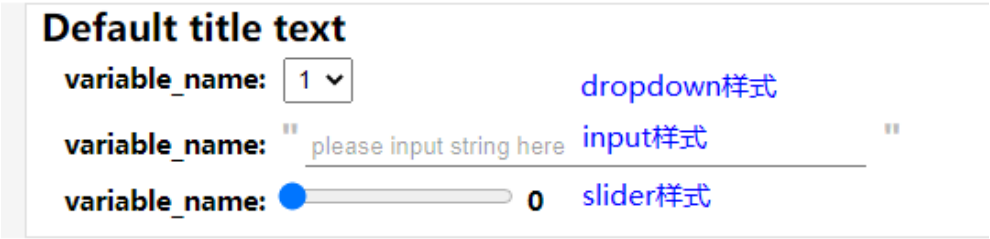


图 6-28 创建“dropdown”类型的表单

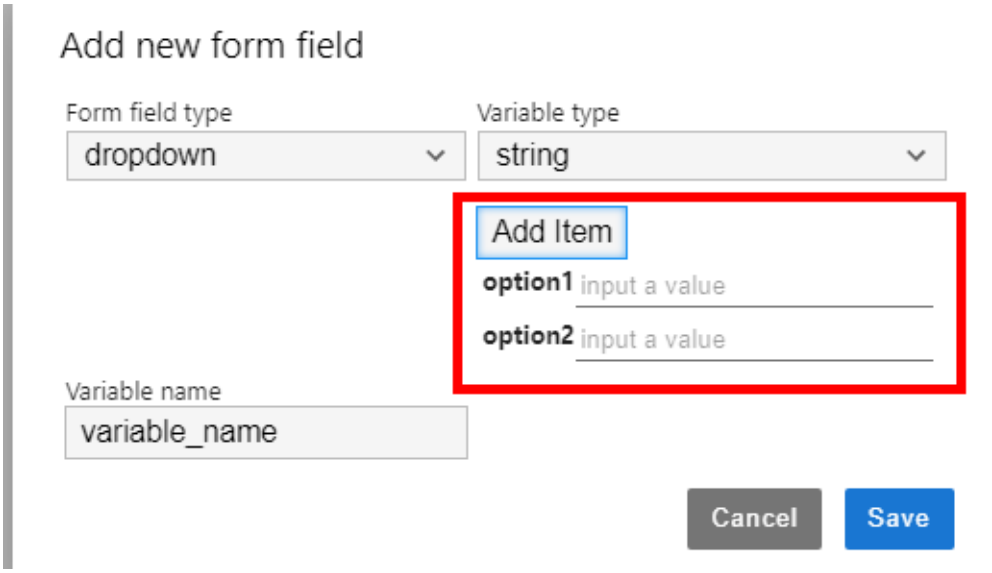
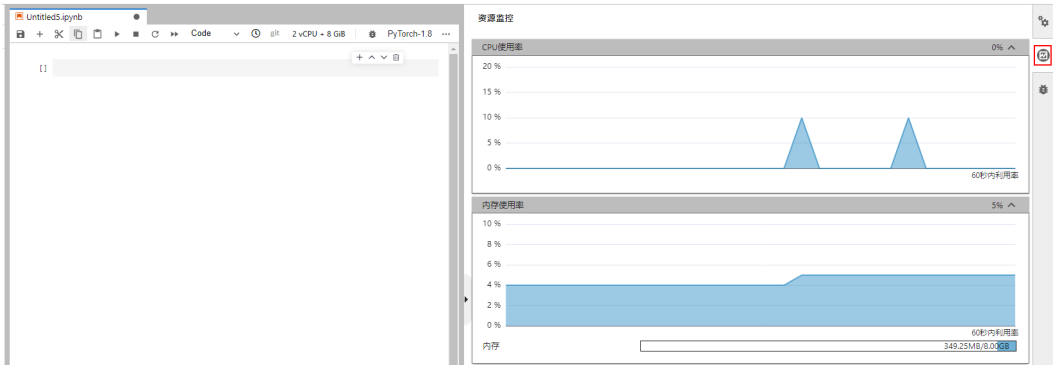


图 6-29 删除表单

资源监控

在使用过程中，如果想了解资源使用情况，可在右侧区域选择“Resource Monitor”，展示“CPU使用率”和“内存使用率”。

图 6-30 资源监控



6.3.3 在 JupyterLab 使用 Git 克隆代码仓

在JupyterLab中使用Git插件可以克隆GitHub开源代码仓库，快速查看及编辑内容，并提交修改后的内容。

前提条件

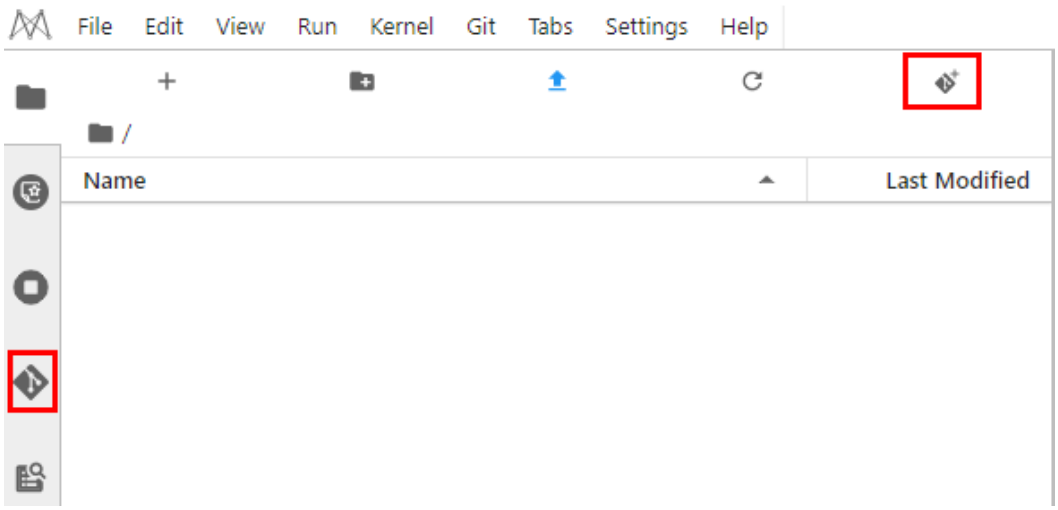
Notebook处于运行中状态。

打开 JupyterLab 的 git 插件

在Notebook列表中，选择一个实例，单击右侧的打开进入“JupyterLab”页面。

图6-31所示图标，为JupyterLab的Git插件。

图 6-31 Git 插件

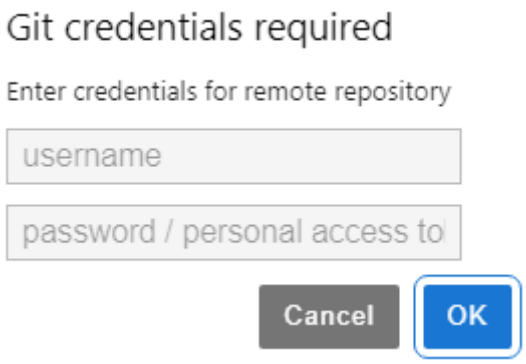


克隆 GitHub 的开源代码仓库

GitHub开源仓库地址：https://github.com/jupyterlab/extension-examplesitHub，单击 ，输入仓库地址，单击确定后即开始克隆，克隆完成后，JupyterLab左侧导航出现代码库文件夹。

克隆 GitHub 的私有仓库

1. 克隆GitHub私有仓库时，会弹出输入个人凭证的对话框，如下图。此时需要输入GitHub中Personal Access Token信息。



2. 查看Personal Access Token步骤如下：
   a. 登录**Github**，打开设置页面。
   b. 单击“Developer settings”。
   c. 单击“Personal access tokens > Generate new token”。
   d. 验证登录账号。
   e. 填写Token描述并选择权限，选择私有仓库访问权限，单击“Generate token”生成Token。
   f. 复制生成的Token到编译构建服务即可。

须知

- Token生成后，请及时保存，下次刷新页面将无法读取，需要重新生成新Token。
- 注意填写有效的Token描述信息，避免误删除导致构建失败。
- 无需使用时及时删除Token，避免信息泄露。

图 6-32 克隆 GitHub 的私有仓库（目前只支持 Personal Access Token 授权）

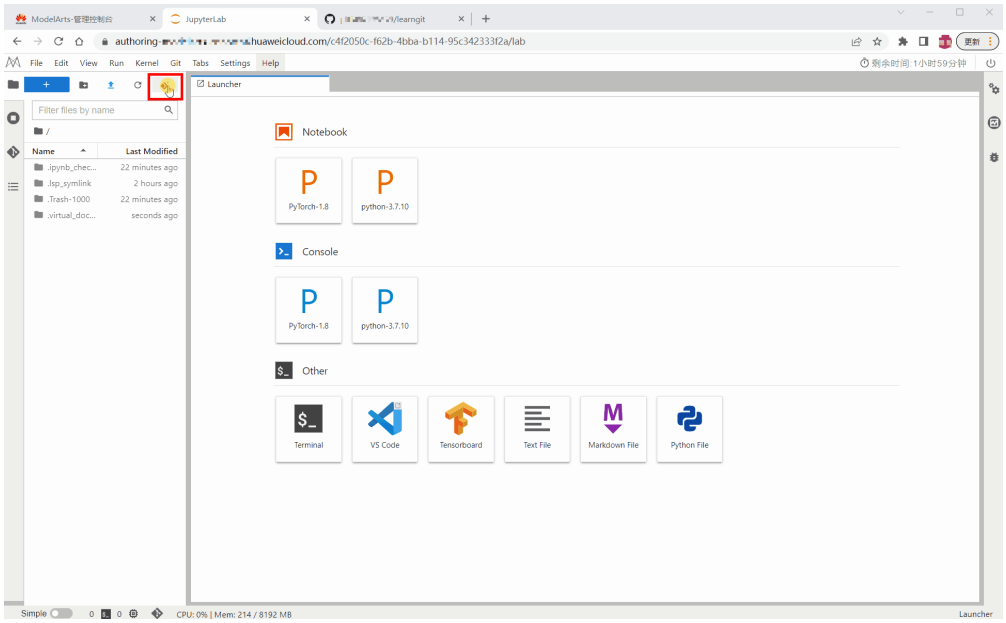
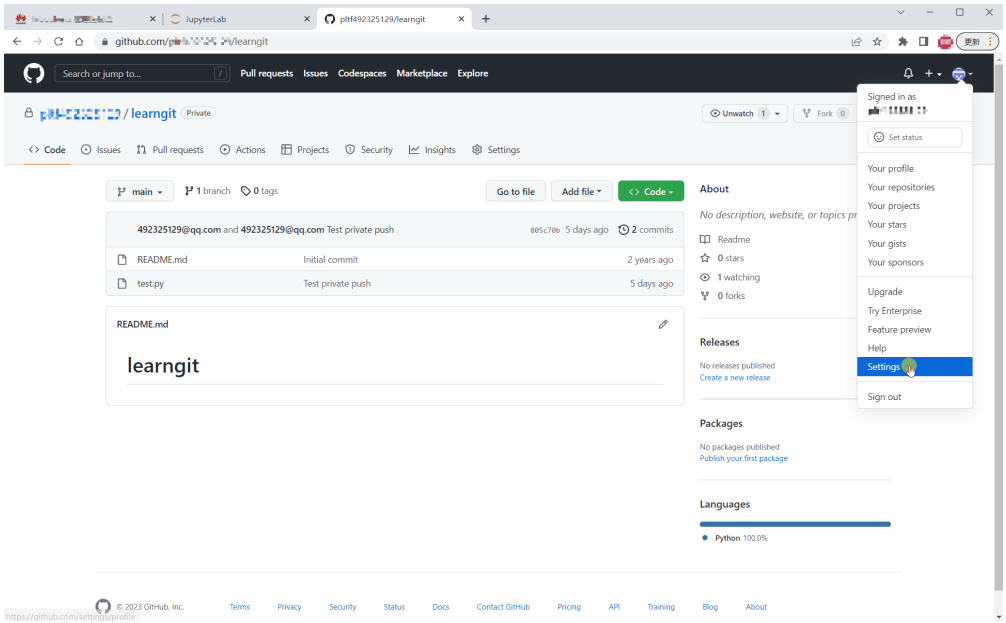


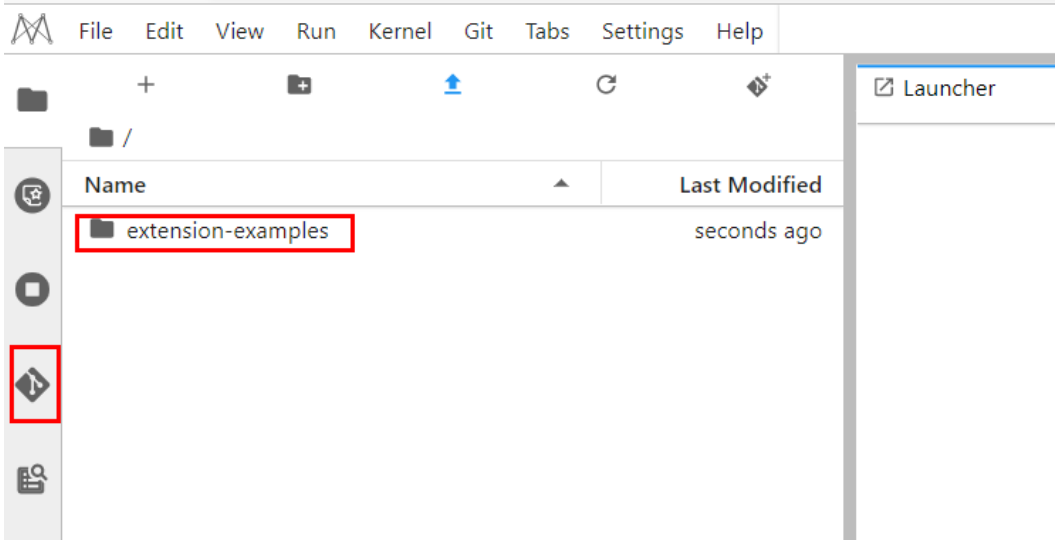
图 6-33 获取 Personal Access Token



查看代码库信息

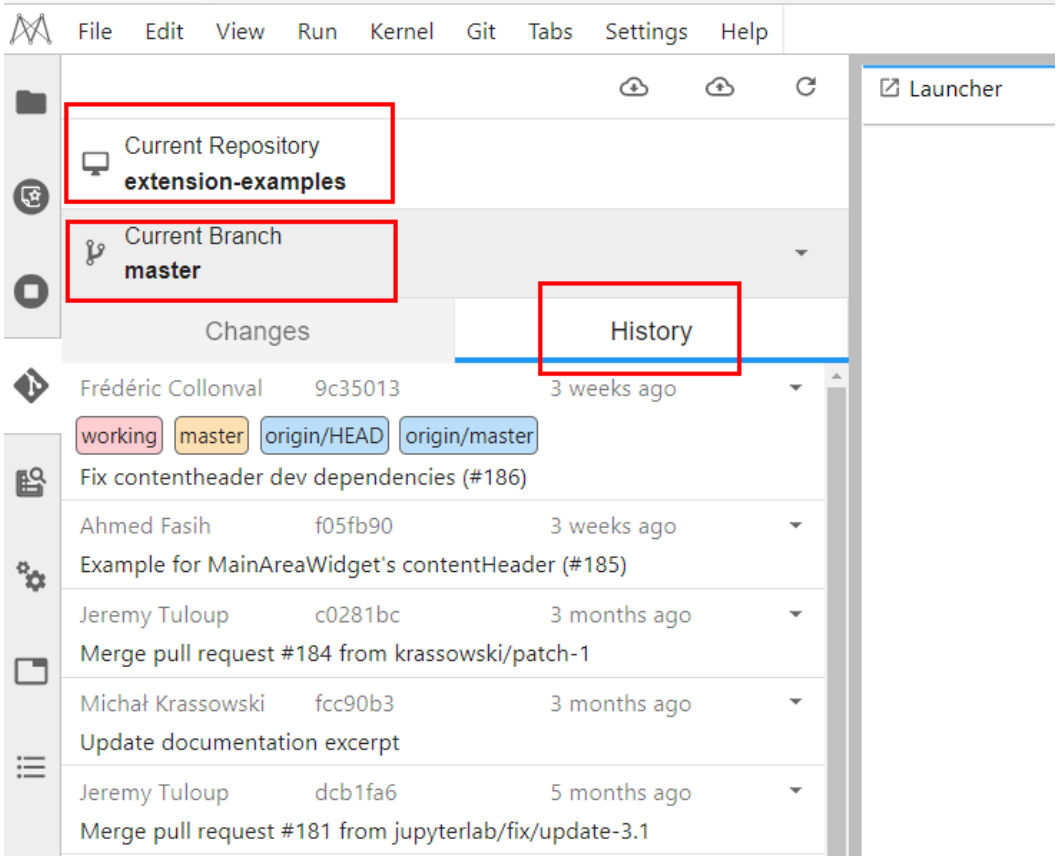
在Name下方列表中，选中您希望使用的文件夹，双击打开，然后单击左侧git插件图标进入此文件夹对应的代码库。

图 6-34 打开文件夹后打开 git 插件



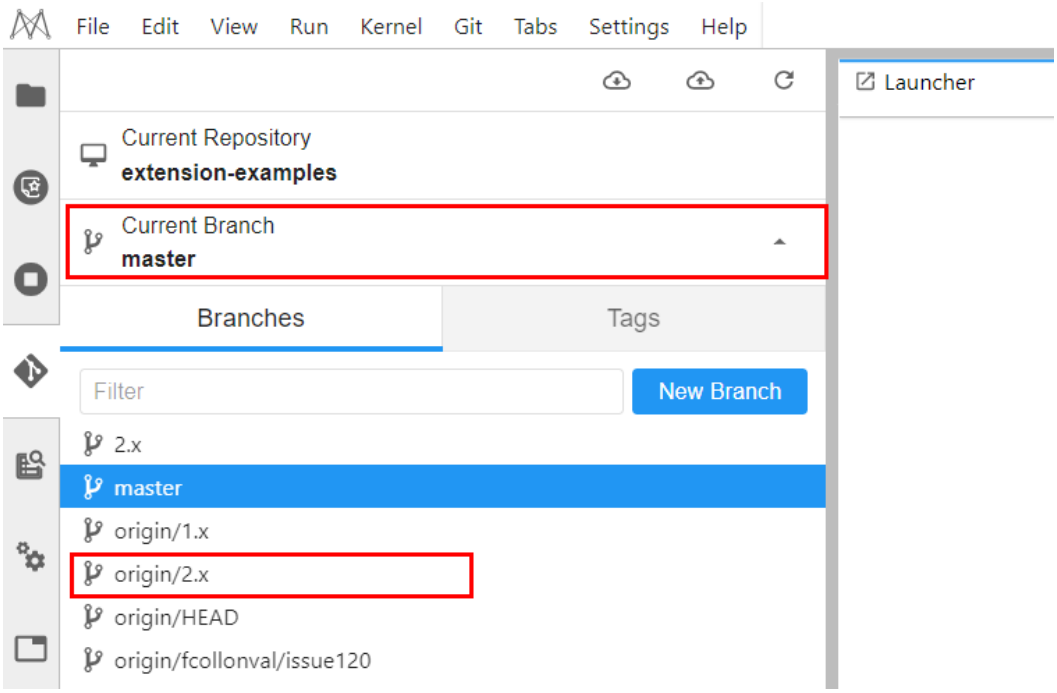
即可看到当前代码库的信息，如仓库名称、分支、历史提交记录等。

图 6-35 查看代码库信息



说明

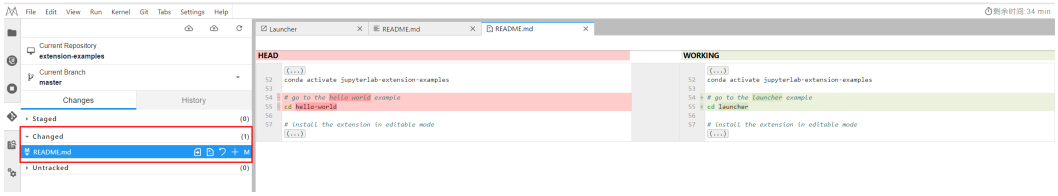
Git插件一般默认克隆master分支，如果要切换分支可单击Current Branch展开所有分支，单击相应分支名称可完成切换。



查看修改的内容

如果修改代码库中的某个文件，在“Changes”页签的“Changed”下可以看到修改的文件，并单击修改文件名称右侧的“Diff this file”，可以看到修改的内容。

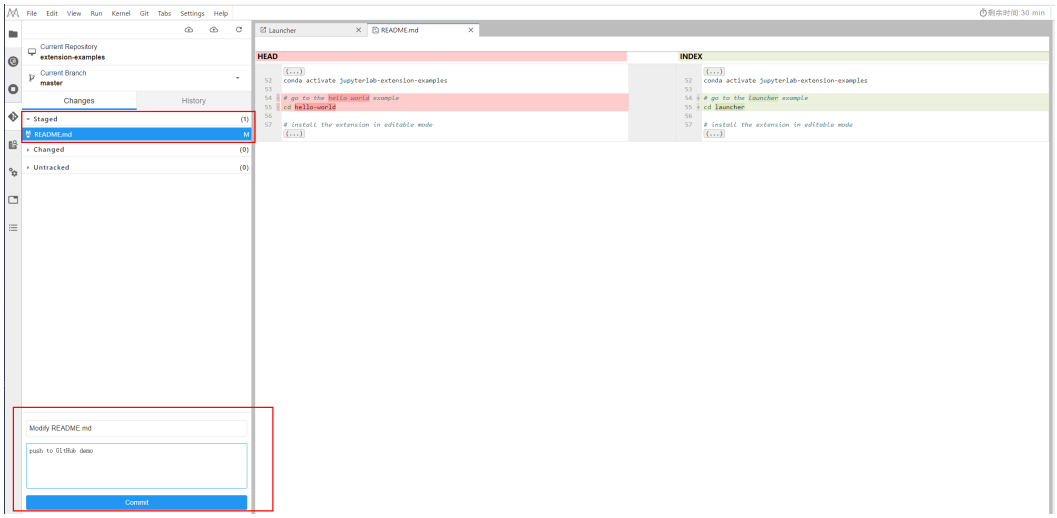
图 6-36 查看修改的内容



提交修改的内容

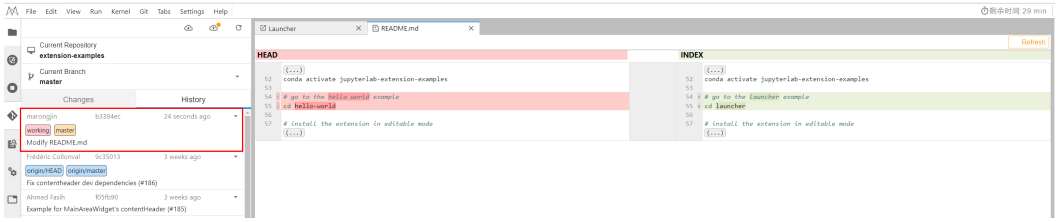
确认修改无误后，单击修改文件名称右侧的“Stage this change”，文件将进入Staged状态，相当于执行了git add命令。在左下方输入本次提交的Message，单击“Commit”，相当于执行了git commit命令。

图 6-37 提交修改内容



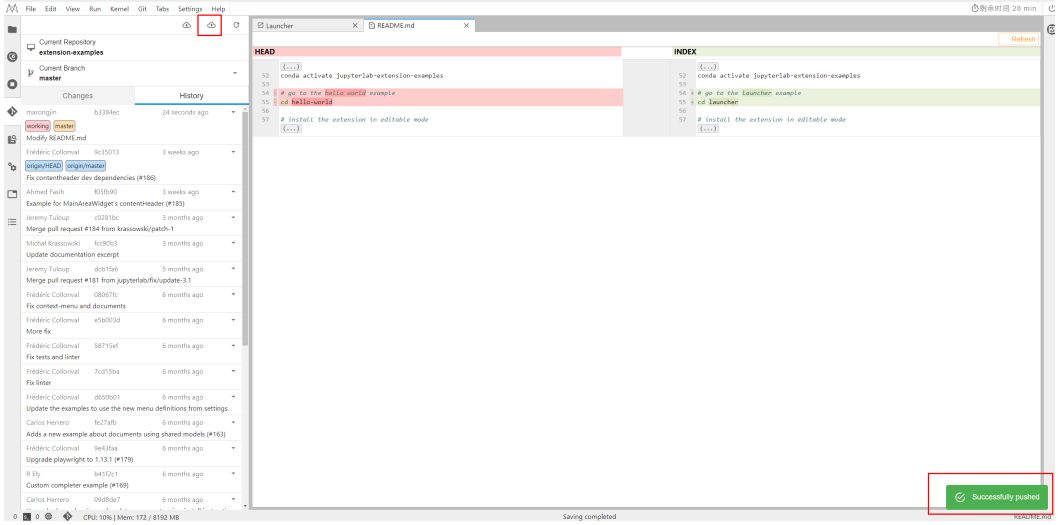
此时，可以在“History”页签下看到本地提交已成功。

图 6-38 查看是否提交成功



单击“push”按钮，相当于执行git push命令，即可提交代码到GitHub仓库中。提交成功后会提示“Successfully completed ”。如果OAuth鉴权的token过期，则此时再push会弹框让输入用户的token或者账户信息，按照提示输入即可。这里推荐使用Personal Access Token授权方式，如果出现密码失效报错请参考**git插件密码失效如何解决?**

图 6-39 提交代码至 GitHub 仓库



完成上述操作后，可以在JupyterLab的git插件页面的History页签，看到“origin/HEAD”和“orgin/master”已指向最新一次的提交。同时在GitHub对应仓库的commit记录中也可以查找到对应的信息。

6.3.4 在 JupyterLab 中创建定时任务

ModelArts Notebook支持创建定时任务。本文档介绍了如何创建定时任务、一键运行Notebook文件，从而提高工作效率。

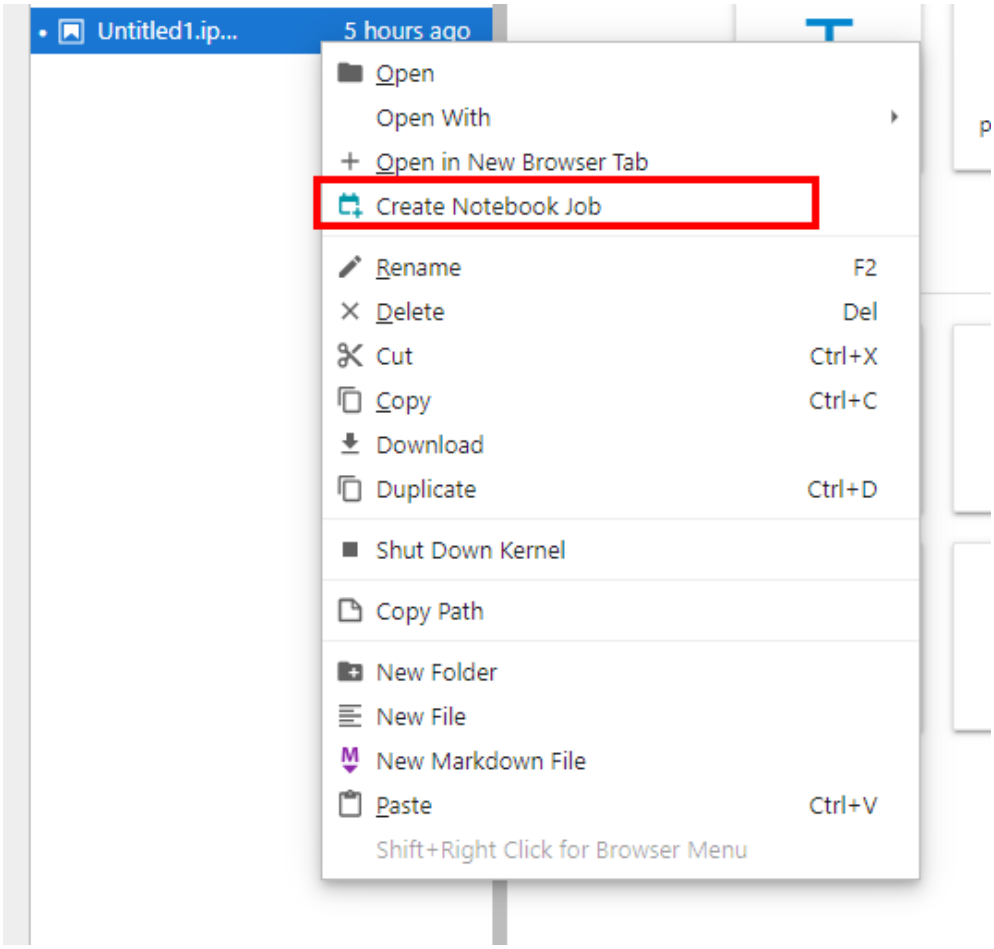
功能亮点

- **一键运行**：允许用户一键运行Notebook文件，无需逐个执行Cell。
- **定时任务调度**：允许用户设置定时执行代码块的时间和频率。支持秒、分钟、小时和每天/每周/月的时间设置。
- **支持参数化执行**：允许用户在运行时向Notebook传递参数，使得Notebook能根据不同需求调整行为。
- **任务管理界面**：提供用户友好的界面，便于查看、添加和删除定时任务。
- **任务执行记录**：记录每次执行任务的状态和输出，方便后续查看和调试。

操作步骤

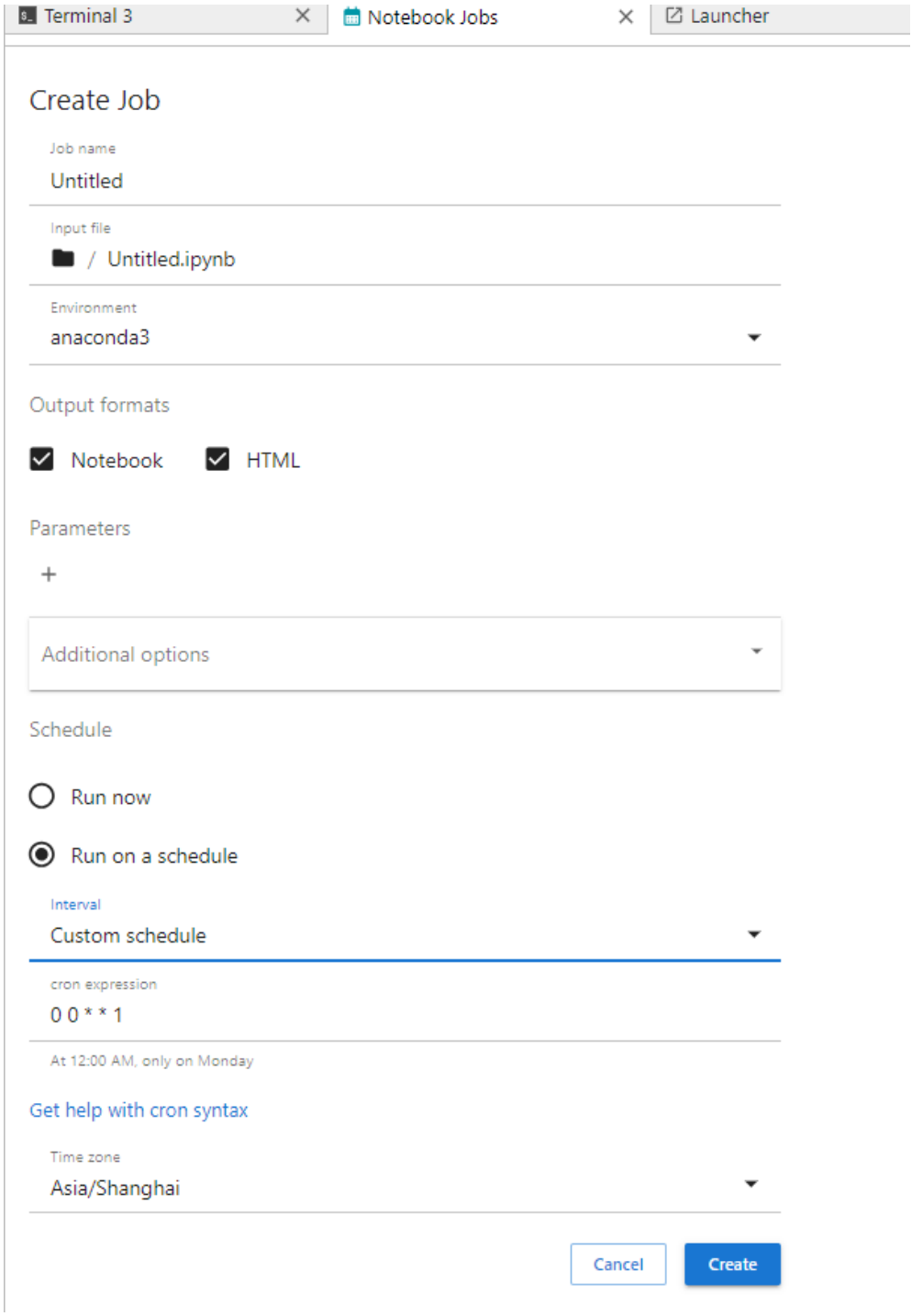
1. 打开ModelArts Notebook。
2. 选中Notebook文件（ipynb文件），创建定时任务。

图 6-40 打开 Notebook Jobs



3. 在Create Job界面，填写参数后单击“create”。

图 6-41 创建定时任务参数填写



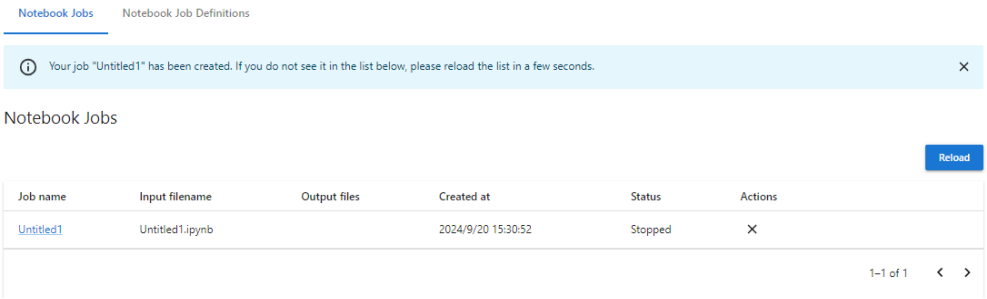
- Job name：定时任务名称。
- Environment：要运行该Notebook的python环境。
- Output formats：执行结果的输出文件类型。
- Parameter：单击+，手动设置运行Notebook的python变量。
- Schedule：任务执行策略，可以立即运行；也可以设置定时策略运行，支持cron表达式。

说明

- cron表达式需要使用linux系统下支持的格式，其他的cron表达式会报错。表达式可能会包含问号，要兼容linux的cron表达式，需将“?”替换为“*”。
- 设置定时任务后，修改文件名称以及文件内容，已经创建好的任务不受影响。

4. 立即运行后，在Notebook Jobs页签可以看到任务运行记录，右上角Reload刷新。

图 6-42 查看定时任务运行记录



5. 任务执行完成后会出现下载按钮，单击文件名称可以看到执行结果。

图 6-43 查看定时任务执行结果

6. 在Notebook Job Definitions页签可以看到所有的任务列表。单击任务名称，进入设置定时任务界面。可以启动，停止，删除定时任务；通过Edit Job Definition更新该定时任务，也可以查看该定时任务的运行历史。

图 6-44 在 Notebook Job Definitions 页签单击任务名称

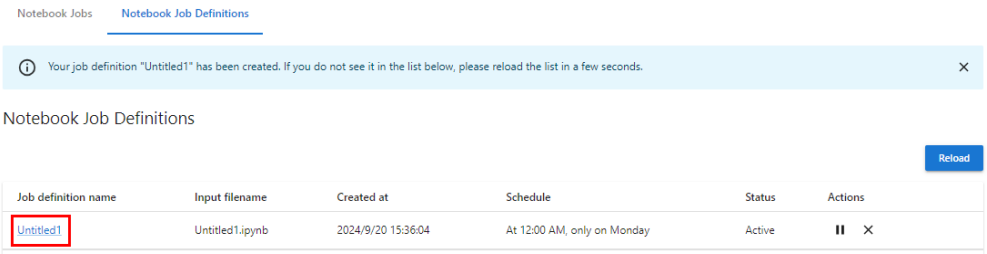
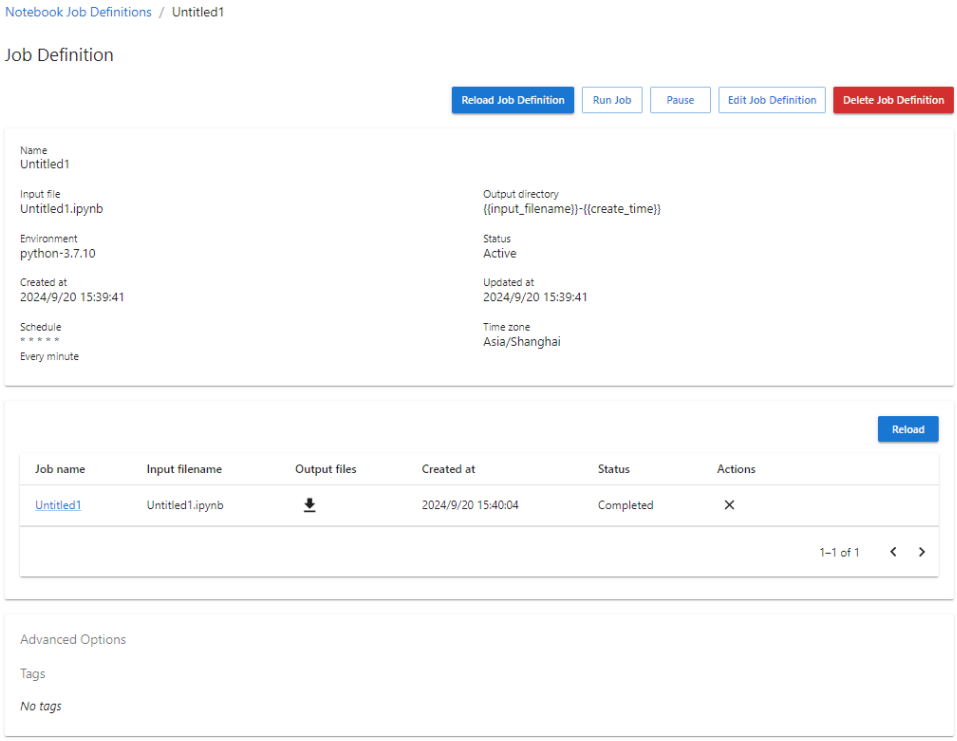


图 6-45 设置定时任务



6.3.5 上传文件至 JupyterLab

6.3.5.1 上传本地文件至 JupyterLab

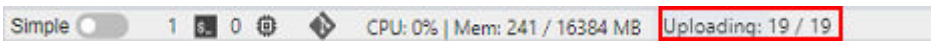
Notebook的JupyterLab中提供了多种方式上传文件。

上传文件要求

- 对于大小不超过100MB的文件直接上传，并展示文件大小、上传进度及速度等详细信息。
- 对于大小超过100MB不超过50GB的文件可以使用OBS中转，系统先将文件上传OBS（对象桶或并行文件系统），然后从OBS下载到Notebook，上传完成后，会将文件从OBS中删除。
- 50GB以上的文件上传通过调用ModelArts SDK或者Moxing完成。
- 对于Notebook当前目录下已经有同文件名称的文件，可以覆盖继续上传，也可以取消。
- 支持10个文件同时上传，其余文件显示“等待上传”。不支持上传文件夹，可以将文件夹压缩成压缩包上传至Notebook后， 在Terminal中解压压缩包。

  ```
  unzip xxx.zip  #在xxx.zip压缩包所在路径直接解压
  ```

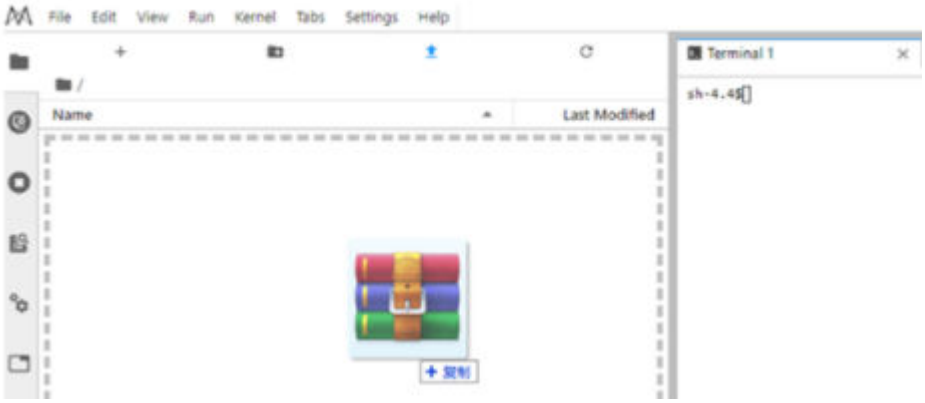
  解压命令的更多使用说明可以在主流搜索引擎中查找Linux解压命令操作。
- 多个文件同时上传时，JupyterLab窗口最下面会显示上传文件总数和已上传文件数。



上传文件入口

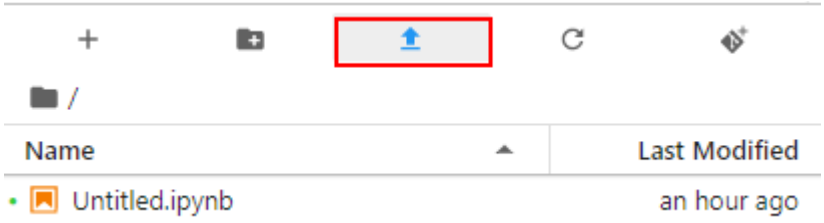
方式一：使用JupyterLab打开一个运行中的Notebook环境。

图 6-46 直接将文件拖拽到 JupyterLab 窗口左边的空白处上传。



方式二：单击窗口上方导航栏的 ModelArts Upload Files按钮，打开文件上传界面，拖拽或者选择本地文件上传。

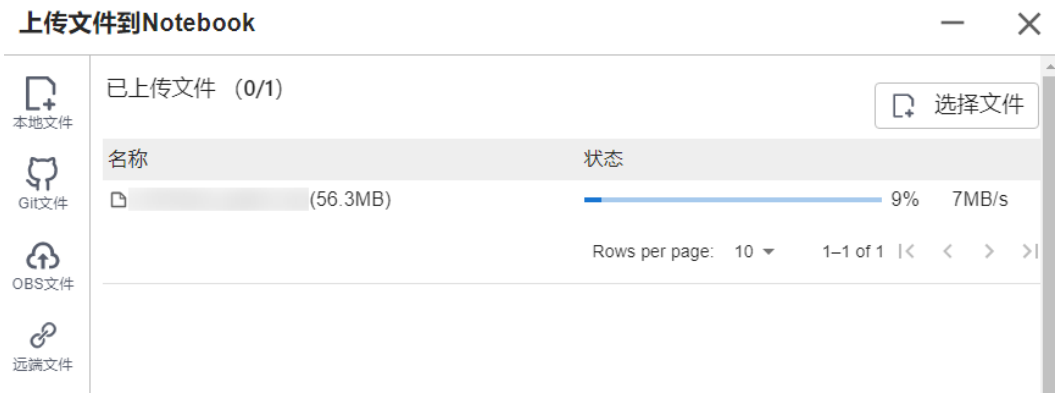
图 6-47 单击窗口上方导航栏的 ModelArts Upload Files 按钮



上传本地小文件（100MB 以内）至 JupyterLab

对于大小不超过100MB的文件直接上传，并展示文件大小、上传进度及速度等详细信息。

图 6-48 上传 100MB 以下小文件



文件上传完成后给出提示。

图 6-49 上传成功

上传本地大文件（100MB~50GB）至 JupyterLab

对于大小超过100MB不超过50GB的文件可以使用OBS中转，系统先将文件上传至OBS（对象桶或并行文件系统），然后从OBS下载到Notebook。下载完成后，ModelArts会将文件自动从OBS中删除。

例如，对于下面这种情况，可以通过“OBS中转”上传。

图 6-50 通过 OBS 中转上传大文件

如果使用OBS中转需要提供一个OBS中转路径，可以通过以下三种方式提供：

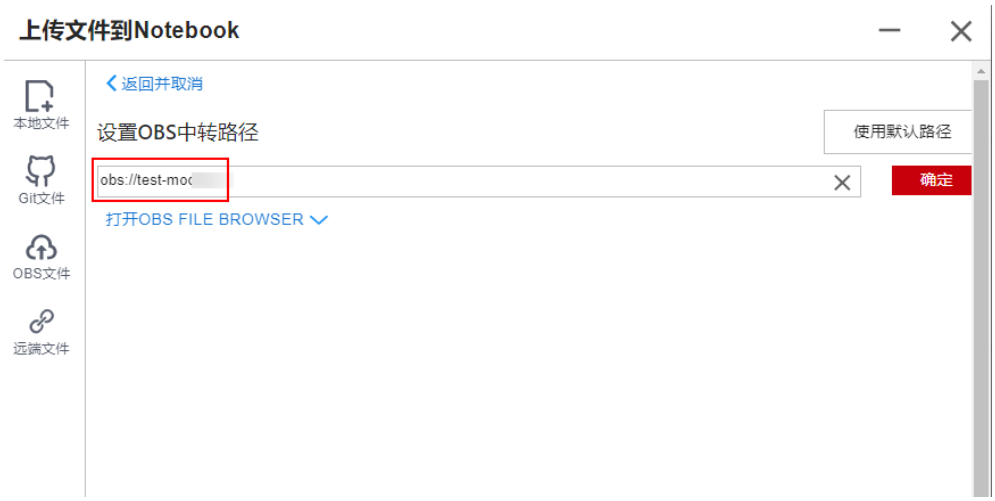
图 6-51 通过 OBS 中转路径上传

说明

仅第一次单击“OBS中转”需要提供OBS中转路径，以后默认使用该路径直接上传，可以通过上传文件窗口左下角的设置按钮 更新OBS中转路径。如图6-55所示。

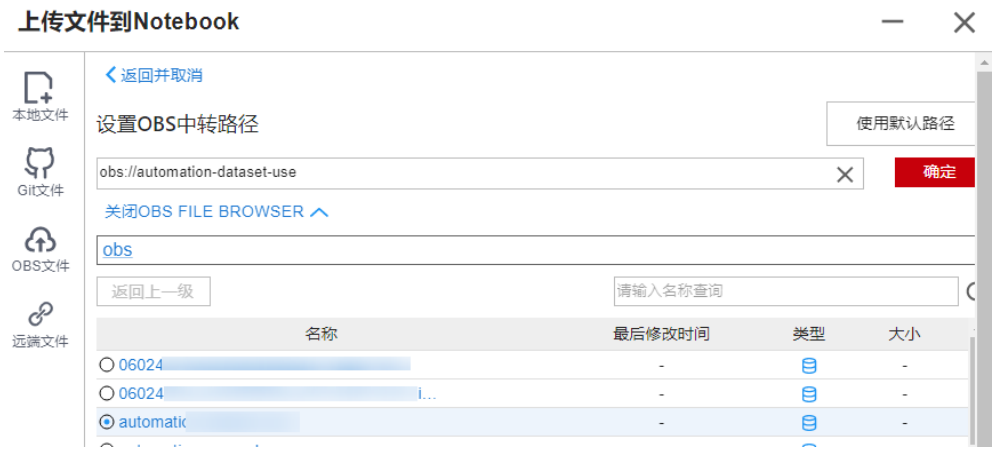
- 方式一：在输入框中直接输入有效的OBS中转路径，然后单击“确定”完成。

图 6-52 输入有效的 OBS 中转路径



- 方式二：打开OBS File Browser选择一个OBS中转路径，然后单击“确定”完成。

图 6-53 打开 OBS File Browser



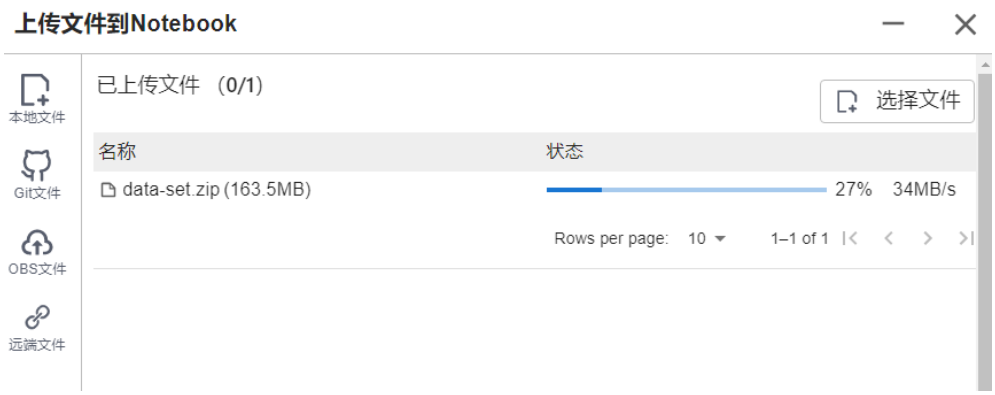
- 方式三：单击“使用默认路径”完成。

图 6-54 使用默认路径上传文件

图 6-55 设置本地文件 OBS 中转路径

完成OBS中转路径设置后，开始上传文件。

图 6-56 上传文件



解压缩文件包

将文件以压缩包形式上传至Notebook JupyterLab后， 可在Terminal中解压缩文件包。

```
unzip xxx.zip  #在xxx.zip压缩包所在路径直接解压
```

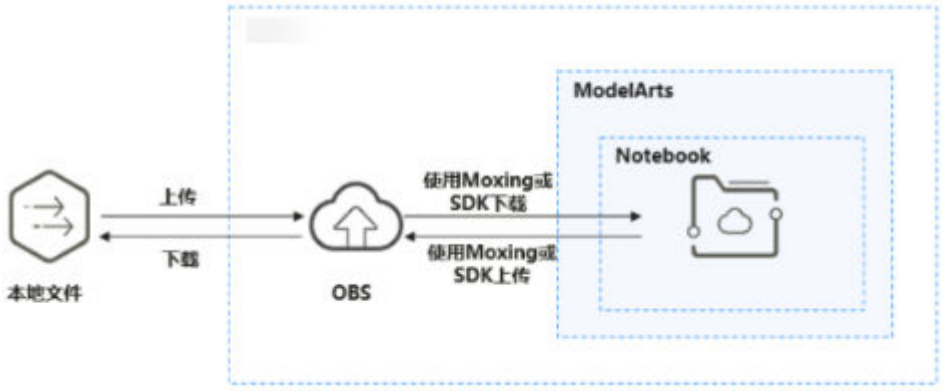
解压命令的更多使用说明可以在主流搜索引擎中查找Linux解压命令操作。

上传本地超大文件（50GB 以上）至 JupyterLab

不支持在Notebook的JupyterLab中直接上传大小超过50GB的文件。

50GB以上的文件需要先从本地上传到OBS中，再在Notebook中调用ModelArts的Moxing接口或者SDK接口读写OBS中的文件。

图 6-57 在 Notebook 中上传下载大文件



具体操作如下：

1. 从本地上传文件至OBS。具体操作请参见**上传文件至OBS桶**。
2. 将OBS中的文件下载到Notebook，可以通过在Notebook中运行代码的方式完成数据下载，具体方式有2种，ModelArts的SDK接口或者调用MoXing接口。
   - 方法一：使用ModelArts SDK接口将OBS中的文件下载到Notebook后进行操作。

     示例代码：

     ```
     from modelarts.session import Session
     session = Session()
     session.obs.copy("obs://bucket-name/obs_file.txt","/home/ma-user/work/")
     ```

   - 方法二：使用如下Moxing接口将OBS中的文件同步到Notebook后进行操作。

     ```
     import moxing as mox

     #下载一个OBS文件夹sub_dir_0，从OBS下载至Notebook
     mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/home/ma-user/work/sub_dir_0')
     #下载一个OBS文件obs_file.txt，从OBS下载至Notebook
     mox.file.copy('obs://bucket_name/obs_file.txt', '/home/ma-user/work/obs_file.txt')
     ```

     如果下载到Notebook中的是zip文件，在Terminal中执行下列命令，解压压缩包。

     ```
     unzip xxx.zip  #在xxx.zip压缩包所在路径直接解压
     ```

代码执行完成后，参考图6-58打开Terminal后执行**ls /home/ma-user/work**命令查看下载到Notebook中的文件。或者在JupyterLab左侧导航中显示下载的文件，如果没有显示，请刷新后查看，如图6-59所示。

图 6-58 打开 Terminal

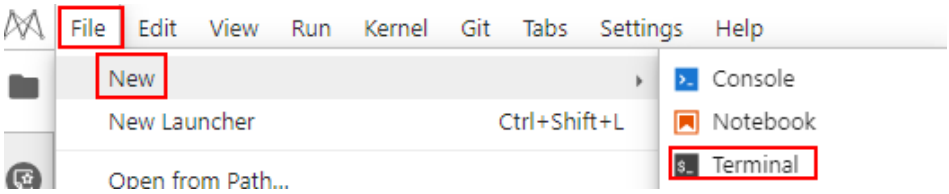
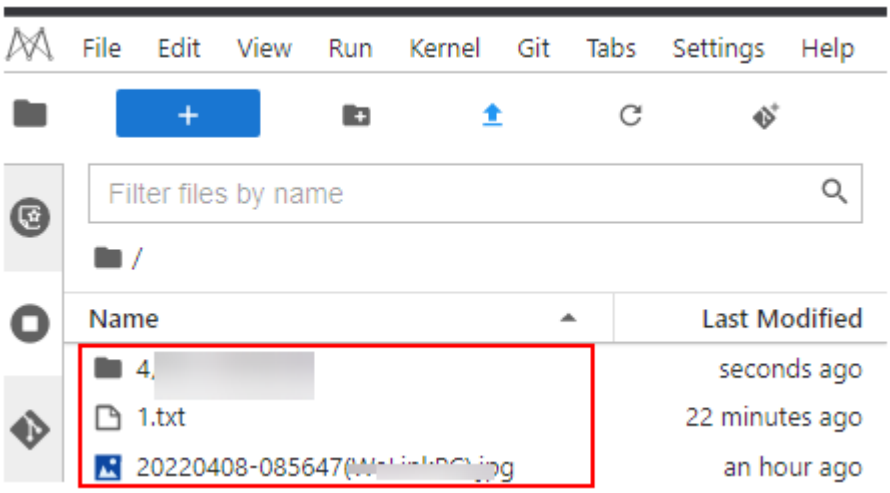


图 6-59 查看下载到 Notebook 中的文件



异常处理

通过OBS下载文件到Notebook中时，提示Permission denied。请依次排查：

- 请确保读取的OBS桶和Notebook处于同一站点区域。不支持跨站点访问OBS桶。
- 请确认操作Notebook的账号有权限读取OBS桶中的数据。

具体请参见**ModelArts中提示OBS路径错误**。

6.3.5.2 克隆 GitHub 开源仓库文件到 JupyterLab

在Notebook的JupyterLab中，支持从GitHub开源仓库Clone文件。

1. 通过JupyterLab打开一个运行中的Notebook。
2. 单击JupyterLab窗口上方导航栏的 ModelArts Upload Files按钮，打开文件上传窗口，选择左侧的 Git文件 进入GitHub开源仓库Clone界面。

图 6-60 上传文件图标

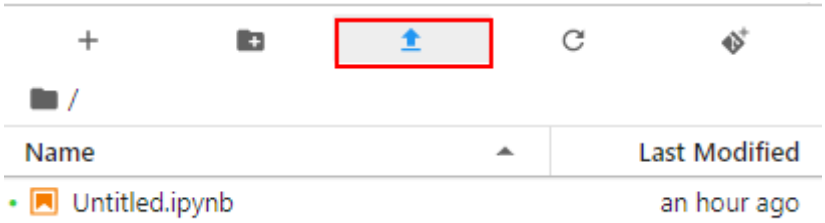
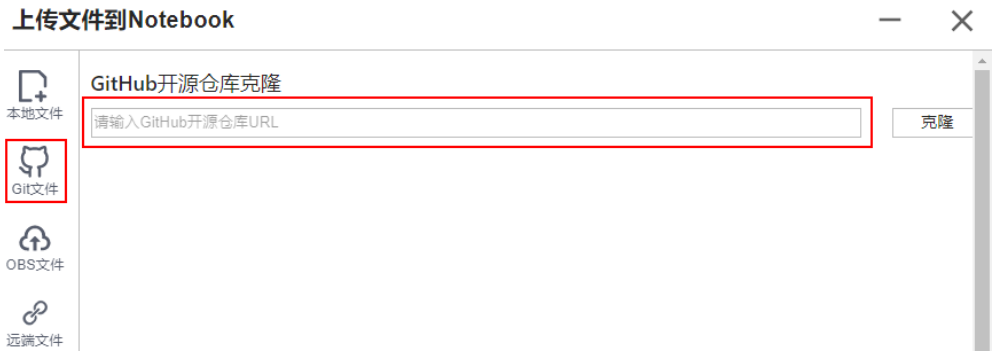


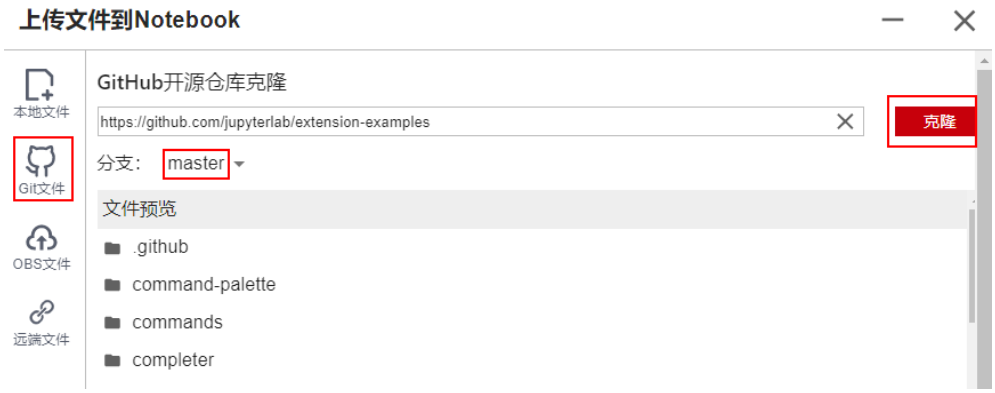
图 6-61 进入 GitHub 开源仓库 Clone 界面



3. 输入有效的GitHub开源仓库地址后会展示该仓库下的文件及文件夹，说明用户输入了有效的仓库地址，同时给出该仓库下所有的分支供选择，选择完成后单击“克隆”开始Clone仓库。

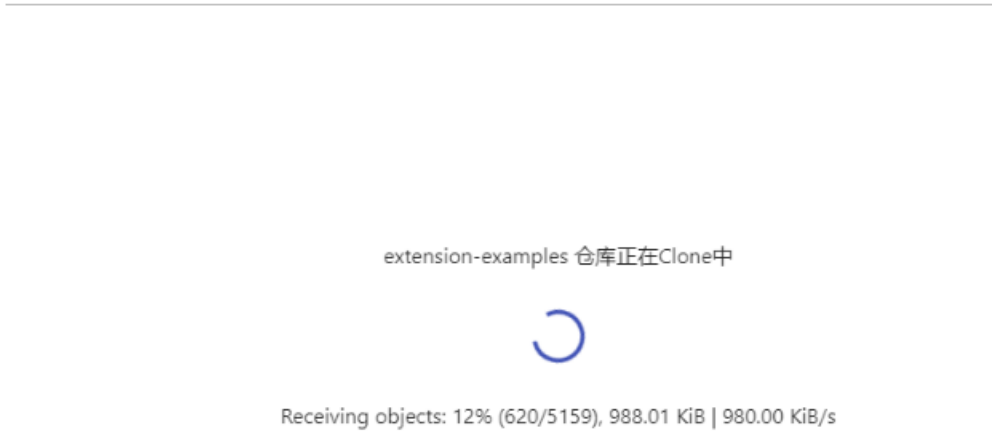
   GitHub开源仓库地址：https://github.com/jupyterlab/extension-examples

图 6-62 输入有效的 GitHub 开源仓库地址



4. Clone仓库的过程中会将进度展示出来。

图 6-63 Clone 仓库的过程



5. Clone仓库成功。

异常处理

- Clone仓库失败。可能是网络原因问题。可以在JupyterLab的Terminal中通过执行git clone https://github.com/jupyterlab/extension-examples.git测试网络连通情况。
- 如果克隆时遇到Notebook当前目录下已有该仓库，系统给出提示仓库名称重复，此时可以单击“覆盖”继续克隆仓库，也可以单击 取消。

6.3.5.3 上传 OBS 文件到 JupyterLab

在Notebook的JupyterLab中，支持将OBS中的文件下载到Notebook。注意：文件大小不能超过10GB，否则会上传失败。

1. 通过JupyterLab打开一个运行中的Notebook。
2. 单击JupyterLab窗口上方导航栏的 ModelArts Upload Files按钮，打开文件上传窗口，选择左侧的 进入OBS文件上传界面。

图 6-64 上传文件图标

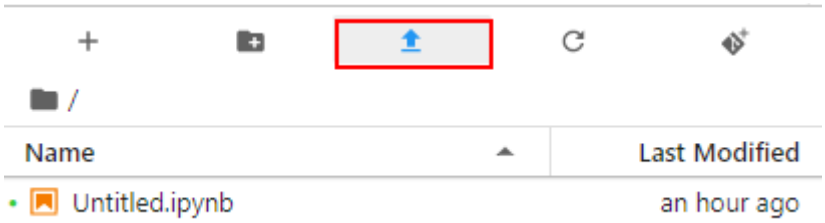
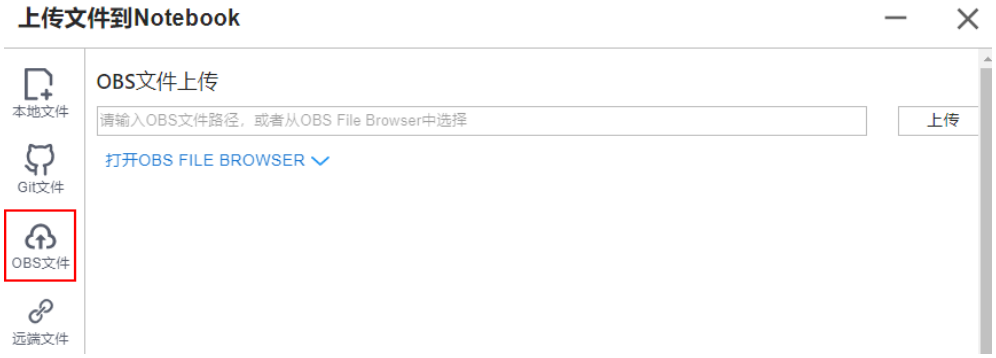
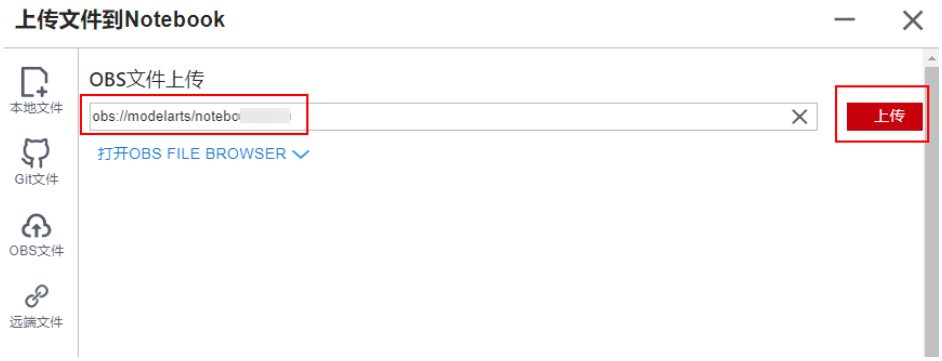


图 6-65 OBS 文件上传界面



3. 需要提供OBS文件路径，可以通过以下两种方式提供：
   - 方式一：在输入框中直接输入有效的OBS文件路径，然后单击“上传”开始传文件。

     图 6-66 输入有效的 OBS 文件路径

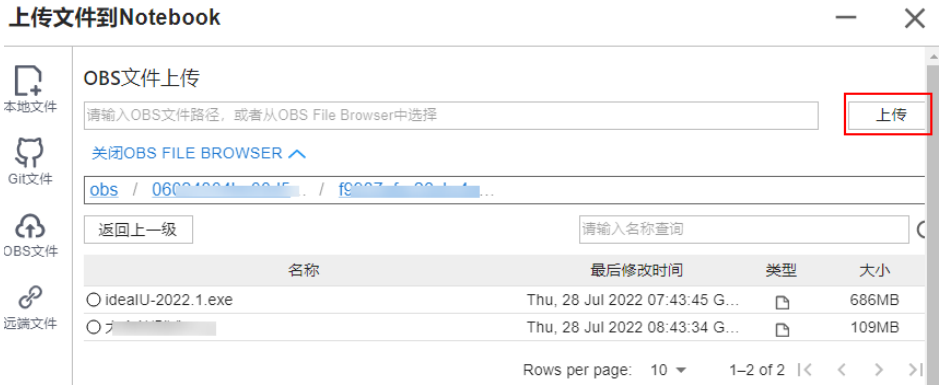
     

     说明

     此处输入的是具体的OBS文件路径，不是文件夹的路径，否则会导致上传失败。

   - 方式二：打开OBS File Browser选择OBS文件路径，然后单击“上传”，开始上传文件。

     图 6-67 上传 OBS 文件

异常处理

提示文件上传失败，有以下三种常见场景。

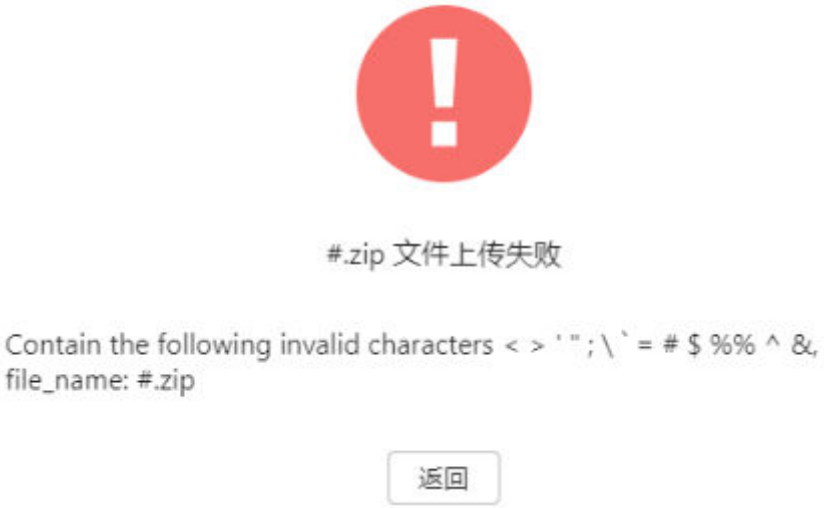
- **异常场景1**

  可能原因：

  - OBS路径没有设置为具体的文件路径，设置成了文件夹。
  - OBS中的文件设置了加密。请前往OBS控制台查看，确保该文件未加密。
  - OBS桶和Notebook不在同一个区域。请确保读取的OBS桶和Notebook处于同一站点区域，不支持跨站点访问OBS桶。具体操作请参见**如何查看OBS桶与ModelArts是否在同一区域**。
  - 没有该OBS桶的访问权限。请确认操作Notebook的账号有权限读取OBS桶中的数据。具体操作请参见**检查您的账号是否有该OBS桶的访问权限**。
  - OBS文件被删除。请确认待上传的OBS文件是否存在。

- **异常场景2**

  图 6-68 文件上传失败

  可能原因：

  文件名包含<>'";\`=#$%^&等特殊字符。

- **异常场景3**

图 6-69 文件上传失败

可能原因：
文件大小超过50GB导致上传失败。

6.3.5.4 上传远端文件至 JupyterLab

在Notebook的JupyterLab中，支持通过远端文件地址下载文件。

要求：远端文件的URL粘贴在浏览器的输入框中时，可以直接下载该文件。

1. 通过JupyterLab打开一个运行中的Notebook。
2. 单击JupyterLab窗口上方导航栏的 ModelArts Upload Files按钮，打开文件上传窗口，选择左侧的 进入远端文件上传界面。

图 6-70 上传文件图标

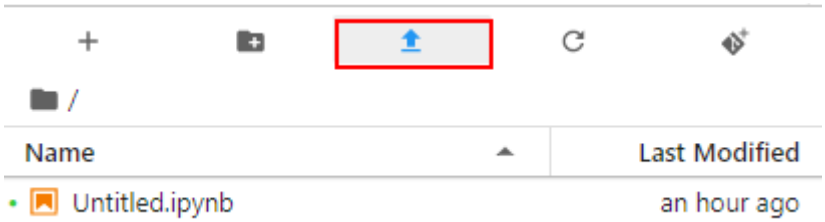
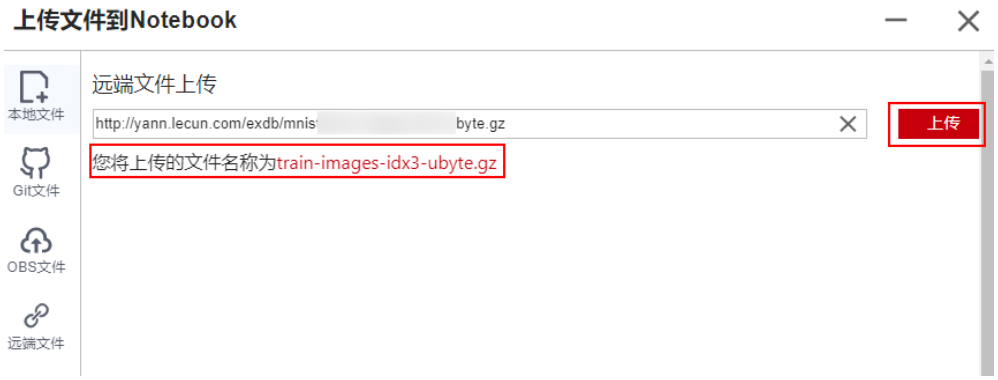


图 6-71 进入远端文件上传界面

3. 输入有效的远端文件URL后，系统会自动识别上传文件名称，单击“上传”，开始上传文件。

图 6-72 输入有效的远端文件 URL



异常处理

远端文件上传失败。可能是网络原因。请先在浏览器中输入该远端文件的URL地址，测试该文件是否能下载。

6.3.6 下载 JupyterLab 文件到本地

在JupyterLab中开发的文件，可以下载至本地。关于如何上传文件至JupyterLab，请参见**上传文件至JupyterLab**。

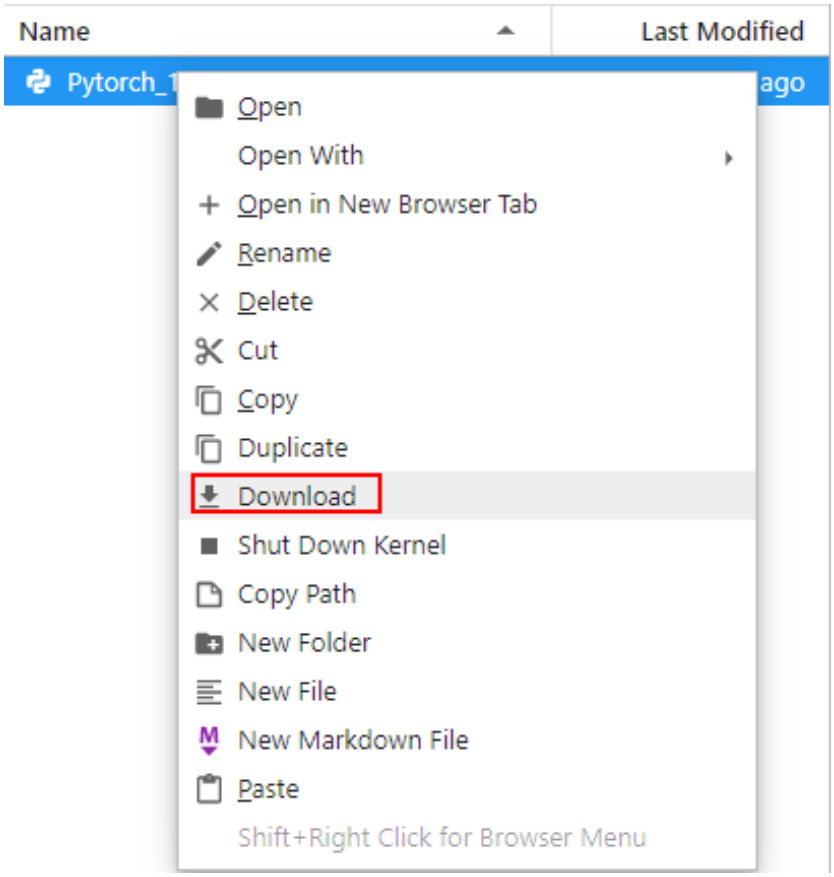
- 不大于100MB的文件，可以直接从JupyterLab中下载到本地，具体操作请参见**从JupyterLab中下载不大于100MB的文件至本地**。
- 大于100MB的文件，需要先从JupyterLab上传到OBS，再通过OBS下载到本地，具体操作请参见**从JupyterLab中下载大于100MB的文件到本地**。

从 JupyterLab 中下载不大于 100MB 的文件至本地

在JupyterLab文件列表中，选择需要下载的文件，单击右键，在操作菜单中选择“Download”下载至本地。

下载的目的路径，为您本地浏览器设置的下载目录。

图 6-73 下载文件



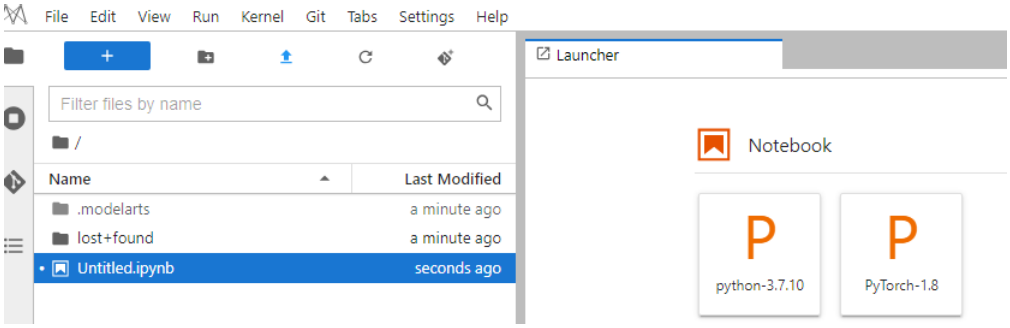
从 JupyterLab 中下载大于 100MB 的文件到本地

大于100MB的文件需要先从Notebook中上传到OBS，再从OBS下载到本地，具体操作如下：

1. 打开Python运行环境。

   以下图为例，在Launcher页面的Notebook区域，单击“python-3.7.10”。请您以实际环境为准。

   图 6-74 打开 Python 运行环境

2. 使用MoXing将目标文件从Notebook上传到OBS中。

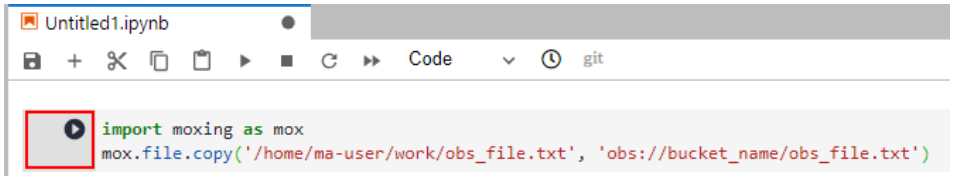
   上传txt、压缩后文件夹的Python示例代码如下。代码中的“/home/ma-user/work/xxx”为文件在Notebook中的存储路径，“obs://bucket_name/xxx”为该文件上传到OBS的存储路径，请您按照实际路径进行修改。

- 上传一个obs_file.txt文件，从Notebook上传至OBS。

  在命令行输入以下代码，按需修改路径后，单击运行代码。在OBS控制台的桶中，可以看到txt对象存在，表明上传成功。

  ```
  import moxing as mox
  mox.file.copy('/home/ma-user/work/obs_file.txt', 'obs://bucket_name/obs_file.txt')
  ```

  图 6-75 运行代码示例

- 上传一个压缩后的sub_dir_0文件夹，从Notebook上传至OBS。

  在命令行输入以下代码，按需修改路径后，单击运行代码。在OBS控制台的桶中，可以看到文件夹对象存在，表明上传成功。

  ```
  import moxing as mox
  mox.file.copy_parallel('/home/ma-user/work/sub_dir_0', 'obs://bucket_name/sub_dir_0')
  ```

3. 使用OBS或ModelArts SDK将OBS中的文件下载到本地。
   - 方式一：使用OBS进行下载

     在OBS中，可以将样例中的“obs_file.txt”下载到本地。如果您的数据较多，推荐OBS Browser+下载数据或文件夹。使用OBS下载文件的操作指导，请参见**下载文件**。
   - 方式二：使用ModelArts SDK进行下载
     i. 在您的本地环境**下载并安装ModelArts SDK**。
     ii. 完成ModelArts SDK的**Session鉴权**。
     iii. 将OBS中的文件下载到本地，详情请参见**从OBS下载数据**。示例代码如下：

     ```
     from modelarts.session import Session

     # 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
     # 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
     __AK = os.environ["HUAWEICLOUD_SDK_AK"]
     __SK = os.environ["HUAWEICLOUD_SDK_SK"]
     # 如果进行了加密还需要进行解密操作
     session = Session(access_key=__AK,secret_key=__SK, project_id='***', region_name='***')

     session.download_data(bucket_path="/bucket_name/obs_file.txt",path="/home/user/obs_file.txt")
     ```

6.3.7 在 JupyterLab 中使用 MindInsight 可视化作业

ModelArts支持在开发环境中开启MindInsight可视化工具。在开发环境中通过小数据集训练调试算法，主要目的是验证算法收敛性、检查是否有训练过程中的问题，方便用户调测。

MindInsight能可视化展现出训练过程中的标量、图像、计算图以及模型超参等信息，同时提供训练看板、模型溯源、数据溯源、性能调试等功能，帮助您更高效地训练调试模型。MindInsight当前支持基于MindSpore引擎的训练作业。MindInsight相关概念请参考**MindSpore官网**。

MindSpore支持将数据信息保存到Summary日志文件中，并通过可视化界面MindInsight进行展示。

前提条件

使用MindSpore引擎编写训练脚本时，为了保证训练结果中输出Summary文件，您需要在脚本中添加收集Summary相关代码。

将数据记录到Summary日志文件中的具体方式请参考**收集Summary数据**。

注意事项

- 在开发环境跑训练作业，在开发环境使用MindInsight，要求先启动MindInsight，后启动训练进程。
- 仅支持单机单卡训练。
- 运行中的可视化作业不单独计费，当停止Notebook实例时，计费停止。
- Summary文件如果存放在OBS中，由OBS单独收费。任务完成后请及时停止Notebook实例，清理OBS数据，避免产生不必要的费用。

在开发环境中创建 MindInsight 可视化作业流程

Step1 创建开发环境并在线打开

Step2 上传Summary数据

Step3 启动MindInsight

Step4 查看训练看板中的可视化数据

Step1 创建开发环境并在线打开

在ModelArts控制台，进入“开发空间> Notebook”页面，创建MindSpore引擎的开发环境实例。创建成功后，单击开发环境实例操作栏右侧的“打开 ”，在线打开运行中的开发环境。

Step2 上传 Summary 数据

在开发环境中使用MindInsight可视化功能，需要用到Summary数据。

Summary数据可以直接传到开发环境的这个路径下/home/ma-user/work/，也可以放到OBS并行文件系统中。

- Summary数据上传到Notebook路径/home/ma-user/work/下的方式，请参见**上传数据至Notebook**。
- Summary数据如果是通过OBS并行文件系统挂载到Notebook中，请将模型训练时产生的Summary文件先上传到OBS并行文件系统，并确保OBS并行文件系统与ModelArts在同一区域。在Notebook中启动MindInsight时，Notebook会自动从挂载的OBS并行文件系统目录中读取Summary数据。

Step3 启动 MindInsight

在开发环境的JupyterLab中打开MindInsight。

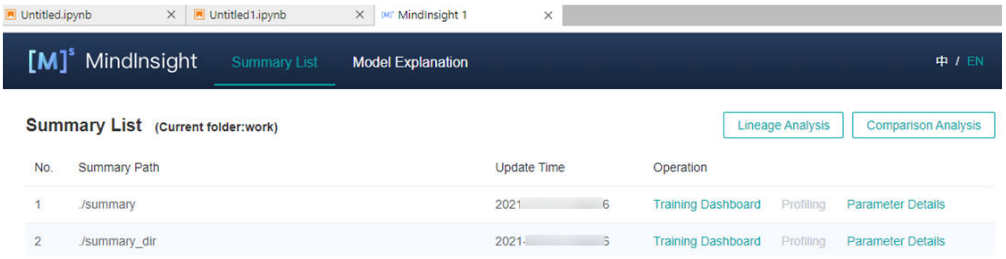
单击 

，直接进入MindInsight可视化界面。

默认读取路径/home/ma-user/work/

当存在两个以及以上工程的log时，界面如下。通过Runs下选择查看相对应的log。

图 6-76 MindInsight 界面（2）



Step4 查看训练看板中的可视化数据

训练看板是MindInsight的可视化组件的重要组成部分，而训练看板的标签包含：标量可视化、参数分布图可视化、计算图可视化、数据图可视化、图像可视化和张量可视化等。

更多功能介绍请参见MindSpore官网资料：**查看训练看板中可视的数据**。

关闭 MindInsight

关闭MindInsight方式如下单击下方  按钮进入MindInsight实例管理界面，该界面记录了所有启动的MindInsight实例，单击对应实例后面的SHUT DOWN即可停止该实例。

图 6-77 单击 SHUT DOWN 停止实例

6.3.8 在 JupyterLab 中使用 TensorBoard 可视化作业

ModelArts支持在开发环境中开启TensorBoard可视化工具。TensorBoard是TensorFlow的可视化工具包，提供机器学习实验所需的可视化功能和工具。

TensorBoard是一个可视化工具，能够有效地展示TensorFlow在运行过程中的计算图、各种指标随着时间的变化趋势以及训练中使用到的数据信息。TensorBoard相关概念请参考**TensorBoard官网**。

TensorBoard可视化训练作业，当前仅支持基于TensorFlow、PyTorch版本镜像，CPU/GPU规格的资源类型。请根据实际局点支持的镜像和资源规格选择使用。

前提条件

为了保证训练结果中输出Summary文件，在编写训练脚本时，您需要在脚本中添加收集Summary相关代码。

TensorFlow引擎的训练脚本中添加Summary代码，具体方式请参见**TensorFlow官方网站**。

注意事项

- 运行中的可视化作业不单独计费，当停止Notebook实例时，计费停止。
- Summary文件数据如果存放在OBS中，由OBS单独收费。任务完成后请及时停止Notebook实例，清理OBS数据，避免产生不必要的费用。

在开发环境中创建 TensorBoard 可视化作业流程

Step1 创建开发环境并在线打开

Step2 上传Summary数据

Step3 启动TensorBoard

Step4 查看训练看板中的可视化数据

Step1 创建开发环境并在线打开

在ModelArts控制台，进入“开发空间 > Notebook”页面，创建TensorFlow或者PyTorch镜像的开发环境实例。创建成功后，单击开发环境实例操作栏右侧的“打开”，在线打开运行中的开发环境。

TensorBoard可视化训练作业，当前仅支持基于TensorFlow、PyTorch镜像，CPU/GPU规格的资源类型。请根据实际局点支持的镜像和资源规格选择使用。

Step2 上传 Summary 数据

在开发环境中使用TensorBoard可视化功能，需要用到Summary数据。

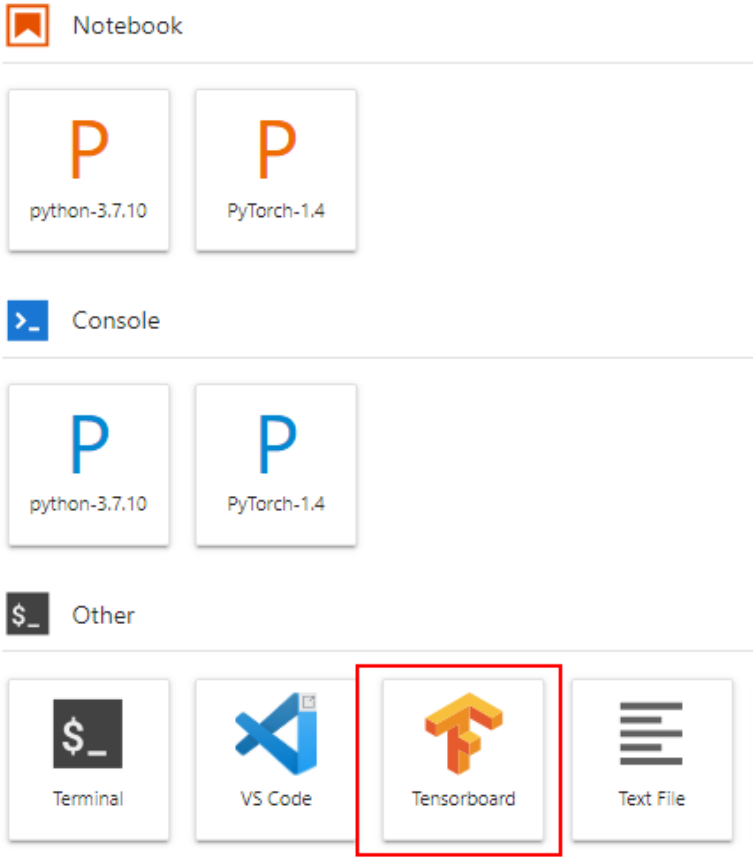
Summary数据可以直接传到开发环境的这个路径下/home/ma-user/work/，也可以放到OBS并行文件系统中。

- Summary数据上传到Notebook路径/home/ma-user/work/下的方式，请参见**上传本地文件至JupyterLab**。
- Summary数据如果是通过OBS并行文件系统挂载到Notebook中，请将模型训练时产生的Summary文件先上传到OBS并行文件系统，并确保OBS并行文件系统与ModelArts在同一区域。在Notebook中启动TensorBoard时，Notebook会自动从挂载的OBS并行文件系统目录中读取Summary数据。

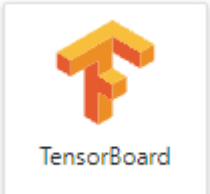
Step3 启动 TensorBoard

在开发环境的JupyterLab中打开TensorBoard。

图 6-78 JupyterLab 中打开 TensorBoard

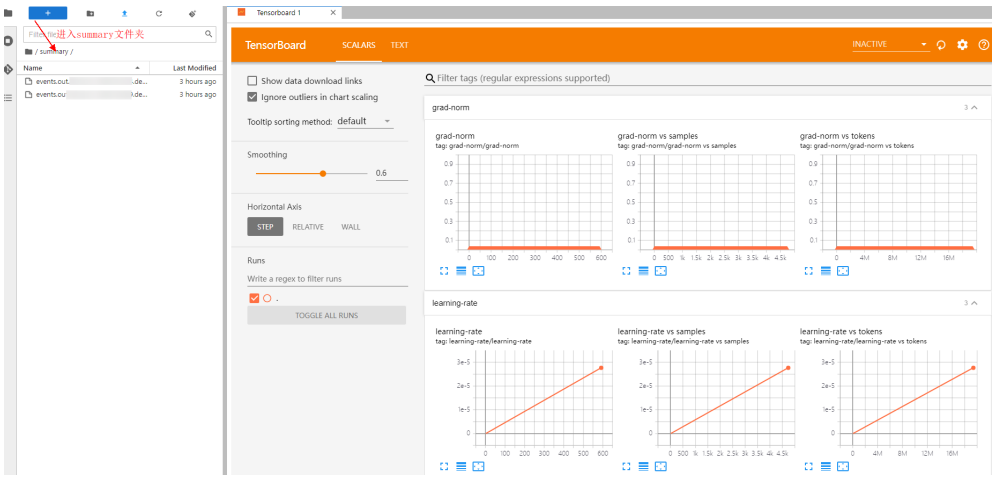


1. 在JupyterLab左侧导航创建名为“summary”的文件夹，将数据上传到“/home/ma-user/work/summary”路径。注：文件夹命名只能为summary否则无法使用。



2. 进入“summary”文件夹，单击方式1 ，直接进入TensorBoard可视化界面。如图6-79所示。

图 6-79 TensorBoard 界面（1）



Step4 查看训练看板中的可视化数据

训练看板是TensorBoard的可视化组件的重要组成部分，而训练看板的标签包含：标量可视化、图像可视化和计算图可视化等。

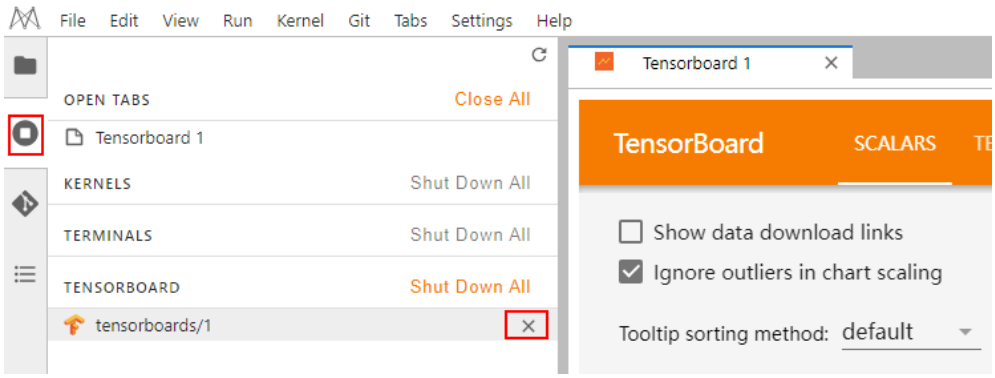
更多功能介绍请参见**TensorBoard官网资料**。

关闭 TensorBoard

关闭TensorBoard方式如下：

单击下图所示的，进入TensorBoard实例管理界面，该界面记录了所有启动的TensorBoard实例，单击对应实例后面的SHUT DOWN即可停止该实例。

图 6-80 单击 SHUT DOWN 停该实例



6.4 通过 PyCharm 远程使用 Notebook 实例

6.4.1 使用 PyCharm Toolkit 插件连接 Notebook

由于AI开发者会使用PyCharm工具开发算法或模型，为方便快速将本地代码提交到ModelArts的训练环境，ModelArts提供了一个PyCharm插件工具PyCharm ToolKit，

协助用户完成SSH远程连接Notebook、代码上传、提交训练作业、将训练日志获取到本地展示等，用户只需要专注于本地的代码开发即可。

本章节介绍了使用PyCharm Toolkit如何连接Notebook。

使用限制

- 当前仅支持2019.2-2023.2之间（包含2019.2和2023.2）版本，包括社区版和专业版。
- 使用PyCharm ToolKit远程连接Notebook开发环境，仅限PyCharm专业版。
- 使用PyCharm ToolKit提交训练作业，社区版和专业版都支持，PyCharm ToolKit latest版本仅限提交新版训练作业。
- PyCharm ToolKit工具仅支持Windows版本的PyCharm。

表 6-11 ToolKit（latest）功能列表

支持的功能	说明	对应操作指导
SSH远程连接	支持SSH远程连接ModelArts的Notebook开发环境。	**配置PyCharm ToolKit远程连接Notebook**
训练模型	支持将本地开发的代码，快速提交至ModelArts并自动创建**新版**训练作业，在训练作业运行期间获取训练日志并展示到本地。	**使用PyCharm ToolKit创建并调试训练作业**
OBS上传下载	上传本地文件或文件夹至OBS，从OBS下载文件或文件夹到本地。	**使用PyCharm上传数据至Notebook**

前提条件

本地已安装2019.2-2023.2之间（包含2019.2和2023.2）版本的PyCharm专业版。SSH远程开发功能只限PyCharm专业版。单击**PyCharm工具下载地址**下载并完成安装。

Step1 下载并安装 PyCharm ToolKit

在PyCharm中选择“File > Settings > Plugins”，在Marketplace里搜索“ModelArts”，单击“Install”即可完成安装。

Step2 创建 Notebook 实例

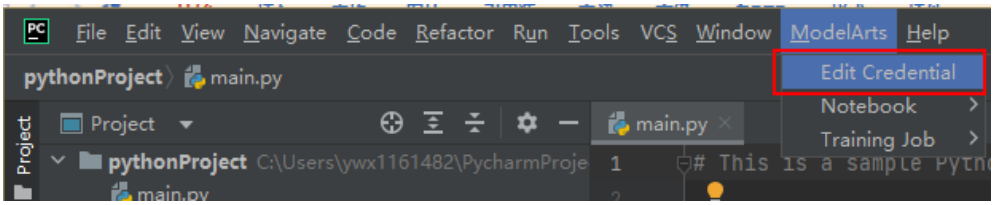
创建一个Notebook实例，并开启远程SSH开发，配置远程访问IP白名单。该实例状态必须处于“运行中”，具体参见**创建Notebook实例**章节。

Step3 登录插件

使用访问密钥完成登录认证操作如下：

1. 打开已安装ToolKit工具的PyCharm，在菜单栏中选择“ModelArts > Edit Credential”。

图 6-81 Edit Credential

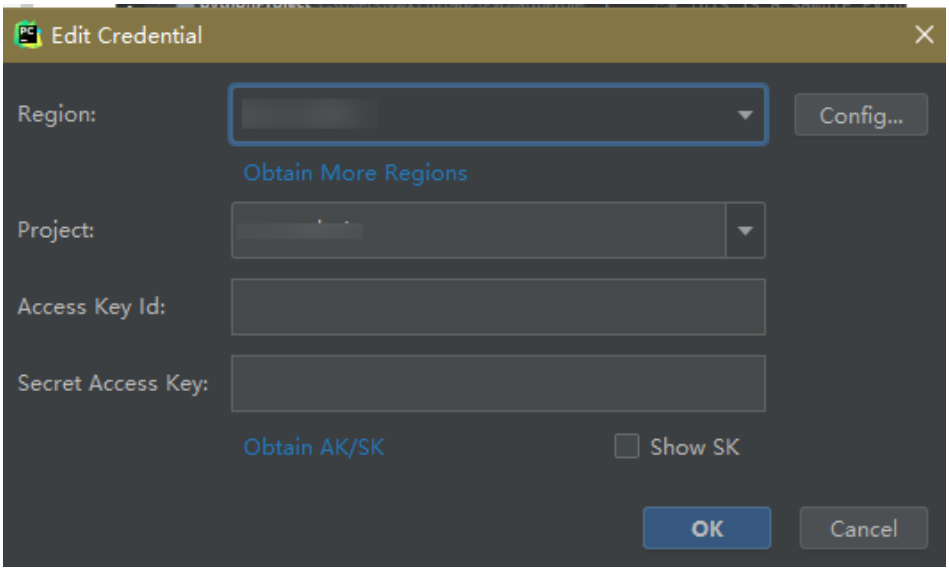


说明

如果菜单栏中找不到“ModelArts > Edit Credential”，可能是PyCharm版本过高，PyCharm toolkit未适配2023.2之后版本的PyCharm工具。请下载2019.2-2023.2之间（包含2019.2和2023.2）版本的PyCharm专业版工具。

2. 在弹出的对话框中，选择您使用的ModelArts所在区域、填写AK、SK（获取方式**参考链接**），然后单击“OK”完成登录。
   - “Region”：从下拉框中选择区域。必须与ModelArts管理控制台在同一区域。
   - “Project”：Region选择后，Project自动填充为Region对应的项目。
   - “Access Key ID”：填写访问密钥的AK。
   - “Secret Access Key”：填写访问密钥的SK。

图 6-82 填写区域和访问密钥



3. 查看认证结果。

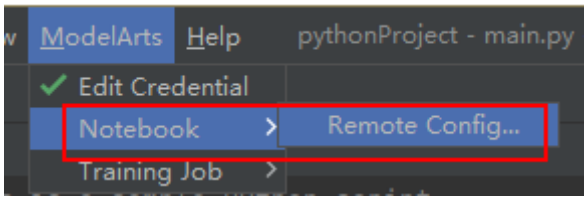
   在Event Log区域中，当提示如下类似信息时，表示访问密钥添加成功。

   ```
   16:01Validate Credential Success: The HUAWEI CLOUDcredential is valid.
   ```

Step4 插件自动化配置

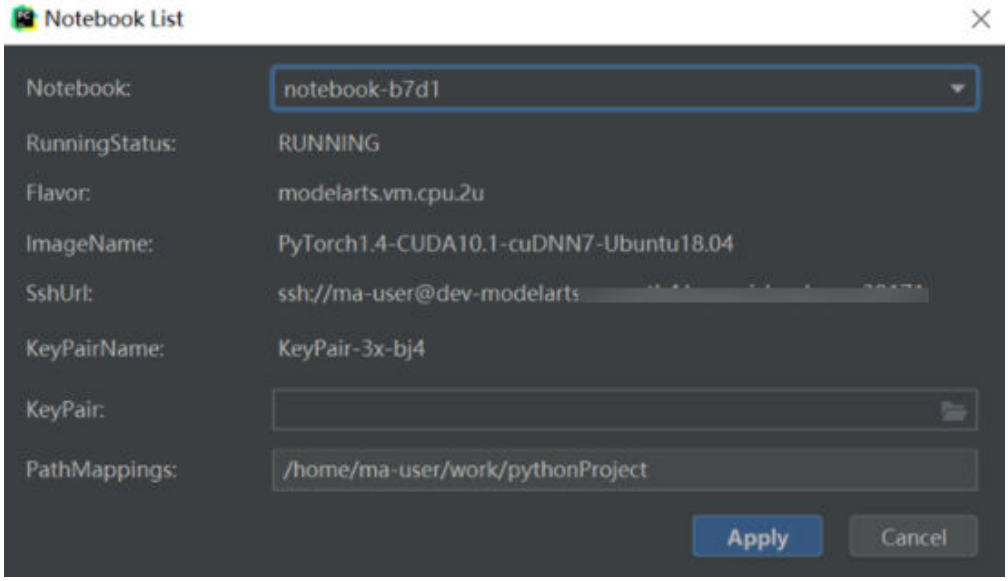
1. 在本地的PyCharm开发环境中，单击“ModelArts > Notebook > Remote Config...”，配置插件。

图 6-83 配置插件



2. 此时，会出现该账号已创建的所有包含SSH功能的Notebook列表，下拉进行选择对应Notebook。

图 6-84 Notebook 列表



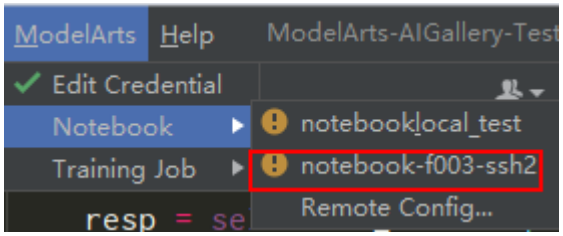
   - KeyPair: 需要选择保存在本地的Notebook对应的keypair认证。即创建Notebook时创建的密钥对文件，创建时会直接保存到浏览器默认的下载文件夹中。
   - PathMappings: 该参数为本地IDE项目和Notebook对应的同步目录，默认为/home/ma-user/work/project名称，可根据自己实际情况更改。
3. 单击“Apply”，配置完成后，重启IDE生效。

   重启后初次进行update python interpreter需要耗费20分钟左右。

Step5 使用插件连接云上 Notebook

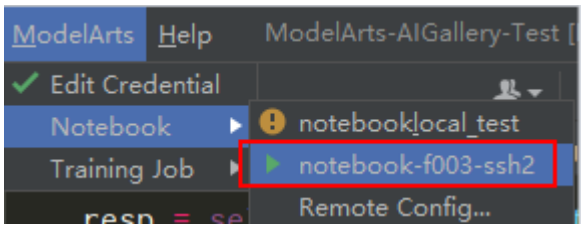
与Notebook断开连接的状态下，单击Notebook名称，根据提示启动本地IDE与Notebook的连接(默认启动时间4小时)。

图 6-85 启动连接 Notebook



连接状态下，单击Notebook名称，根据提示断开本地IDE与云上Notebook的连接。

图 6-86 停止连接 Notebook



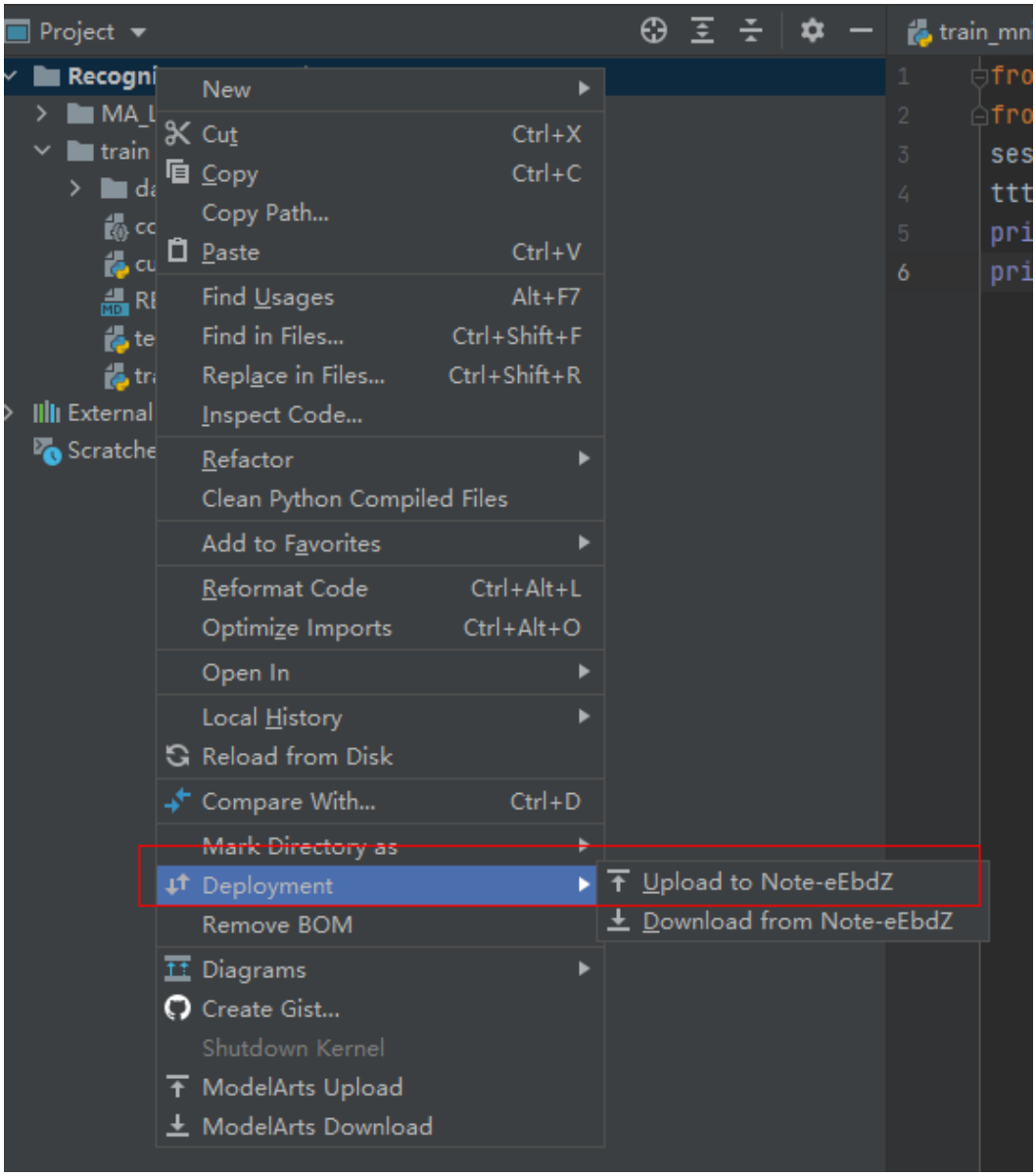
Step6 同步上传本地文件至 Notebook

本地文件中的代码直接复制至本地IDE中即可，本地IDE中会自动同步至云上开发环境。

初始化同步：

在本地IDE的Project目录下，单击右键，选择“Deployment”，单击“Upload to xxx”（Notebook名称），将本地工程文件上传至指定的Notebook。

图 6-87 同步本地文件至 Notebook

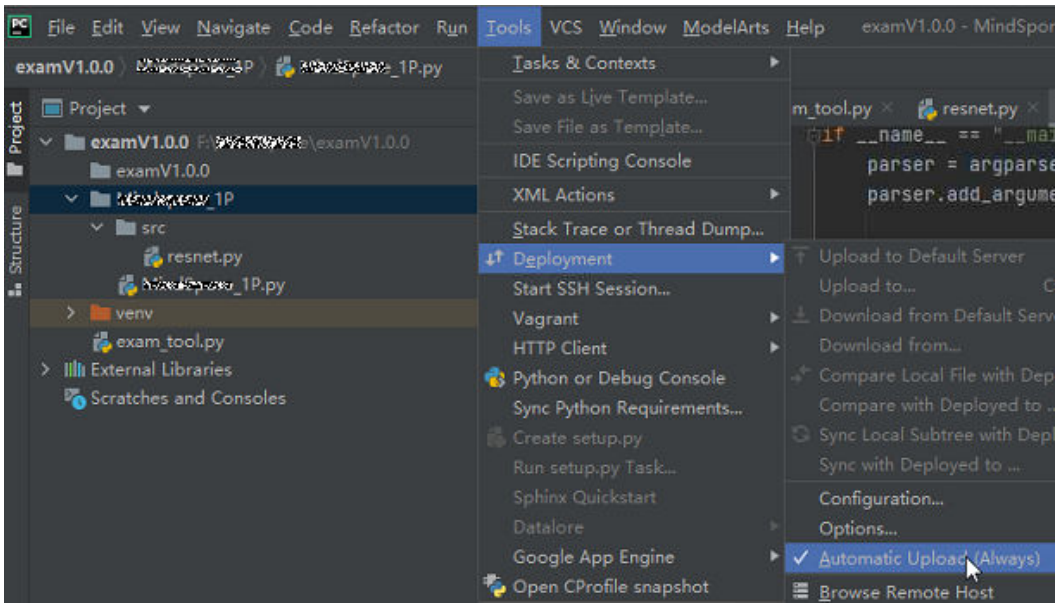


后续同步:

只需修改代码后保存（ctrl+s），即可进行自动同步。

插件安装完成后在本地IDE中开启了“Automatic Upload”，本地目录中的文件会自动上传至云端开发环境Notebook。如果未开启，请参考下图开启自动上传。

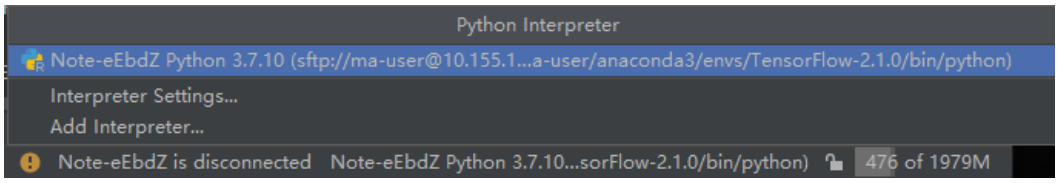
图 6-88 开启自动上传



Step7 远程调试

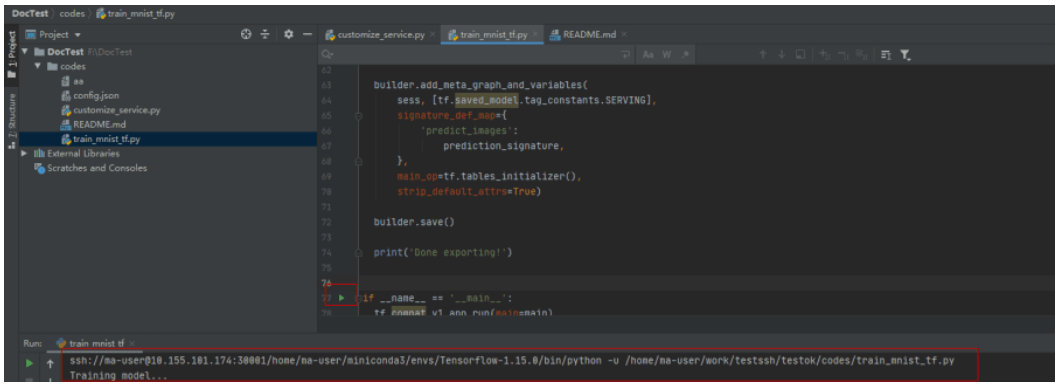
单击本地IDE右下角interpreter，选择Notebook的python解释器。

图 6-89 选择 Python 解释器



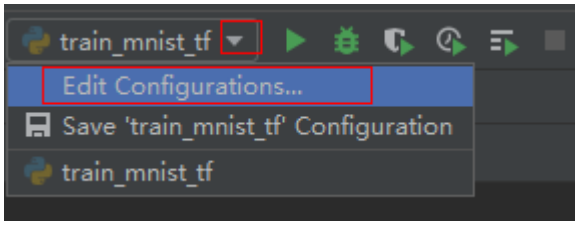
像本地运行代码一样，直接单击运行按钮运行代码即可，此时虽然是在本地IDE点的运行按钮，实际上运行的是云端Notebook里的代码，日志可以回显在本地的日志窗口。

图 6-90 查看运行日志



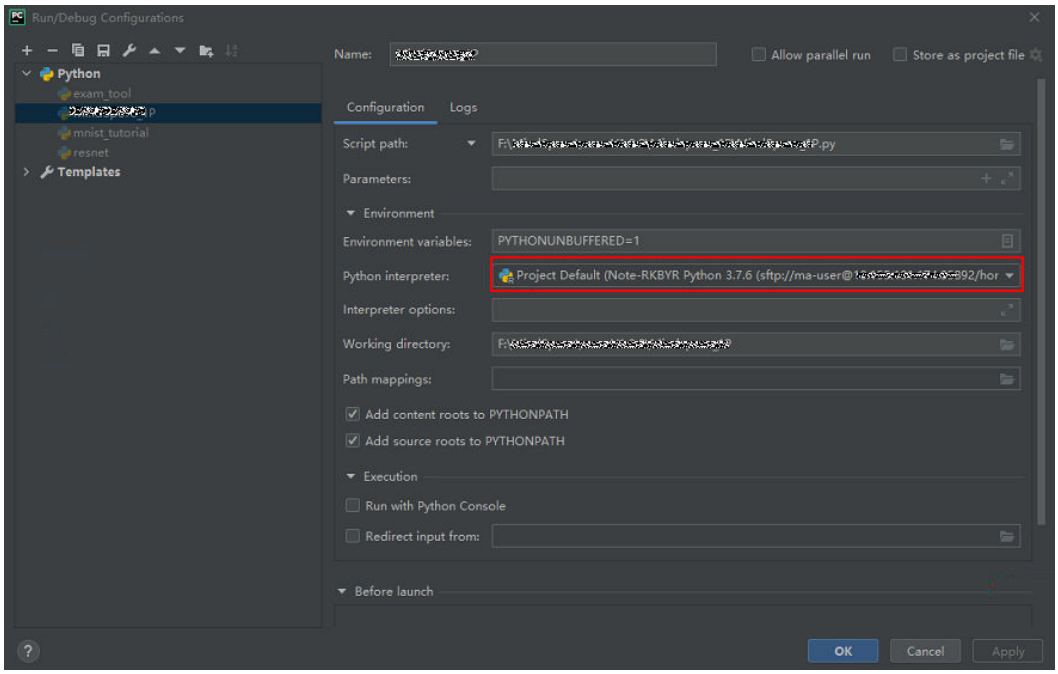
也可以单击本地IDE右上角的Run/Debug Configuration按钮来设置运行参数。

图 6-91 设置运行参数（1）



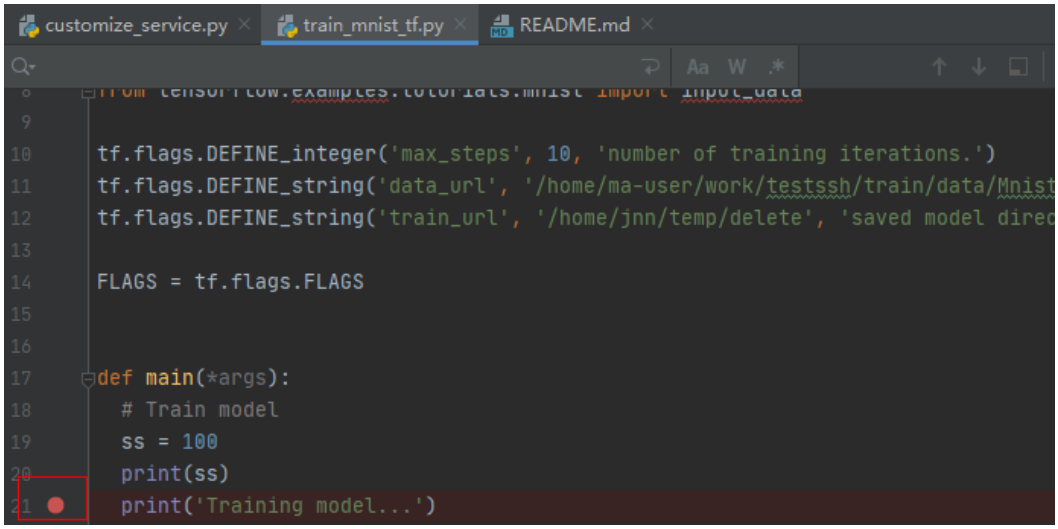
选择远程连接到云上开发环境实例对应的Python解释器。

图 6-92 设置运行参数（2）



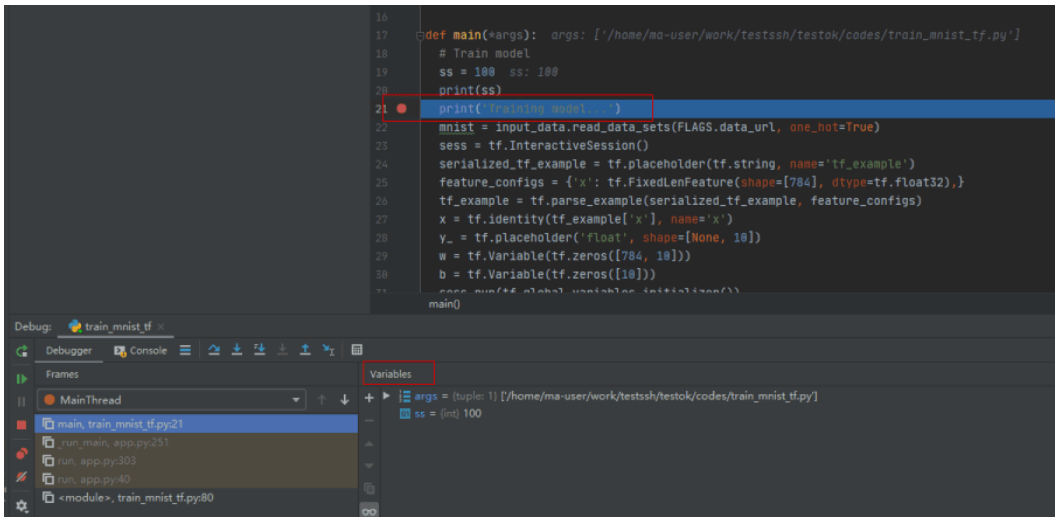
当需要调试代码时，可以直接打断点，然后使用debug方式运行程序。

图 6-93 使用 debug 方式运行程序



此时可以进入debug模式，代码运行暂停在该行，且可以查看变量的值。

图 6-94 Debug 模式下查看变量值



6.4.2 使用 PyCharm 手动连接 Notebook

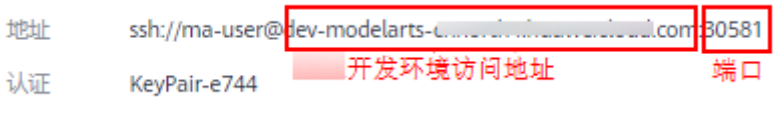
本地IDE环境支持PyCharm和VS Code。通过简单配置，即可用本地IDE远程连接到ModelArts的Notebook开发环境中，调试和运行代码。

本章节介绍基于PyCharm环境访问Notebook的方式。

前提条件

- 本地已安装2019.2及以上版本的PyCharm专业版。SSH远程调试功能只限PyCharm专业版。
- 创建一个Notebook实例，并开启远程SSH开发。该实例状态必须处于“运行中”，具体参见**创建Notebook实例**章节。
- 在Notebook实例详情页面获取开发环境IP地址和端口号。

  图 6-95 Notebook 实例详情页面

- 准备好密钥对。

  密钥对在用户第一次创建时，自动下载，之后使用相同的密钥时不会再有下载界面（用户一定要保存好），或者每次都使用新的密钥对。

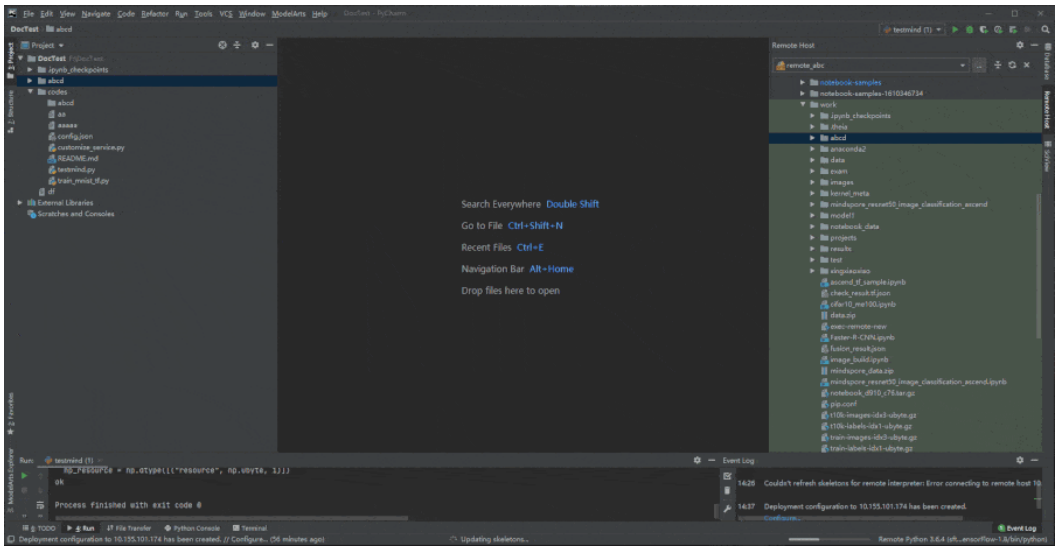
Step1 配置 SSH

1. 在本地的PyCharm开发环境中，单击File -> Settings -> Tools -> SSH Configurations，单击+号，增加一个SSH连接配置。
   - Host：云上开发环境的IP地址，即在开发环境实例页面远程访问模块获取的IP地址。

- Port：云上开发环境的端口，即在开发环境实例页面远程访问模块获取的端口号。
- User name：固定为ma-user。
- Authentication type：Key pair方式。
- Private key file：存放在本地的云上开发环境私钥文件，即在创建开发环境实例时创建并保存的密钥对文件。

2. 单击  将连接重命名，可以自定义一个便于识别的名字，单击OK。
3. 配置完成后，单击Test Connection测试连通性。
4. 选择Yes，显示Successfully connected表示网络可以连通，单击OK。
5. 在最下方再单击OK保存配置。

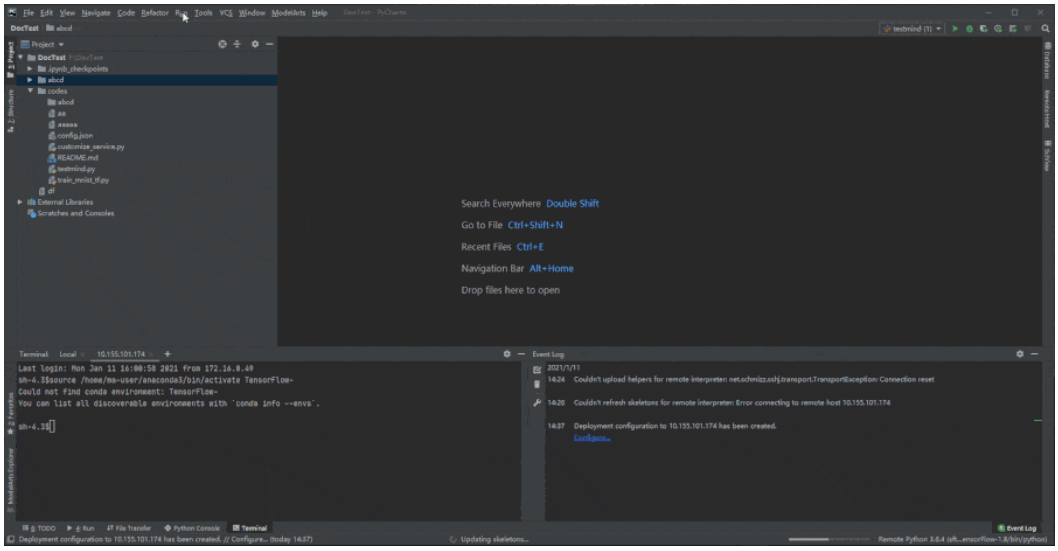
图 6-96 配置 SSH



Step2 获取开发环境预置虚拟环境路径

1. 单击“Tools > Start SSH Session”，则可连接到云端开发环境内。
2. 执行如下命令可在/home/ma-user/下面的README文件查看当前环境内置的Python虚拟环境。
   ```
   cat /home/ma-user/README
   ```
3. 执行source命令可以切换到具体的Python环境中。
4. 执行which python查看python路径并复制出来，以备后续配置云上Python Interpreter使用。

图 6-97 获取开发环境预置虚拟环境路径



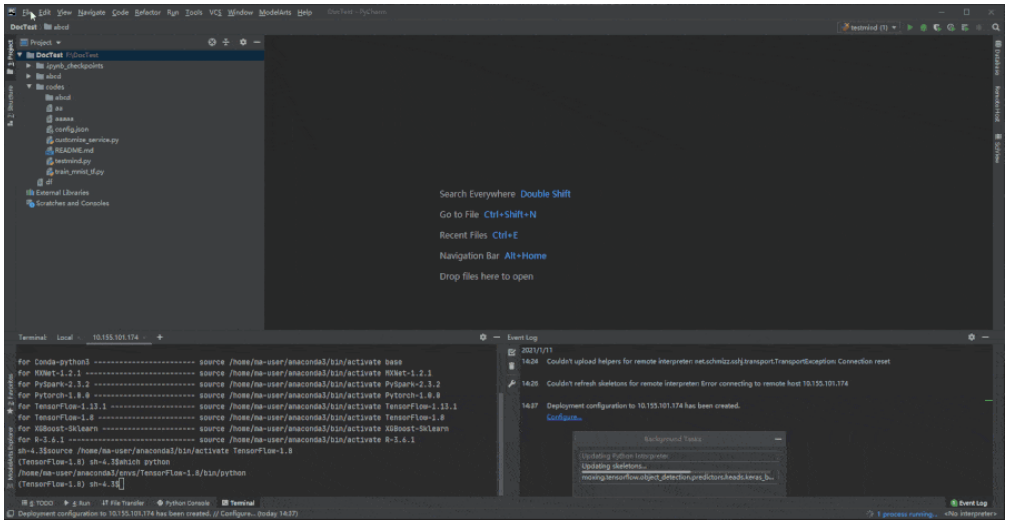
Step3 配置云上 Python Interpreter

1. 单击“File > Settings > Project：PythonProject > Python Interpreter”，单击设置图标，再单击“Add”，添加一个新的interpreter。
2. 选择“Existing server configuration”，在下拉菜单中选择上一步配置好的SSH configuration，单击“Next”。
3. 配置Python Interpreter
   - Interpreter：填写第一步复制的python路径，例如：/home/ma-user/anaconda3/envs/Pytorch-1.0.0/bin/python
     如果路径为~/anaconda3/envs/Pytorch-1.0.0/bin/python，把~替换为/home/ma-user即可。
   - Sync folders: 需要配置本地的工程目录文件同步到云上开发环境中的某个目录，推荐配置为/home/ma-user下的某个目录中（其他目录可能没有访问权限），例如/home/ma-user/work/projects。
4. 单击右侧文件夹图标，勾选上“Automatically upload”选项，以便于本地修改的文件自动上传到容器环境中。
5. 单击“Finish”，结束配置。

   可以看到本地的工程文件已经自动往云上环境上传了。后续本地的文件每修改一次，都会自动的同步到云上的环境中。

   右下角可以看到当前的Interpreter为Remote Interpreter。

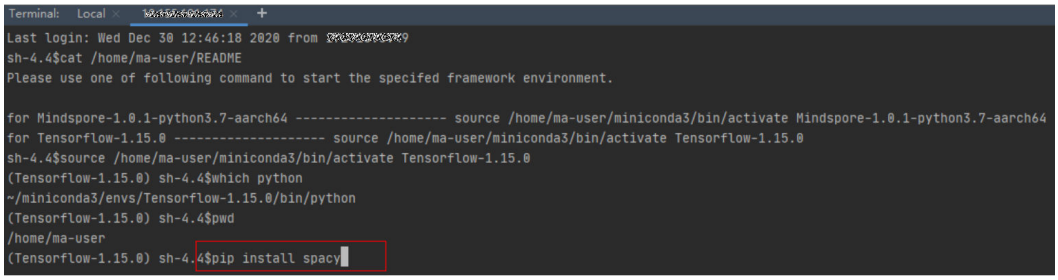
图 6-98 配置云上 Python Interpreter



Step4 云上环境依赖库安装

在进入开发环境后，可以使用不同的虚拟环境，例如TensorFlow、PyTorch等，但是实际开发中，通常还需要安装其他依赖包，此时可以通过Terminal连接到环境里操作。

单击工具栏“Tools >Start SSH session”，选择SSH Configuration中配置的开发环境。可以执行pip install安装所需要的包。

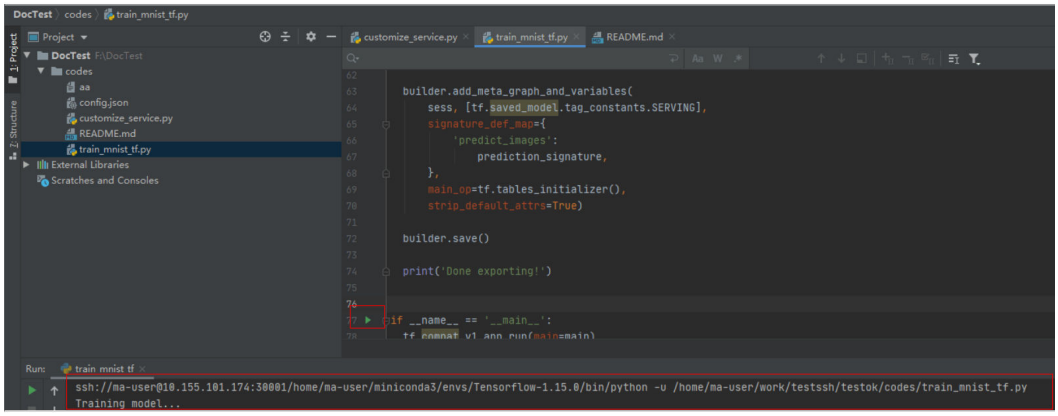


Step5 在开发环境中调试代码

由于已经连接至云端开发环境，此时可以方便地在本地PyCharm中编码、调测并运行。实际运行环境为云上开发环境，资源为云上昇腾AI处理器资源。可以做到本地编写修改代码，直接在云上环境运行。

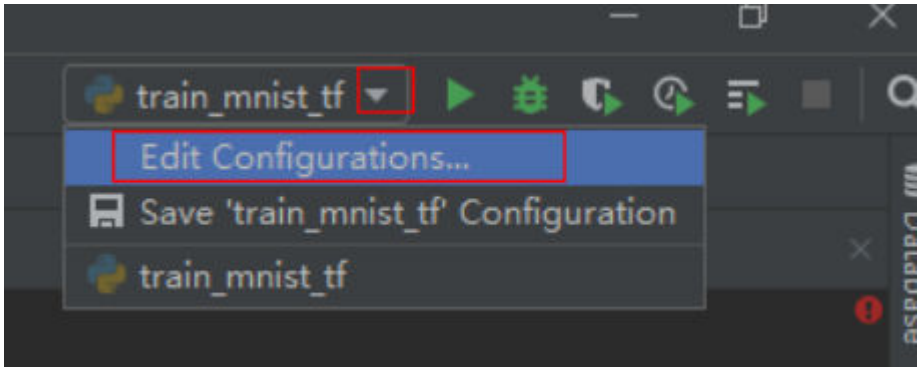
像本地运行代码一样，直接单击运行按钮运行代码即可，此时虽然是在本地IDE单击的运行按钮，实际上运行的是云端开发环境里的代码，日志可以回显在本地的日志窗口。

图 6-99 调试代码



也可以单击右上角的Run/Debug Configuration来设置运行的参数。

图 6-100 设置运行参数



当需要调试代码时，可以直接打断点，然后使用debug方式运行程序。

图 6-101 代码打断点

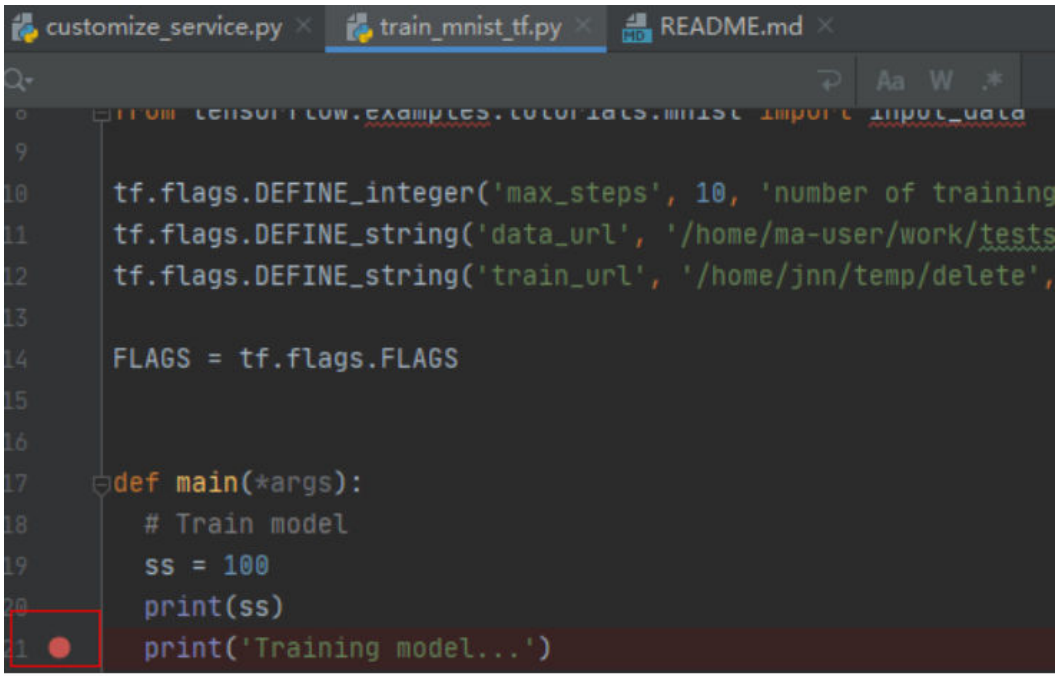
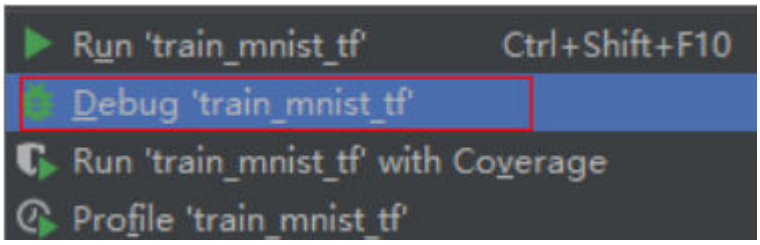
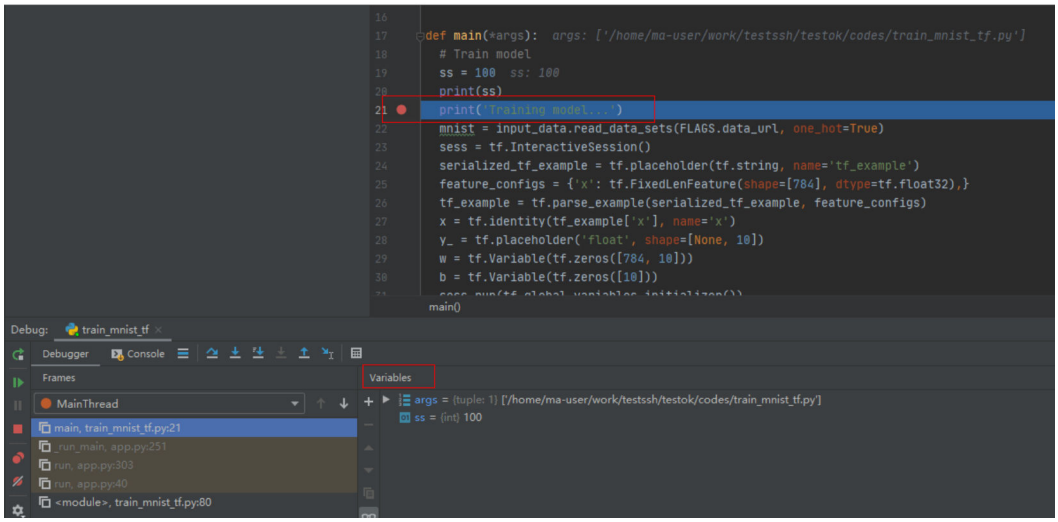


图 6-102 Debug 方式调试



此时可以进入debug模式，代码运行暂停在该行，且可以查看变量的值。

图 6-103 Debug 模式



使用debug方式调试代码的前提是本地的代码和云端的代码是完全一致的，如果不一致可能会导致在本地打断点的行和实际运行时该行的代码并不一样，会出现意想不到的错误。

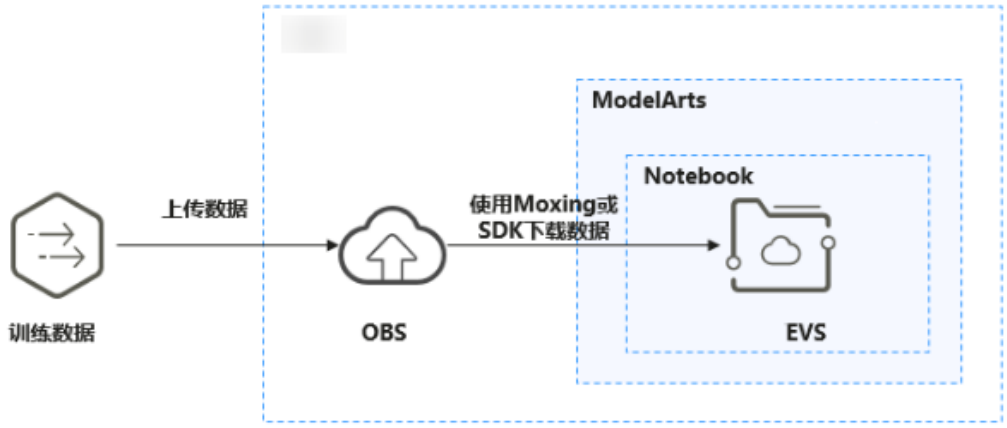
因此在配置云上Python Interpreter时，推荐选择Automatically upload选项，以保证本地的文件修改能自动上传到云端。如果没有选择自动上传，则本地代码修改完后，也可以参考**Step6 同步上传本地文件至Notebook**手动上传目录或代码。

6.4.3 使用 PyCharm 上传数据至 Notebook

不大于500MB数据量，直接复制至本地IDE中即可。

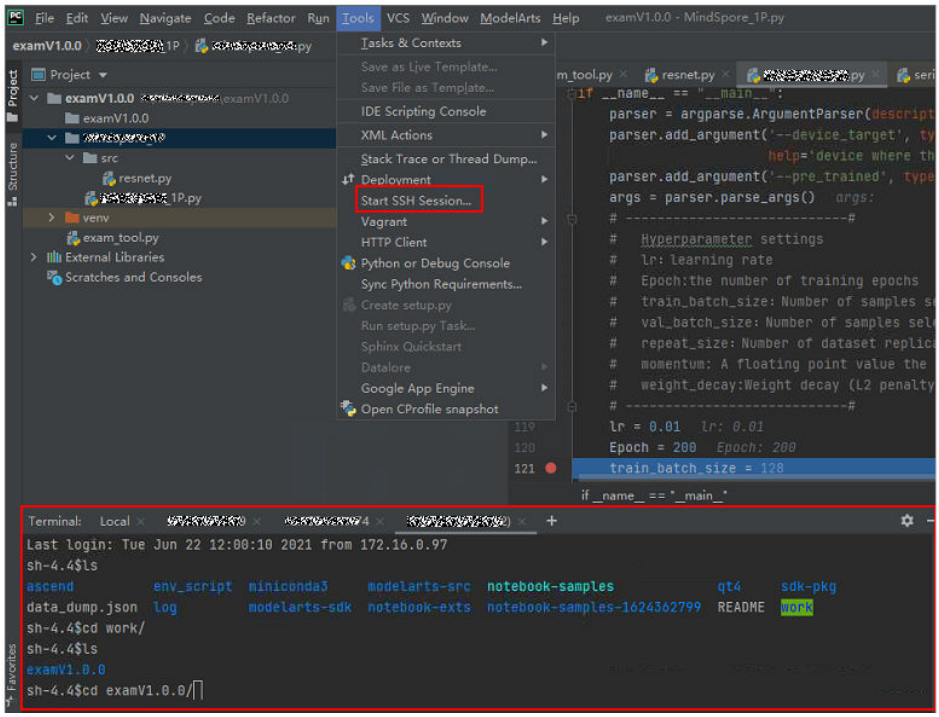
大于500MB数据量，请先上传到OBS中，再从OBS下载到云上Notebook。

图 6-104 数据通过 OBS 中转上传到 Notebook



1. 上传数据至OBS，具体操作请参见**上传文件至OBS桶**。
2. 将OBS中的数据传至Notebook中，通过在本地IDE的Terminal中使用ModelArts提供的Moxing库的文件操作API（mox.file.copy_parallel）完成。
   a. 在PyCharm环境中开启Terminal，VS Code中操作类似。

图 6-105 PyCharm 环境开启 Terminal



b. 在本地IDE的Terminal中使用Moxing上传OBS文件到Notebook的操作示例如下：

```
#手动source进入开发环境
cat /home/ma-user/README
#然后选择要source的环境
source /home/ma-user/miniconda3/bin/activate MindSpore-python3.7-aarch64
#输入python并回车，进入python环境
python
#使用moxing
import moxing as mox
#下载一个OBS文件夹，从OBS下载至EVS（OBS -> EVS）
mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/tmp/sub_dir_0')
```

6.5 通过 VS Code 远程使用 Notebook 实例

6.5.1 VS Code 连接 Notebook 方式介绍

Visual Studio Code (VS Code) 是一个流行的代码编辑器，它支持多种编程语言和开发环境。支持通过VS Code连接和使用Jupyter Notebook。

当用户创建完成支持SSH的Notebook实例后，使用VS Code的开发者可以通过以下方式连接到开发环境中：

- **VS Code ToolKit连接Notebook**

  该方式是指用户在VS Code上使用ModelArts VS Code Toolkit插件提供的登录和连接按钮，连接云上实例。

- **VS Code手动连接Notebook**

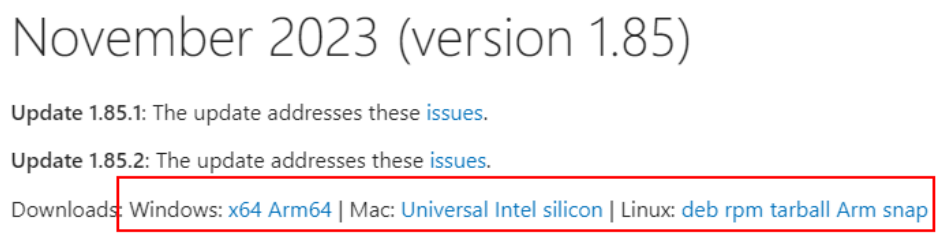
  该方式是指用户使用VS Code Remote SSH插件手工配置连接信息，连接云上实例。

安装 VS Code 软件

使用VS Code连接开发环境时，首先需要安装VS Code软件。

- **VS Code下载方式**：
  下载地址: **https://code.visualstudio.com/updates/v1_85**

  图 6-106 VS Code 的下载位置

- **VS Code版本要求**：
  建议用户使用VS Code 1.85.2版本或者最新版本进行远程连接。
- **VS Code安装指导如下：**
  Windows系统下，下载后直接双击安装包完成安装。
  Linux系统下，执行命令**sudo dpkg -i code_1.85.2-1705561292_amd64.deb**安装。

  说明

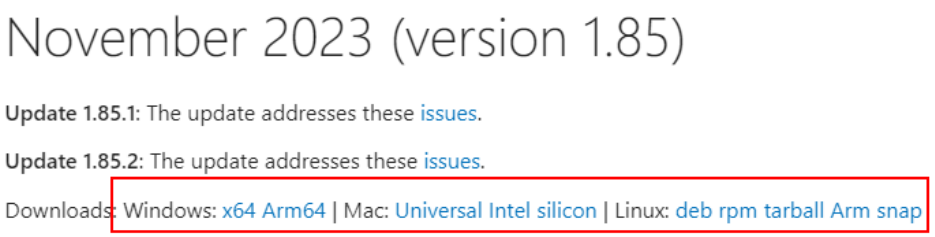
  Linux系统用户，需要在非root用户进行VS Code安装。

6.5.2 安装 VS Code 软件

VS Code下载方式：

下载地址: **https://code.visualstudio.com/updates/v1_85**

图 6-107 VS Code 的下载位置



VS Code版本要求：

建议用户使用VS Code 1.85.2版本进行远程连接。

VS Code安装指导如下：

Windows系统下，下载后直接双击安装包完成安装。

Linux系统下，执行命令**sudo dpkg -i code_1.85.2-1705561292_amd64.deb**安装。

> **说明**
>
> Linux系统用户，需要在非root用户进行VS Code安装。

6.5.3 VS Code ToolKit 连接 Notebook

本节介绍如何在本地使用ModelArts提供的VS Code插件工具VS Code ToolKit，协助用户完成SSH远程连接Notebook。

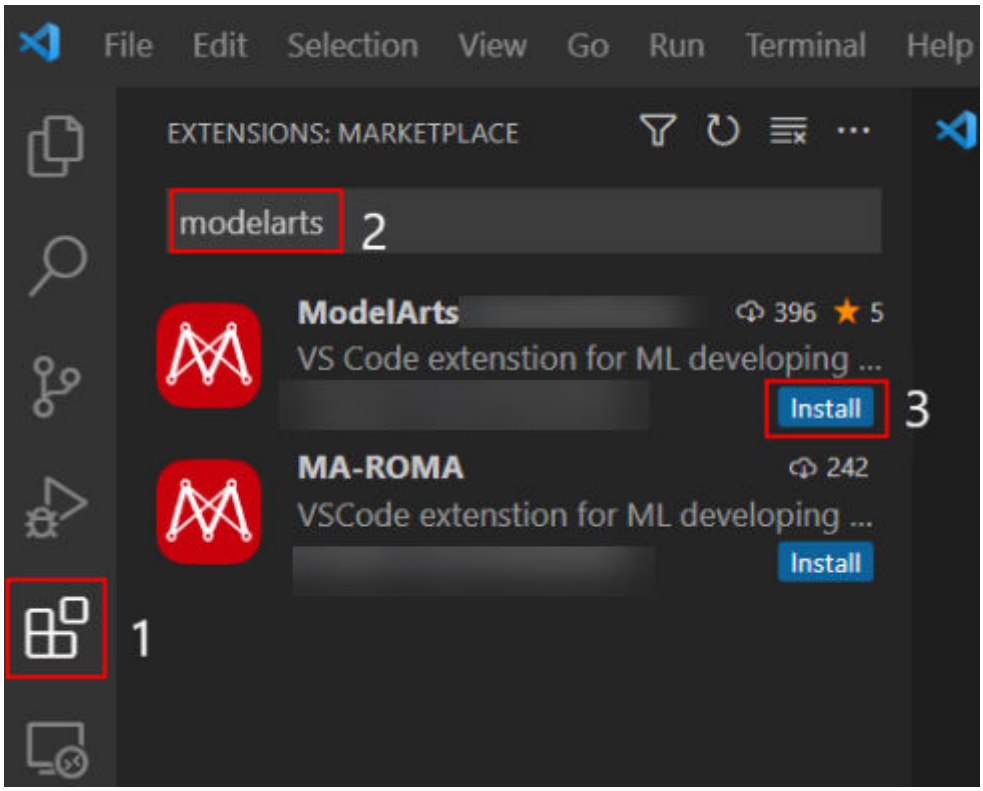
前提条件

已下载并安装VS Code。详细操作请参考**安装VS Code软件**。

Step1 安装 VS Code 插件

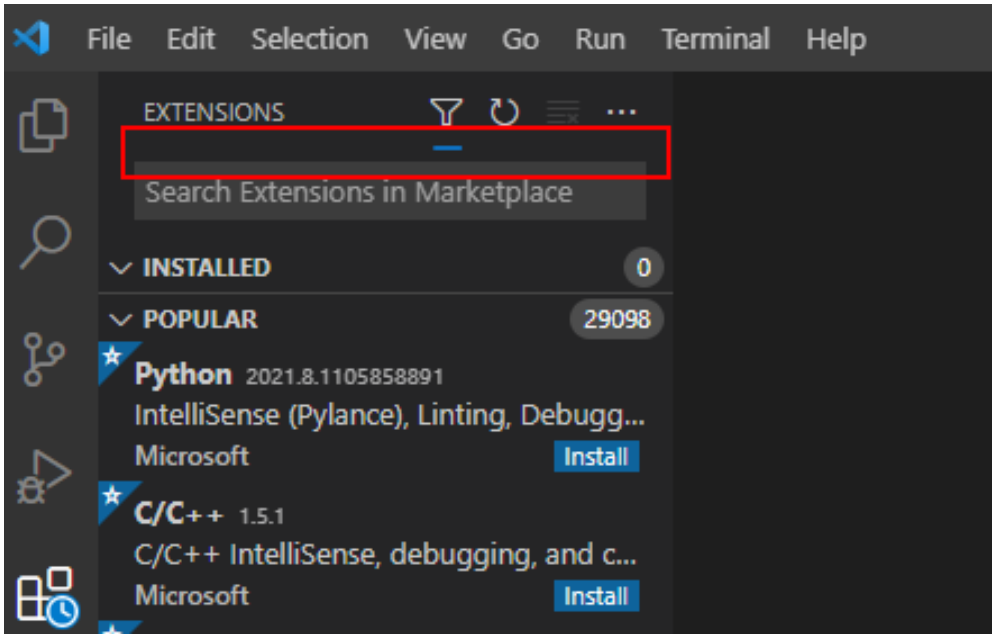
1. 在本地的VS Code开发环境中，如**图6-108**所示，在VS Code扩展中搜索“ModelArts-HuaweiCloud”并单击“安装”。

   图 6-108 安装 VS Code 插件



2. 安装过程预计1~2分钟，如**图6-109**所示，请耐心等待。

图 6-109 安装过程



3. 安装完成后，系统右下角提示安装完成，导航左侧出现ModelArts图标和SSH远程连接图标，表示VS Code插件安装完成。

图 6-110 安装完成提示

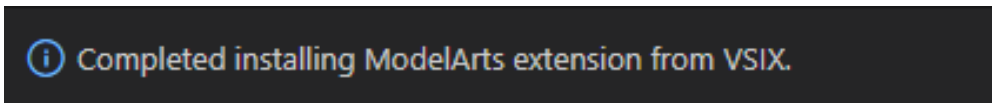
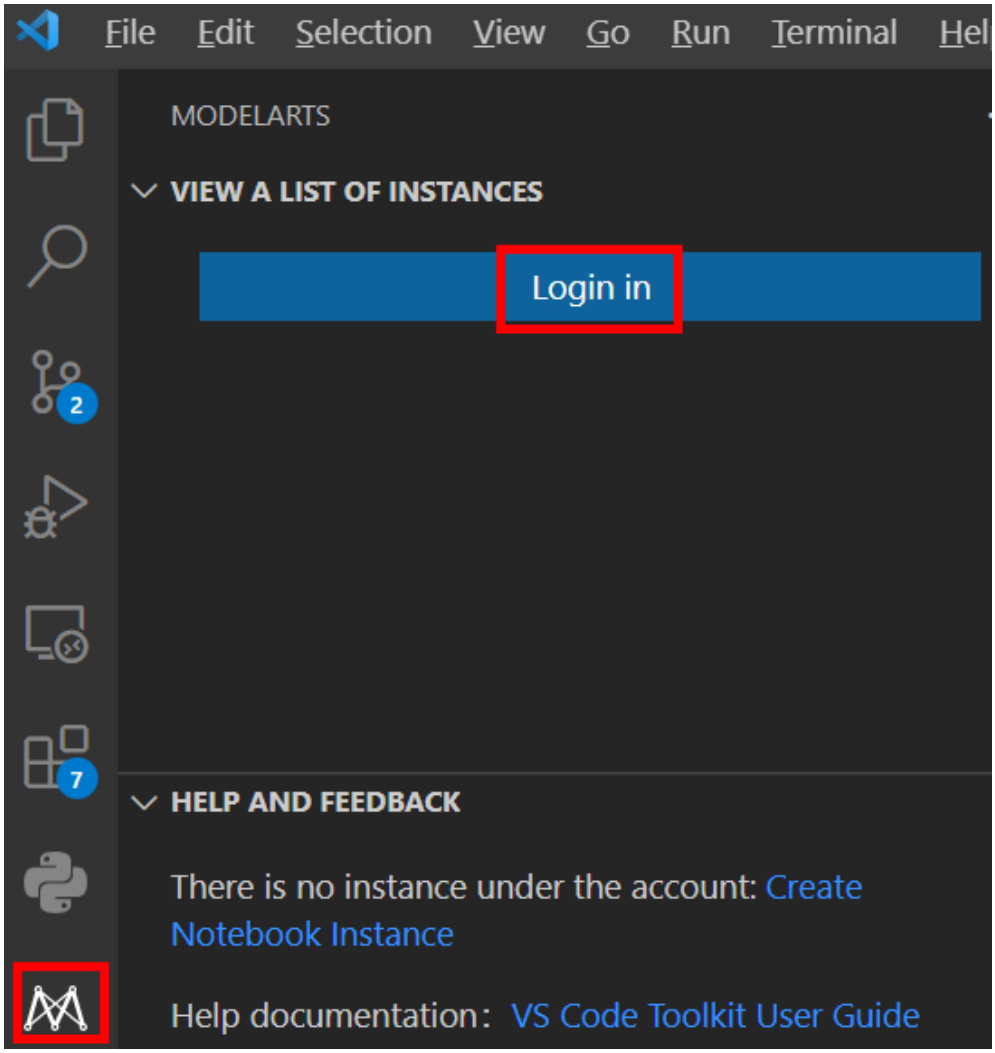
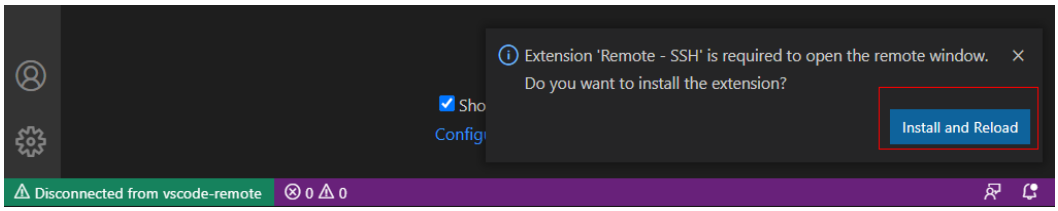


图 6-111 安装完成



当前网络不佳时SSH远程连接插件可能未安装成功，此时无需操作，在**Step4 连接Notebook实例**的**1**之后，会弹出如下图对话框，单击Install and Reload即可。

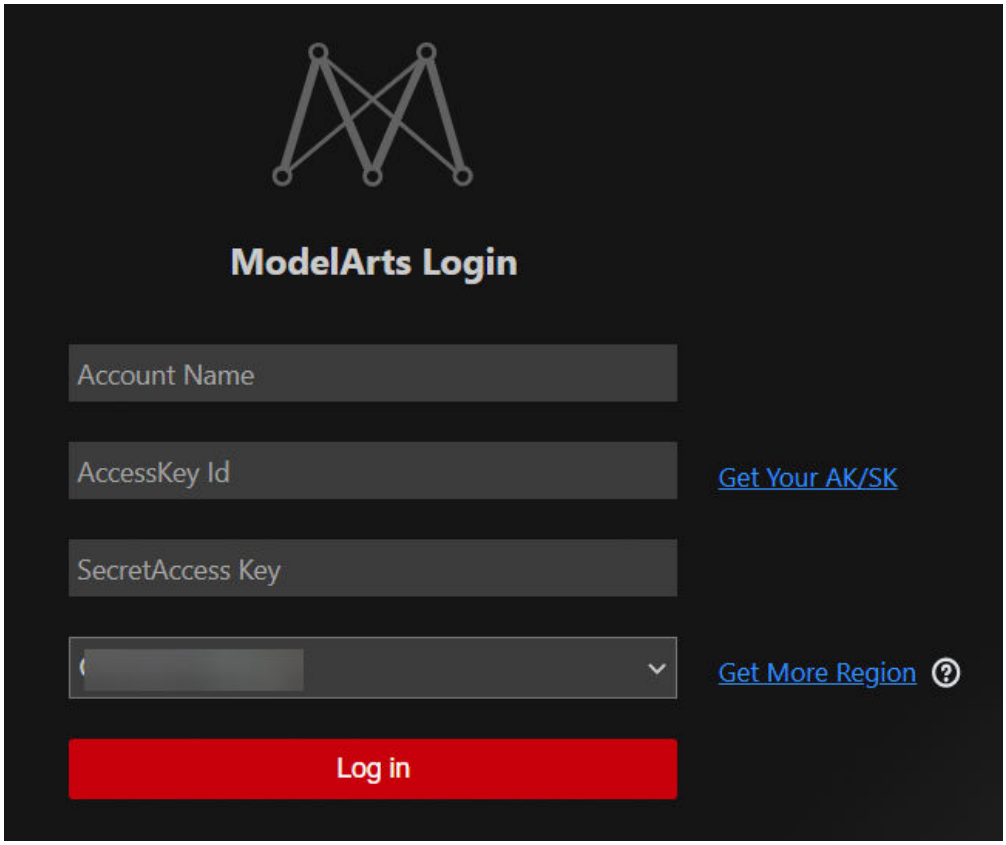
图 6-112 重新连接远程 SSH



Step2 登录 VS Code 插件

1. 在本地的VS Code开发环境中，单击ModelArts图标  ，单击“User Settings”，配置用户登录信息。

图 6-113 登录插件



输入如下用户登录信息，单击“登录”。

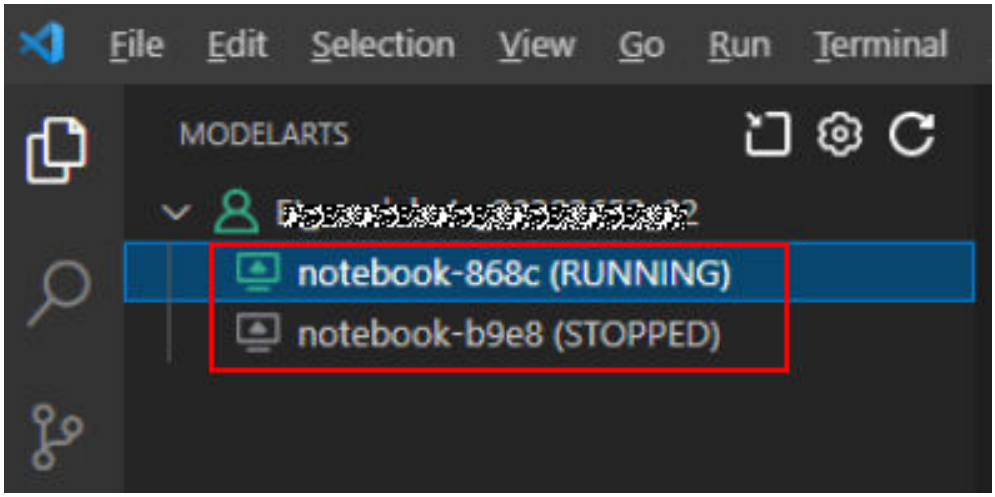
- Name：自定义用户名，仅用于VS Code页面展示，不与任何华为云用户关联。
- AK、SK：在“账号中心 > 我的凭证 > 访问密钥”中创建访问密钥，获取AK、SK（**参考链接**）。
- 选择站点：此处的站点必须和远程连接的Notebook在同一个站点，否则会导致连接失败。

2. 登录成功后显示Notebook实例列表。

说明

此处仅显示ModelArts控制台default工作空间下的Notebook实例。

图 6-114 登录成功



Step3 创建 Notebook 实例

⚠ 注意

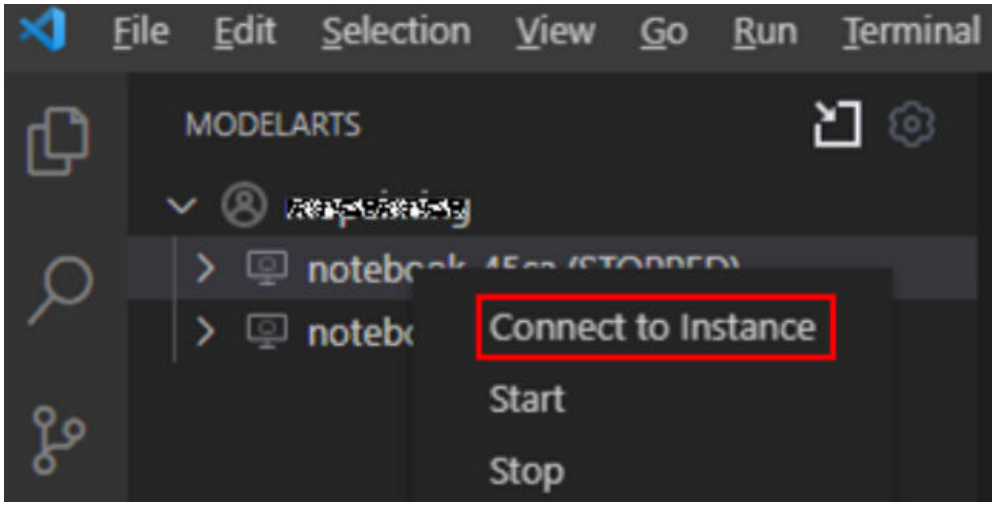
- 创建实例时，需开启“SSH远程开发”，并下载保存密钥对至本地如下目录。
  Windows：C:\Users\{{user}}
  macOS/Linux: Users/{{user}}
- 密钥对在用户第一次创建时自动下载，之后使用相同的密钥时不会再有下载界面（请妥善保管），或者每次都使用新的密钥对。

创建一个Notebook实例，并开启远程SSH开发，具体参见**创建Notebook实例**。

Step4 连接 Notebook 实例

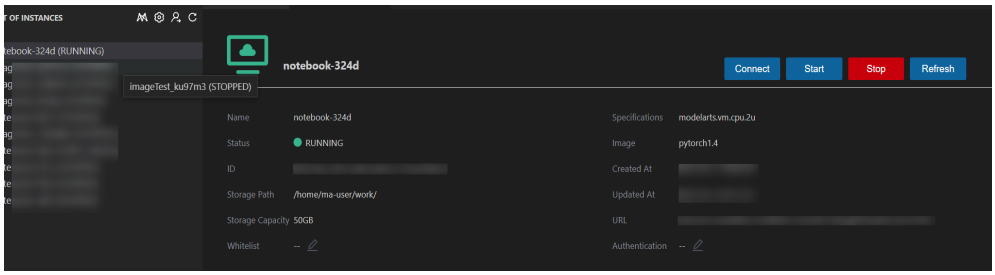
1. 在本地的VS Code开发环境中，右键单击实例名称，单击“Connect to Instance”，启动并连接Notebook实例。
   Notebook实例状态处于“运行中”或“停止”状态都可以，如果Notebook实例是停止状态，连接Notebook时，VS Code插件会先启动实例再去连接。

图 6-115 连接 Notebook 实例



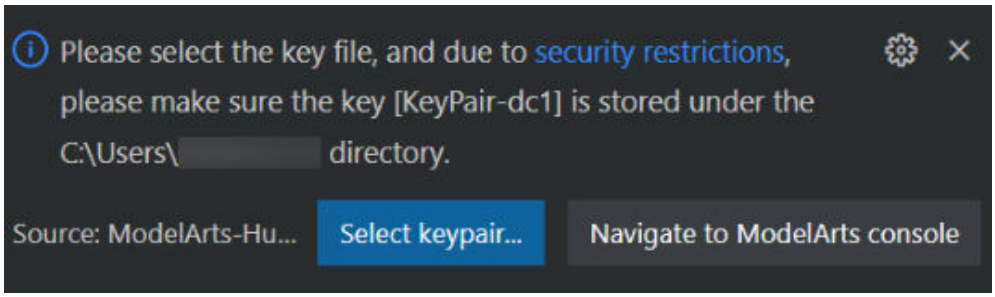
或者单击实例名称，在VS Code开发环境中显示Notebook实例详情页，单击“连接”，系统自动启动该Notebook实例并进行远程连接。

图 6-116 查看 Notebook 实例详情页



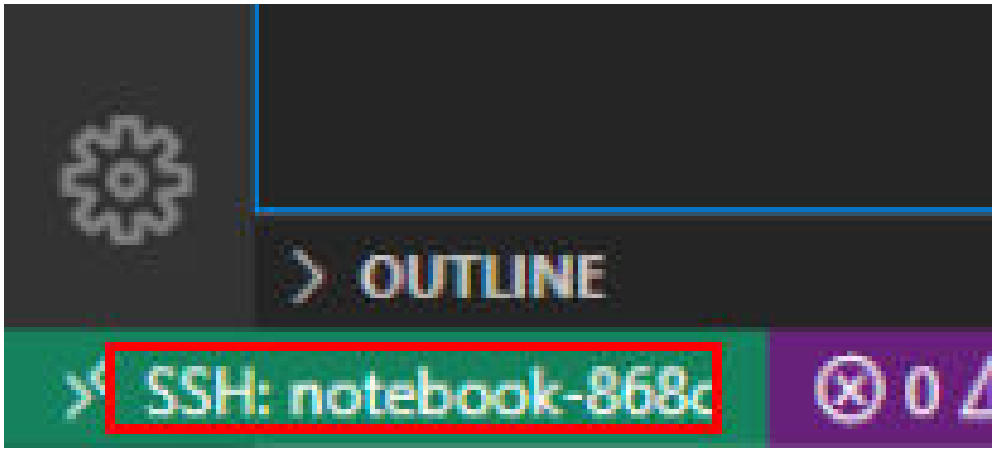
2. 第一次连接Notebook时，系统右下角会提示需要先配置密钥文件。选择本地密钥pem文件，根据系统提示单击“OK”。

图 6-117 配置密钥文件



3. 单击“确定”后，插件自动连接远端Notebook实例。首次连接大约耗时1~2分钟，取决于本地的网络情况。VS Code环境左下角显示类似下图即为连接成功。

图 6-118 连接成功



远程调试代码

1. 在VS Code界面，上传本地代码到云端开发环境。
   a. 单击“File > OpenFolder”，选择要打开的路径，单击“OK”。

   图 6-119 Open Folder

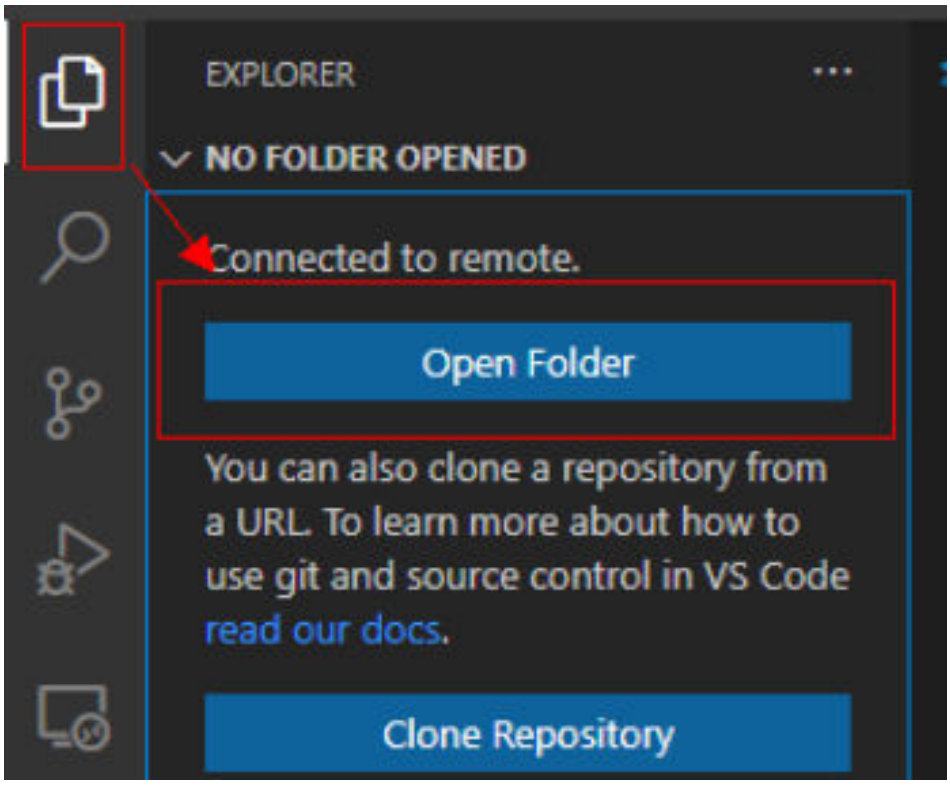
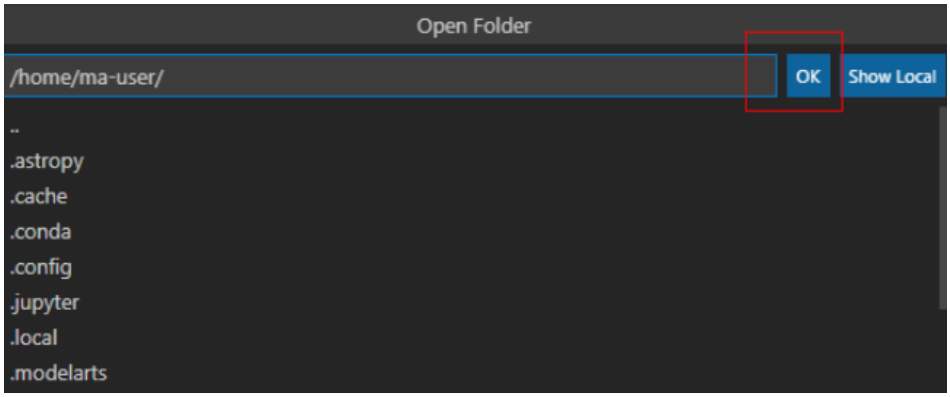


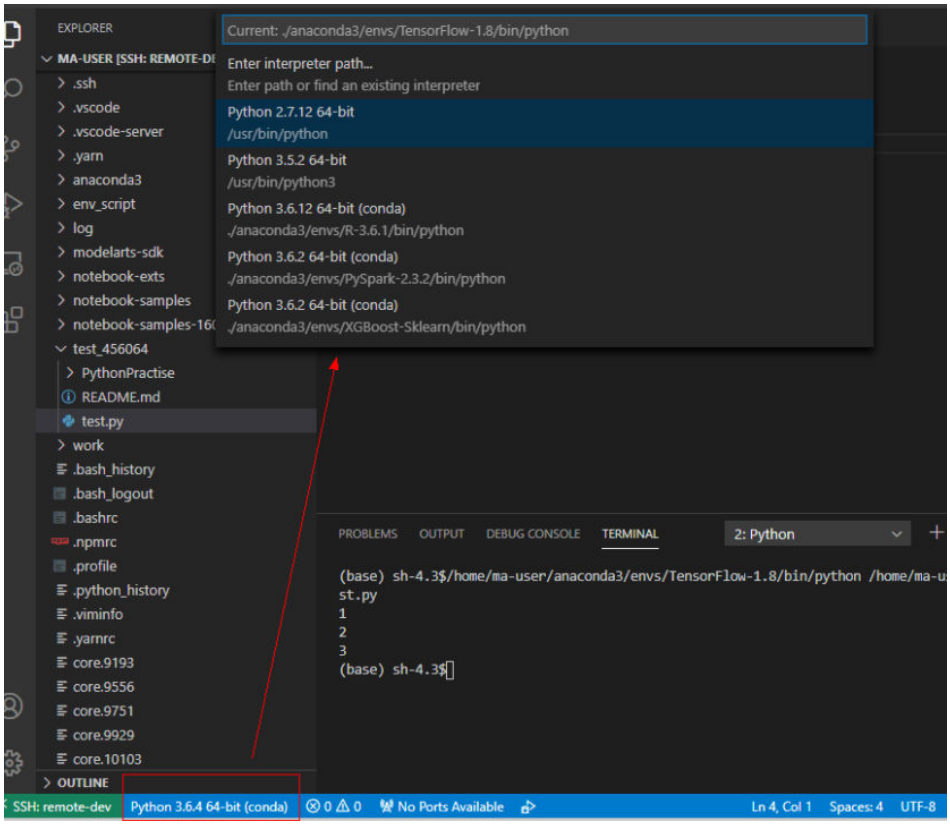
图 6-120 选择文件路径



b. 此时，会在IDE左侧出现该开发环境下的目录结构，选择想要上传的代码及其他文件，拖拽至目录对应的文件夹内即完成本地代码上传至云端。

c. 在VS Code中打开要执行的代码文件，在执行代码之前需要选择合适的Python版本路径，单击下方默认的Python版本路径，此时在上方会出现该远程环境上所有的python版本，选择自己需要的版本即可。

图 6-121 选择 Python 版本



2. 对于打开的代码文件，单击run按钮，即可执行，可以在下方的Terminal中看到代码输出信息。

   - 如果执行较长时间的训练作业，建议使用nohup命令后台运行，否则SSH窗口关闭或者网络断连会影响正在运行的训练作业，命令参考：

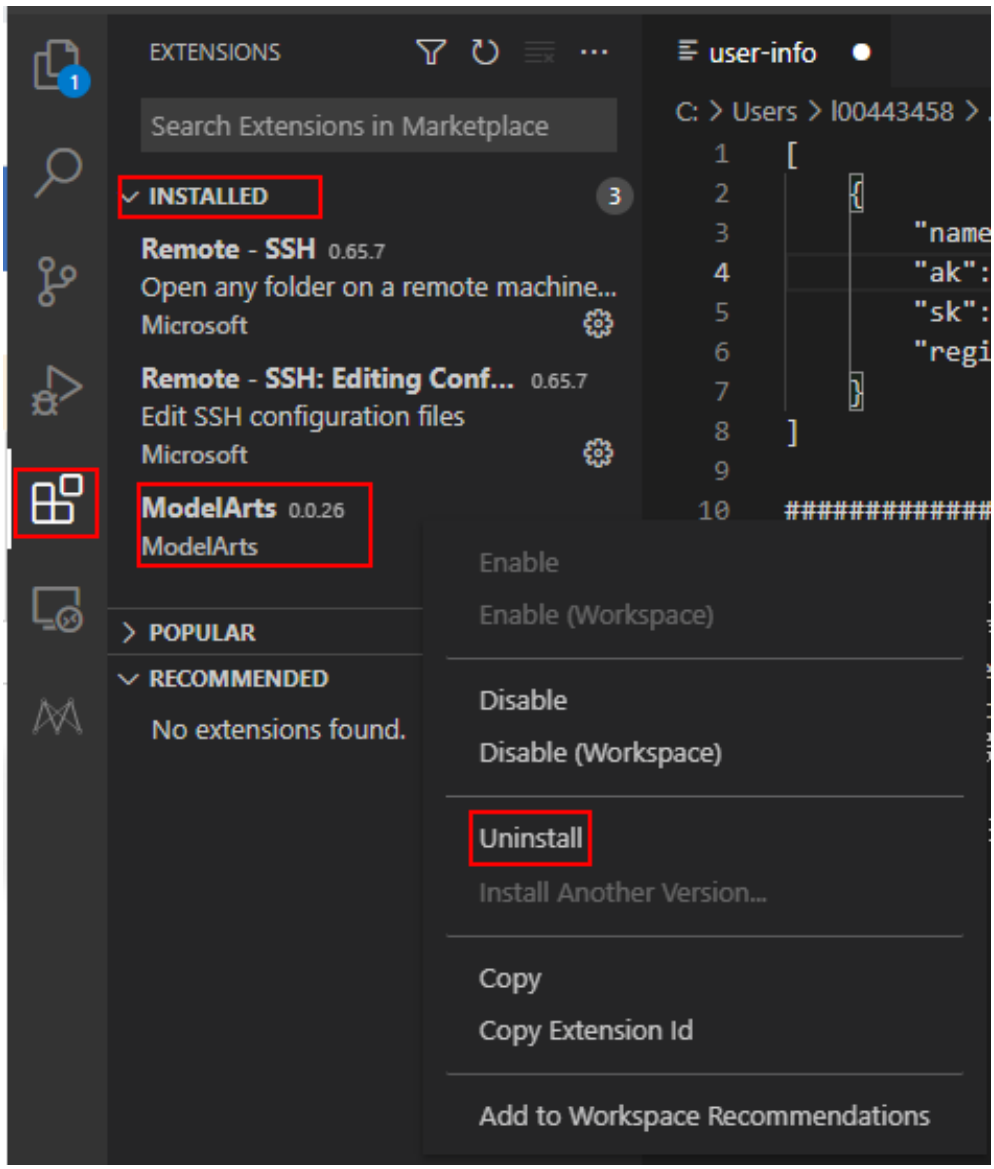
     ```
     nohup your_train_job.sh > output.log 2>&1  &  tail -f  output.log
     ```

- 如果要对代码进行debug调试，步骤如下：
  i. 单击左侧“Run > Run and Debug”。
  ii. 选择当前打开的默认的python代码文件进行调试。
  iii. 对当前代码进行打断点，即在代码左侧进行单击，就会出现小红点。
  iv. 此时，即可按照正常的代码调试步骤对代码调试，在界面左边会显示debug信息，代码上方有相应的调试步骤。

相关操作

卸载VS Code插件操作如图6-122所示。

图 6-122 卸载 VS Code 插件



6.5.4 VS Code 手动连接 Notebook

本地IDE环境支持PyCharm和VS Code。通过简单配置，即可用本地IDE远程连接到ModelArts的Notebook开发环境中，调试和运行代码。

本章节介绍基于VS Code环境访问Notebook的方式。

前提条件

- 已下载并安装VS Code。详细操作请参考**安装VS Code软件**。
- 用户本地PC或服务器的操作系统中建议先安装Python环境，详见**VSCode官方指导**。
- 创建一个Notebook实例，并开启远程SSH开发。该实例状态必须处于“运行中”，具体参见**创建Notebook实例**章节。
- 在Notebook实例详情页面获取开发环境访问地址和端口号。

  图 6-123 Notebook 实例详情页面

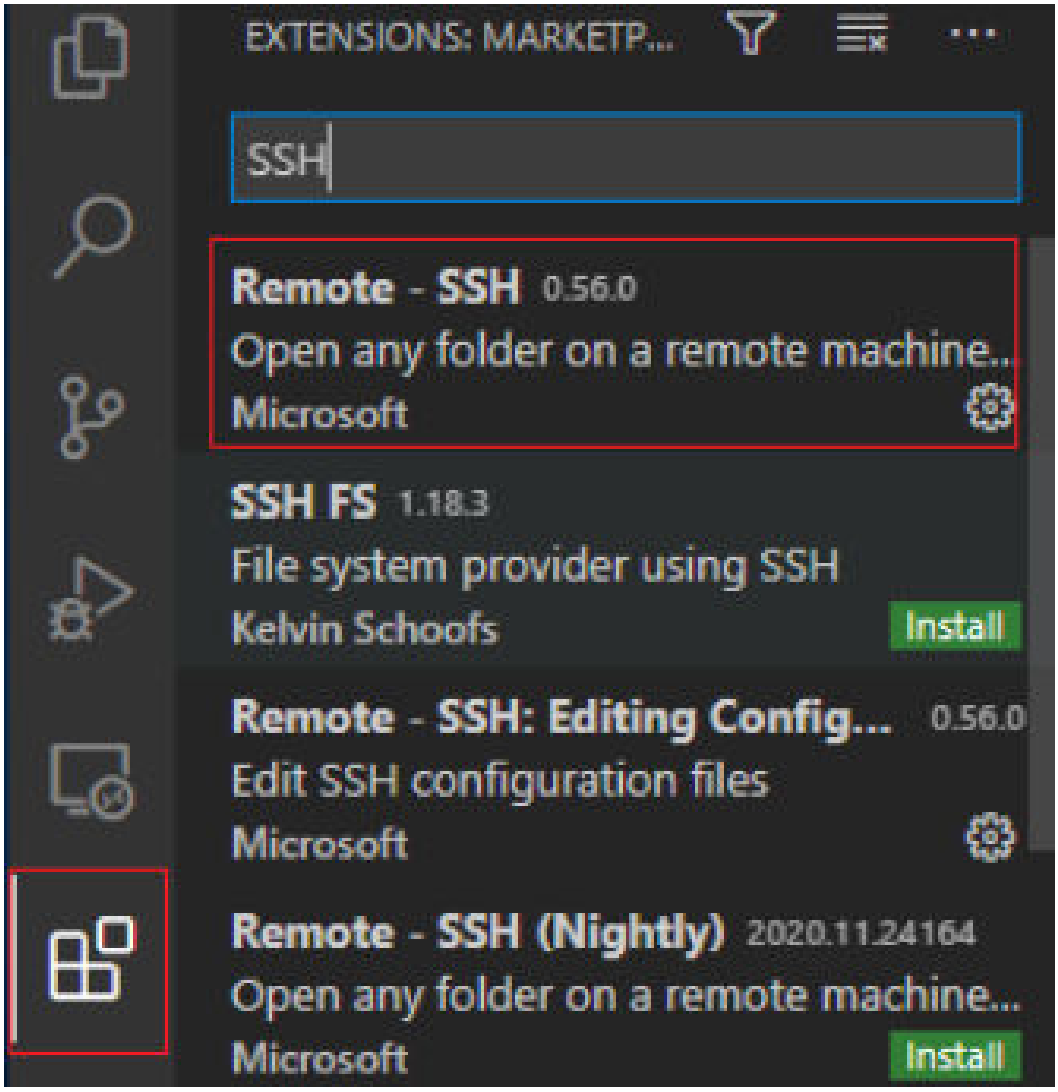
- 准备好密钥对。

  密钥对在用户第一次创建时，自动下载，之后使用相同的密钥时不会再有下载界面（用户一定要保存好），或者每次都使用新的密钥对。

Step1 添加 Remote-SSH 插件

在本地的VS Code开发环境中，单击左侧列表的Extensions图标选项  ，在搜索框中输入SSH，单击Remote-SSH插件的install按钮，完成插件安装。

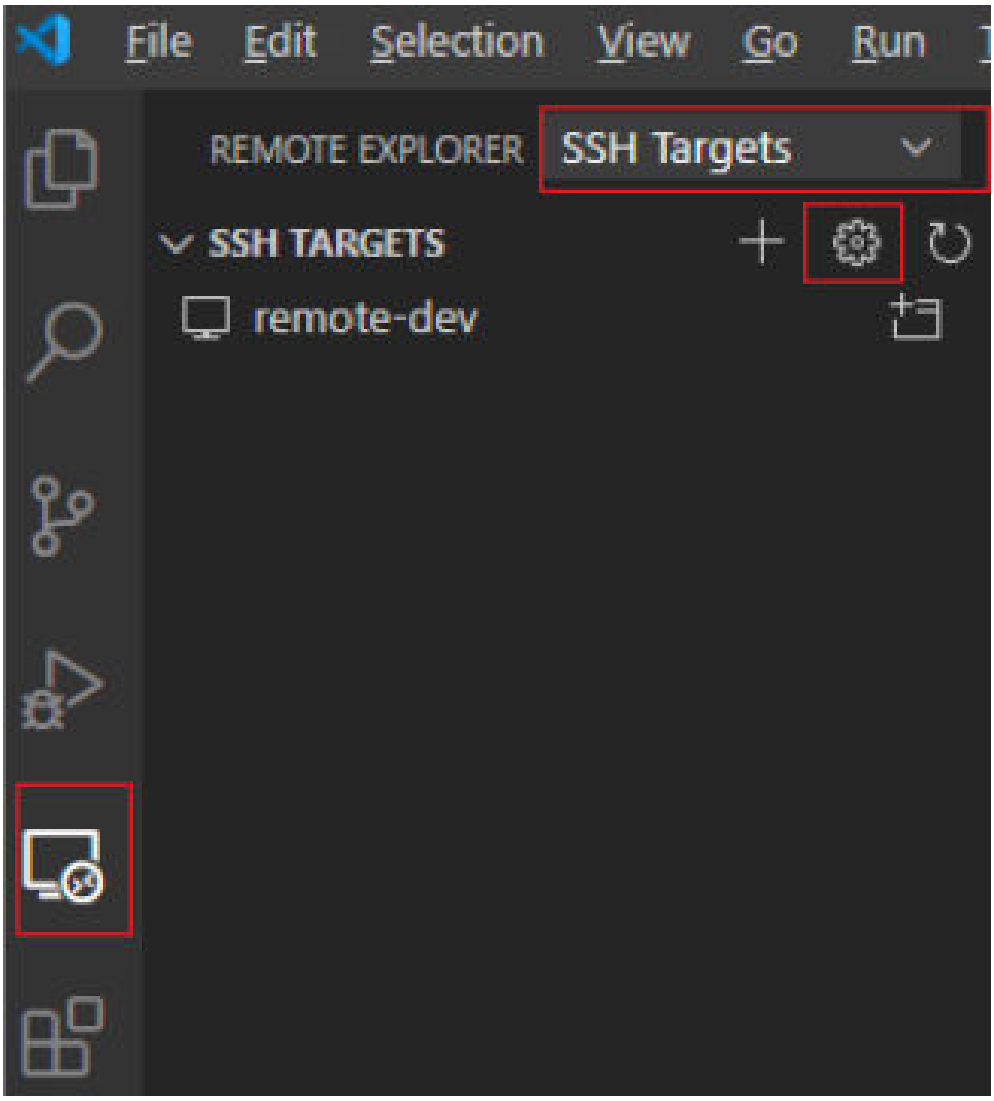
图 6-124 添加 Remote-SSH 插件



Step2 配置 SSH

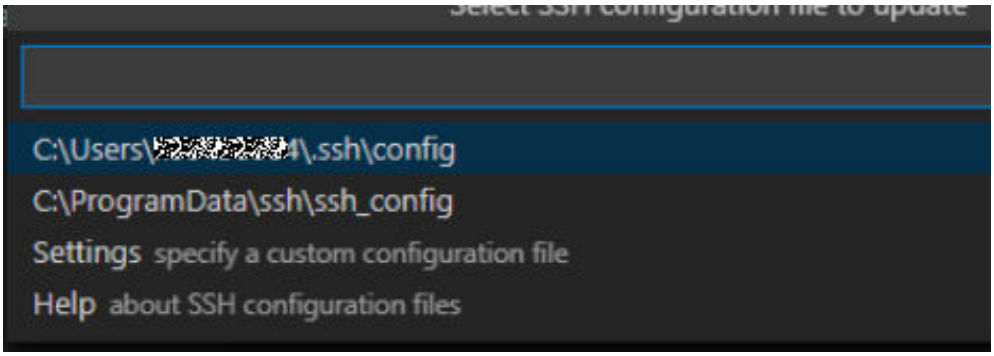
1. 在本地的VS Code开发环境中，单击左侧Remote Explorer按钮，在上方的下拉列表中选择“SSH Target”，再单击页面上的设置按钮，此时会出现SSH配置文件路径。

图 6-125 配置 SSH Targets 页面



2. 单击列表中出现的SSH路径按钮，打开config文件，进行配置。

图 6-126 配置 SSH Config 文件



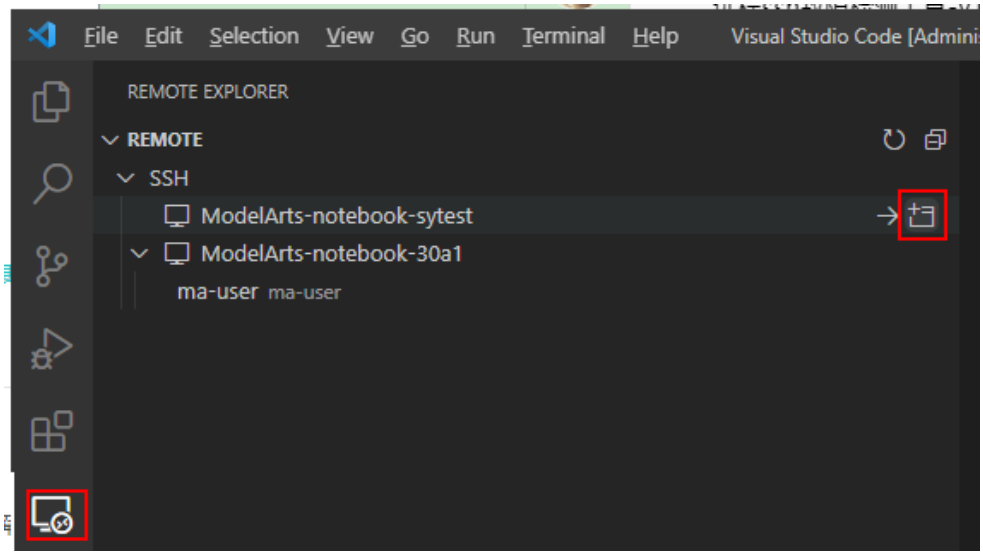

```
HOST remote-dev
    hostname <instance connection host>
    port <instance connection port>
    user ma-user
    IdentityFile ~/.ssh/test.pem
```

```
UserKnownHostsFile=/dev/null
StrictHostKeyChecking no
```

- Host: 自定义设置的云上开发环境名称。
- HostName: 云上开发环境的访问地址，即在开发环境实例页面远程访问模块获取的访问地址。
- Port: 云上开发环境的端口，即在开发环境实例页面远程访问模块获取的端口号。
- User：登录用户只支持ma-user进行登录。
- IdentityFile: 存放在本地的云上开发环境私钥文件，即前提条件**准备好密钥对**中准备的密钥对。

3. 再回到SSH Targets页面，选择远程开发环境名称，单击右侧的Connect to Host in New Window按钮。

图 6-127 打开开发环境



在新打开的选择Notebook运行环境页面中，选择“Linux”。

图 6-128 选择 Notebook 运行环境

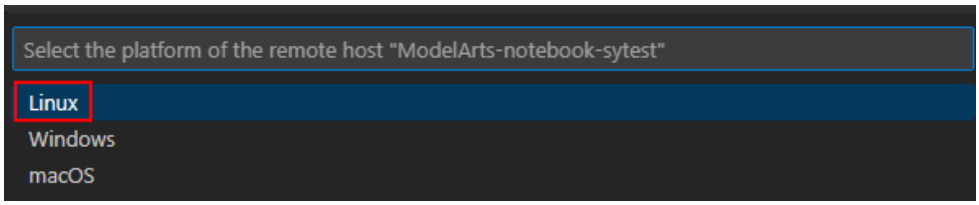
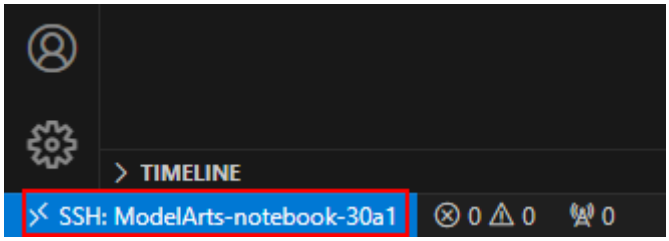


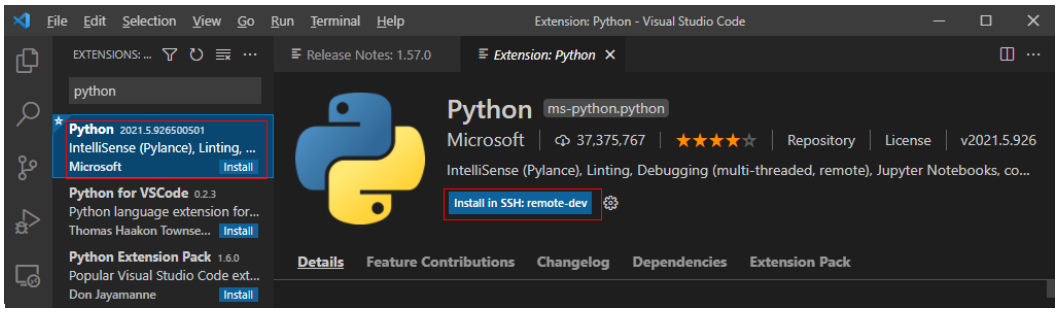
图 6-129 开发环境远程连接成功



Step3 安装云端 Python 插件

在新打开的VS Code界面，单击左侧列表的Extensions选项  ，在搜索框中输入Python，在下拉列表中单击“Install”进行安装。

图 6-130 安装云端 Python 插件



如果安装云端的Python插件不成功时，建议通过离线包的方式安装。

Step4 云上环境依赖库安装

在进入容器环境后，可以使用不同的虚拟环境，例如TensorFlow、PyTorch等，但是实际开发中，通常还需要安装其他依赖包，此时可以通过Terminal连接到环境里操作。

1. 在VS Code环境中，执行Ctrl+Shift+P。
2. 搜Python: Select Interpreter，选择对应的Python环境。
3. 单击页面上方的“Terminal > New Terminal”，此时打开的命令行界面即为远端容器环境命令行。
4. 进入引擎后，通过执行如下命令安装依赖包。
   ```
   pip install spacy
   ```

6.5.5 在 VS Code 中上传下载文件

在 VS Code 中上传数据至 Notebook

不大于500MB数据量，直接复制至本地IDE中即可。

大于500MB数据量，请先上传到OBS中，再从OBS上传到云上开发环境。

操作步骤

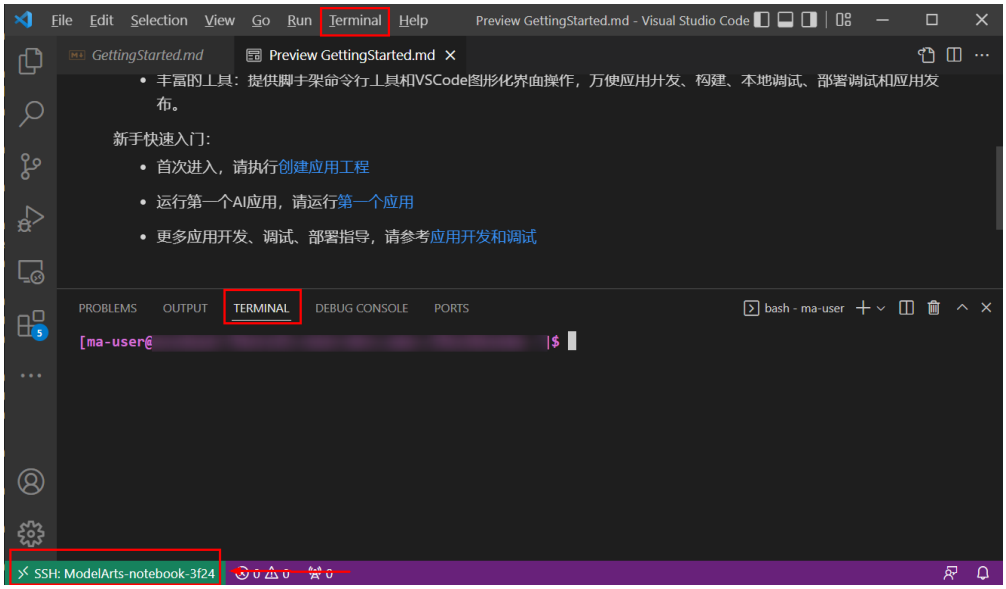
1. 上传数据至OBS。具体操作请参见**上传文件至OBS桶**。

   或者在本地VS Code的Terminal中使用ModelArts SDK完成数据上传至OBS。首先在本地VS Code中单击上方菜单栏的“Terminal”。在Terminal中输入python并回车，进入python环境。

   ```
   python
   ```

   然后在本地VS Code的Terminal中使用ModelArts SDK上传本地文件至OBS，详情请参考**文件传输**进行OBS传输操作。
2. 上传OBS文件到Notebook。在远程连接VS Code的Terminal中使用ModelArts SDK上传OBS文件到Notebook的操作示例如下：

图 6-131 远程连接 VS Code 环境开启 Terminal



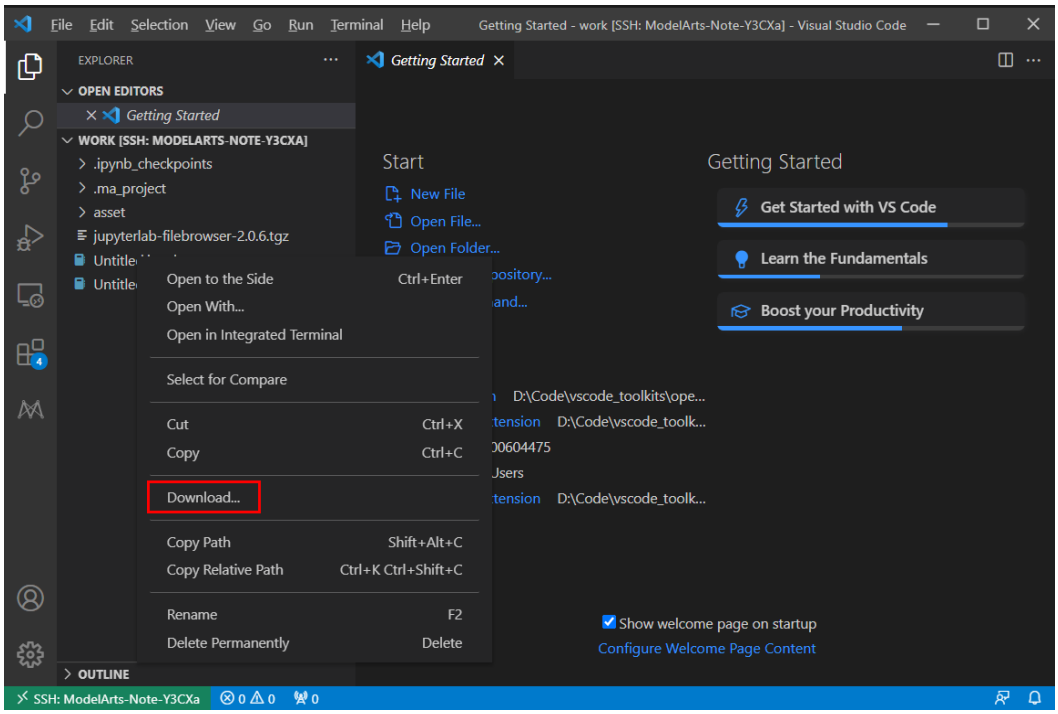

```
#手动source进入开发环境
cat /home/ma-user/README
#然后选择要source的环境
source /home/ma-user/miniconda3/bin/activate MindSpore-python3.7-aarch64
#输入python并回车，进入python环境
python
```

然后参考**上传文件至OBS**进行OBS传输操作。

下载 Notebook 中的文件至本地

在Notebook中开发的文件，可以下载至本地。在本地IDE的Project目录下的Notebook2.0工程单击右键，单击“Download...”将文件下载到本地。

图 6-132 VS Code 环境下载 Notebook 中的文件至本地



6.6 通过 SSH 工具远程使用 Notebook

本节操作介绍在Windows环境中使用PuTTY SSH远程登录云上Notebook实例的操作步骤。

前提条件

- 创建一个Notebook实例，并开启远程SSH开发，配置远程访问IP白名单。该实例状态必须处于“运行中”，具体参见**创建Notebook实例**章节。
- 在Notebook实例详情页面获取开发环境访问地址和端口号。

  图 6-133 Notebook 实例详情页面

- 准备好密钥对文件。

  密钥对在用户第一次创建时，自动下载，之后使用相同的密钥时不会再有下载界面（用户一定要保存好），或者每次都使用新的密钥对。

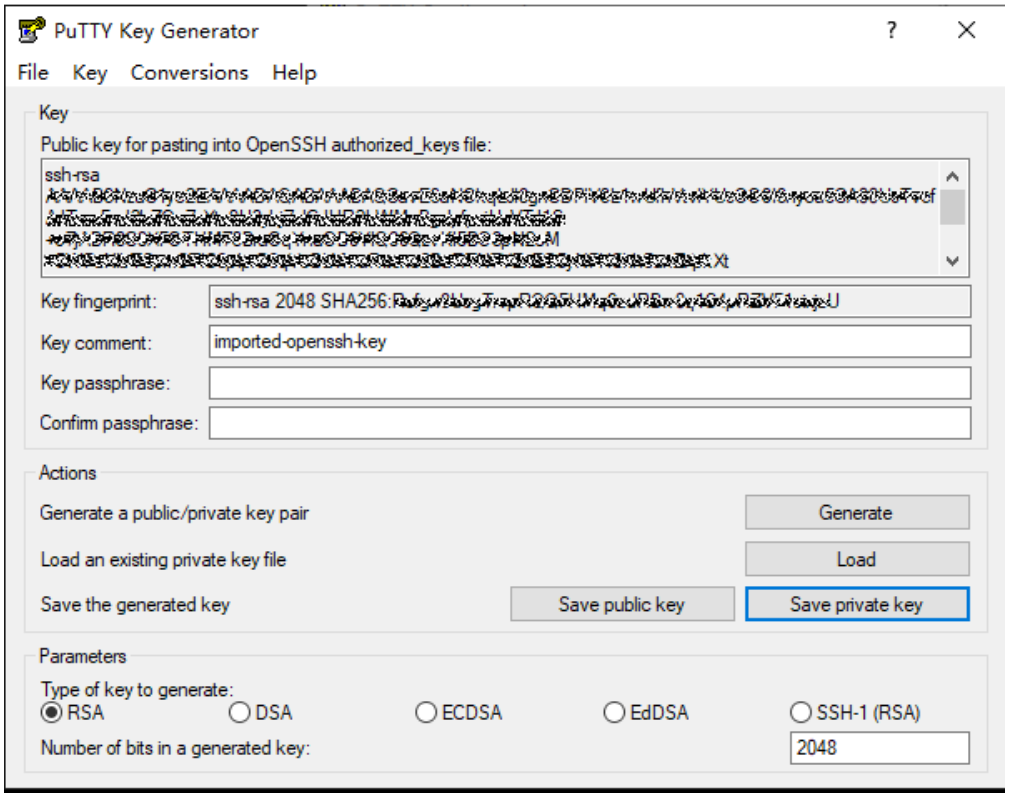
Step1 安装 SSH 工具

下载并安装SSH远程连接工具，以PuTTY为例，**下载链接**。

Step2 使用 puttygen 将密钥对.pem 文件转成.ppk 文件

1. 下载puttygen，并双击运行puttygen。
2. 单击“Load”，上传.pem密钥（即在创建Notebook实例时创建并保存的密钥对文件）。
3. 单击“Save private key”，保存生成的.ppk文件。.ppk文件的名字可以自定义，例如key.ppk。

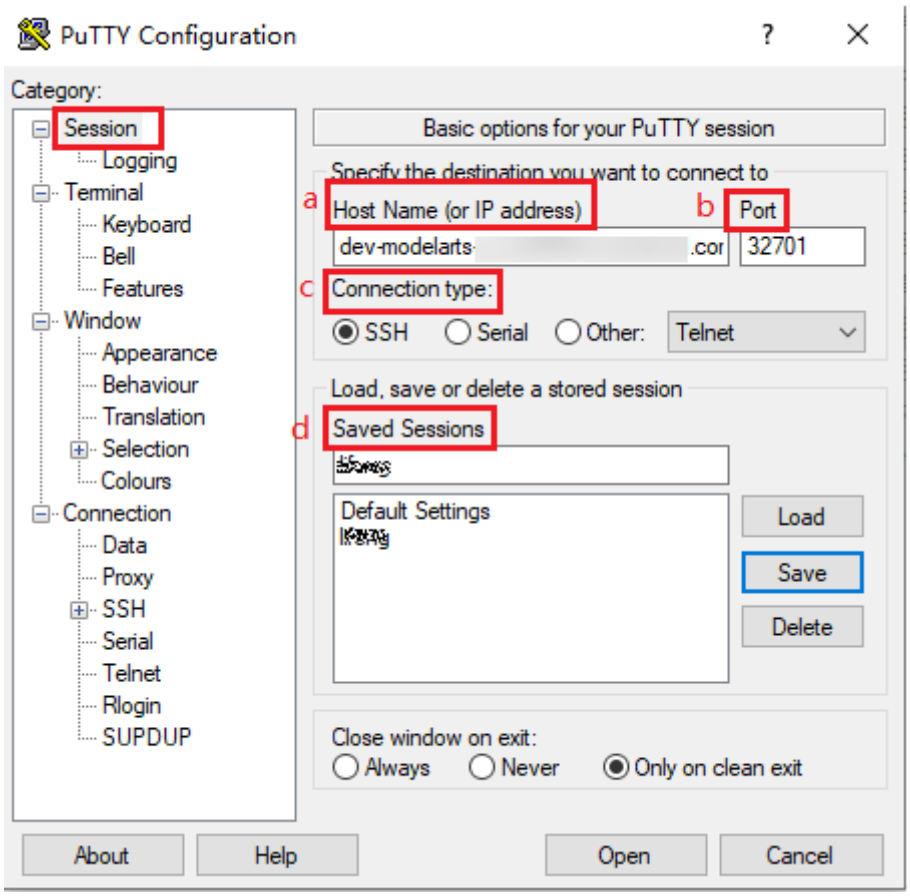
图 6-134 将密钥对.pem 文件转成.ppk 文件



Step3 使用 SSH 工具连接云上 Notebook 实例

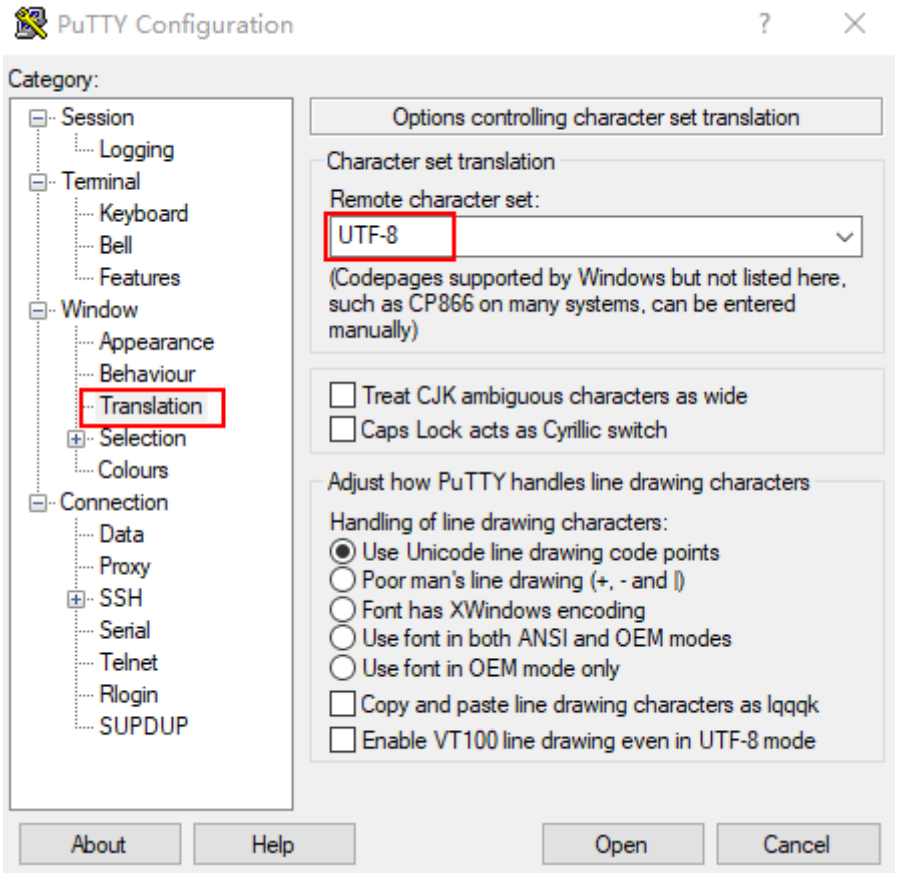
1. 运行PuTTY。
2. 单击“Session”，填写以下参数。
   a. Host Name (or IP address)：云上开发环境Notebook实例的访问地址，即在Notebook实例详情页获取的地址。
   b. Port：云上Notebook实例的端口，即在Notebook实例详情页获取的端口号。例如：32701。
   c. Connection Type：选择SSH。
   d. Saved Sessions：任务名称，在下一次使用PuTTY时就可以单击保存的任务名称，即可打开远程连接。

图 6-135 设置 Session



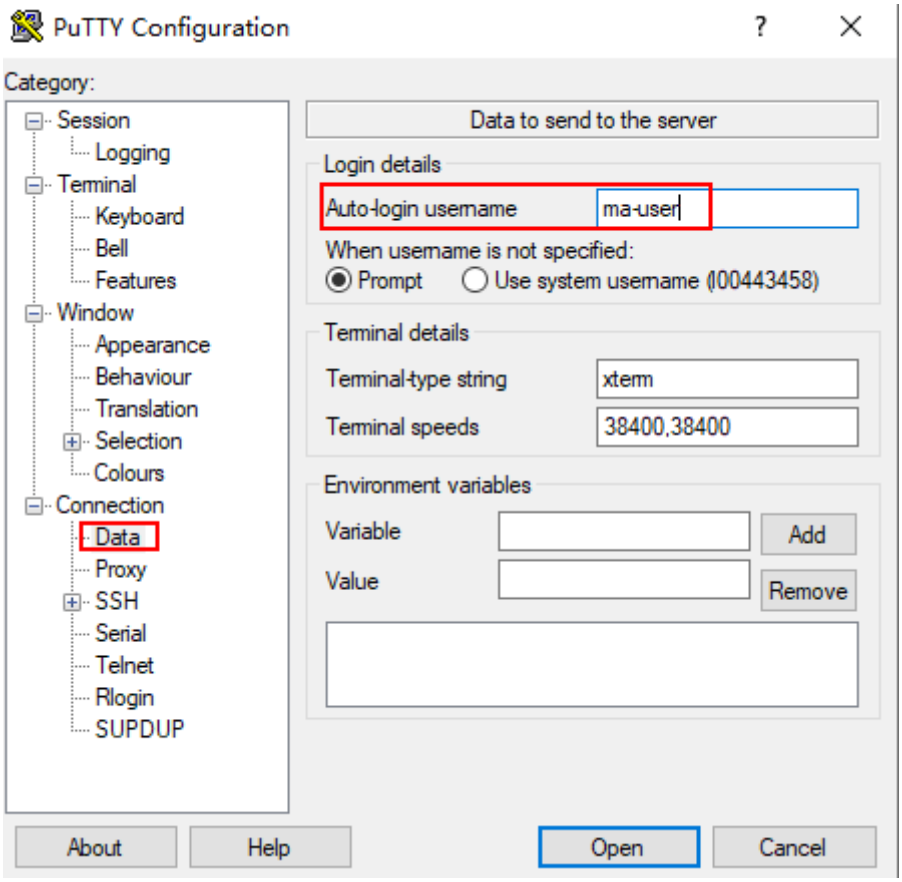
3. 选择“Window > Translation”，在“Remote character set:”中选择“UTF-8”。

图 6-136 设置字符格式

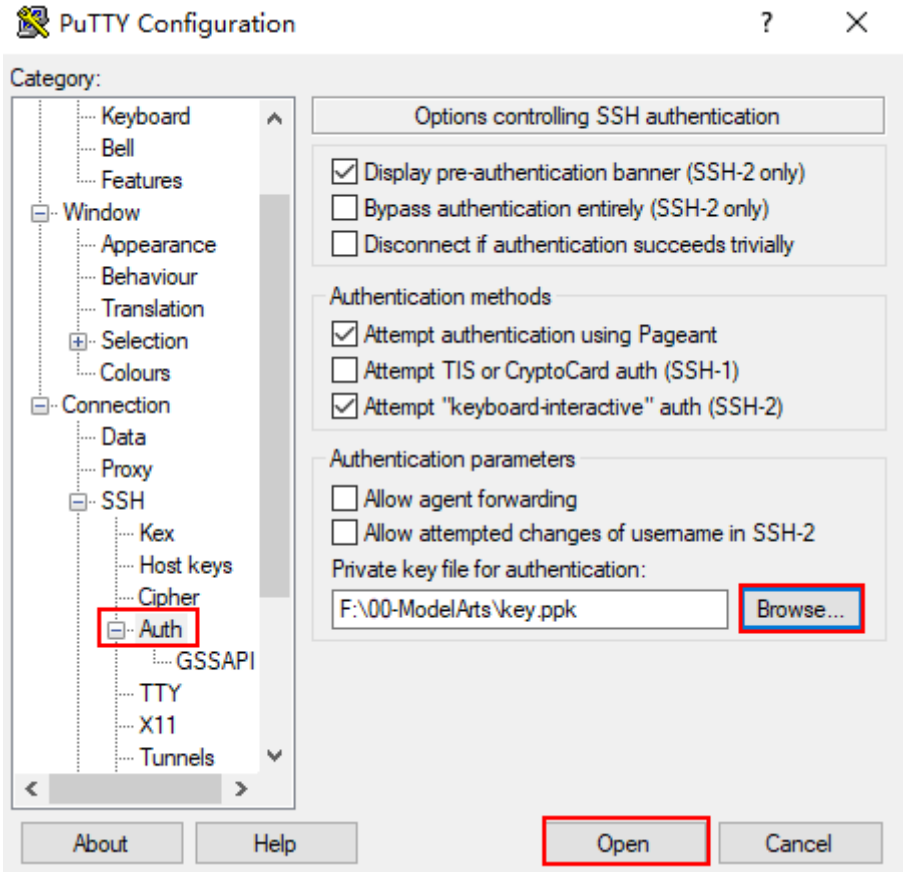


4. 选择“Connection > Data”， 在“Auto-login username” 中填写用户名“ma-user”。

图 6-137 填写用户名

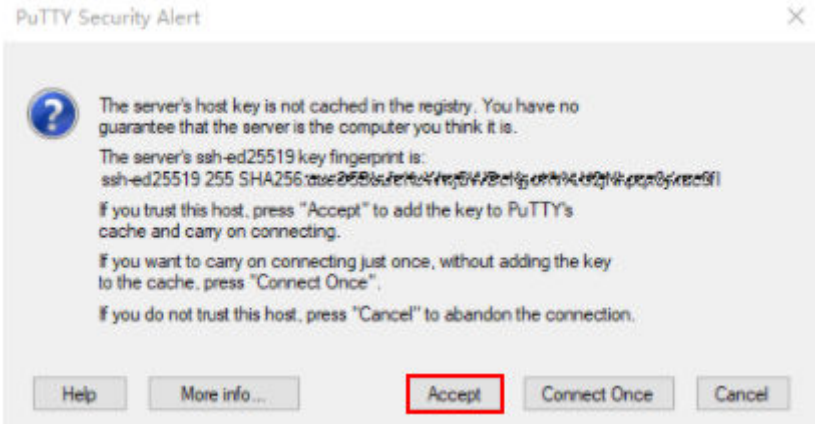


5. 选择“Connection > SSH > Auth”，单击“Browse”，选择“.ppk文件”（由**Step2**密钥对.pem文件生成）。



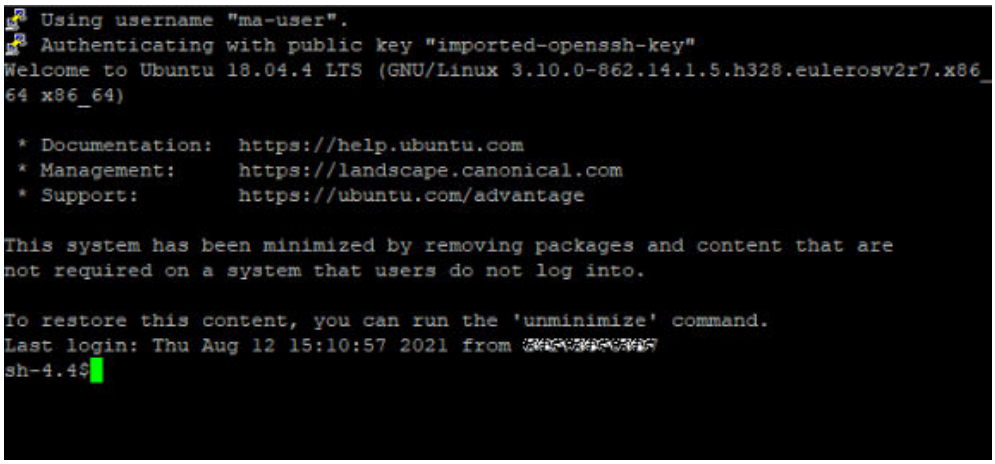
6. 单击“Open”。如果首次登录，PuTTY会显示安全警告对话框，询问是否接受服务器的安全证书。单击“Accept”将证书保存到本地注册表中。

图 6-138 询问是否接受服务器的安全证书



7. 成功连接到云上Notebook实例。

图 6-139 连接到云上 Notebook 实例



6.7 管理 Notebook 实例

6.7.1 查找 Notebook 实例

查找实例

Notebook页面展示了所有创建的实例。如果需要查找特定的实例，可根据筛选条件快速查找。

- 参考**给子账号配置查看所有Notebook实例的权限**后，进入“开发空间>Notebook”页面，打开“查看所有”开关，可以看到IAM项目下所有子账号创建的Notebook实例。
- 按实例名称、实例ID、实例状态、使用的镜像、实例规格、实例描述、创建时间等单个筛选或组合筛选。

给子账号配置查看所有 Notebook 实例的权限

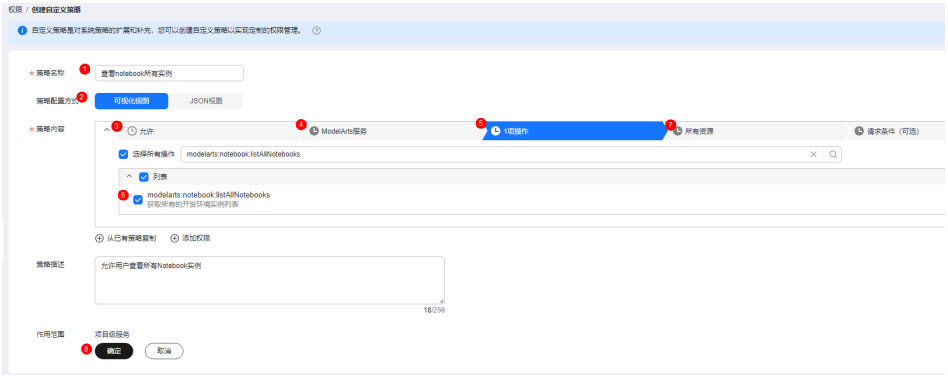
当子账号被授予“listAllNotebooks”和“listUsers”权限时，在Notebook页面上，单击“查看所有”，可以看到IAM项目下所有子账号创建的Notebook实例 。配置该权限后，也可以在Notebook中访问子账号的OBS、SWR等。

1. 使用主用户账号登录ModelArts管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，需要设置两条策略。

   策略1：设置查看Notebook所有实例，如**图6-140**所示，单击“确定”。

   - “策略名称”：设置自定义策略名称，例如：查看Notebook所有实例。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:notebook:listAllNotebooks并选中，所有资源选择默认值。

图 6-140 创建自定义策略



策略2：设置查看Notebook实例创建者信息的策略。

- “策略名称”：设置自定义策略名称，例如：查看所有子账号信息。
- “策略配置方式”：选择可视化视图。
- “策略内容”：允许，云服务中搜索IAM服务并选中，操作列中搜索关键词iam:users:listUsers并选中，所有资源选择默认值。

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的两条自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限查看该用户组内成员创建的所有Notebook实例。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子账号没有在用户组中，也可以通过“用户组管理”功能增加用户。

子账号启动其他用户的 SSH 实例

子账号可以看到所有用户的Notebook实例后，如果要通过SSH方式远程连接其他用户的Notebook实例，需要将SSH密钥对更新成自己的，否则会报错ModelArts.6786。更新密钥对具体操作请参见**修改Notebook SSH远程连接配置**。具体的错误信息提示：ModelArts.6789: 在ECS密钥对管理中找不到指定的ssh密钥对xxx，请更新密钥对并重试。

6.7.2 更新 Notebook 实例

变更镜像

ModelArts允许用户在同一个Notebook实例中切换镜像，方便用户灵活调整实例的AI引擎。Notebook实例状态需在“停止”中才可以变更镜像。

说明

请注意，变更镜像后可能会导致Notebook实例无法启动，镜像对应的Notebook实例规格不匹配，对应的收费规则也会随着镜像的变更而变化，请谨慎操作。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“开发环境 > Notebook”，进入Notebook页面。
2. 在Notebook列表，单击某个Notebook实例操作栏的“更多 > 变更镜像”，在变更镜像窗口选择新的镜像，单击“确定”。

3. 在镜像窗口选择新的镜像，单击“确定”，变更成功后，在Notebook列表页的镜像栏，可以查看到变更后的镜像。

变更 Notebook 实例运行规格

ModelArts允许用户在同一个Notebook实例中切换节点运行规格，方便用户灵活调整规格资源。只有处于“停止”、“运行中”和“启动失败”的Notebook实例才能变更规格。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“开发环境 > Notebook”，进入Notebook页面。
2. 在Notebook列表，单击某个Notebook实例“操作”列的“更多 > 变更实例规格”，在弹出的“变更实例规格”对话框中选择对应规格资源。

图 6-141 选择实例规格

说明

实例规格切换需要该规格所在的集群有其他规格才可以执行，当前上线的部分规格所在集群无其他规格，切换的时候会显示为空，所以不可进行切换。

修改 Notebook SSH 远程连接配置

ModelArts允许用户在Notebook实例中更改SSH配置信息，Notebook实例状态需在“停止”时才可以修改。

在创建Notebook实例时，未配置SSH远程连接，创建完成后，需要开启远程连接时，则可以在Notebook的实例详情页打开SSH的配置信息开关；如果用户设置了允许远程连接Notebook的白名单IP地址，当需要更换一个IP地址远程连接Notebook实例时，则可以在Notebook的实例详情页修改白名单IP地址；如果子用户需要启动别人的Notebook实例时，也可更换密钥对。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“开发环境 > Notebook”，进入Notebook页面。
2. 在Notebook列表，进入Notebook实例详情页。打开SSH远程开发开关，更新密钥对和白名单。

说明

“远程SSH开发”开关可以手动打开的场景，请打开远程SSH开发开关，参考图6-142操作。SSH配置信息更新后，“远程SSH开发”开关打开后不可关闭。

“所选镜像必须配置SSH远程开发”的场景，直接单击“白名单”或“认证”后的 修改。

图 6-142 更新 SSH 配置信息

- 密钥对可单击 选择已有的密钥对或“立即创建”创建新的密钥对。
- 修改远程连接的可访问IP地址后， 原来已经建立的链接依然有效， 当链接关闭后失效；新打开建立的链接只允许当前设置的IP进行访问。
- 此处的IP地址，请填写外网IP地址。如果用户使用的访问机器和华为云ModelArts服务的网络有隔离，则访问机器的外网地址需要在主流搜索引擎中搜索“IP地址查询”获取，而不是使用ipconfig或ifconfig/ip命令在本地查询。

6.7.3 启动/停止/删除实例

启动/停止实例

由于运行中的Notebook将一直耗费资源，您可以通过停止操作，停止资源消耗。对于停止状态的Notebook，可通过启动操作重新使用Notebook。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“开发空间 > Notebook”，进入Notebook管理页面。
2. 执行如下操作启动或停止Notebook。
   - **启动Notebook**：单击“操作”列的“启动”。只有处于“停止”状态的Notebook可以执行启动操作。
   - **停止Notebook**：单击“操作”列的“停止”。只有处于“运行中”状态的Notebook可以执行停止操作。

⚠ 注意

Notebook停止后：

- “/home/ma-user/work”目录以及动态挂载在“/data”下的目录下的数据会保存，其余目录下内容会被清理。例如：用户在开发环境中的其他目录下安装的外部依赖包等，在Notebook停止后会被清理。您可以通过保存镜像的方式保留开发环境设置，具体操作请参考**保存Notebook实例**。
- Notebook实例将停止计费，但如有EVS盘挂载，存储部分仍会继续计费。

删除实例

针对不再使用的Notebook实例，可以删除以释放资源。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“开发空间>Notebook”，进入Notebook页面。
2. 在Notebook列表中，单击操作列的“删除”，在弹出的确认对话框中，确认信息无误，然后输入“DELETE”，单击“确定”，完成删除操作。

⚠ 注意

Notebook删除后不可恢复，请谨慎操作。实例删除后，挂载目录下的数据也将一并删除，请谨慎操作。

6.7.4 保存 Notebook 实例

通过预置的镜像创建Notebook实例，在基础镜像上安装对应的自定义软件和依赖，在管理页面上进行操作，进而完成将运行的实例环境以容器镜像的方式保存下来。镜像保存后，默认工作目录是根目录“/”路径。

保存的镜像中，安装的依赖包不丢失，持久化存储的部分（home/ma-user/work目录的内容）不会保存在最终产生的容器镜像中。VS Code远程开发场景下，在Server端安装的插件不丢失。

说明

当镜像保存失败时，请在Notebook实例详情页查看事件，事件描述请参考**查看Notebook实例事件**。

建议保存的镜像大小不要超过35G，镜像层数不要超过125层，因为节点容器存储Rootfs差异（详细请参考**容器引擎空间分配**），可能会导致镜像保存失败。

- 如使用的是专属资源池，可尝试在“专属资源池>弹性集群”页面按需调整容器引擎空间大小，具体步骤请参考**扩缩容专属资源池的“修改容器引擎空间大小”**。
- 如果问题仍未解决，请联系技术支持。

前提条件

Notebook实例状态为“运行中”。

保存镜像

1. 在Notebook列表中，对于要保存的Notebook实例，单击右侧“操作”列中的“更多 > 保存镜像”，进入“保存镜像”对话框。

图 6-143 保存镜像

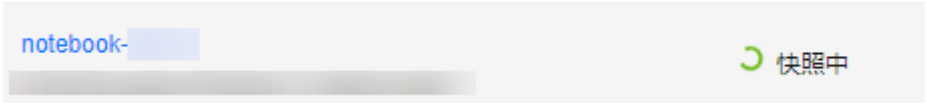
2. 在保存镜像对话框中，设置组织、镜像名称、镜像版本和描述信息。单击“确定”保存镜像。

   在“组织”下拉框中选择一个组织。如果没有组织，可以单击右侧的“立即创建”，创建一个组织。

   同一个组织内的用户可以共享使用该组织内的所有镜像。

3. 镜像会以快照的形式保存，保存过程约5分钟，请耐心等待。此时不可再操作实例。

图 6-144 保存镜像



须知

快照中耗费的时间仍占用实例的总运行时长，如果在快照中时，实例因运行时间到期停止，将导致镜像保存失败。

4. 镜像保存成功后，实例状态变为“运行中”，用户可在“镜像管理”页面查看到该镜像详情。
5. 单击镜像的名称，进入镜像详情页，可以查看镜像版本/ID，状态，资源类型，镜像大小，SWR地址等。

基于自定义镜像创建 Notebook 实例

从Notebook中保存的镜像可以在镜像管理中查询到，可以用于创建新的Notebook实例，完全继承保存状态下的实例软件环境配置。

方式一：在Notebook实例创建页面，镜像类型选择“自定义镜像”，名称选择上述保存的镜像。

图 6-145 创建基于自定义镜像的 Notebook 实例

方式二：在“镜像管理”页面，单击某个镜像的镜像详情，在镜像详情页，单击“创建Notebook”，也会跳转到基于该自定义镜像创建Notebook的页面。

镜像保存时，哪些目录的数据可以被保存

- 可以保存的目录：包括容器构建时静态添加到镜像中的文件和目录，可以保存在镜像环境里。

  例如：安装的依赖包、“/home/ma-user”目录
- 不会被保存的目录：容器启动时动态连接到宿主机的挂载目录或数据卷，这些内容不会被保存在镜像中。可以通过**df -h**命令查看挂载的动态目录，非“/”路径下的不会保存。

  例如：持久化存储的部分“home/ma-user/work”目录的内容不会保存在最终产生的容器镜像中、动态挂载在“/data”下的目录不会被保存。

6.7.5 动态扩充云硬盘 EVS 容量

什么是动态扩容 EVS

存储配置采用云硬盘EVS的Notebook实例， 存储盘是挂载至容器/home/ma-user/work/目录下， 可以在实例运行中的状态下，动态扩充存储盘容量，单次最大动态扩容100GB。

动态扩容 EVS 适用于哪些使用场景

在Notebook开发过程中，初期存储使用量较小时， 创建Notebook可以选择小容量EVS， 比如5G大小； 开发完成后，需要大规模数据集训练，此时再将存储容量扩容至当前阶段所需容量，可以节约成本。

动态扩容 EVS 有什么限制

- Notebook实例的存储配置采用的是云硬盘EVS。

  图 6-146 创建 Notebook 实例时选择云硬盘 EVS 存储

- 单次最大可以扩容100GB，扩容后的总容量不超过4096GB。
- 云硬盘EVS存储容量最大支持4096GB，达到4096GB时，不允许再扩容。

- 实例停止后，扩容后的容量仍然有效。计费也是按照扩容后的云硬盘EVS容量进行计费。
- 云硬盘EVS只要使用就会计费，请在停止Notebook实例后，确认不使用就及时删除数据，释放资源，避免产生费用。

动态扩容 EVS 操作

1. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 选择运行中的Notebook实例，单击实例名称，进入Notebook实例详情页面，单击“扩容”。

图 6-147 Notebook 实例详情页

3. 设置待扩充的存储容量大小，单击“确定”。系统显示“扩容中”，扩容成功后，可以看到扩容后的存储容量。

图 6-148 扩容

图 6-149 扩容中

6.7.6 动态挂载 OBS 并行文件系统

什么是动态挂载 OBS 并行文件系统

并行文件系统（Parallel File System）是对象存储服务（Object Storage Service，OBS）提供的一种经过优化的高性能文件系统，详细介绍可以参见**并行文件系统**。

在ModelArts运行态的Notebook容器中，采用动态挂载特性，将OBS对象存储模拟成本地文件系统。其本质是通过挂载工具，将对象协议转为POSIX文件协议。挂载后应用层可以在容器中正常操作OBS对象。

动态挂载适用于哪些使用场景

场景1：数据集预览和操作，将承载数据集的OBS挂载至Notebook中，可以像本地文件系统一样操作数据集。

场景2：在Notebook中训练时，可直接使用挂载至Notebook容器中的数据集。

动态挂载 OBS 并行文件系统有什么限制

OBS提供两种桶，对象存储（对象桶）和并行文件系统PFS。

ModelArts的Notebook仅支持挂载OBS的**并行文件系统**，挂载至Notebook容器“/data/”的子目录下。

动态挂载 OBS 并行文件系统操作

方式1：通过ModelArts控制台操作

1. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 选择运行中的Notebook实例，单击实例名称，进入Notebook实例详情页面，在“存储配置”页签，单击“添加数据存储”，设置挂载参数。
   a. 设置本地挂载目录，在“/data/”目录下输入一个文件夹名称，例如：demo。挂载时，后台自动会在Notebook容器的“/data/”目录下创建该文件夹，用来挂载OBS文件系统。
   b. 选择存放OBS并行文件系统下的文件夹，单击“确定”。

图 6-150 动态挂载 OBS 并行文件系统

3. 挂载成功后，可以在Notebook实例详情页查看到挂载结果。

图 6-151 挂载成功

6.7.7 查看 Notebook 实例事件

在Notebook的整个生命周期，包括实例的创建、启动、停止、规格变更等关键操作以及实例的运行状态等在后台都有记录，用户可以在Notebook实例详情页中查看具体的事件，通过实例的事件，从而看到实例的运行或者异常等状态详情。在右侧可以手动刷新事件，也可以设置间隔30秒，1分钟，5分钟自动刷新事件。

查看 Notebook 实例事件的方法

单击Notebook名称，进入Notebook详情页，单击“事件”。

Notebook 实例事件列表

表 6-12 实例创建过程的事件列表

事件名称	事件描述	事件级别
Scheduled	实例被调度成功	提示
PullingImage	正在拉取镜像	提示
PulledImage	镜像拉取完毕	提示
NotebookHealthy	实例运行中，处于健康状态	重要
CreateNotebookFailed	创建实例失败	紧急
PullImageFailed	镜像拉取失败	紧急
FailedCreate	Failed to create notebook container. Please contact SRE to check node {node_name}	紧急

事件名称	事件描述	事件级别
CreateContainerError	Failed to create container. Please contact SRE to check node {node_name}	紧急
FailedAttachVolume	Failed to attach volume. Please contact SRE to check node {node_name}	重要
MountVolumeFailed	Mount volume failed; Check whether the DEW secret is correct if the instance cannot change to running in five minutes	紧急
	Mount volume failed; Check if vpc of sfs-turbo is interconnected if the instance cannot change to running in five minutes	紧急
	Mount volume failed; Please contact SRE to check node {node_name} if the instance cannot change to running in five minutes	紧急

表 6-13 实例停止过程的事件列表

事件名称	事件描述	事件级别
StopNotebook	实例停止	重要
StopNotebookResourceIdle	实例因资源空闲即将自动停止或实例因资源空闲自动停止	重要

表 6-14 更新实例过程的事件列表

事件名称	事件描述	事件级别
UpdateName	更新实例名称	提示
UpdateDescription	更新实例描述	提示
UpdateFlavor	更新实例规格	重要
UpdateImage	更新实例镜像	重要

事件名称	事件描述	事件级别
UpdateStorageSize	实例存储正在扩容 （User %s is updating storage size from %sGB to %sGB）	重要
	实例扩容完成 （User %s updated storage size successfully）	重要
UpdateKeyPair	配置实例密钥对 （User %s updated the instance keypair to "{%s}"）	重要
	更新实例密钥对 （User %s updated the instance keypair from %s to %s）	重要
UpdateWhitelist	更新实例访问白名单	重要
UpdateHook	更新自定义脚本	重要
UpdateStorageSizeFailed	资源售罄引起的实例存储扩容失败 （The EVS disk is sold out）	紧急
	内部错误引起的实例扩容失败 （The EVS disk size updated failed. Operations and maintenance personnel are handling the problem）	紧急

表 6-15 镜像保存过程中的事件列表

事件名称	事件描述	事件级别
SaveImage	保存镜像成功	重要
SavedImageFailed	D进程引起的保存镜像失败 （There are processes in 'D' status, please check process status using 'ps -aux' and kill all the 'D' status processes）	紧急

事件名称	事件描述	事件级别
	镜像大小引起的保存镜像失败 （Container size %dG is greater than threshold %dG）	紧急
	层数限制引起的保存镜像失败 （Too many layers in your image）	紧急
	任务超时引起的保存镜像失败 （Operations personnel are handling the problem）	紧急
	SWR故障引起的保存镜像失败 （Failed to save the image because the SWR service is faulty）	紧急
CheckImageSize	The notebook container image size is {image_size}G. {image_size} 表示镜像大小，为可变变量。	提示
CheckImageLayer	The number of original notebook image layers is {layer_number}. {layer_number} 表示镜像层数，为可变变量。	提示
ContainerCommitStarted	Start to commit notebook container.	提示
ContainerCommitSuccess	Notebook container commit successfully.	提示
ImagePushStarted	Start to push notebook image.	提示
ImagePushSuccess	Notebook image push successfully.	提示

事件名称	事件描述	事件级别
ContainerCommitFailed	Failed to commit notebook container. Please contact SRE to check node {node_name}. {node_name}表示节点名称，为可变变量，一般为ip形式，如：192.168.225.161	提示
ImagePushFailed	Failed to push Notebook image. Please contact SRE to check node {node_name}.	提示

表 6-16 实例运行过程的事件列表

事件名称	事件描述	事件级别
NotebookUnhealthy	实例处于不健康状态	紧急
OutOfMemory	实例被OOM掉了	紧急
JupyterProcessKilled	jupyter进程被killed掉了	紧急
CacheVolumeExceedQuota	/cache目录文件大小超过最大限制	紧急
NotebookHealthy	实例从不健康恢复到了健康状态	重要
EVSSoldOut	EVS存储售罄	紧急

表 6-17 OBS 动态挂载产生的事件列表

事件名称	事件描述	事件级别
DynamicMountStorage	挂载OBS存储	重要
DynamicUnmountStorage	卸载OBS存储	重要

表 6-18 用户侧触发的事件

事件名称	事件描述	事件级别
RefreshCredentialsFailed	用户鉴权失败	紧急

6.7.8 Notebook Cache 盘告警上报

创建Notebook时，可以根据业务数据量的大小选择CPU、GPU或者Ascend资源，对GPU或Ascend类型的资源，ModelArts会挂载硬盘至“/cache”目录，用户可以使用此目录来储存临时文件。

当前开发环境的Cache盘使用时，没有容量告警，在使用时很容易超过限制，并直接重启Notebook实例。重启后多种配置重置，会导致用户数据丢弃，环境丢失，造成很不好的使用体验。因此需要提供cache盘使用情况的监控和告警，并将数据上报至AOM平台。

配置流程

1. 填写告警基本信息
2. **设置告警规则**
   a. 监控对象指标配置
   b. 告警触发条件设置
3. **告警通知设置**
   a. 创建主题、设置主题策略、订阅主题
   b. 创建告警行动规则
   c. 选择已创建的行动规则

告警上报配置方法

1. 登录AOM控制台。
2. 单击“告警 > 告警规则”，在“告警规则”界面，单击“添加告警”。
3. 填写告警基本信息。

4. 设置告警规则。
   “规则类型”选择“阈值规则”。
   “监控对象”：选择“选择资源对象”。单击选择资源对象，弹出新窗口。
   - 添加方式：选择“按指标维度添加”。
   - 指标名称：选择“全量指标”，搜索需要监控的cache指标名称然后选中。例如：ma_container_notebook_cache_dir_size_bytes（cache目录的总大小）、ma_container_notebook_cache_dir_util（cache目录的利用率）
   - 指标维度：根据实际需求选择相应的指标维度。例如service_id:xxx，然后单击“确定”。

监控对象设置完成后，选择“统计方式”和“统计周期”。

“告警条件设置”：触发条件根据实际需求设置。

图 6-152 监控对象指标设置

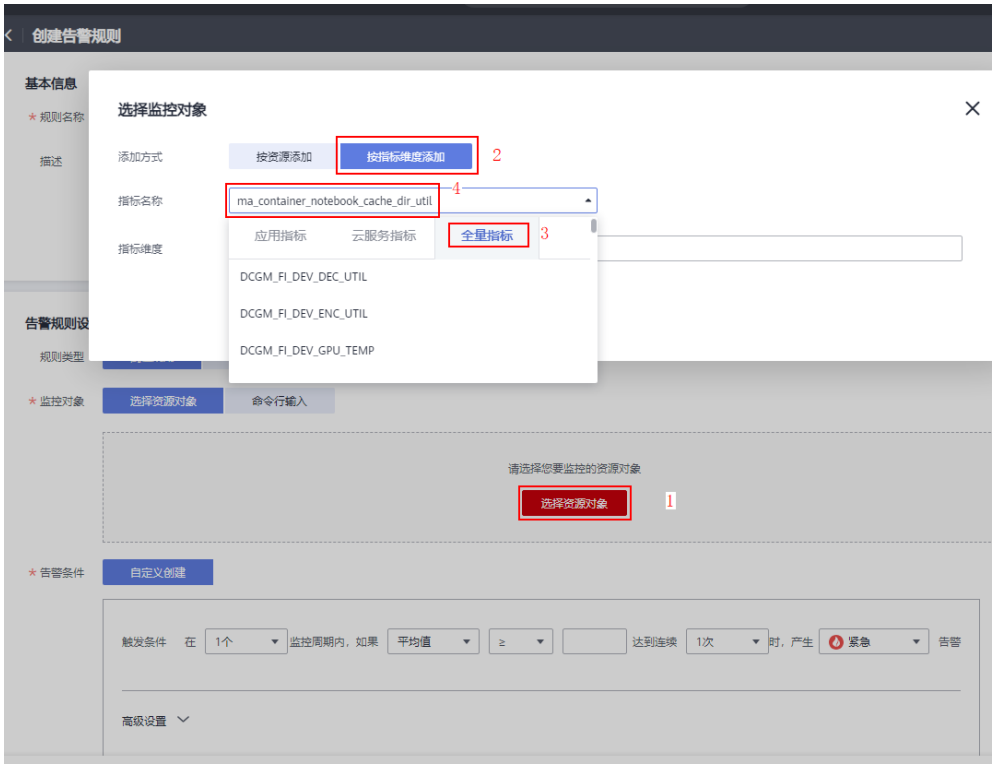


图 6-153 设置指标统计方式

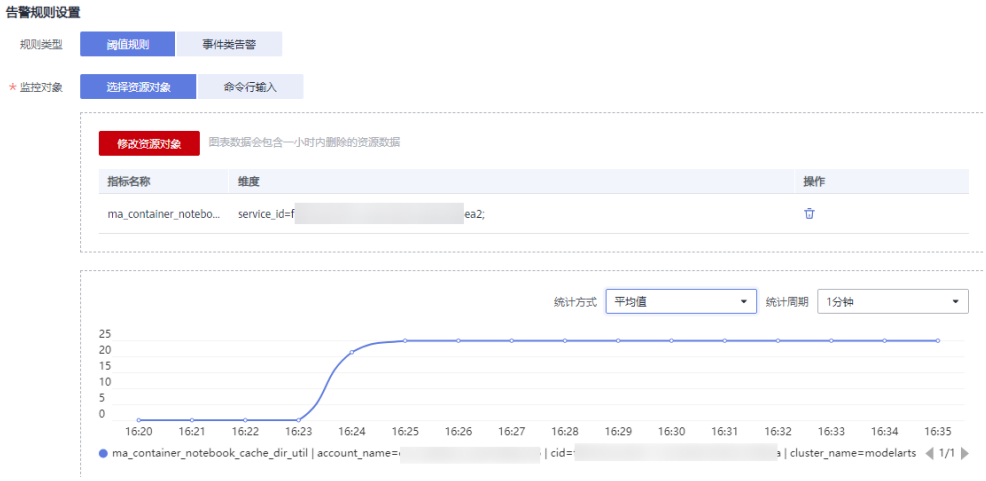


图 6-154 告警条件设置

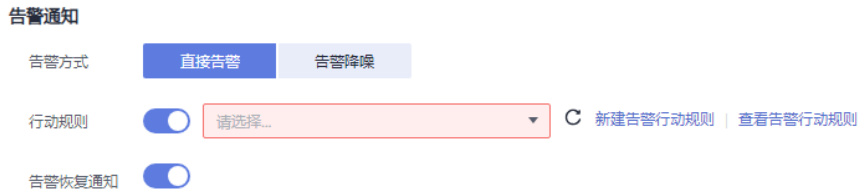
5. 设置告警通知，单击“立即创建”。

   “告警方式”：选择“直接告警”

   “行动规则”：开启开关，选择已创建的行动规则。如果现有列表中的告警行动规则无法满足需要，可单击“新建告警行动规则”添加，详细操作请参考**创建告警行动规则**。

   “告警恢复通知”：开启开关

   图 6-155 设置告警通知

   先在SMN创建一个主题，用于配置告警通知规则。

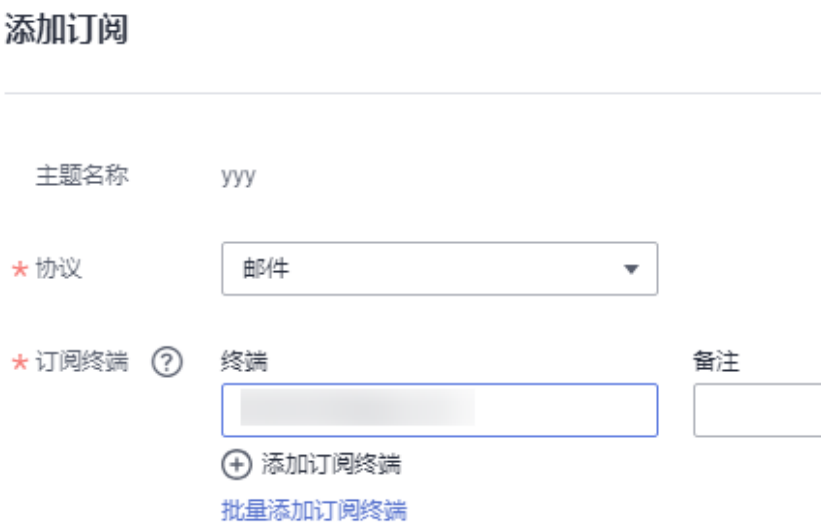
   - **创建主题**
     i. 进入“消息通知服务”控制台，单击“主题管理 > 主题”，进入“主题”页面。
     ii. 单击“创建主题”填写主题名称，选择企业项目后，单击确定即可创建一个主题。
     iii. 单击主题名称“操作”列的“更多 > 设置主题策略”。

     选择APM，即允许AOM的告警触发SMN服务。

图 6-156 设置主题策略

iv. 单击主题名称“操作”列的“添加订阅”。订阅成功后，一旦满足告警条件，那么就会收到通知。

选择合适的协议，如邮件，短信等，并填写终端，如邮件地址，手机号等。单击确认。



此时订阅总数中会出现一条记录，但是处于未确认的状态。

收到邮件后单击“订阅确认”。

此时该订阅记录将处于已确认的状态。

- **创建告警行动规则**

  行动规则即为告警触发时，AOM以怎样的方式来告知用户。启用告警行动规则后，系统根据关联SMN主题与消息模板来发送告警通知。

  根据界面提示填写行动规则名称，选择行动规则类型，选择上一步创建的主题，选择消息模板，然后单击“确定”。

  图 6-157 新建告警行动规则

在之前打开的“创建告警规则”页面的告警通知区域，“行动规则”选择新创建的告警行动规则，单击“立即创建”。

至此，整个告警流程配置完成，一旦满足告警条件，那么就会收到邮件通知。

6.8 ModelArts CLI 命令参考

6.8.1 ModelArts CLI 命令功能介绍

功能介绍

ModelArts CLI，即ModelArts命令行工具，是一个跨平台命令行工具，用于连接ModelArts服务并在ModelArts资源上执行管理命令。用户可以使用交互式命令行提示符或脚本通过终端执行命令。为了方便理解，下面将ModelArts CLI统称为ma-cli。ma-cli支持用户在ModelArts Notebook及线下虚拟机中与云端服务交互，使用ma-cli命令可以实现命令自动补全、鉴权、镜像构建、提交ModelArts训练作业、提交DLI Spark作业、OBS数据复制等。

使用场景

- ma-cli已经集成在ModelArts开发环境Notebook中，可以直接使用。

  登录ModelArts控制台，在“开发空间 > Notebook”中创建Notebook实例，打开Terminal，使用ma-cli命令。

- ma-cli在本地Windows/Linux环境中需要安装后在本地Terminal中使用。安装步骤具体可参考**（可选）本地安装ma-cli**。

  说明

  - ma-cli不支持在git-bash上使用。
  - 推荐使用Linux Bash、ZSH、Fish，WSL或PowerShell等Terminal。在使用过程中，注意您的敏感信息数据保护，避免敏感信息泄露。

命令预览

```
$ ma-cli -h
Usage: ma-cli [OPTIONS] COMMAND [ARGS]...

Options:
  -V, -v, --version       1.2.1
  -C, --config-file TEXT  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help          Show this message and exit.

Commands:
  configure        Configures authentication and endpoints info for the CLI.
  image            Support get registered image list、register or unregister image、debug image, build image
in Notebook.
  obs-copy         Copy file or directory between OBS and local path.
  ma-job           ModelArts job submission and query job details.
  dli-job          DLI spark job submission and query job details.
  auto-completion  Auto complete ma-cli command in terminal, support "bash(default)/zsh/fish".
```

其中，-C、-D、-P，-h参数属于全局可选参数。

- -C表示在执行此命令时可以手动指定鉴权配置文件，默认使用~/.modelarts/ma-cli-profile.yaml配置文件；
- -P表示鉴权文件中的某一组鉴权信息，默认是DEFAULT；
- -D表示是否开启debug模式（默认关闭），当开启debug模式后，命令的报错堆栈信息将会打印出来，否则只会打印报错信息；
- -h表示显示命令的帮助提示信息。

命令说明

表 6-19 ma-cli 支持的命令

命令	命令详情
configure	ma-cli鉴权命令，支持用户名密码、AK/SK
image	ModelArts镜像构建、镜像注册、查询已注册镜像信息等
obs-copy	本地和OBS文件/文件夹间的相互复制
ma-job	ModelArts训练作业管理，包含作业提交、资源查询等
dli-job	DLI Spark任务提交及资源管理
auto-completion	命令自动补全

6.8.2 （可选）本地安装 ma-cli

使用场景

本文以Windows系统为例，介绍如何在Windows环境中安装ma-cli。

Step1：安装 ModelArts SDK

参考**本地安装ModelArts SDK**完成SDK的安装。

Step2：下载 ma-cli

1. **下载ma-cli软件包**。
2. 完成软件包签名校验。
   a. **下载软件包签名校验文件**。
   b. 安装openssl并执行如下命令进行签名校验。

```
openssl cms -verify -binary -in D:\ma_cli-latest-py3-none-any.whl.cms -inform DER -content D:\ma_cli-latest-py3-none-any.whl  -noverify > ./test
```

说明

本示例以软件包在D:\举例，请根据软件包实际路径修改。

Step3：安装 ma-cli

1. 在本地环境cmd中执行命令**python --version**，确认环境已经安装完成Python。（Python版本需大于3.7.x且小于3.10.x版本，推荐使用3.7.x版本）

```
C:\Users\xxx>python --version
Python *.*.*
```

2. 执行命令**pip --version**，确认Python通用包管理工具pip已经存在。

```
C:\Users\xxx>pip --version
pip **.*.* from c:\users\xxx\appdata\local\programs\python\python**\lib\site-packages\pip (python *.*)
```

3. 执行如下命令，安装ma-cli。

pip install *{ma-cli软件包路径}*\ma_cli-latest-py3-none-any.whl

```
C:\Users\xxx>pip install C:\Users\xxx\Downloads\ma_cli-latest-py3-none-any.whl
.....
Successfully installed ma_cli.*.*.*
```

在安装ma-cli时会默认同时安装所需的依赖包。当显示“Successfully installed”时，表示ma-cli安装完成。

说明

如果在安装过程中报错提示缺少相应的依赖包，请根据报错提示执行如下命令进行依赖包安装。

pip install xxxx

其中，xxxx为依赖包的名称。

6.8.3 ma-cli auto-completion 自动补全命令

命令行自动补全是指用户可以在Terminal中输入命令前缀通过Tab键自动提示支持的ma-cli命令。ma-cli自动补全功能需要手动在Terminal中激活。执行**ma-cli auto-**

completion命令，用户根据提示的补全命令，复制并在当前Terminal中执行，就可以自动补全ma-cli的命令。目前支持Bash、Fish及Zsh三种Shell，默认是Bash。

以Bash命令为例：在Terminal中执行**eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"**激活自动补全功能。

```
eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"
```

此外，可以通过“ma-cli auto-completion Fish”或“ma-cli auto-completion Fish”命令查看“Zsh”、“Fish”中的自动补全命令。

命令概览

```
$ ma-cli auto-completion -h
Usage: ma-cli auto-completion [OPTIONS] [[Bash|Zsh|Fish]]

  Auto complete ma-cli command in terminal.

  Example:

  # print bash auto complete command to terminal
  ma-cli auto-completion Bash

Options:
  -H, -h, --help  Show this message and exit.
```

```
# 默认显示Bash Shell自动补全命令

$ ma-cli auto-completion

Tips: please paste following shell command to your terminal to activate auto complation.

[ OK ] eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"

# 执行上述命令，此时Terminal已经支持自动补全

$ eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"
```

```
# 显示Fish Shell自动补全命令
$ ma-cli auto-completion Fish
Tips: please paste following shell command to your terminal to activate auto complation.

[ OK ] eval (env _MA_CLI_COMPLETE=fish_source ma-cli)
```

6.8.4 ma-cli configure 鉴权命令

鉴权信息说明

- 在虚拟机及个人PC场景，需要配置鉴权信息，目前支持用户名密码鉴权（默认）和AK/SK鉴权；
- 在使用账号认证时，需要指定username和password；在使用IAM用户认证时，需要指定account、username和password；
- 在ModelArts Notebook中可以不用执行鉴权命令，默认使用委托信息，不需要手动进行鉴权操作；
- 如果用户在ModelArts Notebook中也配置了鉴权信息，那么将会优先使用用户指定的鉴权信息。

  说明

  在鉴权时，注意您的敏感信息数据保护，避免敏感信息泄露。

命令参数总览

```
$ ma-cli configure -h
Usage: ma-cli configure [OPTIONS]

Options:
  -auth, --auth [PWD|AKSK|ROMA]  Authentication type.
  -rp, --region-profile PATH      ModelArts region file path.
  -a, --account TEXT              Account of an IAM user.
  -u, --username TEXT             Username of an IAM user.
  -p, --password TEXT             Password of an IAM user
  -ak, --access-key TEXT          User access key.
  -sk, --secret-key TEXT          User secret key.
  -r, --region TEXT               The region you want to visit.
  -pi, --project-id TEXT          User project id.
  -C, --config-file TEXT          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help                  Show this message and exit.
```

表 6-20 鉴权命令参数说明

参数名	参数类型	是否必选	参数说明
-auth / --auth	String	否	鉴权方式，支持PWD（用户名密码）、AKSK（access key和secret key），默认是PWD。
-rp / --region-profile	String	否	指定ModelArts region配置文件信息。
-a / --account	String	否	IAM租户账号，在使用IAM用户认证场景时需要指定，属于PWD鉴权的一部分。
-u / --username	String	否	用户名，在使用账号认证时表示账号名，IAM认证时表示IAM用户名，在云星账号场景不需要指定，属于PWD鉴权的一部分。
-p / --password	String	否	密码，属于PWD鉴权的一部分。
-ak / --access-key	String	否	access key，属于AKSK鉴权的一部分。
-sk / --secret-key	String	否	secret key，属于AKSK鉴权的一部分。
-r / --region	String	否	region名称，如果不填会默认使用REGION_NAME环境变量的值。
-pi / --project-id	String	否	项目ID，如果不填会默认使用对应region的值，或者使用PROJECT_ID环境变量。
-P / --profile	String	否	鉴权配置项，默认是DEFAULT。
-C / --config-file	String	否	配置文件本地路径，默认路径为~/.modelarts/ma-cli-profile.yaml。

配置用户名密码鉴权

以在虚拟机上使用**ma-cli configure**为例，介绍如何配置用户名密码进行鉴权。

说明

以下样例中所有以${}装饰的字符串都代表一个变量，用户可以根据实际情况指定对应的值。

比如${your_password}表示输入用户自己的密码信息。

```
# 默认使用DEFAULT鉴权配置项，默认提示账号、用户名及密码（其中账号和用户名如果不需要填写可以使用Enter跳过）
$ ma-cli configure --auth PWD --region ${your_region}
account: ${your_account}
username: ${your_username}
password: ${your_password}  # 输入在控制台不会回显
```

AKSK 鉴权

如下命令表示使用AKSK进行鉴权，需要交互式输入AK及SK信息。默认提示AK和SK，且输入在控制台不会回显。

注意

以下样例中所有以${}装饰的字符串都代表一个变量，用户可以根据实际情况指定对应的值。

比如${access key}表示输入用户自己的access key。

```
ma-cli configure --auth AKSK
access key [***]: ${access key}
secret key [***]: ${secret key}
```

执行完鉴权命令后，将会在~/.modelarts/ma-cli-profile.yaml配置文件中保存相应的鉴权信息。

6.8.5 ma-cli image 镜像构建支持的命令

ma-cli image命令支持：查询用户已注册的镜像、查询/加载镜像构建模板、Dockerfile镜像构建、查询/清理镜像构建缓存、注册/取消注册镜像、调试镜像是否可以在Notebook中使用等。具体命令及功能可执行**ma-cli image -h**命令查看。

镜像构建命令总览

```
$ ma-cli image -h
Usage: ma-cli image [OPTIONS] COMMAND [ARGS]...
  Support get registered image list, register or unregister image, debug image, build image in Notebook.

Options:
  -H, -h, --help  Show this message and exit.

Commands:
  add-template, at  List build-in dockerfile templates.
  build             Build docker image in Notebook.
  debug             Debug SWR image as a Notebook in ECS.
  df                Query disk usage.
  get-image, gi     Query registered image in ModelArts.
  get-template, gt  List build-in dockerfile templates.
  prune             Prune image build cache.
  register          Register image to ModelArts.
  unregister        Unregister image from ModelArts.
```

表 6-21 镜像构建支持的命令

命令	命令详情
get-template	查询镜像构建模板。
add-template	加载镜像构建模板。
get-image	查询ModelArts已注册镜像。
register	注册SWR镜像到ModelArts镜像管理。
unregister	取消注册ModelArts镜像管理中的已注册镜像。
build	基于指定的Dockerfile构建镜像 （只支持ModelArts Notebook里使用）。
df	查询镜像构建缓存（只支持ModelArts Notebook里使用）。
prune	清理镜像构建缓存 （只支持ModelArts Notebook里使用）。
debug	在ECS上调试SWR镜像是否能在ModelArts Notebook中使用 （只支持已安装docker环境的ECS）。

使用 ma-cli image get-template 命令查询镜像构建模板

ma-cli提供了一些常用的镜像构建模板，模板中包含了在ModelArts Notebook上进行Dockerfile开发的牵引指导。

```
$ ma-cli image get-template -h
Usage: ma-cli image get-template [OPTIONS]

  List build-in dockerfile templates.

  Example:

  # List build-in dockerfile templates
  ma-cli image get-template [--filer <filter_info>]  [--page-num <yourPageNum>] [--page-size
<yourPageSize>]

Options:
  --filter TEXT                 filter by keyword.
  -pn, --page-num INTEGER RANGE   Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [x>=1]
  -D, --debug                    Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT             CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                 Show this message and exit.
(PyTorch-1.4) [ma-user work]$
```

表 6-22 参数说明

参数名	参数类型	是否必选	参数说明
--filter	String	否	根据模板名称关键字过滤模板列表。
-pn / --page-num	Int	否	镜像页索引，默认是第1页。
-ps / --page-size	Int	否	每页显示的镜像数量，默认是20。

示例：查看镜像构建模板。

```
ma-cli image get-template
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image get-template
Template Name                                  Description
---------------------------------------------  ------------------------------------------------------------------------------------------
customize_from_ubuntu_18.04_to_modelarts       Add ma-user, apt install packages and create a new conda environment with pip based on scratch ubuntu 18.04
upgrade_current_notebook_apt_packages          Install apt packages like ffmpeg, gcc-8, g++-8 based on current Notebook image
migrate_3rd_party_image_to_modelarts           General template for migrating your own or open source image to ModelArts
migrate_official_torch_110_cu113_image_to_modelarts  Reconstructing and migrating the official torch 1.10.0 with cuda11.3 image to ModelArts
build_handwritten_number_inference_application  Create a new AI application, used to generate an image to deploy and infer in ModelArts
update_dli_image_pip_package                   Install pip packages based on DLI image
forward_compat_cuda_11_image_to_modelarts      Migrate and forward compat cuda-11.x to ModelArts by upgrading only user-mode CUDA components
```

使用 ma-cli image add-template 命令加载镜像构建模板

ma-cli可以使用**add-template**命令将镜像模板加载到指定文件夹下，默认路径为当前命令所在的路径。

比如${current_dir}/.ma/${template_name}/。也可以通过--dest命令指定保存的路径。当保存的路径已经有同名的模板文件夹时，可以使用--force | -f参数进行强制覆盖。

```
$ ma-cli image add-template -h
Usage: ma-cli image add-template [OPTIONS] TEMPLATE_NAME

  Add buildin dockerfile templates into disk.

  Example:

  # List build-in dockerfile templates
  ma-cli image add-template customize_from_ubuntu_18.04_to_modelarts --force

Options:
  --dst TEXT          target save path.
  -f, --force         Override templates that has been installed.
  -D, --debug         Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT  CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help      Show this message and exit.
```

表 6-23 参数说明

参数名	参数类型	是否必选	参数说明
--dst	String	否	加载模板到指定路径，默认是当前路径。

参数名	参数类型	是否必选	参数说明
-f / --force	Bool	否	是否强制覆盖已存在的同名模板，默认不覆盖。

示例：加载**customize_from_ubuntu_18.04_to_modelarts**镜像构建模板。

```
ma-cli image add-template customize_from_ubuntu_18.04_to_modelarts
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image add-template customize_from_ubuntu_18.04_to_modelarts
[ OK ] Successfully add configuration template [ customize_from_ubuntu_18.04_to_modelarts ] under folder [ /home/ma-user/work/.ma/customize_from_ubuntu_18.04_to_modelarts ]
```

使用 ma-cli image get-image 查询 ModelArts 已注册镜像

Dockerfile一般需要提供一个基础镜像的地址，目前支持从docker hub等开源镜像仓拉取公开镜像，以及SWR的公开或私有镜像。其中ma-cli提供了查询ModelArts预置镜像和用户已注册镜像列表及SWR地址。

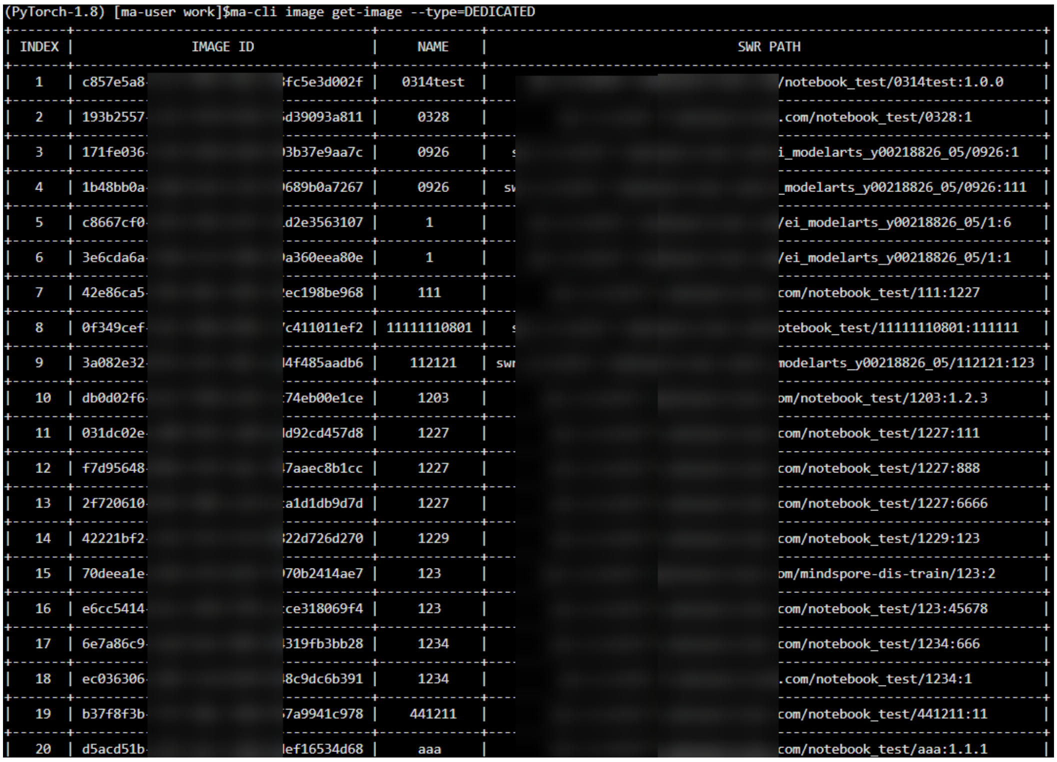
```
$ma-cli image get-image -h
Usage: ma-cli image get-image [OPTIONS]

  Get registered image list.

  Example:

  # Query images by image type and only image id, show name and swr_path
  ma-cli image get-image --type=DEDICATED

  # Query images by image id
  ma-cli image get-image --image-id ${image_id}

  # Query images by image type and show more information
  ma-cli image get-image --type=DEDICATED -v

  # Query images by image name
  ma-cli image get-image --filter=torch

Options:
  -t, --type [BUILD_IN|DEDICATED|ALL]
                                  Image type(default ALL)
  -f, --filter TEXT               Image name to filter
  -v, --verbose                   Show detailed information on image.
  -i, --image-id TEXT             Get image details by image id
  -n, --image-name TEXT           Get image details by image name
  -wi, --workspace-id TEXT        The workspace where you want to query image(default "0")
  -pn, --page-num INTEGER RANGE   Specify which page to query  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query  [x>=1]
  -C, --config-file PATH          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                  Show this message and exit.
```

表 6-24 参数说明

参数名	参数类型	是否必选	参数说明
-t / --type	String	否	查询的镜像类型，支持BUILD_IN、DEDICATED和ALL三种查询类型。 ● BUILD_IN：预置镜像 ● DEDICATED：用户已注册的自定义镜像 ● ALL：所有镜像
-f / --filter	String	否	镜像名关键字。根据镜像名关键字过滤镜像列表。
-v / --verbose	Bool	否	显示详细的信息开关，默认关闭。
-i / --image-id	String	否	查询指定镜像ID的镜像详情。
-n / --image-name	String	否	查询指定镜像名称的镜像详情。
-wi / --workspace-id	String	否	查询指定工作空间下的镜像信息。
-pn / --page-num	Int	否	镜像页索引，默认是第1页。
-ps / --page-size	Int	否	每页显示的镜像数量，默认是20。

示例：查询ModelArts已注册的自定义镜像。

```
ma-cli image get-image --type=DEDICATED
```



(PyTorch-1.8) [ma-user work]$ma-cli image get-image --type=DEDICATED

INDEX	IMAGE ID	NAME	SWR PATH
1	c857e5a8-[redacted]fc5e3d002f	0314test	[redacted]/notebook_test/0314test:1.0.0
2	193b2557-[redacted]d39093a811	0328	[redacted].com/notebook_test/0328:1
3	171fe036-[redacted]3b37e9aa7c	0926	s[redacted]i_modelarts_y00218826_05/0926:1
4	1b48bb0a-[redacted]689b0a7267	0926	sw[redacted]_modelarts_y00218826_05/0926:111
5	c8667cf0-[redacted]d2e3563107	1	[redacted]/ei_modelarts_y00218826_05/1:6
6	3e6cda6a-[redacted]a360eea80e	1	[redacted]/ei_modelarts_y00218826_05/1:1
7	42e86ca5-[redacted]ec198be968	111	[redacted]com/notebook_test/111:1227
8	0f349cef-[redacted]c411011ef2	11111110801	s[redacted]otebook_test/11111110801:111111
9	3a082e32-[redacted]4f485aadb6	112121	swr[redacted]modelarts_y00218826_05/112121:123
10	db0d02f6-[redacted]74eb00e1ce	1203	[redacted]om/notebook_test/1203:1.2.3
11	031dc02e-[redacted]d92cd457d8	1227	[redacted]com/notebook_test/1227:111
12	f7d95648-[redacted]7aaec8b1cc	1227	[redacted]com/notebook_test/1227:888
13	2f720610-[redacted]a1d1db9d7d	1227	[redacted]com/notebook_test/1227:6666
14	42221bf2-[redacted]22d726d270	1229	[redacted]com/notebook_test/1229:123
15	70deea1e-[redacted]70b2414ae7	123	[redacted]om/mindspore-dis-train/123:2
16	e6cc5414-[redacted]ce318069f4	123	[redacted]com/notebook_test/123:45678
17	6e7a86c9-[redacted]319fb3bb28	1234	[redacted]com/notebook_test/1234:666
18	ec036306-[redacted]8c9dc6b391	1234	[redacted].com/notebook_test/1234:1
19	b37f8f3b-[redacted]7a9941c978	441211	[redacted]com/notebook_test/441211:11
20	d5acd51b-[redacted]ef16534d68	aaa	[redacted]com/notebook_test/aaa:1.1.1

使用 ma-cli image build 命令在 ModelArts Notebook 中进行镜像构建

使用**ma-cli image build**命令基于指定的Dockerfile进行镜像构建，仅支持在ModelArts Notebook里使用该命令。

```
$ ma-cli image build -h
Usage: ma-cli image build [OPTIONS] FILE_PATH

  Build docker image in Notebook.

  Example:

  # Build a image and push to SWR
  ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile -swr my_organization/
my_image:0.0.1

  # Build a image and push to SWR, dockerfile context path is current dir
  ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile -swr my_organization/
my_image:0.0.1 -context .

  # Build a local image and save to local path and OBS
  ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile --target ./build.tar --
obs_path obs://bucket/object --swr-path my_organization/my_image:0.0.1

Options:
  -t, --target TEXT        Name and optionally a tag in the 'name:tag' format.
  -swr, --swr-path TEXT    SWR path without swr endpoint, eg:organization/image:tag.  [required]
  --context DIRECTORY      build context path.
  -arg, --build-arg TEXT   build arg for Dockerfile.
  -obs, --obs-path TEXT    OBS path to save local built image.
  -f, --force              Force to overwrite the existing swr image with the same name and tag.
  -C, --config-file PATH   Configure file path for authorization.
  -D, --debug              Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT       CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help           Show this message and exit.
```

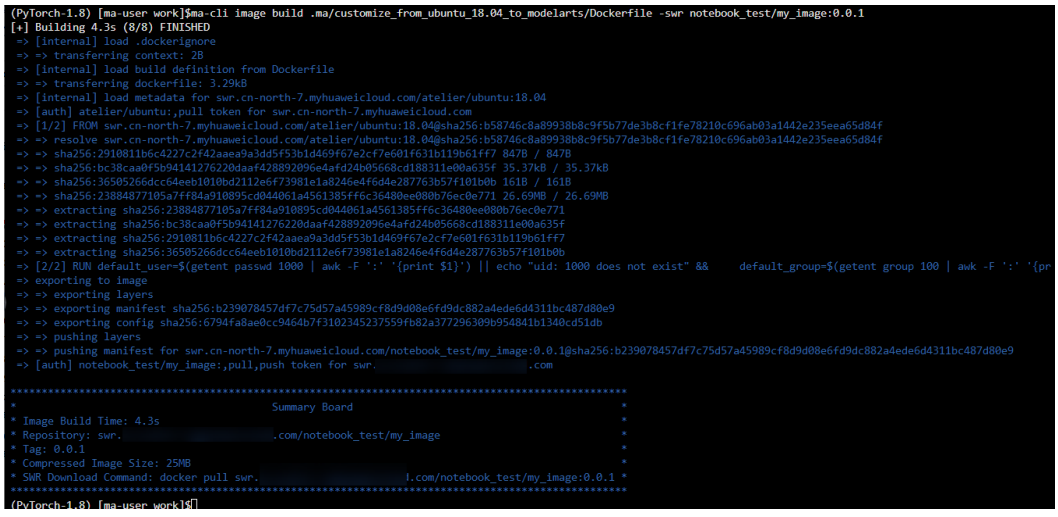

表 6-25 参数说明

参数名	参数类型	是否必选	参数说明
FILE_PATH	String	是	Dockerfile文件所在的路径。
-t / --target	String	否	表示构建生成的tar包保存在本地的路径，默认是当前文件夹目录。
-swr / --swr-path	String	是	SWR镜像名称，遵循organization/image_name:tag格式，针对于构建保存tar包场景可以省略。
--context	String	否	Dockerfile构建时的上下文信息路径，主要用于数据复制。
-arg / --build-arg	String	否	指定构建参数，多个构建参数可以使用--build-arg VERSION=18.04 --build-arg ARCH=X86_64
-obs / --obs-path	String	否	将生成的tar包自动上传到OBS中。
-f / --force	Bool	否	是否强制覆盖已存在的SWR镜像，默认不覆盖。

示例：在ModelArts Notebook里进行镜像构建。

```
ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile -swr notebook_test/my_image:0.0.1
```

其中“.ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile”为Dockerfile文件所在路径，“notebook_test/my_image:0.0.1”为构建的新镜像的SWR路径。



使用 ma-cli image df 命令在 ModelArts Notebook 中查询镜像构建缓存

使用**ma-cli image df**命令查询镜像构建缓存，仅支持在ModelArts Notebook里使用该命令。

```
$ ma-cli image df -h
Usage: ma-cli image df [OPTIONS]
```

```
Query disk usage used by image-building in Notebook.

Example:

# Query image disk usage
ma-cli image df

Options:
  -v, --verbose      Show detailed information on disk usage.
  -D, --debug        Debug Mode. Shows full stack trace when error occurs.

  -h, -H, --help     Show this message and exit.
```

表 6-26 参数说明

参数名	参数类型	是否必选	参数说明
-v / --verbose	Bool	否	显示详细的信息开关，默认关闭。

示例：在ModelArts Notebook里查看所有镜像缓存。

```
ma-cli image df
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image df
ID                                                              RECLAIMABLE     SIZE            LAST ACCESSED
iwrrwsi9pdcjafe1ij6d0r918                                       true            98.50MB
cp52c4q81ud2abu2vp7sj5vyt                                       true            1.04MB
4jbo6v06r2w1575ddq3w8g12e                                       true            139.68kB
ojdjw5mok71s1nh2cauant051                                       true            86.86kB
k2jm6g061n5twmz7gmonmqjsh                                       true            16.55kB
efu5kwgig1ve44fe7smbrncnh*                                      true            8.19kB
uzikwqk5taxnslvajm14jrbje*                                      true            4.10kB
2g8p0qcb014g3qva7ucawkv87*                                      true            4.10kB
Reclaimable:    99.80MB
Total:          99.80MB
```

示例：显示镜像缓存占用磁盘的详细信息。

```
ma-cli image df --verbose
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image df --verbose
ID:             iwrrwsi9pdcjafe1ij6d0r918
Created at:     2023-03-28 12:23:28.353759532 +0000 UTC
Mutable:        false
Reclaimable:    true
Shared:         false
Size:           98.50MB
Description:    pulled from swr                 .com/atelier/ubuntu:18.04@sha256:b5874                                  ea65d84f
Usage count:    1
Last used:      2023-03-28 12:23:28.37337776 +0000 UTC
Type:           regular

ID:             cp52c4q81ud2abu2vp7sj5vyt
Parents:        iwrrwsi9pdcjafe1ij6d0r918
Created at:     2023-03-28 12:23:28.366910223 +0000 UTC
Mutable:        false
Reclaimable:    true
Shared:         false
Size:           1.04MB
Description:    pulled from swr.                .com/atelier/ubuntu:18.04@sha256:b5874                                  e235eea65d84f
Usage count:    1
Last used:      2023-03-28 12:23:28.38560437 +0000 UTC
Type:           regular

ID:             4jbo6v06r2w1575ddq3w8g12e
Parents:        k2jm6g061n5twmz7gmonmqjsh
Created at:     2023-03-28 12:23:30.681643727 +0000 UTC
Mutable:        false
Reclaimable:    true
Shared:         false
Size:           139.68kB
Description:    mount / from exec /bin/sh -c default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" &&     default_group=$(getent
up 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" &&     if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then         userdel -r ${defau
user};     fi &&     if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then         groupdel -f ${default_group};     fi &&     groupadd -g 100 ma-group
useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user &&     chmod -R 750 /home/ma-user
Usage count:    2
Last used:      2023-03-28 12:25:39.149080471 +0000 UTC
Type:           regular
```

使用 ma-cli image prune 命令在 ModelArts Notebook 中清理镜像构建缓存

使用**ma-cli image prune**命令清理镜像构建缓存，仅支持在ModelArts Notebook里使用该命令。

```
$ ma-cli image prune -h
Usage: ma-cli image prune [OPTIONS]
```

```
Prune image build cache by image-building in Notebook.

Example:

# Prune image build cache
ma-cli image prune

Options:
  -ks, --keep-storage INTEGER  Amount of disk space to keep for cache below this limit (in MB) (default: 0).
  -kd, --keep-duration TEXT    Keep cache newer than this limit, support second(s), minute(m) and hour(h)
(default:   0s).
  -v, --verbose                Show more verbose output.
  -D, --debug                  Debug Mode. Shows full stack trace when error occurs.
  -h, -H, --help               Show this message and exit.
```

表 6-27 参数说明

参数名	参数类型	是否必选	参数说明
-ks / --keep-storage	Int	否	清理缓存时保留的缓存大小，单位是MB，默认是0，表示全部清理。
-kd / --keep-duration	String	否	清理缓存时保留较新的缓存，只清除历史缓存，单位为s（秒）、m（分钟）、h（小时），默认是0s，表示全部清理。
-v / --verbose	Bool	否	显示详细的信息开关，默认关闭。

示例：清理保留1MB镜像缓存。

```
ma-cli image prune -ks 1
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image prune -ks 1
ID                                                                      RECLAIMABLE     SIZE    LAST ACCESSED
uzikwqk5taxnslvajm14jrbje*                                              true            4.10kB
4jbo6v06r2wl575ddq3w8g12e                                               true            139.68kB
k2jm6g061n5twmz7gmonmqjsh                                               true            16.55kB
ojdjw5mok71s1nh2cauant051                                               true            86.86kB
cp52c4q81ud2abu2vp7sj5vyt                                               true            1.04MB
iwrrwsi9pdcjafe1ij6d0r918                                               true            98.50MB
Total:  99.79MB
```

使用 ma-cli image register 命令注册 SWR 镜像到 ModelArts 镜像管理

调试完成后，使用**ma-cli image register**命令将新镜像注册到ModelArts镜像管理服务中，进而在能够在ModelArts中使用该镜像。

```
$ma-cli image register -h
Usage: ma-cli image register [OPTIONS]

Register image to ModelArts.

Example:

# Register image into ModelArts service
ma-cli image register --swr-path=xx

# Share SWR image to DLI service
ma-cli image register -swr xx -td
```



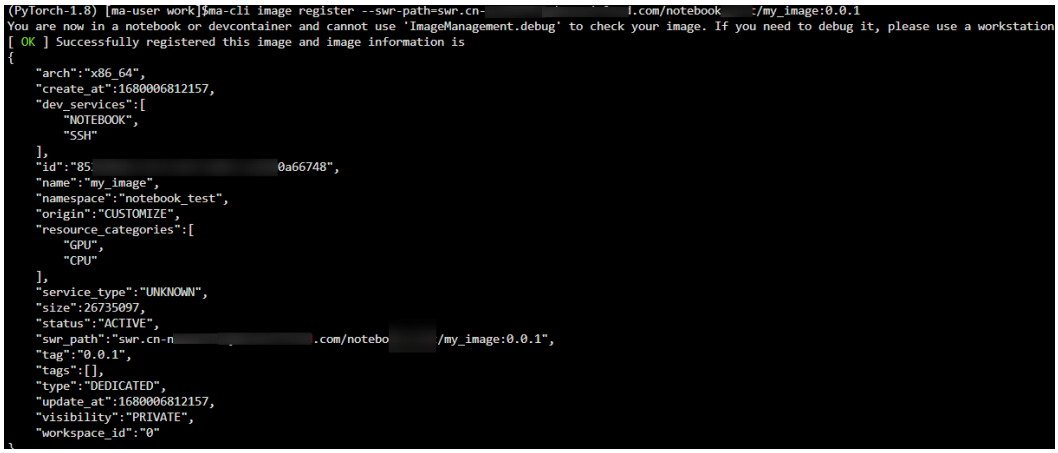
```
  # Register image into ModelArts service and specify architecture to be 'AARCH64'
  ma-cli image register --swr-path=xx --arch AARCH64

Options:
  -swr, --swr-path TEXT           SWR path without swr endpoint, eg:organization/image:tag.  [required]
  -a, --arch [X86_64|AARCH64]     Image architecture (default: X86_64).
  -s, --service [NOTEBOOK|MODELBOX]
                                  Services supported by this image(default NOTEBOOK).
  -rs, --resource-category [CPU|GPU|ASCEND]
                                  The resource category supported by this image (default: CPU and GPU).
  -wi, --workspace-id TEXT        The workspace to register this image (default: "0").
  -v, --visibility [PUBLIC|PRIVATE]
                                  PUBLIC: every user can use this image. PRIVATE: only image owner can use this
image (Default: PRIVATE).
  -td, --to-dli                   Register swr image to DLI, which will share SWR image to DLI service.
  -d, --description TEXT          Image description (default: "").
  -C, --config-file PATH          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help                  Show this message and exit.
```

表 6-28 参数说明

参数名	参数类型	是否必选	参数说明
-swr / --swr-path	String	是	需要注册的镜像的SWR路径。
-a / --arch	String	否	注册镜像的架构，X86_64或者AARCH64，默认是X86_64。
-s / --service	String	否	注册镜像的服务类型，NOTEBOOK或者MODELBOX，默认是NOTEBOOK。 可以输入多个值，如-s NOTEBOOK -s MODELBOX。
-rs / --resource-category	String	否	注册镜像能够使用的资源类型，默认是CPU和GPU。
-wi / --workspace-id	String	否	注册镜像到指定的工作空间，workspace ID默认是0。
-v / --visibility	Bool	否	注册的镜像可见性，PRIVATE（仅自己可见）或者PUBLIC（所有用户可见），默认是PRIVATE。
-td / --to-dli	Bool	否	注册镜像到DLI服务。
-d/ --description	String	否	填写镜像描述，默认为空。

示例：注册SWR镜像到ModelArts。

```
ma-cli image register --swr-path=xx
```



使用 ma-cli image unregister 命令取消已注册的镜像

使用**ma-cli image unregister**命令将注册的镜像从ModelArts中删除。

```
$ ma-cli image unregister -h
Usage: ma-cli image unregister [OPTIONS]

  Unregister image from ModelArts.

  Example:

  # Unregister image
  ma-cli image unregister --image-id=xx

  # Unregister image and delete it from swr
  ma-cli image unregister --image-id=xx -d

Options:
  -i, --image-id TEXT     Unregister image details by image id.  [required]
  -d, --delete-swr-image  Delete the image from swr.
  -C, --config-file PATH  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help          Show this message and exit.
```

表 6-29 参数说明

参数名	参数类型	是否必选	参数说明
-i / -image-id	String	是	需要取消注册的镜像ID。
-d / --delete-swr-image	Bool	否	取消注册后同步删除SWR镜像开关，默认关闭。

在 ECS 上调试 SWR 镜像是否能在 ModelArts Notebook 中使用

ma-cli支持在ECS上调试SWR镜像是否可以在ModelArts开发空间中运行，发现镜像中可能存在的问题。

```
ma-cli image debug -h
Usage: ma-cli image debug [OPTIONS]

  Debug SWR image as a Notebook in ECS.

  Example:

  # Debug cpu notebook image
  ma-cli image debug --swr-path=xx --service=NOTEBOOK --region=

  # Debug gpu notebook image
  ma-cli image debug --swr-path=xx --service=NOTEBOOK --region=  --gpu

Options:
  -swr, --swr-path TEXT           SWR path without SWR endpoint, eg:organization/image:tag.  [required]
  -r, --region TEXT               Region name.  [required]
  -s, --service [NOTEBOOK|MODELBOX]
                                  Services supported by this image(default NOTEBOOK).
  -a, --arch [X86_64|AARCH64]     Image architecture(default X86_64).
  -g, --gpu                       Use all gpus to debug.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help                  Show this message and exit.
```

表 6-30 参数说明

参数名	参数类型	是否必选	参数说明
-swr / --swr-path	String	是	需要调试的镜像的SWR路径。
-r / --region	String	是	需要调试的镜像所在的区域。
-s / --service	String	否	调试镜像的服务类型，NOTEBOOK或者MODELBOX，默认是NOTEBOOK。
-a / --arch	String	否	调试镜像的架构，X86_64或者AARCH64，默认是X86_64。
-g / --gpu	Bool	否	使用GPU进行调试开关，默认关闭。

6.8.6 ma-cli ma-job 训练作业支持的命令

使用**ma-cli ma-job**命令可以提交训练作业，查询训练作业日志、事件、使用的AI引擎、资源规格及停止训练作业等。

```
$ ma-cli ma-job -h
Usage: ma-cli ma-job [OPTIONS] COMMAND [ARGS]...

  ModelArts job submission and query jod details.

Options:
  -h, -H, --help  Show this message and exit.

Commands:
```



```
delete      Delete training job by job id.
get-engine  Get job engines.
get-event   Get job running event.
get-flavor  Get job flavors.
get-job     Get job details.
get-log     Get job log details.
get-pool    Get job engines.
stop        Stop training job by job id.
submit      Submit training job.
```

表 6-31 训练作业支持的命令

命令	命令详情
get-job	查询ModelArts训练作业列表及详情。
get-log	查询ModelArts训练作业运行日志。
get-engine	查询ModelArts训练AI引擎。
get-event	查询ModelArts训练作业事件。
get-flavor	查询ModelArts训练资源规格。
get-pool	查询ModelArts训练专属池。
stop	停止ModelArts训练作业。
submit	提交ModelArts训练作业。
delete	删除指定作业id的训练作业。

使用 ma-cli ma-job get-job 命令查询 ModelArts 训练作业

使用**ma-cli ma-job get-job**命令可以查看训练作业列表或某个作业详情。

```
$ ma-cli ma-job get-job -h
Usage: ma-cli ma-job get-job [OPTIONS]

  Get job details.

  Example:

  # Get train job details by job name
  ma-cli ma-job get-job -n ${job_name}

  # Get train job details by job id
  ma-cli ma-job get-job -i ${job_id}

  # Get train job list
  ma-cli ma-job get-job --page-size 5 --page-num 1

Options:

  -i, --job-id TEXT               Get training job details by job id.
  -n, --job-name TEXT             Get training job details by job name.
  -pn, --page-num INTEGER         Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [1<=x<=50]
  -v, --verbose                   Show detailed information about training job details.
  -C, --config-file TEXT          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help                  Show this message and exit.
```

表 6-32 参数说明

参数名	参数类型	是否必选	参数说明
-i / --job-id	String	否	查询指定训练作业ID的任务详情。
-n / --job-name	String	否	查询指定任务名称的训练作业或根据任务名称关键字过滤训练作业。
-pn / --page-num	Int	否	页面索引，默认是第1页。
-ps / --page-size	Int	否	每页显示的训练作业数量，默认是10。
-v / --verbose	Bool	否	显示详细的信息开关，默认关闭。

- 示例：查询指定任务ID的训练作业。
  ma-cli ma-job get-job -i *b63e90xxx*

```
(PyTorch-1.4) [ma-user work]$ma-cli ma-job get-job -i b63e90ba-91
+--------------+--------------------------------------+---------+-----------+----------+-------------+-------------+-----------+
|      id      |                 name                 |  status |  user_name| duration |  create_time|  start_time | descripti |
+--------------+--------------------------------------+---------+-----------+----------+-------------+-------------+-----------+
| b63e90ba-    | workflow_created_job_ed3a963f-5438-4a99-9a19-c97ce | Complet | ei_modela | 00h:01m:1| 2023-03-29  | 2023-03-29  |           |
|              | 88c48b8                              | ed      |           | 6s       | 03:41:21    | 03:41:30    |           |
+--------------+--------------------------------------+---------+-----------+----------+-------------+-------------+-----------+
```

- 示例：根据任务名称关键字“auto”过滤训练作业。
  ma-cli ma-job get-job -n *auto*

```
(PyTorch-1.4) [ma-user work]$ma-cli ma-job get-job -n auto
+------+----------+------+-----------+----------+-------------+-------------+
| inde |    id    | name |  status   | user_name| duration    | create_time | start_time  |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  1   | 9b495c   |      | Complete  |          | 00h:01m:3   | 2023-03-29  | 2023-03-29  |
|      |          |      |    d      |          |    1s       | 07:03:08    | 07:05:20    |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  2   | af2147f5-|      | Terminat  |          | 00h:10m:4   | 2023-03-29  | 2023-03-29  |
|      |          |      |    ed     |          |    9s       | 06:52:16    | 06:52:32    |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  3   | 2c1855b1-|      |  Failed   |          | 00h:37m:2   | 2023-03-29  | 2023-03-29  |
|      |          |      |           |          |    9s       | 03:22:31    | 03:22:58    |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  4   | 4525b3c9-|      |  Failed   |          | 00h:00m:0   | 2023-03-29  | 2023-03-29  |
|      |          |      |           |          |    1s       | 03:19:41    | 03:19:49    |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  5   | 4234455d-|      | Terminat  |          | 00h:00m:0   | 2023-03-29  |     N/A     |
|      |          |      |    ed     |          |    0s       | 02:25:18    |             |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  6   | 9810ae49-|      | Terminat  |          | 00h:09m:0   | 2023-03-29  | 2023-03-29  |
|      |          |      |    ed     |          |    6s       | 02:19:49    | 02:20:13    |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  7   | 90c7de89-|      | Abnormal  |          | 00h:00m:0   | 2023-03-29  |     N/A     |
|      |          |      |           |          |    0s       | 01:43:18    |             |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  8   | fc740dc5-|      | Terminat  |          | 00h:00m:0   | 2023-03-29  |     N/A     |
|      |          |      |    ed     |          |    0s       | 01:22:19    |             |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  9   | 5d16fdfe-|      | Terminat  |          | 00h:00m:0   | 2023-03-29  |     N/A     |
|      |          |      |    ed     |          |    0s       | 01:11:26    |             |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
|  10  | 3737e56d-|      | Complete  |          | 00h:05m:5   | 2023-03-29  | 2023-03-29  |
|      |          |      |    d      |          |    9s       | 00:59:28    | 01:04:20    |
+------+----------+------+-----------+----------+-------------+-------------+-------------+
```

使用 ma-cli ma-job submit 命令提交 ModelArts 训练作业

执行**ma-cli ma-job submit**命令提交ModelArts训练作业。

ma-cli ma-job submit命令需要指定一个位置参数YAML_FILE表示作业的配置文件路径，如果不指定该参数，则表示配置文件为空。配置文件是一个YAML格式的文件，里面的参数就是命令的option参数。此外，如果用户在命令行中同时指定YAML_FILE配置文件和option参数，命令行中指定的option参数的值将会覆盖配置文件相同的值。

```
$ma-cli ma-job submit -h
Usage: ma-cli ma-job submit [OPTIONS] [YAML_FILE]...
```



```
Submit training job.

Example:

ma-cli ma-job submit --code-dir obs://your_bucket/code/
                     --boot-file main.py
                     --framework-type PyTorch
                     --working-dir /home/ma-user/modelarts/user-job-dir/code
                     --framework-version pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
                     --data-url obs://your_bucket/dataset/
                     --log-url obs://your_bucket/logs/
                     --train-instance-type modelarts.vm.cpu.8u
                     --train-instance-count 1

Options:
  --name TEXT                   Job name.
  --description TEXT            Job description.
  --image-url TEXT              Full swr custom image path.
  --uid TEXT                    Uid for custom image (default: 1000).
  --working-dir TEXT            ModelArts training job working directory.
  --local-code-dir TEXT         ModelArts training job local code directory.
  --user-command TEXT           Execution command for custom image.
  --pool-id TEXT                Dedicated pool id.
  --train-instance-type TEXT    Train worker specification.
  --train-instance-count INTEGER  Number of workers.
  --data-url TEXT               OBS path for training data.
  --log-url TEXT                OBS path for training log.
  --code-dir TEXT               OBS path for source code.
  --output TEXT                 Training output parameter with OBS path.
  --input TEXT                  Training input parameter with OBS path.
  --env-variables TEXT          Env variables for training job.
  --parameters TEXT             Training job parameters (only keyword parameters are supported).
  --boot-file TEXT              Training job boot file path behinds `code_dir`.
  --framework-type TEXT         Training job framework type.
  --framework-version TEXT      Training job framework version.
  --workspace-id TEXT           The workspace where you submit training job(default "0")
  --policy [regular|economic|turbo|auto]
                                Training job policy, default is regular.
  --volumes TEXT                Information about the volumes attached to the training job.
  -q, --quiet                   Exit without waiting after submit successfully.
  -C, --config-file PATH        Configure file path for authorization.
  -D, --debug                   Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT            CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                Show this message and exit.
```

表 6-33 参数说明

参数名	参数类型	是否必选	参数说明
YAML_FILE	String	否	表示训练作业的配置文件，如果不传则表示配置文件为空。
--code-dir	String	是	训练源代码的OBS路径。
--data-url	String	是	训练数据的OBS路径。
--log-url	String	是	存放训练生成日志的OBS路径。

参数名	参数类型	是否必选	参数说明
--train-instance-count	String	是	训练作业实例数，默认是1，表示单节点。
--boot-file	String	否	当使用自定义镜像或自定义命令时可以省略，当使用预置命令提交训练作业时需要指定该参数。
--name	String	否	训练作业名称。
--description	String	否	训练作业描述信息。
--image-url	String	否	自定义镜像SWR地址，遵循organization/image_name:tag
--uid	String	否	自定义镜像运行的UID，默认值1000。
--working-dir	String	否	运行算法时所在的工作目录。
--local-code-dir	String	否	算法的代码目录下载到训练容器内的本地路径。
--user-command	String	否	自定义镜像执行命令。需为/home下的目录。 当code-dir以file://为前缀时，当前字段不生效。
--pool-id	String	否	训练作业选择的资源池ID。可在ModelArts管理控制台，单击左侧“专属资源池”，在专属资源池列表中查看资源池ID。
--train-instance-type	String	否	训练作业选择的资源规格。
--output	String	否	训练的输出信息，指定后，训练作业将会把训练脚本中指定输出参数对应训练容器的输出目录上传到指定的OBS路径。如果需要指定多个参数，可以使用--output output1=obs://bucket/output1 --output output2=obs://bucket/output2
--input	String	否	训练的输入信息，指定后，训练作业将会把对应OBS上的数据下载到训练容器，并将数据存储路径通过指定的参数传递给训练脚本。如果需要指定多个参数，可以使用--input data_path1=obs://bucket/data1 --input data_path2=obs://bucket/data2
--env-variables	String	否	训练时传入的环境变量，如果需要指定多个参数，可以使用--env-variables ENV1=env1 --env-variables ENV2=env2

参数名	参数类型	是否必选	参数说明
--parameters	String	否	训练入参，可以通过--parameters "--epoch 0 --pretrained"指定多个参数。
--framework-type	String	否	训练作业选择的引擎规格。
--framework-version	String	否	训练作业选择的引擎版本。
-q / --quiet	Bool	否	提交训练作业成功后直接退出，不再同步打印作业状态。
--workspace-id	String	否	作业所处的工作空间，默认值为“0”。
--policy	String	否	训练资源规格模式，可选值regular、economic、turbo、auto。
--volumes	String	否	挂载EFS，如果需要指定多个参数，可以使用--volumes。 "local_path=/xx/yy/zz;read_only=false;nfs_server_path=xxx.xxx.xxx.xxx:/" -volumes "local_path=/xxx/yyy/zzz;read_only=false;nfs_server_path=xxx.xxx.xxx.xxx:/"

示例：基于 ModelArts 预置镜像提交训练作业

指定命令行options参数提交训练作业

```
ma-cli ma-job submit --code-dir obs://your-bucket/mnist/code/ \
                --boot-file main.py \
                --framework-type PyTorch \
                --working-dir /home/ma-user/modelarts/user-job-dir/code \
                --framework-version pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 \
                --data-url obs://your-bucket/mnist/dataset/MNIST/ \
                --log-url obs://your-bucket/mnist/logs/ \
                --train-instance-type modelarts.vm.cpu.8u \
                --train-instance-count 1  \
                -q
```

使用预置镜像的train.yaml样例：

```
# .ma/train.yaml样例（预置镜像）
# pool_id: pool_xxxx
train-instance-type: modelarts.vm.cpu.8u
train-instance-count: 1
data-url: obs://your-bucket/mnist/dataset/MNIST/
code-dir: obs://your-bucket/mnist/code/
```

```
working-dir: /home/ma-user/modelarts/user-job-dir/code
framework-type: PyTorch
framework-version: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
boot-file: main.py
log-url: obs://your-bucket/mnist/logs/

##[Optional] Uncomment to set uid when use custom image mode
uid: 1000

##[Optional] Uncomment to upload output file/dir to OBS from training platform
output:
   - name: output_dir
     obs_path: obs://your-bucket/mnist/output1/

##[Optional] Uncomment to download input file/dir from OBS to training platform
input:
   - name: data_url
     obs_path: obs://your-bucket/mnist/dataset/MNIST/

##[Optional] Uncomment pass hyperparameters
parameters:
   - epoch: 10
   - learning_rate: 0.01
   - pretrained:

##[Optional] Uncomment to use dedicated pool
pool_id: pool_xxxx

##[Optional] Uncomment to use volumes attached to the training job
volumes:
  - efs:
     local_path: /xx/yy/zz
     read_only: false
     nfs_server_path: xxx.xxx.xxx.xxx:/
```

示例：基于自定义镜像创建训练作业

指定命令行options参数提交训练作业

```
ma-cli ma-job submit --image-url atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-
x86_64-20220926104358-041ba2e \
                --code-dir obs://your-bucket/mnist/code/ \
                --user-command "export LD_LIBRARY_PATH=/usr/local/cuda/compat:$LD_LIBRARY_PATH &&
cd /home/ma-user/modelarts/user-job-dir/code && /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/
python main.py" \
                --data-url obs://your-bucket/mnist/dataset/MNIST/ \
                --log-url obs://your-bucket/mnist/logs/ \
                --train-instance-type modelarts.vm.cpu.8u \
                --train-instance-count 1  \
                -q
```

使用自定义镜像的train.yaml样例：

```
# .ma/train.yaml样例（自定义镜像）
image-url: atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-
x86_64-20220926104358-041ba2e
user-command: export LD_LIBRARY_PATH=/usr/local/cuda/compat:$LD_LIBRARY_PATH && cd /home/ma-
user/modelarts/user-job-dir/code && /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python main.py
train-instance-type: modelarts.vm.cpu.8u
train-instance-count: 1
data-url: obs://your-bucket/mnist/dataset/MNIST/
code-dir: obs://your-bucket/mnist/code/
log-url: obs://your-bucket/mnist/logs/

##[Optional] Uncomment to set uid when use custom image mode
uid: 1000

##[Optional] Uncomment to upload output file/dir to OBS from training platform
```

```
output:
  - name: output_dir
    obs_path: obs://your-bucket/mnist/output1/

##[Optional] Uncomment to download input file/dir from OBS to training platform
input:
  - name: data_url
    obs_path: obs://your-bucket/mnist/dataset/MNIST/

##[Optional] Uncomment pass hyperparameters
parameters:
  - epoch: 10
  - learning_rate: 0.01
  - pretrained:

##[Optional] Uncomment to use dedicated pool
pool_id: pool_xxxx

##[Optional] Uncomment to use volumes attached to the training job
volumes:
 - efs:
    local_path: /xx/yy/zz
    read_only: false
    nfs_server_path: xxx.xxx.xxx.xxx:/
```

使用 ma-cli ma-job get-log 命令查询 ModelArts 训练作业日志

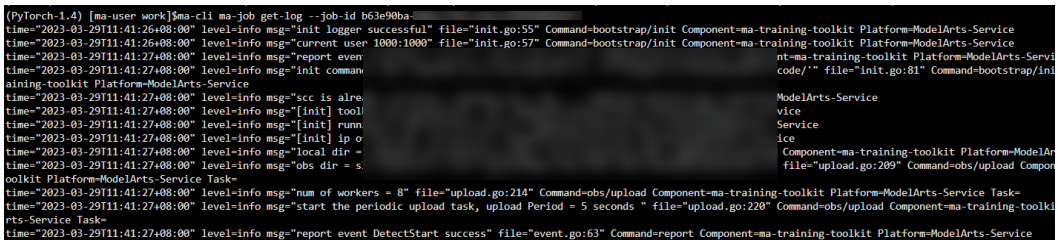
执行**ma-cli ma-job get-log**命令查询ModelArts训练作业日志。

```
$ ma-cli ma-job get-log -h
Usage: ma-cli ma-job get-log [OPTIONS]

  Get job log details.

  Example:

  # Get job log by job id
  ma-cli ma-job get-log --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get training job details by job id.  [required]
  -t, --task-id TEXT      Get training job details by task id (default "worker-0").
  -C, --config-file TEXT  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help          Show this message and exit.
```

参数名	参数类型	是否必选	参数说明
-i / --job-id	String	是	查询指定训练作业ID的任务日志。
-t / --task-id	String	否	查询指定task的日志，默认是work-0。

示例：查询指定训练作业ID的作业日志。

```
ma-cli ma-job get-log --job-id b63e90baxxx
```



使用 ma-cli ma-job get-event 命令查询 ModelArts 训练作业事件

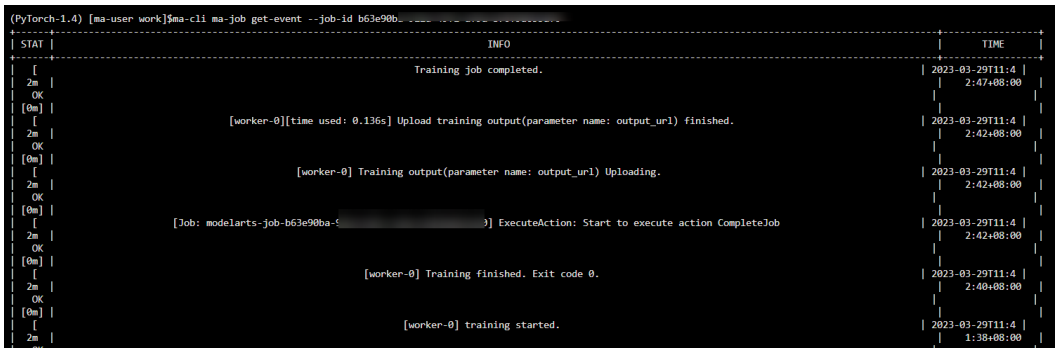
执行**ma-cli ma-job get-event**命令查看ModelArts训练作业事件。

```
$ ma-cli ma-job get-event -h
Usage: ma-cli ma-job get-event [OPTIONS]

  Get job running event.

  Example:

  # Get training job running event
  ma-cli ma-job get-event --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get training job event by job id.  [required]
  -C, --config-file TEXT  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help          Show this message and exit.
```

参数名	参数类型	是否必选	参数说明
-i / --job-id	String	是	查询指定训练作业ID的事件。

示例：查看指定ID的训练作业的事件详情等。

```
ma-cli ma-job get-event --job-id b63e90baxxx
```



使用 ma-cli ma-job get-engine 命令查询 ModelArts 训练 AI 引擎

执行**ma-cli ma-job get-engine**命令查询ModelArts训练使用的AI引擎。

```
$ ma-cli ma-job get-engine -h
Usage: ma-cli ma-job get-engine [OPTIONS]

  Get job engine info.

  Example:

  # Get training job engines
  ma-cli ma-job get-engine

Options:
  -v, --verbose           Show detailed information about training engines.
  -C, --config-file TEXT  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help          Show this message and exit.
```

表 6-34 参数说明

参数名	参数类型	是否必选	参数说明
-v / --verbose	Bool	否	显示详细的信息开关，默认关闭。

示例：查看训练作业的AI引擎。

```
ma-cli ma-job get-engine
```

```
(PyTorch-1.4) [ma-user work]$ma-cli ma-job get-engine
+-------+-----------------------------------------------------+-----------------------+----------+
| index |                      engine id                      |      engine name      | run user |
+-------+-----------------------------------------------------+-----------------------+----------+
|   1   |                  caffe-1.0.0-python2.7              |         Caffe         |          |
+-------+-----------------------------------------------------+-----------------------+----------+
|   2   |                  horovod-cp36-tf-1.16.2             |        Horovod        |          |
+-------+-----------------------------------------------------+-----------------------+----------+
|   3   | horovod_0.20.0-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 |        Horovod        |   1102   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   4   | horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64 |        Horovod        |   1102   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   5   |              kungfu-0.2.2-tf-1.13.1-python3.6       |         KungFu        |          |
+-------+-----------------------------------------------------+-----------------------+----------+
|   6   |       mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64 |          MPI          |   1102   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   7   |      mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64 | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   8   |      mindspore_1.8.0-cann_5.1.2-py_3.7-euler_2.8.3-aarch64 | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   9   |      mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3-aarch64 | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   10  |                  mxnet-1.2.1-python3.6              |         MXNet         |          |
+-------+-----------------------------------------------------+-----------------------+----------+
|   11  |   optverse_0.2.0-pygrassland_1.1.0-py_3.7-ubuntu_18.04-x86_64 |           OR          |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   12  |                  pytorch-cp36-1.0.0                 |        PyTorch        |          |
+-------+-----------------------------------------------------+-----------------------+----------+
|   13  |                  pytorch-cp36-1.3.0                 |        PyTorch        |          |
+-------+-----------------------------------------------------+-----------------------+----------+
|   14  |                  pytorch-cp36-1.4.0                 |        PyTorch        |          |
+-------+-----------------------------------------------------+-----------------------+----------+
|   15  |      pytorch_1.8.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64 | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   16  |       pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 |        PyTorch        |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   17  |      pytorch_1.8.1-cann_5.1.2-py_3.7-euler_2.8.3-aarch64 | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   18  |      pytorch_1.8.1-cann_6.0.0-py_3.7-euler_2.8.3-aarch64 | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   19  |       pytorch_1.8.1-cuda_11.1-py_3.7-ubuntu_18.04-x86_64 |        PyTorch        |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   20  |       pytorch_1.8.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 |        PyTorch        |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   21  |       pytorch_1.9.1-cuda_11.1-py_3.7-ubuntu_18.04-x86_64 |        PyTorch        |   1000   |
+-------+-----------------------------------------------------+-----------------------+----------+
|   22  |                  ray-cp36-0.7.4                     |          Ray          |          |
+-------+-----------------------------------------------------+-----------------------+----------+
```

使用 ma-cli ma-job get-flavor 命令查询 ModelArts 训练资源规格

执行**ma-cli ma-job get-flavor**命令查询ModelArts训练的资源规格。

```
$ ma-cli ma-job get-flavor -h
Usage: ma-cli ma-job get-flavor [OPTIONS]

  Get job flavor info.

  Example:

  # Get training job flavors
  ma-cli ma-job get-flavor

Options:
```

```
-t, --flavor-type [CPU|GPU|Ascend]
                        Type of training job flavor.
-v, --verbose           Show detailed information about training flavors.
-C, --config-file TEXT      Configure file path for authorization.
-D, --debug             Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
-H, -h, --help          Show this message and exit.
```

表 6-35 参数说明

参数名	参数类型	是否必选	参数说明
-t / --flavor-type	String	否	资源规格类型，如果不指定默认返回所有的资源规格。
-v / --verbose	Bool	否	显示详细的信息开关，默认关闭。

示例：查看训练作业的资源规格及类型。

```
ma-cli ma-job get-flavor
```

```
(PyTorch-1.4) [ma-user work]$ma-cli ma-job get-flavor
+-------+-------------------------------+-----------------------------------+-------------+
| index |           flavor id           |            flavor name            | flavor type |
+-------+-------------------------------+-----------------------------------+-------------+
|   1   |     modelarts.kat1.8xlarge    | Computing NPU(8*Ascend) instance  |    Ascend   |
+-------+-------------------------------+-----------------------------------+-------------+
|   2   |     modelarts.kat1.xlarge     |  Computing NPU(Ascend) instance   |    Ascend   |
+-------+-------------------------------+-----------------------------------+-------------+
|   3   |      modelarts.vm.cpu.2u      |     Computing CPU(2U) instance    |     CPU     |
+-------+-------------------------------+-----------------------------------+-------------+
|   4   |      modelarts.vm.cpu.8u      |     Computing CPU(8U) instance    |     CPU     |
+-------+-------------------------------+-----------------------------------+-------------+
|   5   |  modelarts.vm.cpu.8u16g.119   |     Computing CPU(8U) instance    |     CPU     |
+-------+-------------------------------+-----------------------------------+-------------+
|   6   |    modelarts.vm.v100.large    |    Computing GPU(V100) instance   |     GPU     |
+-------+-------------------------------+-----------------------------------+-------------+
|   7   | modelarts.vm.v100.large.free  |    Computing GPU(V100) instance   |     GPU     |
+-------+-------------------------------+-----------------------------------+-------------+
```

使用 ma-cli ma-job stop 命令停止 ModelArts 训练作业

执行**ma-cli ma-job stop**命令，可停止指定作业id的训练作业。

```
$ ma-cli ma-job stop -h
Usage: ma-cli ma-job stop [OPTIONS]

  Stop training job by job id.

  Example:
```

```
Stop training job by job id
ma-cli ma-job stop --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get training job event by job id.  [required]
  -y, --yes               Confirm stop operation.
  -C, --config-file TEXT  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help          Show this message and exit.
```

表 6-36 参数说明

参数名	参数类型	是否必选	参数说明
-i / --job-id	String	是	ModelArts训练作业ID。
-y / --yes	Bool	否	强制关闭指定训练作业。

示例：停止运行中的训练作业。

```
ma-cli ma-job stop --job-id efd3e2f8xxx
```

6.8.7 ma-cli dli-job 提交 DLI Spark 作业支持的命令

```
$ma-cli dli-job -h
Usage: ma-cli dli-job [OPTIONS] COMMAND [ARGS]...

  DLI spark job submission and query job details.

Options:
  -h, -H, --help  Show this message and exit.

Commands:
  get-job       Get DLI spark job details.
  get-log       Get DLI spark log details.
  get-queue     Get DLI spark queues info.
  get-resource  Get DLI resources info.
  stop          Stop DLI spark job by job id.
  submit        Submit dli spark batch job.
  upload        Upload local file or OBS object to DLI resources.
```

表 6-37 提交 DLI Spark 作业命令总览

命令	命令详情
get-job	查询DLI Spark作业列表及详情。
get-log	查询DLI Spark运行日志。
get-queue	查询DLI队列。
get-resource	查询DLI分组资源。
stop	停止DLI Spark作业。
submit	提交DLI Spark作业。

命令	命令详情
upload	上传本地文件或OBS文件到DLI分组资源。

使用 ma-cli dli-job get-job 命令查询 DLI Spark 作业

执行**ma-cli dli-job get-job**查询DLI Spark作业列表或单个作业详情。

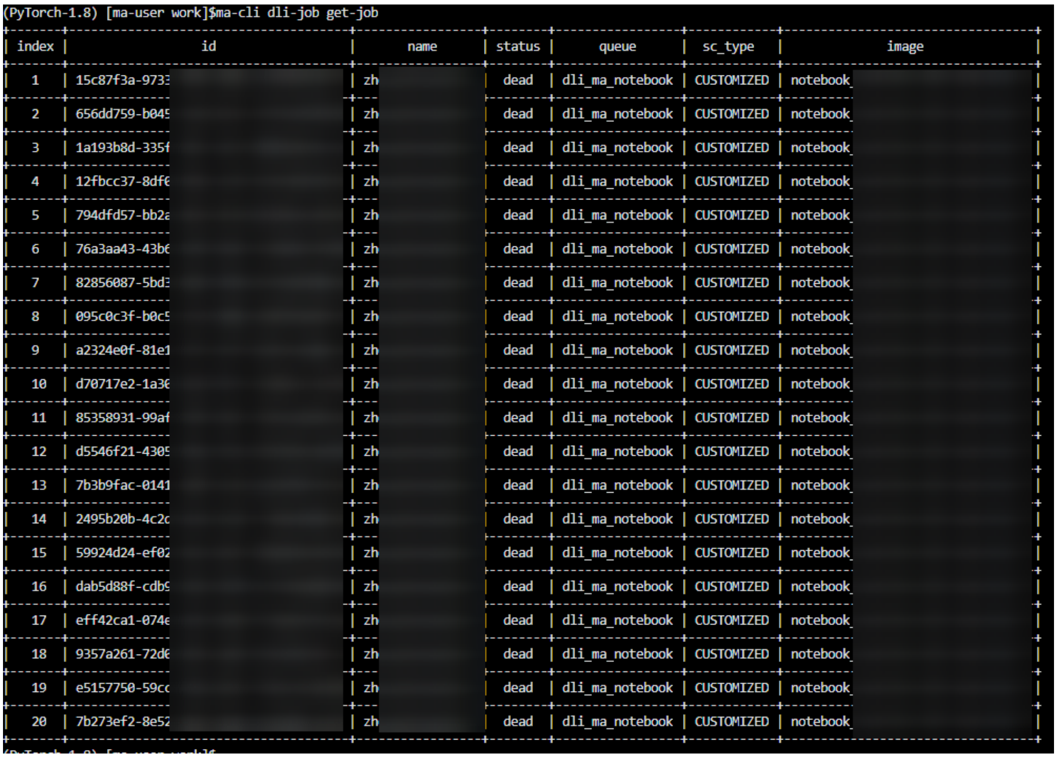
```
ma-cli dli-job get-job -h
Usage: ma-cli dli-job get-job [OPTIONS]

  Get DLI Spark details.

  Example:

  # Get DLI Spark job details by job name
  ma-cli dli-job get-job -n ${job_name}

  # Get DLI Spark job details by job id
  ma-cli dli-job get-job -i ${job_id}

  # Get DLI Spark job list
  ma-cli dli-job get-job --page-size 5 --page-num 1

Options:
  -i, --job-id TEXT               Get DLI Spark job details by job id.
  -n, --job-name TEXT             Get DLI Spark job details by job name.
  -pn, --page-num INTEGER RANGE   Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [x>=1]
  -v, --verbose                   Show detailed information about DLI Spark job details.
  -C, --config-file PATH          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                  Show this message and exit.
```

表 6-38 参数说明

参数名	参数类型	是否必选	参数说明
-i / --job-id	String	否	查询指定DLI Spark作业ID的任务详情。
-n / --job-name	String	否	查询指定作业名称的DLI Spark作业或根据作业名称关键字过滤DLI Spark作业。
-pn / --page-num	Int	否	作业索引页，默认是第1页。
-ps / --page-size	Int	否	每页显示的作业数量，默认是20。
-v / --verbose	Bool	否	显示详细的信息开关，默认关闭。

示例：查询DLI Spark所有作业。

```
ma-cli dli-job get-job
```



(PyTorch-1.8) [ma-user work]$ma-cli dli-job get-job

index	id	name	status	queue	sc_type	image
1	15c87f3a-9733[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
2	656dd759-b045[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
3	1a193b8d-335f[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
4	12fbcc37-8df[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
5	794dfd57-bb2a[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
6	76a3aa43-43b[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
7	82856087-5bd[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
8	095c0c3f-b0c[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
9	a2324e0f-81e1[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
10	d70717e2-1a3[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
11	85358931-99af[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
12	d5546f21-4305[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
13	7b3b9fac-0141[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
14	2495b20b-4c2[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
15	59924d24-ef02[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
16	dab5d88f-cdb[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
17	eff42ca1-074[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
18	9357a261-72d[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
19	e5157750-59c[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
20	7b273ef2-8e52[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]

使用 ma-cli dli-job submit 命令提交 DLI Spark 作业

执行**ma-cli dli-job submit**命令提交DLI Spark作业。

ma-cli dli-job submit命令需要指定一个位置参数YAML_FILE表示作业的配置文件路径，如果不指定该参数，则表示配置文件为空。配置文件是一个YAML格式的文件，里面的参数就是命令的option参数。此外，如果用户在命令行中同时指定YAML_FILE配置文件和option参数，命令行中指定的option参数的值将会覆盖配置文件相同的值。

命令参数预览

```
ma-cli dli-job submit -h
Usage: ma-cli dli-job submit [OPTIONS] [YAML_FILE]...

  Submit DLI Spark job.

  Example:

  ma-cli dli-job submit  --name test-spark-from-sdk
                         --file test/sub_dli_task.py
                         --obs-bucket dli-bucket
                         --queue dli_test
                         --spark-version 2.4.5
                         --driver-cores 1
                         --driver-memory 1G
                         --executor-cores 1
                         --executor-memory 1G
                         --num-executors 1

Options:
  --file TEXT                     Python file or app jar.
  -cn, --class-name TEXT          Your application's main class (for Java / Scala apps).
  --name TEXT                     Job name.
  --image TEXT                    Full swr custom image path.
  --queue TEXT                    Execute queue name.
  -obs, --obs-bucket TEXT         DLI obs bucket to save logs.
  -sv, --spark-version TEXT       Spark version.
```

```
-st, --sc-type [A|B|C]          Compute resource type.
--feature [basic|custom|ai]     Type of the Spark image used by a job (default: basic).
-ec, --executor-cores INTEGER   Executor cores.
-em, --executor-memory TEXT     Executor memory (eg. 2G/2048MB).
-ne, --num-executors INTEGER    Executor number.
-dc, --driver-cores INTEGER     Driver cores.
-dm, --driver-memory TEXT       Driver memory (eg. 2G/2048MB).
--conf TEXT                     Arbitrary Spark configuration property (eg. <PROP=VALUE>).
--resources TEXT                Resources package path.
--files TEXT                    Files to be placed in the working directory of each executor.
--jars TEXT                     Jars to include on the driver and executor class paths.
-pf, --py-files TEXT            Python files to place on the PYTHONPATH for Python apps.
--groups TEXT                   User group resources.
--args TEXT                     Spark batch job parameter args.
-q, --quiet                     Exit without waiting after submit successfully.
-C, --config-file PATH          Configure file path for authorization.
-D, --debug                     Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
-H, -h, --help                  Show this message and exit.
```

yaml文件预览

```
# dli-demo.yaml
name: test-spark-from-sdk
file: test/sub_dli_task.py
obs-bucket: ${your_bucket}
queue: dli_notebook
spark-version: 2.4.5
driver-cores: 1
driver-memory: 1G
executor-cores: 1
executor-memory: 1G
num-executors: 1

## [Optional]
jars:
  - ./test.jar
  - obs://your-bucket/jars/test.jar
  - your_group/test.jar

## [Optional]
files:
  - ./test.csv
  - obs://your-bucket/files/test.csv
  - your_group/test.csv

## [Optional]
python-files:
  - ./test.py
  - obs://your-bucket/files/test.py
  - your_group/test.py

## [Optional]
resources:
  - name: your_group/test.py
    type: pyFile
  - name: your_group/test.csv
    type: file
  - name: your_group/test.jar
    type: jar
  - name: ./test.py
    type: pyFile
  - name: obs://your-bucket/files/test.py
    type: pyFile

## [Optional]
groups:
  - group1
  - group2
```

指定options参数提交DLI Spark作业示例：

```
$ ma-cli dli-job submit --name test-spark-from-sdk \
                       --file test/sub_dli_task.py \
                       --obs-bucket ${your_bucket} \
                       --queue dli_test \
                       --spark-version 2.4.5 \
                       --driver-cores 1 \
                       --driver-memory 1G \
                       --executor-cores 1 \
                       --executor-memory 1G \
                       --num-executors 1
```

表 6-39 参数说明

参数名	参数类型	是否必选	参数说明
YAML_FILE	String	否	DLI Spark作业的配置文件本地路径，如果不传则表示配置文件为空。
--file	String	是	程序运行入口文件，支持本地文件路径、OBS路径或者用户已上传到DLI资源管理系统的类型为jar或pyFile的程序包名。
-cn / --class_name	String	是	批处理作业的Java/Spark主类。
--name	String	否	创建时用户指定的作业名称，不能超过128个字符。
--image	String	否	自定义镜像路径，格式为：组织名/镜像名:镜像版本。当用户设置“**feature**”为“custom”时，该参数生效。用户可通过与“**feature**”参数配合使用，指定作业运行使用自定义的Spark镜像。
-obs / --obs-bucket	String	否	保存Spark作业的obs桶，需要保存作业时配置该参数。同时也可作为提交本地文件到resource的中转站。
-sv/ --spark-version	String	否	作业使用Spark组件的版本号。
-st / `--sc-type	String	否	如果当前Spark组件版本为2.3.2，则不填写该参数。 如果当前Spark组件版本为2.3.3，则在“**feature**”为“basic”或“ai”时填写。如果不填写，则使用默认的Spark组件版本号2.3.2。
--feature	String	否	作业特性。表示用户作业使用的Spark镜像类型，默认值为basic。 ● basic：表示使用DLI提供的基础Spark镜像。 ● custom：表示使用用户自定义的Spark镜像。 ● ai：表示使用DLI提供的AI镜像。

参数名	参数类型	是否必选	参数说明
--queue	String	否	用于指定队列，填写已创建DLI的队列名。必须为通用类型的队列。队列名称的获取请参考**表6-41**。
-ec / --executor-cores	String	否	Spark应用每个Executor的CPU核数。该配置项会替换sc_type中对应的默认参数。
-em / --executor-memory	String	否	Spark应用的Executor内存，参数配置例如2G，2048M。该配置项会替换“**sc_type**”中对应的默认参数，使用时必须带单位，否则会启动失败。
-ne / --num-executors	String	否	Spark应用Executor的个数。该配置项会替换sc_type中对应的默认参数。
-dc / --driver-cores	String	否	Spark应用Driver的CPU核数。该配置项会替换sc_type中对应的默认参数。
-dm / --driver-memory	String	否	Spark应用的Driver内存，参数配置例如2G，2048M。该配置项会替换“**sc_type**”中对应的默认参数，使用时必须带单位，否则会启动失败。
--conf	Array of String	否	batch配置项，参考**Spark Configuration**。如果需要指定多个参数，可以使用--conf conf1 --conf conf2。
--resources	Array of String	否	资源包名称。支持本地文件，OBS路径及用户已上传到DLI资源管理系统的文件。如果需要指定多个参数，可以使用--resources resource1 --resources resource2。
--files	Array of String	否	用户已上传到DLI资源管理系统的类型为file的资源包名。也支持指定OBS路径，例如：obs://桶名/包名。同时也支持本地文件。如果需要指定多个参数，可以使用--files file1 --files file2。
--jars	Array of String	否	用户已上传到DLI资源管理系统的类型为jar的程序包名。也支持指定OBS路径，例如：obs://桶名/包名。也支持本地文件。如果需要指定多个参数，可以使用--jars jar1 --jars jar2。
-pf /--python-files	Array of String	否	用户已上传到DLI资源管理系统的类型为pyFile的资源包名。也支持指定OBS路径，例如：obs://桶名/包名。也支持本地文件。如果需要指定多个参数，可以使用--python-files py1 --python-files py2。
--groups	Array of String	否	资源分组名称，如果需要指定多个参数，可以使用--groups group1 --groups group2。

参数名	参数类型	是否必选	参数说明
--args	Array of String	否	传入主类的参数，即应用程序参数。如果需要指定多个参数，可以使用--args arg1 --args arg2。
-q / --quiet	Bool	否	提交DLI Spark作业成功后直接退出，不再同步打印任务状态。

示例

- 通过YAML_FILE文件提交DLI Spark作业。

```
$ma-cli dli-job submit dli_job.yaml
```

```
(PyTorch-1.4) [ma-user work]$ma-cli dli-job submit ./dli-job.yaml
[ OK ] Current DLI job id is: 01b698b8-9fd6-4a8e-bc3c-6821c6405b14
[ OK ] starting
[ OK ] running
[ OK ] success
[ OK ] Successfully submit DLI spark job [ 01b698b8-9fd6-4a8e-bc3c-6821c6405b14 ].
```

- 指定命令行options参数提交DLI Spark作业。

```
$ma-cli dli-job submit --name test-spark-from-sdk \
>                     --file test/jumpstart-trainingjob-gallery-pytorch-sample.ipynb \
>                     --queue dli_ma_notebook \
>                     --spark-version 2.4.5 \
>                     --driver-cores 1 \
>                     --driver-memory 1G \
>                     --executor-cores 1 \
>                     --executor-memory 1G \
>                     --num-executors 1
```

```
(PyTorch-1.4) [ma-user work]$ma-cli dli-job submit --name test-spark-from-sdk \
>                     --file test/jumpstart-trainingjob-gallery-pytorch-sample.ipynb \
>                     --queue dli_ma_notebook \
>                     --spark-version 2.4.5 \
>                     --driver-cores 1 \
>                     --driver-memory 1G \
>                     --executor-cores 1 \
>                     --executor-memory 1G \
>                     --num-executors 1
[ OK ] Current DLI job id is: ae856c20-e9ae-49ca-8409-7a02652297b8
[ OK ] starting
```

使用 ma-cli dli-job get-log 命令查询 DLI Spark 运行日志

执行**ma-cli dli-job get-log**命令查询DLI Spark作业后台的日志。

```
$ ma-cli dli-job get-log -h
Usage: ma-cli dli-job get-log [OPTIONS]

  Get DLI spark job log details.

  Example:

  # Get job log by job id
  ma-cli dli-job get-log --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get DLI spark job details by job id.  [required]
  -C, --config-file TEXT  Configure file path for authorization.
```

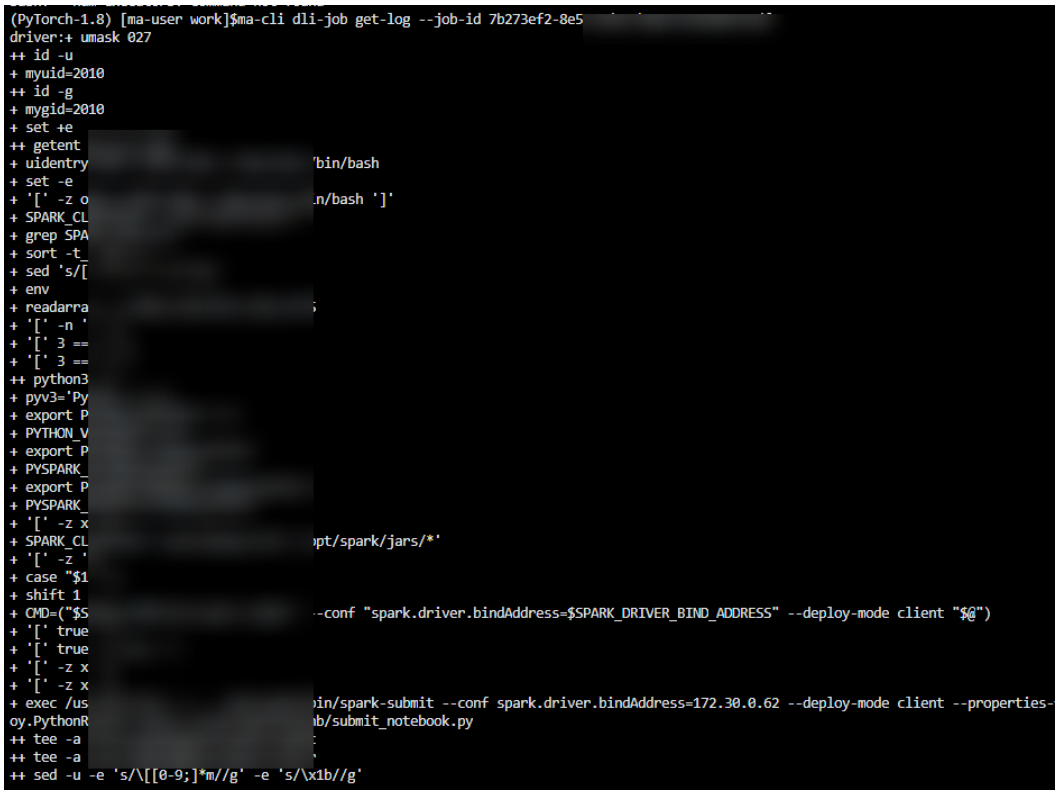
```
-D, --debug           Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT    CLI connection profile to use. The default profile is "DEFAULT".
-H, -h, --help        Show this message and exit.
```

表 6-40 参数说明

参数名	参数类型	是否必选	参数说明
-i / --job-id	String	是	查询指定DLI Spark作业ID的任务日志。

示例：查询指定作业ID的DLI Spark作业运行日志。

```
ma-cli dli-job get-log --job-id ${your_job_id}
```



使用 ma-cli dli-job get-queue 命令查询 DLI 队列

执行**ma-cli dli-job get-queue**命令查询DLI队列。

```
ma-cli dli-job get-queue -h
Usage: ma-cli dli-job get-queue [OPTIONS]

  Get DLI queues info.

  Example:

  # Get DLI queue details by queue name
  ma-cli dli-job get-queue --queue-name $queue_name}

Options:
  -pn, --page-num INTEGER RANGE   Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [x>=1]
```

```
-n, --queue-name TEXT          Get DLI queue details by queue name.
-t, --queue-type [sql|general|all]
                               DLI queue type (default "all").
-tags, --tags TEXT             Get DLI queues by tags.
-C, --config-file PATH         Configure file path for authorization.
-D, --debug                    Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT             CLI connection profile to use. The default profile is "DEFAULT".
-H, -h, --help                 Show this message and exit.
```

表 6-41 参数说明

参数名	参数类型	是否必选	参数说明
-n / --queue-name	String	否	指定需要查询的DLI队列名称。
-t / --queue-type	String	否	指定查询的DLI队列类型，支持sql、general和all，默认是all。
-tags / --tags	String	否	指定查询的DLI队列tags。
-pn / --page-num	Int	否	DLI队列页索引，默认是第1页。
-ps / --page-size	Int	否	每页显示的DLI队列数量，默认是20。

示例：查询队列名为“dli_ma_notebook”的队列信息。

```
ma-cli dli-job get-queue --queue-name dli_ma_notebook
```

```
(PyTorch-1.8) [ma-user work]$ ma-cli dli-job get-queue --queue-name dli_ma_notebook
{'chargingMode': 1,
 'create_time': 1668585417422,
 'cuCount': 16,
 'cu_spec': 16,
 'description': '',
 'enterprise_project_id': '0',
 'is_success': True,
 'message': '',
 'owner': '                              ',
 'queueName': 'dli_ma_notebook',
 'queueType': 'general',
 'queue_id': 8____,
 'resource_id': '242e7af8-c9d1                          e',
 'resource_mode': 1,
 'resource_type': 'container',
 'support_spark_versions': ['2.3.2', '2.4.5', '3.1.1']}
```

使用 ma-cli dli-job get-resource 命令查询 DLI 分组资源

执行**ma-cli dli-job get-resource**命令获取DLI资源详细信息，如资源名称，资源类型等。

```
$ ma-cli dli-job get-resource -h
Usage: ma-cli dli-job get-resource [OPTIONS]

  Get DLI resource info.

  Example:

  # Get DLI resource details by resource name
```

```
 ma-cli dli-job get-resource --resource-name ${resource_name}

Options:
  -n, --resource-name TEXT        Get DLI resource details by resource name.
  -k, --kind [jar|pyFile|file|modelFile]
                                  DLI resources type.
  -g, --group TEXT                Get DLI resources by group.
  -tags, --tags TEXT              Get DLI resources by tags.
  -C, --config-file TEXT          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                  Show this message and exit.
```

表 6-42 参数说明

参数名	参数类型	是否必选	参数说明
-n / --resource-name	String	否	按DLI分组资源名称查询DLI资源详细信息。
-k / --kind	String	否	按DLI分组资源类型查询DLI资源详细信息，支持jar、pyFile、file和modelFile。
-g / --group	String	否	按DLI分组资源组名查询DLI资源组详细信息。
-tags / --tags	String	否	通过DLI分组资源tags获取DLI资源详细信息。

示例：查询所有DLI分组资源信息。

```
ma-cli dli-job get-resource
```

使用 ma-cli dli-job upload 命令上传文件到 DLI 分组资源

ma-cli dli-job upload命令支持将本地文件或OBS文件上传到DLI资源组。

```
$ ma-cli dli-job upload -h
Usage: ma-cli dli-job upload [OPTIONS] PATHS...

  Upload DLI resource.

  Tips: --obs-path is need when upload local file.

  Example:

  # Upload an OBS path to DLI resource
  ma-cli dli-job upload obs://your-bucket/test.py -g test-group --kind pyFile

  # Upload a local path to DLI resource
  ma-cli dli-job upload ./test.py -g test-group -obs ${your-bucket} --kind pyFile

  # Upload local path and OBS path to DLI resource
  ma-cli dli-job upload ./test.py obs://your-bucket/test.py -g test-group -obs ${your-bucket}
```



```
Options:
  -k, --kind [jar|pyFile|file]  DLI resources type.
  -g, --group TEXT              DLI resources group.
  -tags, --tags TEXT            DLI resources tags, follow --tags `key1`=`value1`.
  -obs, --obs-bucket TEXT       OBS bucket for upload local file.
  -async, --is-async            whether to upload resource packages in asynchronous mode. The default value is
False.
  -C, --config-file TEXT        Configure file path for authorization.
  -D, --debug                   Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT            CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                Show this message and exit.
```

表 6-43 参数说明

参数名	参数类型	是否必选	参数说明
PATHS	String	是	需要上传到DLI分组资源的本地文件路径或者obs路径，支持同时传入多个路径。
-k / --kind	String	否	上传文件的类型，支持jar、pyFile和file。
-g / --group	String	否	上传文件的DLI分组名。
-tags / --tags	String	否	上传文件的tag。
-obs / --obs-bucket	String	否	如果上传文件包含本地路径，则需要指定一个OBS桶作为中转。
-async / --is-async	Bool	否	异步上传文件，推荐使用。

示例

- 上传本地文件到DLI分组资源
  ma-cli dli-job upload ./test.py -obs ${your-bucket} --kind pyFile

  ```
  (PyTorch-1.8) [ma-user work]$ma-cli dli-job upload ./test.py -obs obs://[redacted] --kind pyFile
  [ OK ] Upload ['test.py']  successfully.
  ```

- 上传OBS文件到DLI分组资源
  ma-cli dli-job upload obs://your-bucket/test.py --kind pyFile

  ```
  (PyTorch-1.8) [ma-user work]$ma-cli dli-job upload obs://[redacted]/test.py --kind pyFile
  [ OK ] Upload ['test.py']  successfully.
  ```

使用 ma-cli dli-job stop 命令停止 DLI Spark 作业

执行**ma-cli dli-job stop**命令停止DLI Spark作业。

```
$ ma-cli dli-job stop -h
Usage: ma-cli dli-job stop [OPTIONS]

  Stop DLI spark job by job id.

  Example:

  Stop training job by job id
  ma-cli dli-job stop --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get DLI spark job event by job id.  [required]
```

```
-y, --yes               Confirm stop operation.
-C, --config-file TEXT  Configure file path for authorization.
-D, --debug             Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
-H, -h, --help          Show this message and exit.
```

表 6-44 参数说明

参数名	参数类型	是否必选	参数说明
-i / --job-id	String	是	DLI Spark作业ID。
-y / --yes	Bool	否	强制关闭指定DLI Spark作业。

示例

```
ma-cli dli-job stop -i ${your_job_id}
```

6.8.8 使用 ma-cli obs-copy 命令复制 OBS 数据

使用**ma-cli obs-copy [SRC] [DST]**可以实现本地和OBS文件或文件夹的相互复制。

```
$ma-cli obs-copy -h
Usage: ma-cli obs-copy [OPTIONS     ] SRC DST

  Copy file or directory to between OBS and local path. Example:

  # Upload local file to OBS path
  ma-cli obs-copy ./test.zip obs://your-bucket/copy-data/

  # Upload local directory to OBS path
  ma-cli obs-copy ./test/ obs://your-bucket/copy-data/

  # Download OBS file to local path
  ma-cli obs-copy obs://your-bucket/copy-data/test.zip ./test.zip

  # Download OBS directory to local path
  ma-cli obs-copy obs://your-bucket/copy-data/ ./test/

Options:
  -d, --drop-last-dir     Whether to drop last directory when copy folder. if True, the last directory of the
source folder will not copy to the destination folder.  [default: False]
  -C, --config-file PATH  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help          Show this message and exit.
```

表 6-45 参数说明

参数名	参数类型	是否必选	参数说明
-d / --drop-last-dir	Bool	否	如果指定，在复制文件夹时不会将源文件夹最后一级目录复制至目的文件夹下，仅对文件夹复制有效。

命令示例

上传文件到OBS中

```
$ ma-cli obs-copy ./test.csv obs://${your_bucket}/test-copy/
[ OK ] local src path: [ /home/ma-user/work/test.csv ]
[ OK ] obs dst path: [ obs://${your_bucket}/test-copy/ ]
```

上传文件夹到OBS中，对应上传到OBS的目录为obs://${your_bucket}/test-copy/data/

```
$ ma-cli obs-copy /home/ma-user/work/data/ obs://${your_bucket}/test-copy/
[ OK ] local src path: [ /home/ma-user/work/data/ ]
[ OK ] obs dst path: [ obs://${your_bucket}/test-copy/ ]
```

上传文件夹到OBS中，并指定--drop-last-dir，对应上传到OBS的目录为obs://${your_bucket}/test-copy/

```
$ ma-cli obs-copy /home/ma-user/work/data/ obs://${your_bucket}/test-copy/ --drop-last-dir
[ OK ] local src path: [ /home/ma-user/work/data ]
[ OK ] obs dst path: [ obs://${your_bucket}/test-copy/ ]
```

从OBS下载文件夹到本地磁盘中

```
$ ma-cli obs-copy obs://${your_bucket}/test-copy/ ~/work/test-data/
[ OK ] obs src path: [ obs://${your_bucket}/test-copy/ ]
[ OK ] local dst path: [ /home/ma-user/work/test-data/ ]
```

6.9 在 Notebook 中使用 Moxing 命令

6.9.1 MoXing Framework 功能介绍

MoXing Framework模块为MoXing提供基础公共组件，例如访问华为云的OBS服务，和具体的AI引擎解耦，在ModelArts支持的所有AI引擎(TensorFlow、MXNet、PyTorch、MindSpore等)下均可以使用。目前，提供的MoXing Framework功能中主要包含操作OBS组件，即下文中描述的mox.file接口。

说明

Moxing主要使用场景为提升从OBS读取和下载数据的易用性，适配对象为OBS对象桶，对于OBS并行文件系统部分接口可能存在问题，不建议使用。生产业务代码开发建议直接调用OBS Python SDK，详情请参见**Python SDK接口概览**。

为什么要用 mox.file

使用Python打开一个本地文件，如下所示：

```
with open('/tmp/a.txt', 'r') as f:
  print(f.read())
```

OBS目录以“obs://”开头，比如“obs://bucket/XXX.txt”。用户无法直接使用open方法打开OBS文件，上面描述的打开本地文件的代码将会报错。

OBS提供了很多方式和工具给用户使用，如SDK、API、console、OBS Browser等，ModelArts mox.file提供了一套更为方便地访问OBS的API，允许用户通过一系列模仿操作本地文件系统的API来操作OBS文件。例如，可以使用以下代码来打开一个OBS上的文件。

```
import moxing as mox
with mox.file.File('obs://bucket_name/a.txt', 'r') as f:
  print(f.read())
```

例如，列举一个本地路径会使用如下Python代码。

```
import os
os.listdir('/tmp/my_dir/')
```

如果要列举一个OBS路径，mox.file则需要如下代码：

```
import moxing as mox
mox.file.list_directory('obs://bucket_name/my_dir/')
```

引入 MoXing Framework 模块

使用MoXing Framework前，您需要在代码的开头先引入MoXing Framework模块。

执行如下代码，引入MoXing模块。

```
import moxing as mox
```

引入 MoXing Framework 的相关说明

在引入MoXing模块后，Python的标准logging模块会被设置为INFO级别，并打印版本号信息。可以通过以下API重新设置logging的等级。

```
import logging

from moxing.framework.util import runtime
runtime.reset_logger(level=logging.WARNING)
```

可以在引入moxing之前，配置环境变量MOX_SILENT_MODE=1，来防止MoXing打印版本号。使用如下Python代码来配置环境变量，需要在import moxing之前就将环境变量配置好。

```
import os
os.environ['MOX_SILENT_MODE'] = '1'
import moxing as mox
```

引入 moxing framework 的数据下载加速特性的相关说明

在使用基于ModelArts预置镜像的训练作业时，可以引入moxing framework的数据下载加速特性。加速特性适用场景为：文件数在100w~1000w的场景、单个大文件及文件大小大于20GB的场景。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型训练 > 训练作业”，进入训练作业管理页面。
2. 单击右上角“创建训练作业”进入创建训练作业页面，在“环境变量”中设置“MA_MOXING_FWVER=2.2.8.0aa484aa”以安装最新moxing framework版本，其他参数填写请参见**创建训练作业**。 配置完成后，可以在训练作业脚本中使用“moxing.file.copy_parallel”接口加速数据下载。
3. 需要时可以通过在训练作业的“环境变量”中设置“MOX_C_ACCELERATE=0”，来关闭数据下载加速特性。

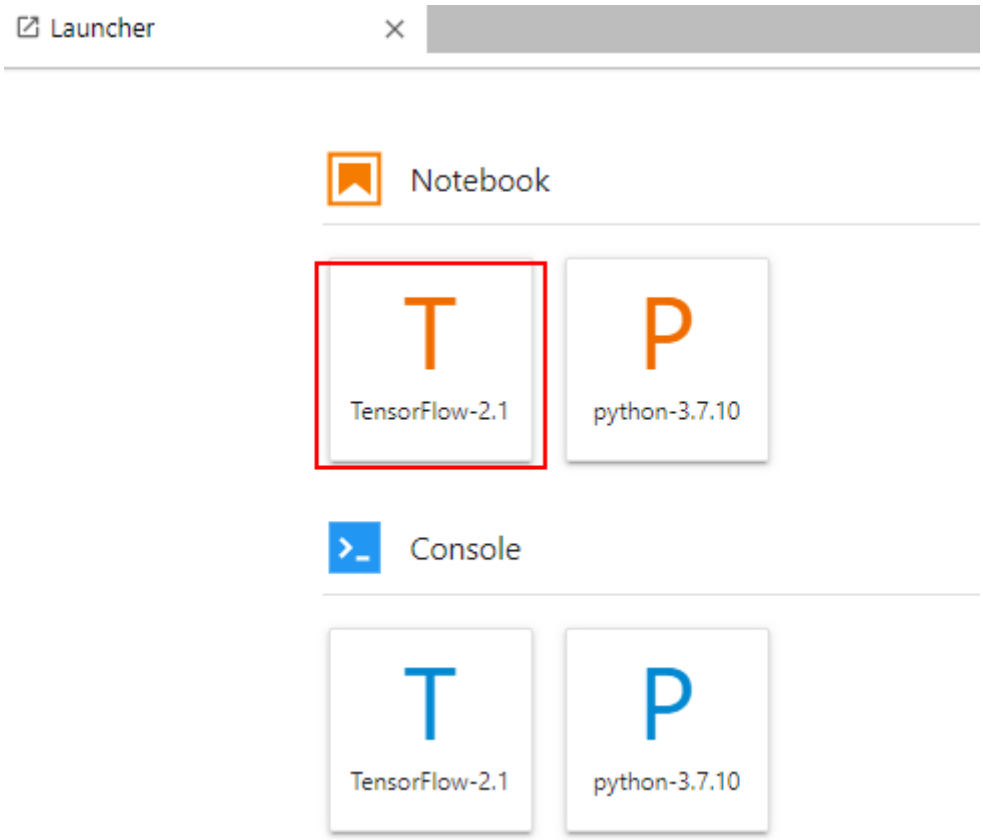
6.9.2 Notebook 中快速使用 MoXing

本文档介绍如何在ModelArts中调用MoXing Framework接口。

进入 ModelArts，创建 Notebook 实例

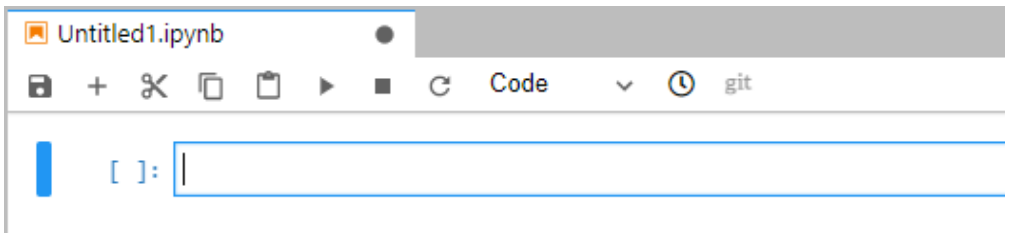
1. 登录ModelArts管理控制台，在左侧菜单栏中选择“开发空间>Notebook”，进入“Notebook”管理页面。
2. 单击“创建”进入“创建Notebook”页面，参考**创建Notebook实例**填写信息并完成Notebook实例创建。
3. 当Notebook实例创建完成后，且状态为“运行中”时，单击“操作”列中的“打开”，进入“JupyterLab Notebook”开发页面。
4. 在JupyterLab的“Launcher”页签下，以TensorFlow为例，您可以单击TensorFlow，创建一个用于编码的文件。

图 6-158 选择不同的 AI 引擎



文件创建完成后，系统默认进入“JupyterLab”编码页面。

图 6-159 进入编码页面



调用 mox.file

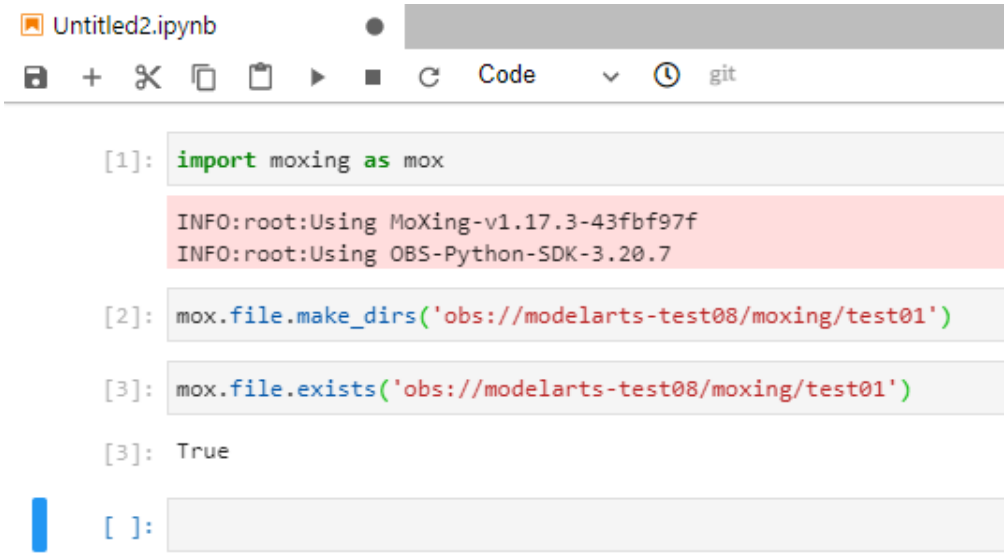
输入如下代码，实现如下几个简单的功能。

1. 引入MoXing Framework。
2. 在已有的“modelarts-test08/moxing”目录下，创建一个“test01”文件夹。
3. 调用代码检查“test01”文件夹是否存在，如果存在，表示上一个操作已成功。

```
import moxing as mox

mox.file.make_dirs('obs://modelarts-test08/moxing/test01')
mox.file.exists('obs://modelarts-test08/moxing/test01')
```

执行结果如图6-160所示。注意，每输入一行代码，单击下“Run”运行。您也可以进入OBS管理控制台，检查“modelarts-test08/moxing”目录，查看“test01”文件夹是否已创建成功。更多MoXing的常用操作请参见MoXing常用操作的样例代码。

图 6-160 运行示例



复制数据到 OBS

在Notebook的在JupyterLab的服务界面，将文件yolov8_train_ascend.zip，复制到已有的OBS桶中，示例代码如下。

```
import os
import zipfile
import moxing as mox
mox.file.copy('yolov8_train_ascend.zip','obs://pcb-data-me/pcb.zip')
```

6.9.3 mox.file 与本地接口的对应关系和切换

API 对应关系

- Python：指本地使用Python对本地文件的操作接口。支持一键切换为对应的MoXing文件操作接口（mox.file）。
- mox.file：指MoXing框架中用于文件操作的接口，其与python接口一一对应关系。

- tf.gfile：指MoXing文件操作接口一一对应的TensorFlow相同功能的接口，在MoXing中，无法自动将文件操作接口自动切换为TensorFlow的接口，下表呈现内容仅表示功能类似，帮助您更快速地了解MoXing文件操作接口的功能。

表 6-46 API 对应关系

Python（本地文件操作接口）	mox.file（MoXing文件操作接口）	tf.gfile（TensorFlow文件操作接口）
glob.glob	mox.file.glob	tf.gfile.Glob
os.listdir	mox.file.list_directory(..., recursive=False)	tf.gfile.ListDirectory
os.makedirs	mox.file.make_dirs	tf.gfile.MakeDirs
os.mkdir	mox.file.mk_dir	tf.gfile.MkDir
os.path.exists	mox.file.exists	tf.gfile.Exists
os.path.getsize	mox.file.get_size	-
os.path.isdir	mox.file.is_directory	tf.gfile.IsDirectory
os.remove	mox.file.remove(..., recursive=False)	tf.gfile.Remove
os.rename	mox.file.rename	tf.gfile.Rename
os.scandir	mox.file.scan_dir	-
os.stat	mox.file.stat	tf.gfile.Stat
os.walk	mox.file.walk	tf.gfile.Walk
open	mox.file.File	tf.gfile.FastGFile(tf.gfile.Gfile)
shutil.copyfile	mox.file.copy	tf.gfile.Copy
shutil.copytree	mox.file.copy_parallel	-
shutil.rmtree	mox.file.remove(..., recursive=True)	tf.gfile.DeleteRecursively

6.9.4 MoXing 常用操作的样例代码

读写操作

- 读取一个OBS文件。

  例如读取“obs://bucket_name/obs_file.txt”文件内容，返回string（字符串类型）。

  ```
  import moxing as mox
  file_str = mox.file.read('obs://bucket_name/obs_file.txt')
  ```

  也可以使用打开文件对象并读取的方式来实现，两者是等价的。

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
  file_str = f.read()
```

- 从文件中读取一行，返回string，以换行符结尾。同样可以打开OBS的文件对象。

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
  file_line = f.readline()
```

- 从文件中读取所有行，返回一个list，每个元素是一行，以换行符结尾。

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
  file_line_list = f.readlines()
```

- 以二进制模式读取一个OBS文件。

  例如读取“obs://bucket_name/obs_file.bin”文件内容，返回bytes（字节类型）。

```
import moxing as mox
file_bytes = mox.file.read('obs://bucket_name/obs_file.bin', binary=True)
```

  也可以使用打开文件对象并读取的方式来实现，两者是等价的。

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.bin', 'rb') as f:
  file_bytes = f.read()
```

  以二进制模式打开的文件也支持读取一行或者读取所有行，用法不变。

- 将字符串写入一个文件。

  例如将字符串“Hello World!”写入OBS文件“obs://bucket_name/obs_file.txt”中。

```
import moxing as mox
mox.file.write('obs://bucket_name/obs_file.txt', 'Hello World!')
```

  也可以使用打开文件对象并写入的方式来实现，两者是等价的。

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'w') as f:
  f.write('Hello World!')
```

  说明

  用写入模式打开文件或者调用mox.file.write时，如果被写入文件不存在，则会创建，如果已经存在，则会覆盖。

- 追加一个OBS文件。

  例如将字符串“Hello World!”追加到“obs://bucket_name/obs_file.txt”文件中。

```
import moxing as mox
mox.file.append('obs://bucket_name/obs_file.txt', 'Hello World!')
```

  也可以使用打开文件对象并追加的方式来实现，两者是等价的。

```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'a') as f:
  f.write('Hello World!')
```

  用追加模式打开文件或者调用**mox.file.append**时，如果被写入文件不存在，则会创建，如果已经存在，则直接追加。

  当被追加的源文件比较大时，例如“obs://bucket_name/obs_file.txt”文件大小超过5MB时，追加一个OBS文件的性能比较低。

  说明

  如果以写入模式或追加模式打开文件，当调用write方法时，待写入内容只是暂时的被存在的缓冲区，直到关闭文件对象（退出with语句时会自动关闭文件对象）或者主动调用文件对象的close()方法或flush()方法时，文件内容才会被写入。

列举操作

- 列举一个OBS目录，只返回顶层结果（相对路径），不做递归列举。

  例如列举“obs://bucket_name/object_dir”，返回该目录下所有的文件和文件夹，不会递归查询。

  假设“obs://bucket_name/object_dir”中有如下结构

  ```
  bucket_name
      |- object_dir
        |- dir0
          |- file00
        |- file1
  ```

  调用如下代码：

  ```
  import moxing as mox
  mox.file.list_directory('obs://bucket_name/object_dir')
  ```

  返回一个list：

  ```
  ['dir0', 'file1']
  ```

- 递归列举一个OBS目录，返回目录中所有的文件和文件夹（相对路径），并且会做递归查询。

  假设obs://bucket_name/object_dir中有如下结构。

  ```
  bucket_name
      |- object_dir
        |- dir0
          |- file00
        |- file1
  ```

  调用如下代码：

  ```
  import moxing as mox
  mox.file.list_directory('obs://bucket_name/object_dir', recursive=True)
  ```

  返回一个list：

  ```
  ['dir0', 'dir0/file00', 'file1']
  ```

创建文件夹操作

创建一个OBS目录（即OBS文件夹）。支持递归创建，即如果“sub_dir_0”文件夹不存在，也会自动被创建，如果已经存在，则不起任何作用。

```
import moxing as mox
mox.file.make_dirs('obs://bucket_name/sub_dir_0/sub_dir_1')
```

查询操作

- 判断一个OBS文件是否存在，如果存在则返回**True**，如果不存在则返回**False**。

  ```
  import moxing as mox
  mox.file.exists('obs://bucket_name/sub_dir_0/file.txt')
  ```

- 判断一个OBS文件夹是否存在，如果存在则返回**True**，如果不存在则返回**False**。

  ```
  import moxing as mox
  mox.file.exists('obs://bucket_name/sub_dir_0/sub_dir_1')
  ```

  说明

  由于OBS允许同名的文件和文件夹（Unix操作系统不允许），如果存在同名的文件和文件夹，例如“obs://bucket_name/sub_dir_0/abc”，当调用mox.file.exists时，不论abc是文件还是文件夹，都会返回True。

- 判断一个OBS路径是否为文件夹，如果是则返回**True**，否则返回**False**。

  ```
  import moxing as mox
  mox.file.is_directory('obs://bucket_name/sub_dir_0/sub_dir_1')
  ```

> **说明**
>
> 由于OBS允许同名的文件和文件夹（Unix操作系统不允许），如果存在同名的文件和文件夹，例如obs://bucket_name/sub_dir_0/abc，当调用mox.file.is_directory时，会返回True。

- 获取一个OBS文件的大小，单位为bytes。

  例如获取“obs://bucket_name/obs_file.txt”的文件大小。

  ```
  import moxing as mox
  mox.file.get_size('obs://bucket_name/obs_file.txt')
  ```

- 递归获取一个OBS文件夹下所有文件的大小，单位为bytes。

  例如获取“obs://bucket_name/object_dir”目录下所有文件大小的总和。

  ```
  import moxing as mox
  mox.file.get_size('obs://bucket_name/object_dir', recursive=True)
  ```

- 获取一个OBS文件或文件夹的stat信息，stat信息中包含如下信息。
  - length：文件大小。
  - mtime_nsec：创建时间戳。
  - is_directory：是否为目录。

  例如查询一个OBS文件“obs://bucket_name/obs_file.txt”，此文件地址也可以替换成一个文件夹地址。

  ```
  import moxing as mox
  stat = mox.file.stat('obs://bucket_name/obs_file.txt')
  print(stat.length)
  print(stat.mtime_nsec)
  print(stat.is_directory)
  ```

删除操作

- 删除一个OBS文件。

  例如删除“obs://bucket_name/obs_file.txt”。

  ```
  import moxing as mox
  mox.file.remove('obs://bucket_name/obs_file.txt')
  ```

- 删除一个OBS目录，并且递归的删除这个目录下的所有内容。如果这个目录不存在，则会报错。

  例如删除“obs://bucket_name/sub_dir_0”下的所有内容。

  ```
  import moxing as mox
  mox.file.remove('obs://bucket_name/sub_dir_0', recursive=True)
  ```

移动和复制操作

- 移动一个OBS文件或文件夹。移动操作本身是用“复制+删除”来实现的。
  - 一个OBS文件移动到另一个OBS文件，例如将“obs://bucket_name/obs_file.txt”移动到“obs://bucket_name/obs_file_2.txt”。

    ```
    import moxing as mox
    mox.file.rename('obs://bucket_name/obs_file.txt', 'obs://bucket_name/obs_file_2.txt')
    ```

    > **说明**
    >
    > 移动和复制操作不可以跨桶，必须在同一个桶内操作。

  - 从OBS移动到本地，例如将“obs://bucket_name/obs_file.txt”移动到“/tmp/obs_file.txt”。

    ```
    import moxing as mox
    mox.file.rename('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
    ```

  - 从本地移动到OBS，例如将“/tmp/obs_file.txt”移动到“obs://bucket_name/obs_file.txt”。

```
import moxing as mox
mox.file.rename('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
```

  - 从本地移动到本地，例如将“/tmp/obs_file.txt”移动到“/tmp/obs_file_2.txt”，该操作相当于**os.rename**。

```
import moxing as mox
mox.file.rename('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
```

所有的移动操作均可以操作文件夹，如果操作的是文件夹，那么则会递归移动文件夹下所有的内容。

- 复制一个文件。**mox.file.copy**仅支持对文件操作，如果要对文件夹进行操作，请使用**mox.file.copy_parallel**。
  - 从OBS复制到OBS，例如将“obs://bucket_name/obs_file.txt”复制到“obs://bucket_name/obs_file_2.txt”。

```
import moxing as mox
mox.file.copy('obs://bucket_name/obs_file.txt', 'obs://bucket_name/obs_file_2.txt')
```

  - 将OBS文件复制到本地，也就是下载一个OBS文件。例如下载“obs://bucket_name/obs_file.txt”到本地“/tmp/obs_file.txt”。

```
import moxing as mox
mox.file.copy('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
```

  - 将本地文件复制到OBS，也就是上传一个OBS文件，例如上传“/tmp/obs_file.txt”到“obs://bucket_name/obs_file.txt”。

```
import moxing as mox
mox.file.copy('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
```

  - 将本地文件复制到本地，操作等价于**shutil.copyfile**，例如将“/tmp/obs_file.txt”复制到“/tmp/obs_file_2.txt”。

```
import moxing as mox
mox.file.copy('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
```

- 复制一个文件夹。**mox.file.copy_parallel**仅支持对文件夹操作，如果要对文件进行操作，请使用**mox.file.copy**。
  - 从OBS复制到OBS，例如将obs://bucket_name/sub_dir_0复制到obs://bucket_name/sub_dir_1

```
import moxing as mox
mox.file.copy_parallel('obs://bucket_name/sub_dir_0', 'obs://bucket_name/sub_dir_1')
```

  - 将OBS文件夹复制到本地，也就是下载一个OBS文件夹。例如下载“obs://bucket_name/sub_dir_0”到本地“/tmp/sub_dir_0”。

```
import moxing as mox
mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/tmp/sub_dir_0')
```

  - 将本地文件夹复制到OBS，也就是上传一个OBS文件夹，例如上传“/tmp/sub_dir_0”到“obs://bucket_name/sub_dir_0”。

```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', 'obs://bucket_name/sub_dir_0')
```

  - 将本地文件夹复制到本地，操作等价于**shutil.copytree**，例如将“/tmp/sub_dir_0”复制到“/tmp/sub_dir_1”。

```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', '/tmp/sub_dir_1')
```

6.9.5 MoXing 进阶用法的样例代码

如果您已经熟悉了常用操作，同时熟悉MoXing Framework API文档以及常用的Python编码，您可以参考本章节使用MoXing Framework的一些进阶用法。

读取完毕后将文件关闭

当读取OBS文件时，实际调用的是HTTP连接读取网络流，注意要记得在读取完毕后将文件关闭。为了防止忘记文件关闭操作，推荐使用with语句，在with语句退出时会自动调用**mox.file.File**对象的**close()**方法：


```
import moxing as mox
with mox.file.File('obs://bucket_name/obs_file.txt', 'r') as f:
  data = f.readlines()
```

利用 pandas 读或写一个 OBS 文件

- 利用pandas读一个OBS文件。

  ```
  import pandas as pd
  import moxing as mox
  with mox.file.File("obs://bucket_name/b.txt", "r") as f:
    csv = pd.read_csv(f)
  ```

- 利用pandas写一个OBS文件。

  ```
  import pandas as pd
  import moxing as mox
  df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
  with mox.file.File("obs://bucket_name/b.txt", "w") as f:
    df.to_csv(f)
  ```

利用文件对象读取图片

使用opencv打开一张图片时，无法传入一个OBS路径，需要利用文件对象读取，考虑以下代码是无法读取到该图片的。

```
import cv2
cv2.imread('obs://bucket_name/xxx.jpg', cv2.IMREAD_COLOR)
```

修改为如下代码：

```
import cv2
import numpy as np
import moxing as mox
img = cv2.imdecode(np.fromstring(mox.file.read('obs://bucket_name/xxx.jpg', binary=True), np.uint8),
cv2.IMREAD_COLOR)
```

将一个不支持 OBS 路径的 API 改造成支持 OBS 路径的 API

pandas中对h5的文件读写**to_hdf**和**read_hdf**既不支持OBS路径，也不支持输入一个文件对象，考虑以下代码会出现错误。

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]}, index=['a', 'b', 'c'])
df.to_hdf('obs://wolfros-net/hdftest.h5', key='df', mode='w')
pd.read_hdf('obs://wolfros-net/hdftest.h5')
```

通过重写pandas源码API的方式，将该API改造成支持OBS路径的形式。

- 写h5到OBS = 写h5到本地缓存 + 上传本地缓存到OBS + 删除本地缓存
- 从OBS读h5 = 下载h5到本地缓存 + 读取本地缓存 + 删除本地缓存

即将以下代码写在运行脚本的最前面，就能使运行过程中的**to_hdf**和**read_hdf**支持OBS路径。

```
import os
import moxing as mox
import pandas as pd
from pandas.io import pytables
from pandas.core.generic import NDFrame

to_hdf_origin = getattr(NDFrame, 'to_hdf')
read_hdf_origin = getattr(pytables, 'read_hdf')


def to_hdf_override(self, path_or_buf, key, **kwargs):
  tmp_dir = '/cache/hdf_tmp'
```



```
  file_name = os.path.basename(path_or_buf)
  mox.file.make_dirs(tmp_dir)
  local_file = os.path.join(tmp_dir, file_name)
  to_hdf_origin(self, local_file, key, **kwargs)
  mox.file.copy(local_file, path_or_buf)
  mox.file.remove(local_file)


def read_hdf_override(path_or_buf, key=None, mode='r', **kwargs):
  tmp_dir = '/cache/hdf_tmp'
  file_name = os.path.basename(path_or_buf)
  mox.file.make_dirs(tmp_dir)
  local_file = os.path.join(tmp_dir, file_name)
  mox.file.copy(path_or_buf, local_file)
  result = read_hdf_origin(local_file, key, mode, **kwargs)
  mox.file.remove(local_file)
  return result

setattr(NDFrame, 'to_hdf', to_hdf_override)
setattr(pytables, 'read_hdf', read_hdf_override)
setattr(pd, 'read_hdf', read_hdf_override)
```

利用 MoXing 使 h5py.File 支持 OBS

```
import os
import h5py
import numpy as np
import moxing as mox

h5py_File_class = h5py.File

class OBSFile(h5py_File_class):
  def __init__(self, name, *args, **kwargs):
    self._tmp_name = None
    self._target_name = name
    if name.startswith('obs://'):
      self._tmp_name = name.replace('/', '_')
      if mox.file.exists(name):
        mox.file.copy(name, os.path.join('cache', 'h5py_tmp', self._tmp_name))
      name = self._tmp_name

    super(OBSFile, self).__init__(name, *args, **kwargs)

  def close(self):
    if self._tmp_name:
      mox.file.copy(self._tmp_name, self._target_name)

    super(OBSFile, self).close()


setattr(h5py, 'File', OBSFile)

arr = np.random.randn(1000)
with h5py.File('obs://bucket/random.hdf5', 'r') as f:
  f.create_dataset("default", data=arr)

with h5py.File('obs://bucket/random.hdf5', 'r') as f:
  print(f.require_dataset("default", dtype=np.float32, shape=(1000,)))
```

7 数据准备与处理

7.1 数据准备使用流程

ModelArts是面向AI开发者的一站式开发平台，能够支撑开发者从数据到模型的全流程开发过程，包含数据处理、算法开发、模型训练、模型部署等操作。并且提供AI Gallery功能，能够在市场内与其他开发者分享数据、算法、模型等。为了能帮用户快速准备大量高质量的数据，ModelArts数据管理提供了全流程的数据准备、数据处理和数据标注能力。

ModelArts数据管理为用户准备高质量的AI数据提供了以下主要能力：

- 解决用户获取数据的问题。
  - 提供多种数据接入方式，支持用户从OBS，MRS，DLI以及DWS等服务导入用户的数据。
  - 提供18+数据增强算子，帮助用户扩增数据，增加训练用的数据量。
- 帮助用户提高数据的质量。
  - 提供图像、文本、音频、视频等多种格式数据的预览，帮助用户识别数据质量。
  - 提供对数据进行多维筛选的能力，用户可以根据样本属性、标注信息等进行样本筛选。
  - 提供12+标注工具，方便用户进行精细化、场景化和专业化的数据标注。

  - 提供基于样本和标注结果进行特征分析，帮助用户整体了解数据的质量。
- 提升用户数据准备的效率。
  - 提供数据版本管理能力，帮助用户提升数据管理的效率。
  - 提供交互式标注、智能标注等能力，提升用户数据标注的效率。
  - 提供团队标注以及团队标注流程管理能力，帮助用户提升大批量数据标注的能力。

7.2 创建 ModelArts 数据集

在ModelArts进行数据准备，首先需要先创建一个数据集，后续的操作如数据导入、数据分析、数据标注等，都是基于数据集来进行的。

说明

数据集功能仅在以下Region支持：华北-北京四、西南-贵阳一、中国-香港、亚太-新加坡、亚太-曼谷、亚太-雅加达、非洲-约翰内斯堡、拉美-圣地亚哥、拉美-圣保罗一、拉美-墨西哥城二。

数据集的类型

当前ModelArts支持如下格式的数据集。

- 图片：对图像类数据进行处理，支持 .jpg、.png、.jpeg、.bmp四种图像格式，支持用户进行图像分类、物体检测、图像分割类型的标注。
- 音频：对音频类数据进行处理，支持.wav格式，支持用户进行声音分类、语音内容、语音分割三种类型的标注。
- 文本：对文本类数据进行处理，支持.txt、.csv格式，支持用户进行文本分类、命名实体、文本三元组三种类型的标注。
- 视频：对视频类数据进行处理，支持.mp4格式，支持用户进行视频标注。
- 自由格式：管理的数据可以为任意格式，目前不支持标注，适用于无需标注或开发者自行定义标注的场景。如果您的数据集需存在多种格式数据，或者您的数据格式不符合其他类型数据集时，可选择自由格式的数据集。
- 表格

  表格：适合表格等结构化数据处理。数据格式支持csv。不支持标注，支持对部分表格数据进行预览，但是最多支持100条数据预览。

不同类型数据集支持的功能列表

其中，不同类型的数据集支持不同的功能，如智能标注、团队标注等。详细信息参考**表7-1**。

表 7-1 不同类型的数据集支持的功能

数据集类型	标注类型	创建数据集	导入数据	导出数据	发布数据集	修改数据集	管理版本	智能标注	团队标注	自动分组	数据特征
图片	图像分类	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持
	物体检测	支持	支持	支持	支持	支持	支持	支持	支持	支持	支持
	图像分割	支持	支持	支持	支持	支持	支持	-	-	支持	-
音频	声音分类	支持	支持	-	支持	支持	支持	-	-	-	-
	语音内容	支持	支持	-	支持	支持	支持	-	-	-	-
	语音分割	支持	支持	-	支持	支持	支持	支持	-	-	-
文本	文本分类	支持	支持	-	支持	支持	支持	支持	-	-	-
	命名实体	支持	支持	-	支持	支持	支持	支持	-	-	-
	文本三元组	支持	支持	-	支持	支持	支持	支持	-	-	-
视频	视频	支持	支持	-	支持	支持	支持	-	-	-	-

数据集类型	标注类型	创建数据集	导入数据	导出数据	发布数据集	修改数据集	管理版本	智能标注	团队标注	自动分组	数据特征
自由格式	自由格式	支持	-	支持	支持	支持	-	-	-	-	-
表格	表格	支持	支持	-	支持	支持	支持	-	-	-	-

规格限制

- 除表格类型之外的数据集（如视频、文本、音频等），单个数据集的最大样本数量限制：1000000，最大标签数量限制：10000。
- 除图片类型之外的数据集（如视频、文本、音频等），单个样本大小限制：5GB。
- 针对图片类数据集（物体检测、图像分类、图像分割），单个图片大小限制：25MB。
- 单个manifest文件大小限制：5GB。
- 文本文件单行大小限制：100KB。
- 数据集标注结果文件大小限制：100MB。

前提条件

- 数据集功能需要获取访问OBS权限，在未进行委托授权之前，无法使用此功能。在使用数据集功能之前，请前往“权限管理”页面，使用委托完成访问授权。
- 已创建用于存储数据的OBS桶及文件夹。并且，数据存储的OBS桶与ModelArts在同一区域。当前不支持OBS并行文件系统，请选择OBS对象存储。
- ModelArts不支持加密的OBS桶，创建OBS桶时，请勿开启桶加密。

创建数据集（图片、音频、文本、视频、自由格式）

1. 登录**ModelArts管理控制台**，在左侧菜单栏中选择“资产管理> 数据集”，进入数据集管理页面。
2. 单击“创建数据集”，进入“创建数据集”页面，根据数据类型以及数据标注要求，选择创建不同类型的数据集。填写数据集基本信息。

图 7-1 参数填写

- 名称：数据集的名称，可自定义您的数据集。
- 描述：该数据集的详情信息。
- 数据类型：根据实际需求，选择对应的数据类型。
- 数据来源：

  i. OBS导入数据

  用户在OBS中有准备好的数据时，选择“OBS”、“导入路径”、“数据标注状态”和“标注格式”（当数据标注状态选择“已标注”时，需要填写该参数）和“数据集输出位置”。针对不同类型的数据集，数据输入支持的标注格式不同，ModelArts目前支持的标注格式及其说明请参见**不同类型数据集支持的功能列表**。

  ii. 从本地上传数据。

  ModelArts还支持从本地上传数据。本地上传时选择“上传数据存储路径”、“数据标注状态”和“数据集输出路径”。单击“文件上传”，上传您本地的数据。并选择“标注格式”（当数据标注状态为“已标注”时，需要关注该参数）。针对不同类型的数据集，数据输入支持的标注格式不同，ModelArts目前支持的标注格式及其说明请参见**不同类型数据集支持的功能列表**。

  图 7-2 数据来源选择本地上传

- 更多参数填写请参见**表7-2**。

表 7-2 数据集的详细参数

参数名称	说明
导入路径	选择需要导入数据的OBS路径，此位置会作为数据集的数据存储路径。 **说明** “导入路径”不支持OBS并行文件系统下的路径，请选择OBS对象桶。 创建数据集时，此OBS路径下的数据会导入数据集，后续如果直接在OBS中修改数据，会造成数据集的数据与OBS的数据不一致，可能导致部分数据不可用。如果需要在数据集中修改数据，建议使用**同步数据源**或4章节**从OBS目录导入数据到数据集**功能。 超出数据集的样本和标签配额，会导致数据无法正常导入。
数据标注状态	选择数据的标注状态，分为“未标注”和“已标注”。 选择“已标注”时，需指定标注格式，并保证数据文件满足相应的格式规范，否则可能存在导入失败的情况。 仅图片（物体检测、图像分类、图像分割）、音频（声音分类）、文本（文本分类）类型的标注任务支持导入已标注数据。
数据集输出位置	选择数据集输出位置的OBS路径，此位置会存放输出的标注信息等文件。 **说明** • 请确保您的OBS路径以字母、数字、下划线命名，不能包含特殊字符，例如：~'@#$%^&*{}[]:;+=<>/以及空格。 • “数据集输出位置”不能与“数据输入路径”为同一路径，且不能是“数据输入路径”的子目录。 • “数据集输出位置”建议选择一个空目录。 • “数据集输出位置”不支持OBS并行文件系统下的路径，请选择OBS对象桶。
高级特征选项-按标签导入	默认关闭，可通过勾选高级选项提供增强功能。 如“按标签导入”：系统将自动获取此数据集的标签，您可以单击“添加标签”添加相应的标签。此字段为可选字段，您也可以在导入数据集后，在标注数据操作时，添加或删除标签。

3. 参数填写完成，单击“提交”，即可完成数据集的创建。

创建数据集（表格）

1. 登录**ModelArts管理控制台**，在左侧菜单栏中选择“资产管理> 数据集”，进入数据集管理页面。
2. 单击“创建数据集”，进入“创建数据集”页面，根据数据类型以及数据标注要求，选择创建表格类型的数据集。填写数据集基本信息。

图 7-3 表格类型的参数

- 名称：数据集的名称，可自定义您的数据集。
- 描述：该数据集的详情信息。
- 数据类型：根据实际需求，选择对应的数据类型。
- 更多参数填写请参考**表7-3**。

表 7-3 数据集的详细参数

参数名称	说明
数据源（“OBS”）	• “文件路径”：单击输入框右侧按钮，可打开当前账号下的所有OBS桶，请选择需要导入的数据文件所在目录。 • “导入是否包含表头”：默认开启，表示导入文件包含表头。 – 如果您的原始表格中已包含表头，开启时，会将导入文件的第一行（表头）作为列名，无需再手动修改Schema信息。 – 如果您的原始表格中没有表头，需手动关闭该开关，并更改Schema信息中的“列名”为attr_1、attr_2、……、attr_n，其中attr_n为最后一列，代表预测列。 OBS的详细功能说明，请参见《**OBS用户指南**》。

参数名称	说明
数据源（“DWS”）	• “集群名称”：系统自动将当前账号下的DWS集群展现在列表中，您可以在下拉框中选择您所需的DWS集群。 • “数据库名称”：根据选择的DWS集群，填写数据所在的数据库名称。 • “表名称”：根据选择的数据库，填写数据所在的表。 • “用户名”：输入DWS集群管理员用户的用户名。 • “密码”：输入DWS集群管理员用户的密码。 DWS的详细功能说明，请参见《**DWS用户指南**》。 **说明** 从DWS导入数据，需要借助DLI的功能，如果用户没有访问DLI服务的权限，需根据页面提示创建DLI的委托。
数据源（“DLI”）	• “队列名称”：系统自动将当前账号下的DLI队列展现在列表中，您可以在下拉框中选择您所需的队列。 • “数据库名称”：根据选择的队列展现所有的数据库，请在下拉框中选择您所需的数据库。 • “表名称”：根据选择的数据库展现此数据库中的所有表。请在下拉框中选择您所需的表。 DLI的详细功能说明，请参见《**DLI用户指南**》。
数据源（“MRS”）	• “集群名称”：系统自动将当前账号下的MRS集群展现在此列表中，但是流式集群不支持导入操作。请在下拉框中选择您所需的集群。 • “文件路径”：根据选择的集群，输入对应的文件路径，此文件路径为HDFS路径。 • “导入是否包含表头”：开启表示导入时将表头同时导入。 MRS的详细功能说明，请参见《**MRS用户指南**》。
本地上传	上传数据存储路径：选择对应的数据存储的OBS路径。
Schema信息	表格的列名和对应类型，需要跟导入数据的列数保持一致。请根据您导入的数据输入“列名”，同时选择此列的“类型”。其中支持的类型见**表7-4**。 单击“添加Schema信息”，即可增加一行列。创建数据集时必须指定schema，且一旦创建不支持修改。 从OBS数据源导入数据，会自动获取文件路径下csv文件的schema，如果多个csv文件的schema不一致会报错。 **说明** 从OBS选择数据后，Schema信息的列名会自动带出，且默认为表格中的第一行数据。为确保预测代码的正确性，请您手动更改Schema信息中的“列名”为attr_1、attr_2、……、attr_n，其中attr_n为最后一列，代表预测列。

参数名称	说明
数据集输出位置	选择表格数据存储路径（OBS路径），此位置会存放由数据源导入的数据。此位置不能和OBS数据源中的文件路径相同或为其子目录。 创建表格数据集后，在存储路径下会自动生成以下4个目录。 ● annotation：版本发布目录，每次发布版本，会在此目录下生成和版本名称相同的子目录。 ● data：数据存放目录，导入的数据会放在此目录。 ● logs：日志存放目录。 ● temp：临时工作目录。

表 7-4 Schema 数据类型说明

类型	描述	存储空间	范围
String	字符串	-	-
Short	有符号整数	2字节	-32768-32767
Int	有符号整数	4字节	-2147483648～2147483647
Long	有符号整数	8字节	-9223372036854775808～9223372036854775807
Double	双精度浮点型	8字节	-
Float	单精度浮点型	4字节	-
Byte	有符号整数	1字节	-128-127
Date	日期类型，描述了特定的年月日，格式：yyyy-MM-dd，例如2014-05-29	-	-
Timestamp	时间戳，表示日期和时间。格式：yyyy-MM-dd HH:mm:ss	-	-
Boolean	布尔类型	1字节	TRUE/FALSE

说明

使用CSV文件时，需要注意以下两点：

- 当数据类型选择String时，默认会把双引号内的数据当作一条，所以同一行数据需要保证双引号闭环，否则会导致数据过大，无法显示。
- 当CSV文件的某一行的列数与定义的Schema不同，则会忽略当前行。

3. 参数填写完成后，单击“提交”，即可完成数据集的创建。

修改数据集基本信息

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理>数据集”，进入“数据集”管理页面。
2. 在数据集列表中，单击操作列的“更多>修改”。修改数据集基本信息，然后单击“确定”完成修改。

表 7-5 参数说明

参数	说明
名称	数据集的名称，支持1~64位可见字符。名称只能是字母、中文、数字、下划线或者中划线组成的合法字符串。且只能以字母或者中文字符开头。
描述	数据集的简要描述。

7.3 导入数据到 ModelArts 数据集

7.3.1 数据导入方式介绍

数据集创建完成后，您还可以通过导入数据的操作，接入更多数据。ModelArts支持从不同数据源导入数据。

- 从OBS导入数据到ModelArts数据集
- 从DLI导入数据到ModelArts数据集
- 从MRS导入数据到ModelArts数据集
- 从DWS导入数据到ModelArts数据集
- 从本地上传数据到ModelArts数据集

文件型数据来源

文件型数据集支持从两种数据源导入数据：“OBS”和“本地上传”。导入后，导入目录下的数据会复制至数据集的数据源路径下。

- OBS：又分为从OBS目录或从Manifest文件两种导入方式，需要将导入的数据或Manifest文件提前存储至OBS目录中。
- 本地上传：将本地数据直接通过Internet上传至OBS指定目录后，再导入数据集。

表格型数据来源

表格数据集支持从5种数据源导入数据，分别为对象存储服务（OBS）、数据仓库服务（DWS）、数据湖探索服务（DLI）、MapReduce服务（MRS）和本地上传。

数据集中的数据导入入口

数据集中的数据导入有5个入口。

- 创建数据集时直接从设置的数据导入路径中自动同步数据。
- 创建完数据集后，在数据集列表页面的操作栏单击“导入”，导入数据。

  图 7-4 在数据集列表页导入数据

- 在数据集列表页面，单击某个数据集的名称，进入数据集详情页中，单击“导入>导入”，导入数据。

  图 7-5 在数据集详情页中导入数据

- 在数据集列表页面，单击某个数据集的名称，进入数据集详情页中，单击“同步数据源”，同步OBS中的数据。

  图 7-6 在数据集详情页中同步数据源

- 在数据标注的标注作业详情中添加数据。

图 7-7 标注作业详情中添加数据

7.3.2 从 OBS 导入数据到 ModelArts 数据集

7.3.2.1 从 OBS 导入数据到数据集场景介绍

导入方式

OBS导入数据方式分为“OBS目录”和“Manifest文件”两种。

- OBS目录：指需要导入的数据集已提前存储至OBS目录中。此时需选择用户具备权限的OBS路径，且OBS路径内的目录结构需满足规范，详细规范请参见**从OBS目录导入数据规范说明**。当前只有“图像分类”、“物体检测”、“表格”、“文本分类”和“声音分类”类型的数据集，支持从OBS目录导入数据。其他类型只支持Manifest文件导入数据集的方式。
- Manifest文件：指数据集为Manifest文件格式，Manifest文件定义标注对象和标注内容的对应关系，且Manifest文件已上传至OBS中。Manifest文件的规范请参见**从Manifest文件导入规范说明**。

说明

导入“物体检测”类型数据集前，您需要保证标注文件的标注范围不超过原始图片大小，否则可能存在导入失败的情况。

表 7-6 不同类型数据集支持的导入方式

数据集类型	标注类型	OBS目录导入	Manifest文件导入
图片	图像分类	支持 可以导入未标注或已标注数据 已标注数据格式规范：**图像分类**	支持 可以导入未标注或已标注数据 已标注数据格式规范：**图像分类**

数据集类型	标注类型	OBS目录导入	Manifest文件导入
	物体检测	支持 可以导入未标注或已标注数据 已标注数据格式规范：**物体检测**	支持 可以导入未标注或已标注数据 已标注数据格式规范：**物体检测**
	图像分割	支持 可以导入未标注或已标注数据 已标注数据格式规范：**图像分割**	支持 可以导入未标注或已标注数据 已标注数据格式规范：**图像分割**
音频	声音分类	支持 导入的是未标注或已标注数据 格式规范：**声音分类**	支持 可以导入未标注或已标注数据 已标注数据格式规范：**声音分类**
	语音内容	支持 导入的是未标注数据	支持 可以导入未标注或已标注数据 已标注数据格式规范：**语音内容**
	语音分割	支持 导入的是未标注数据	支持 可以导入未标注或已标注数据 已标注数据格式规范：**语音分割**
文本	文本分类	支持 导入的是未标注或已标注数据 已标注数据格式规范：**文本分类**	支持 可以导入未标注或已标注数据 已标注数据格式规范：**文本分类**
	命名实体	支持 导入的是未标注数据	支持 可以导入未标注或已标注数据 已标注数据格式规范：**文本命名实体**
	文本三元组	支持 导入的是未标注数据	支持 可以导入未标注或已标注数据 已标注数据格式规范：**文本三元组**
视频	视频	支持 导入的是未标注数据	支持 可以导入未标注或已标注数据 已标注数据格式规范：**视频标注**

数据集类型	标注类型	OBS目录导入	Manifest文件导入
其他	自由格式	支持 导入的是未标注数据	-
表格	表格	支持 还支持从DWS、DLI、MRS导入数据。 格式规范：**表格**	-

7.3.2.2 从 OBS 目录导入数据到数据集

前提条件

- 已存在创建完成的数据集。
- 准备需要导入的数据，具体可参见**从OBS目录导入数据规范说明**。
- 需导入的数据，已存储至OBS中。Manifest文件也需要存储至OBS。详细指导请参见**创建OBS桶用于ModelArts存储数据**。
- 确保数据存储的OBS桶与ModelArts在同一区域，并确保用户具有OBS桶的操作权限。

文件型数据从 OBS 目录导入操作

不同类型的数据集，导入操作界面的示意图存在区别，请参考界面信息了解当前类型数据集的示意图。当前操作指导以图像分类的数据集为例。

1. 登录**ModelArts管理控制台**，在左侧菜单栏中选择“资产管理 >数据集”，进入“数据集”管理页面。
2. 在数据集所在行，单击操作列的“导入”。或者，您可以单击数据集名称，进入数据集“概览”页，在页面右上角单击“导入”。
3. 在“导入”对话框中，参考如下说明填写参数，然后单击“确定”。
   - “数据来源”：“OBS”
   - “导入方式”：“目录”。
   - “导入路径”：数据存储的OBS路径。
   - “数据标注状态”：已标注。
   - “高级特征选项 ”：默认关闭，可通过勾选高级选项提供增强功能。

     如“按标签导入”：系统将自动获取此数据集的标签，您可以单击“添加标签”添加相应的标签。此字段为可选字段，您也可以在导入数据集后，在标注数据操作时，添加或删除标签。

图 7-8 导入数据集-OBS

导入成功后，数据将自动同步到数据集中。您可以在“数据集”页面，单击数据集的名称，查看详细数据，并可以通过创建标注任务进行数据标注。

文件型数据标注状态

数据标注状态分为“未标注”和“已标注”。

- 未标注：仅导入标注对象（指待标注的图片，文本等），不导入标注内容（指标注结果信息）。
- 已标注：同时导入标注对象和标注内容，当前“自由格式”的数据集不支持导入标注内容。

  为了确保能够正确读取标注内容，要求用户严格按照规范存放数据：

  导入方式选择目录时，需要用户选择“标注格式”，并按照标注格式的要求存放数据，详细规范请参见**标注格式**章节。

  导入方式选择manifest时，需要满足manifest文件的规范。

说明

- 数据标注状态选择“已标注”，您需要保证目录或manifest文件满足相应的格式规范，否则可能存在导入失败的情况。
- 导入已标注的文件，导入完成后，请检查您导入的数据是否为已标注状态。

表格数据集从 OBS 导入操作

ModelArts支持从OBS导入表格数据，即csv文件。

表格数据集导入说明：

- 导入成功的前提是，数据源的schema需要与创建数据集指定的schema保持一致。其中schema指表格的列名和类型，创建数据集时一旦指定，不支持修改。
- 从OBS导入csv文件，不会校验数据类型，但是列数需要跟数据集的schema保持一致。如果数据格式不合法，会将数据置为null，详见**表7-4**。
- 导入的csv文件要求如下：需要选择文件所在目录，其中csv文件的列数需要跟数据集schema一致。支持自动获取csv文件的schema。

```
├─dataset-import-example
│      table_import_1.csv
│      table_import_2.csv
│      table_import_3.csv
│      table_import_4.csv
```

7.3.2.3 从 Manifest 文件导入数据到数据集

前提条件

- 已存在创建完成的数据集。
- 准备需要导入的数据，具体可参见**从Manifest文件导入规范说明**。
- 需导入的数据，已存储至OBS中。Manifest文件也需要存储至OBS。
- 确保数据存储的OBS桶与ModelArts在同一区域，并确保用户具有OBS桶的操作权限。

文件型数据从 Manifest 导入操作

不同类型的数据集，导入操作界面的示意图存在区别，请参考界面信息了解当前类型数据集的示意图。当前操作指导以图片数据集为例。

1. 登录**ModelArts管理控制台**，在左侧菜单栏中选择“资产管理>数据集”，进入“数据集”管理页面。
2. 在数据集所在行，单击操作列的“导入”。或者，您可以单击数据集名称，进入数据集“概览”页，在页面右上角单击“导入”。
3. 在“导入”对话框中，参考如下说明填写参数，然后单击“确定”。
   - “数据来源”：“OBS”
   - “导入方式”：“manifest”。
   - “Manifest文件”：存储Manifest文件的OBS路径。
   - “数据标注状态”：已标注。
   - “高级特征选项 ”：默认关闭，可通过勾选高级选项提供增强功能。

“按标签导入”：系统将自动获取此数据集的标签，您可以单击“添加标签”添加。此字段为可选字段，您也可以在导入数据集后，在标注数据操作时，添加或删除标签。

“只导入难例”：难例指manifest文件中的“hard”属性，勾选此参数，表示此导入操作，只导入manifest文件“hard”属性中数据信息。

图 7-9 导入 manifest 文件

导入成功后，数据将自动同步到数据集中。您可以在“数据集”页面，单击数据集的名称，查看详细数据，并可以通过创建标注任务进行数据标注。

文件型数据标注状态

数据标注状态分为“未标注”和“已标注”。

- 未标注：仅导入标注对象（指待标注的图片，文本等），不导入标注内容（指标注结果信息）。
- 已标注：同时导入标注对象和标注内容，当前“自由格式”的数据集不支持导入标注内容。

  为了确保能够正确读取标注内容，要求用户严格按照规范存放数据：

  导入方式选择目录时，需要用户选择“标注格式”，并按照标注格式的要求存放数据。

  导入方式选择manifest时，需要满足manifest文件的规范，详细规范请参见**标注格式**章节。

  说明

  数据标注状态选择“已标注”，您需要保证目录或manifest文件满足相应的格式规范，否则可能存在导入失败的情况。

7.3.2.4 从 OBS 目录导入数据规范说明

导入数据集时，使用存储在OBS的数据时，数据的存储目录以及文件名称需满足ModelArts的规范要求。

当前只有“图像分类”、“物体检测”、“图像分割”、“文本分类”和“声音分类”标注类型支持按标注格式导入。

其中，“表格”类型的数据集，支持从OBS、DWS、DLI和MRS等数据源导入数据。

说明

- 从OBS目录导入数据时，当前操作用户需具备此OBS路径的读取权限。
- 同时确保数据存储的OBS桶与ModelArts在同一区域。

图像分类

图像分类的数据支持两种格式：

- ModelArts imageNet 1.0：目录方式，只支持单标签

  相同标签的图片放在一个目录里，并且目录名字即为标签名。当存在多层目录时，则以最后一层目录为标签名。

  示例如下所示，其中Cat和Dog分别为标签名。

  ```
  dataset-import-example
  ├─Cat
  │      10.jpg
  │      11.jpg
  │      12.jpg
  │
  │
  └─Dog
         1.jpg
         2.jpg
         3.jpg
  ```

- ModelArts image classification 1.0：txt标签文件，支持多标签

  当目录下存在对应的txt文件时，以txt文件内容作为图像的标签。

  示例如下所示，import-dir-1和imort-dir-2为导入子目录。

  ```
  dataset-import-example
  ├─import-dir-1
  │      10.jpg
  │      10.txt
  │      11.jpg
  │      11.txt
  │      12.jpg
  │      12.txt
  └─import-dir-2
         1.jpg
         1.txt
         2.jpg
         2.txt
  ```

  - 单标签的标签文件示例，如1.txt文件内容如下所示：

    ```
    Cat
    ```

  - 多标签的标签文件示例，如2.txt文件内容如下所示：

    ```
    Cat
    Dog
    ```

  只支持JPG、JPEG、PNG、BMP格式的图片。单张图片大小不能超过5MB，且单次上传的图片总大小不能超过8MB。

物体检测

支持两种格式：

- ModelArts PASCAL VOC 1.0

  物体检测的简易模式要求用户将标注对象和标注文件存储在同一目录，并且一一对应，如标注对象文件名为“IMG_20180919_114745.jpg”，那么标注文件的文件名应为“IMG_20180919_114745.xml”。

  物体检测的标注文件需要满足PASCAL VOC格式，格式详细说明请参见**表7-14**。

  示例：

  ```
  ├─dataset-import-example
  │      IMG_20180919_114732.jpg
  │      IMG_20180919_114732.xml
  │      IMG_20180919_114745.jpg
  │      IMG_20180919_114745.xml
  │      IMG_20180919_114945.jpg
  │      IMG_20180919_114945.xml
  ```

  标注文件的示例如下：

  ```
  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <annotation>
     <folder>NA</folder>
     <filename>bike_1_1593531469339.png</filename>
     <source>
        <database>Unknown</database>
     </source>
     <size>
        <width>554</width>
        <height>606</height>
        <depth>3</depth>
     </size>
     <segmented>0</segmented>
     <object>
        <name>Dog</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
           <xmin>279</xmin>
           <ymin>52</ymin>
           <xmax>474</xmax>
           <ymax>278</ymax>
        </bndbox>
     </object>
     <object>
        <name>Cat</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <occluded>0</occluded>
        <bndbox>
           <xmin>279</xmin>
           <ymin>198</ymin>
           <xmax>456</xmax>
           <ymax>421</ymax>
        </bndbox>
     </object>
  </annotation>
  ```

  只支持JPG、JPEG、PNG、BMP格式的图片，单张图片大小不能超过5MB，且单次上传的图片总大小不能超过8MB。

- YOLO：

  YOLO数据集目录应该遵循以下结构，格式不规范将导致导入任务失败。

```
└─ yolo_dataset/
    │
    ├── obj.names  # 标签集文件
    ├── obj.data   # 记录数据集信息的文件及路径信息(相对路径)
    ├── train.txt  # 训练集中各图片路径信息(相对路径)
    ├── valid.txt  # 验证集中各图片路径信息(相对路径)
    │
    ├── obj_train_data/  # 训练集的图片与对应的标注文件所在目录
    │   ├── image1.txt  # image1对应的带标签bbox列表
    │   ├── image1.jpg
    │   ├── image2.txt
    │   ├── image2.jpg
    │   ├── ...
    │
    ├── obj_valid_data/  # 验证集的图片与对应的标注文件所在目录
    │   ├── image101.txt
    │   ├── image101.jpg
    │   ├── image102.txt
    │   ├── image102.jpg
    │   ├── ...
```

YOLO数据集只支持train和valid子集。如果导入的数据集包括除了上述之外的子集，这些其他子集将被忽略。

- obj.data应包含以下内容，train和valid子集必须至少有一个，其中文件路径均为相对路径。

  ```
  classes = 5 # 可选
  names = <path/to/obj.names>#例如 obj.names
  train = <path/to/train.txt>#例如 train.txt
  valid = <path/to/valid.txt>#可选， 例如valid.txt
  backup = backup/ # 可选
  ```

- obj.names文件记录标签列表。每一行的行标作为该标签的index。

  ```
  label1  # label1的index： 0
  label2  # label2的index： 1
  label3
  ...
  ```

- train.txt和valid.txt文件结构如下，其中文件路径均为相对路径，且文件列表与目录下的文件需一一对应：

  ```
  <path/to/image1.jpg>#例如 obj_train_data/image.jpg
  <path/to/image2.jpg>#例如 obj_train_data/image.jpg
  ...
  ```

- obj_train_data/ 和obj_valid_data/目录下的.txt文件应该包含对应图像的bbox标签信息，每行代表一个bbox标注。

  ```
  # image1.txt:
  # <label_index> <x_center> <y_center> <width> <height>
  0 0.250000 0.400000 0.300000 0.400000
  3 0.600000 0.400000 0.400000 0.266667
  ```

  其中x_center、y_center、width和height分别表示归一化后的目标框中心点x坐标、归一化后的目标框中心点y坐标、归一化后的目标框宽度、归一化后的目标框高度。

只支持JPG、JPEG、PNG、BMP格式的图片，单张图片大小不能超过5MB，且单次上传的图片总大小不能超过8MB。

图像分割

ModelArts image segmentation 1.0：

- 要求用户将标注对象和标注文件存储在同一目录，并且一一对应，如标注对象文件名为“IMG_20180919_114746.jpg”，那么标注文件的文件名应为“IMG_20180919_114746.xml”。

  图像分割的标注文件基于PASCAL VOC格式增加了字段mask_source和mask_color，格式详细说明请参见**表7-10**。

示例：

```
├─dataset-import-example
│      IMG_20180919_114732.jpg
│      IMG_20180919_114732.xml
│      IMG_20180919_114745.jpg
│      IMG_20180919_114745.xml
│      IMG_20180919_114945.jpg
│      IMG_20180919_114945.xml
```

标注文件的示例如下：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<annotation>
   <folder>NA</folder>
   <filename>image_0006.jpg</filename>
   <source>
      <database>Unknown</database>
   </source>
   <size>
      <width>230</width>
      <height>300</height>
      <depth>3</depth>
   </size>
   <segmented>1</segmented>
   <mask_source>obs://xianao/out/dataset-8153-Jmf5ylLjRmSacj9KevS/annotation/V001/
segmentationClassRaw/image_0006.png</mask_source>
   <object>
      <name>bike</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
      <mask_color>193,243,53</mask_color>
      <occluded>0</occluded>
      <polygon>
         <x1>71</x1>
         <y1>48</y1>
         <x2>75</x2>
         <y2>73</y2>
         <x3>49</x3>
         <y3>69</y3>
         <x4>68</x4>
         <y4>92</y4>
         <x5>90</x5>
         <y5>101</y5>
         <x6>45</x6>
         <y6>110</y6>
         <x7>71</x7>
         <y7>48</y7>
      </polygon>
   </object>
</annotation>
```

文本分类

文本分类支持导入“txt”和“csv”两种文件类型，文本的编码格式支持“UTF-8”和“GBK”。

文本分类的标注对象和标注文件有2种存放模式。

- ModelArts text classfication combine 1.0：文本和标注合并，文本分类的标注对象和标注内容在一个文本文件内，标注对象与标注内容之间，多个标注内容之间可分别指定分隔符。

  例如，文本文件的内容如下所示。标注对象与标注内容之间采用tab键分隔。

  ```
  手感很好，反应速度很快，不知道以后怎样	positive
  三个月前买了一个用的非常好果断把旧手机替换下来尤其在待机方面表现得尤为明显	positive
  没充一会电源怎么也会发热呢音量健不好用回弹不好	negative
  ```



```
算是给自己的父亲节礼物吧物流很快下单不到24小时就到货了耳机更赞有些低音炮的感觉入耳很紧不会掉棒棒哒  positive
```

- ModelArts text classfication 1.0：文本和标注分离，文本分类的标注对象和标注文件均为文本文件，并且以行数进行对应，如标注文件中的第一行表示的是标注对象文件中的第一行的标注。

  例如，标注对象“COMMENTS_20180919_114745.txt”的内容如下所示。

  ```
  手感很好，反应速度很快，不知道以后怎样
  三个月前买了一个用的非常好果断把旧手机替换下来尤其在待机方面性能好
  没充一会电源怎么也会发热呢音量健不好用回弹不好
  算是给自己的父亲节礼物吧物流很快下单不到24小时就到货了耳机更赞有些低音炮的感觉入耳很紧不会掉棒棒哒
  ```

  标注文件“COMMENTS_20180919_114745_result.txt”的内容。

  ```
  positive
  negative
  negative
  positive
  ```

  此数据格式要求将标注对象和标注文件存储在同一目录，并且一一对应，如标注对象文件名为“COMMENTS_20180919_114745.txt”，那么标注文件名为“COMMENTS _20180919_114745_result.txt”。

  数据文件存储示例：

  ```
  ├─dataset-import-example
  │      COMMENTS_20180919_114732.txt
  │      COMMENTS _20180919_114732_result.txt
  │      COMMENTS _20180919_114745.txt
  │      COMMENTS _20180919_114745_result.txt
  │      COMMENTS _20180919_114945.txt
  │      COMMENTS _20180919_114945_result.txt
  ```

声音分类

ModelArts audio classfication dir 1.0：要求用户将相同标签的声音文件放在一个目录里，并且目录名字即为标签名。

示例：

```
dataset-import-example
├─Cat
│      10.wav
│      11.wav
│      12.wav
│
└─Dog
       1.wav
       2.wav
       3.wav
```

表格

支持从OBS导入csv文件，需要选择文件所在目录，其中csv文件的列数需要跟数据集schema一致。支持自动获取csv文件的schema。

```
├─dataset-import-example
│      table_import_1.csv
│      table_import_2.csv
│      table_import_3.csv
│      table_import_4.csv
```


7.3.2.5 从 Manifest 文件导入规范说明

Manifest文件中定义了标注对象和标注内容的对应关系。此导入方式是指导入数据集时，使用Manifest文件。选择导入Manifest文件时，可以从OBS导入。当从OBS导入Manifest文件时，需确保当前用户具备Manifest文件所在OBS路径的权限。

说明

Manifest文件编写规范要求较多，推荐使用OBS目录导入方式导入新数据。一般此功能常用于不同区域或不同账号下ModelArts的数据迁移，即当您已在某一区域使用ModelArts完成数据标注，发布后的数据集可从输出路径下获得其对应的Manifest文件。在获取此Manifest文件后，可将此数据集导入其他区域或者其他账号的ModelArts中，导入后的数据已携带标注信息，无需重复标注，提升开发效率。

Manifest文件描述的是原始文件和标注信息，可用于标注、训练、推理场景。Manifest文件中也可以只有原始文件信息，没有标注信息，如用于推理场景，或用于生成未标注的数据集。Manifest文件需满足如下要求：

- Manifest文件使用UTF-8编码。
- Manifest文件使用json lines格式（jsonlines.org），一行一个json对象。

  ```
  {"source": "/path/to/image1.jpg", "annotation": … }
  {"source": "/path/to/image2.jpg", "annotation": … }
  {"source": "/path/to/image3.jpg", "annotation": … }
  ```

  为了说明方便，下面的Manifest例子格式化为多行的json对象。
- Manifest文件可以由用户、第三方工具或ModelArts数据标注生成，其文件名没有特殊要求，可以为任意合法文件名。为了ModelArts系统内部使用方便，ModelArts数据标注功能生成的文件名由如下字符串组成：“DatasetName-VersionName.manifest”。例如，“animal-v20190123113030412 3.manifest”。

图像分类

```
{
  "source":"s3://path/to/image1.jpg",
  "usage":"TRAIN",
  "hard":"true",
  "hard-coefficient":0.8,
  "id":"0162005993f8065ef47eefb59d1e4970",
  "annotation": [
    {
      "type": "modelarts/image_classification",
      "name": "cat",
      "property": {
        "color":"white",
        "kind":"Persian cat"
      },
      "hard":"true",
      "hard-coefficient":0.8,
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    },
    {
      "type": "modelarts/image_classification",
      "name":"animal",
      "annotated-by":"modelarts/active-learning",
      "confidence": 0.8,
      "creation-time":"2019-01-23 11:30:30"
    }],
  "inference-loc":"/path/to/inference-output"
}
```


表 7-7 字段说明

字段	是否必选	说明
source	是	被标注对象的URI。数据来源的类型及示例请参考**表7-8**。
usage	否	默认为空，取值范围： ● TRAIN：指明该对象用于训练。 ● EVAL：指明该对象用于评估。 ● TEST：指明该对象用于测试。 ● INFERENCE：指明该对象用于推理。 如果没有给出该字段，则使用者自行决定如何使用该对象。
id	否	此参数为系统导出的样本id，导入时可以不用填写。
annotation	否	如果不设置，则表示未标注对象。annotation值为一个对象列表，详细参数请参见**表7-9**。
inference-loc	否	当此文件由推理服务生成时会有该字段，表示推理输出的结果文件位置。

表 7-8 数据来源类型

类型	示例
OBS	“source”：“s3://path-to-jpg”
Content	“source”：“content://I love machine learning”

表 7-9 annotation 对象说明

字段	是否必选	说明
type	是	标签类型。取值范围为： ● image_classification：图像分类 ● text_classification：文本分类 ● text_entity：文本命名实体 ● object_detection：对象检测 ● audio_classification：声音分类 ● audio_content：声音内容 ● audio_segmentation：声音起止点
name	是/否	对于分类是必选字段，对于其他类型为可选字段，本示例为图片分类名称。

字段	是否必选	说明
id	是/否	标签ID。对于三元组是必选字段，对于其他类型为可选字段。三元组的实体标签ID格式为“E+数字”，比如“E1”、“E2”，三元组的关系标签ID格式为“R+数字”，例如“R1”、“R2”。
property	否	包含对标注的属性，例如本示例中Cat有两个属性，颜色（color）和品种（kind）。
hard	否	表示是否是难例。“True”表示该标注是难例，“False”表示该标注不是难例。
annotated-by	否	默认为“human”，表示人工标注。 ● human
creation-time	否	创建该标注的时间。是用户写入标注的时间，不是Manifest生成时间。
confidence	否	表示机器标注的置信度。范围为0～1。

图像分割

```
{
  "annotation": [{
    "annotation-format": "PASCAL VOC",
    "type": "modelarts/image_segmentation",
    "annotation-loc": "s3://path/to/annotation/image1.xml",
    "creation-time": "2020-12-16 21:36:27",
    "annotated-by": "human"
  }],
  "usage": "train",
  "source": "s3://path/to/image1.jpg",
  "id": "16d196c19bf61994d7deccafa435398c",
  "sample-type": 0
}
```

- “source”、“usage”、“annotation”等参数说明与图像分类一致，详细说明请参见表7-7。
- “annotation-loc”：对于图像分割、物体检测是必选字段，对于其他类型是可选字段，标注文件的存储路径。
- “annotation-format”：描述标注文件的格式，可选字段，默认为“PASCAL VOC”。目前只支持“PASCAL VOC”。
- “**sample-type**”：样本格式，0表示图片，1表示文本，2表示语音，4表示表格，6表示视频。

表 7-10 PASCAL VOC 格式说明

字段	是否必选	说明
folder	是	表示数据源所在目录。
filename	是	被标注文件的文件名。

字段	是否必选	说明
size	是	表示图像的像素信息。 • width：必选字段，图片的宽度。 • height：必选字段，图片的高度。 • depth：必选字段，图片的通道数。
segmented	是	表示是否用于分割。
mask_source	否	表示图像分割保存的mask路径。
object	是	表示物体检测信息，多个物体标注会有多个object体。 • name：必选字段，标注内容的类别。 • pose：必选字段，标注内容的拍摄角度。 • truncated：必选字段，标注内容是否被截断（0表示完整）。 • occluded：必选字段，标注内容是否被遮挡（0表示未遮挡）。 • difficult：必选字段，标注目标是否难以识别（0表示容易识别）。 • confidence：可选字段，标注目标的置信度，取值范围0-1之间。 • bndbox：必选字段，标注框的类型，可选值请参见**表7-11**。 • mask_color：必选字段，标签的颜色，以RGB值表示。

表 7-11 标注框类型描述

type	形状	标注信息
polygon	多边形	各点坐标。 <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>100<y2> <x3>250<x3> <y3>150<y3> <x4>200<x4> <y4>200<y4> <x5>100<x5> <y5>200<y5> <x6>50<x6> <y6>150<y6> <x7>100<x7> <y7>100<y7>

示例：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<annotation>
    <folder>NA</folder>
    <filename>image_0006.jpg</filename>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>230</width>
        <height>300</height>
        <depth>3</depth>
    </size>
    <segmented>1</segmented>
    <mask_source>obs://xianao/out/dataset-8153-Jmf5ylLjRmSacj9KevS/annotation/V001/
segmentationClassRaw/image_0006.png</mask_source>
    <object>
        <name>bike</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <mask_color>193,243,53</mask_color>
        <occluded>0</occluded>
        <polygon>
            <x1>71</x1>
            <y1>48</y1>
            <x2>75</x2>
            <y2>73</y2>
            <x3>49</x3>
            <y3>69</y3>
            <x4>68</x4>
            <y4>92</y4>
            <x5>90</x5>
            <y5>101</y5>
            <x6>45</x6>
            <y6>110</y6>
```



```
            <x7>71</x7>
            <y7>48</y7>
        </polygon>
    </object>
</annotation>
```

文本分类

```
{
  "source": "content://I like this product ",
  "id":"XGDVGS",
  "annotation": [
    {
      "type": "modelarts/text_classification",
      "name": " positive",
      "annotated-by": "human",
      "creation-time": "2019-01-23 11:30:30"
    } ]
}
```

content字段是指被标注的文本（UTF-8编码，可以是中文），其他参数解释与**图像分类**相同，请参见**表7-7**。

文本命名实体

```
{
  "source":"content://Michael Jordan is the most famous basketball player in the world.",
  "usage":"TRAIN",
  "annotation":[
    {
      "type":"modelarts/text_entity",
      "name":"Person",
      "property":{
        "@modelarts:start_index":0,
        "@modelarts:end_index":14
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    },
    {
      "type":"modelarts/text_entity",
      "name":"Category",
      "property":{
        "@modelarts:start_index":34,
        "@modelarts:end_index":44
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    }
  ]
}
```

“source”、“usage”、“annotation”等参数说明与**图像分类**一致，详细说明请参见**表7-7**。

其中，property的参数解释如**表7-12**所示。例如，当“"source":"content://Michael Jordan"”时，如果要提取“Michael”，则对应的“start_index”为“0”，“end_index”为“7”。

表 7-12 property 参数说明

参数名	数据类型	说明
@modelarts:start_index	Integer	文本的起始位置，值从0开始，包括start_index所指的字符。
@modelarts:end_index	Integer	文本的结束位置，但不包括end_index所指的字符。

文本三元组

```
{
  "source":"content://"Three Body" is a series of long science fiction novels created by Liu Cix.",
  "usage":"TRAIN",
  "annotation":[
    {
      "type":"modelarts/text_entity",
      "name":"Person",
      "id":"E1",
      "property":{
        "@modelarts:start_index":67,
        "@modelarts:end_index":74
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    },
    {
      "type":"modelarts/text_entity",
      "name":"Book",
      "id":"E2",
      "property":{
        "@modelarts:start_index":0,
        "@modelarts:end_index":12
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    },
    {
      "type":"modelarts/text_triplet",
      "name":"Author",
      "id":"R1",
      "property":{
        "@modelarts:from":"E1",
        "@modelarts:to":"E2"
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    },
    {
      "type":"modelarts/text_triplet",
      "name":"Works",
      "id":"R2",
      "property":{
        "@modelarts:from":"E2",
        "@modelarts:to":"E1"
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    }
  ]
}
```

“source”、“usage”、“annotation”等参数说明与**图像分类**一致，详细说明请参见**表7-7**。

其中，property的参数解释如表5 property参数说明所示。其中，“@modelarts:start_index”和“@modelarts:end_index”和文本命名实体的参数说明一致。例如，当“source”：“content://"Three Body" is a series of long science fiction novels created by Liu Cix."时，“Liu Cix”是实体Person（人物），“Three Body”是实体Book（书籍），Person指向Book的关系是Author（作者），Book指向Person的关系是Works（作品）。

表 7-13 property 参数说明

参数名	数据类型	说明
@modelarts:start_index	Integer	三元组实体的起始位置，值从0开始，包括start_index所指的字符。
@modelarts:end_index	Integer	三元组实体的结束位置，但不包括end_index所指的字符。
@modelarts:from	String	三元组关系的起始实体id。
@modelarts:to	String	三元组关系的指向实体id。

物体检测

```
{
   "source":"s3://path/to/image1.jpg",
   "usage":"TRAIN",
   "hard":"true",
   "hard-coefficient":0.8,
   "annotation": [
      {
         "type":"modelarts/object_detection",
         "annotation-loc": "s3://path/to/annotation1.xml",
         "annotation-format":"PASCAL VOC",
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      }]
}
```

- “source”、“usage”、“annotation”等参数说明与图像分类一致，详细说明请参见表7-7。
- “annotation-loc”：对于物体检测、图像分割是必选字段，对于其他类型是可选字段，标注文件的存储路径。
- “annotation-format”：描述标注文件的格式，可选字段，默认为“PASCAL VOC”。目前只支持“PASCAL VOC”。

表 7-14 PASCAL VOC 格式说明

字段	是否必选	说明
folder	是	表示数据源所在目录。
filename	是	被标注文件的文件名。

字段	是否必选	说明
size	是	表示图像的像素信息。 • width：必选字段，图片的宽度。 • height：必选字段，图片的高度。 • depth：必选字段，图片的通道数。
segmented	是	表示是否用于分割。
object	是	表示物体检测信息，多个物体标注会有多个object体。 • name：必选字段，标注内容的类别。 • pose：必选字段，标注内容的拍摄角度。 • truncated：必选字段，标注内容是否被截断（0表示完整）。 • occluded：必选字段，标注内容是否被遮挡（0表示未遮挡）。 • difficult：必选字段，标注目标是否难以识别（0表示容易识别）。 • confidence：可选字段，标注目标的置信度，取值范围0-1之间。 • bndbox：必选字段，标注框的类型，可选值请参见**表7-15**。

表 7-15 标注框类型描述

type	形状	标注信息
point	点	点的坐标。 <x>100<x> <y>100<y>
line	线	各点坐标。 <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>200<y2>
bndbox	矩形框	左上和右下两个点坐标。 <xmin>100<xmin> <ymin>100<ymin> <xmax>200<xmax> <ymax>200<ymax>

type	形状	标注信息
polygon	多边形	各点坐标。 <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>100<y2> <x3>250<x3> <y3>150<y3> <x4>200<x4> <y4>200<y4> <x5>100<x5> <y5>200<y5> <x6>50<x6> <y6>150<y6>
circle	圆形	圆心坐标和半径。 <cx>100<cx> <cy>100<cy> <r>50<r>

示例：

```
<annotation>
  <folder>test_data</folder>
  <filename>260730932.jpg</filename>
  <size>
     <width>767</width>
     <height>959</height>
     <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
     <name>point</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
     <occluded>0</occluded>
     <difficult>0</difficult>
     <point>
        <x1>456</x1>
        <y1>596</y1>
     </point>
  </object>
  <object>
     <name>line</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
     <occluded>0</occluded>
     <difficult>0</difficult>
     <line>
        <x1>133</x1>
        <y1>651</y1>
        <x2>229</x2>
        <y2>561</y2>
     </line>
  </object>
```



```
  <object>
    <name>bag</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <bndbox>
      <xmin>108</xmin>
      <ymin>101</ymin>
      <xmax>251</xmax>
      <ymax>238</ymax>
    </bndbox>
  </object>
  <object>
    <name>boots</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <hard-coefficient>0.8</hard-coefficient>
    <polygon>
      <x1>373</x1>
      <y1>264</y1>
      <x2>500</x2>
      <y2>198</y2>
      <x3>437</x3>
      <y3>76</y3>
      <x4>310</x4>
      <y4>142</y4>
    </polygon>
  </object>
  <object>
    <name>circle</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <circle>
      <cx>405</cx>
      <cy>170</cy>
      <r>100<r>
    </circle>
  </object>
</annotation>
```

声音分类

```
{
"source":
"s3://path/to/pets.wav",
   "annotation": [
      {
         "type": "modelarts/audio_classification",
         "name":"cat",
         "annotated-by":"human",
         "creation-time":"2019-01-23 11:30:30"
      }
   ]
}
```

“source”、“usage”、“annotation”等参数说明与**图像分类**一致，详细说明请参见**表7-7**。

语音内容

```
{
   "source":"s3://path/to/audio1.wav",
   "annotation":[
```

```
    {
      "type":"modelarts/audio_content",
      "property":{
        "@modelarts:content":"Today is a good day."
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    }
  ]
}
```

- “source”、“usage”、“annotation”等参数说明与**图像分类**一致，详细说明请参见**表7-7**。
- “property”中的“@modelarts:content”参数，数据类型为“String”，表示语音内容。

语音分割

```
{
  "source":"s3://path/to/audio1.wav",
  "usage":"TRAIN",
  "annotation":[
    {

"type":"modelarts/audio_segmentation",
      "property":{
        "@modelarts:start_time":"00:01:10.123",
        "@modelarts:end_time":"00:01:15.456",

        "@modelarts:source":"Tom",

        "@modelarts:content":"How are you?"
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    },
    {
      "type":"modelarts/audio_segmentation",
      "property":{
        "@modelarts:start_time":"00:01:22.754",
        "@modelarts:end_time":"00:01:24.145",
        "@modelarts:source":"Jerry",
        "@modelarts:content":"I'm fine, thank you."
      },
      "annotated-by":"human",
      "creation-time":"2019-01-23 11:30:30"
    }
  ]
}
```

- “source”、“usage”、“annotation”等参数说明与**图像分类**一致，详细说明请参见**表7-7**。
- “property”的参数解释如**表7-16**所示。

表 7-16 “property”参数说明

参数名	数据类型	描述
@modelarts:start_time	String	声音的起始时间，格式为“hh:mm:ss.SSS”。 其中“hh”表示小时，“mm”表示分钟，“ss”表示秒，“SSS”表示毫秒。

参数名	数据类型	描述
@modelarts:end_time	String	声音的结束时间，格式为“hh:mm:ss.SSS”。 其中“hh”表示小时，“mm”表示分钟，“ss”表示秒，“SSS”表示毫秒。
@modelarts:source	String	声音来源。
@modelarts:content	String	声音内容。

视频标注

```
{
  "annotation": [{
    "annotation-format": "PASCAL VOC",
    "type": "modelarts/object_detection",
    "annotation-loc": "s3://path/to/annotation1_t1.473722.xml",
    "creation-time": "2020-10-09 14:08:24",
    "annotated-by": "human"
  }],
  "usage": "train",
  "property": {
    "@modelarts:parent_duration": 8,
    "@modelarts:parent_source": "s3://path/to/annotation1.mp4",
    "@modelarts:time_in_video": 1.473722
  },
  "source": "s3://input/path/to/annotation1_t1.473722.jpg",
  "id": "43d88677c1e9a971eeb692a80534b5d5",
  "sample-type": 0
}
```

- “source”、“usage”、“annotation”等参数说明与图像分类一致，详细说明请参见表7-7。
- “annotation-loc”：对于物体检测、是必选字段，对于其他类型是可选字段，标注文件的存储路径。
- “annotation-format”：描述标注文件的格式，可选字段，默认为“PASCAL VOC”。目前只支持“PASCAL VOC”。
- “**sample-type**”：样本格式，0表示图片，1表示文本，2表示语音，4表示表格，6表示视频。

表 7-17 property 参数说明

参数名	数据类型	说明
@modelarts:parent_duration	Double	标注视频的时长，单位：秒。
@modelarts:time_in_video	Double	标注的视频帧的时间戳，单位：秒。
@modelarts:parent_source	String	标注视频的OBS路径。

表 7-18 PASCAL VOC 格式说明

字段	是否必选	说明
folder	是	表示数据源所在目录。
filename	是	被标注文件的文件名。
size	是	表示图像的像素信息。 • width：必选字段，图片的宽度。 • height：必选字段，图片的高度。 • depth：必选字段，图片的通道数。
segmented	是	表示是否用于分割。
object	是	表示物体检测信息，多个物体标注会有多个object体。 • name：必选字段，标注内容的类别。 • pose：必选字段，标注内容的拍摄角度。 • truncated：必选字段，标注内容是否被截断（0表示完整）。 • occluded：必选字段，标注内容是否被遮挡（0表示未遮挡）。 • difficult：必选字段，标注目标是否难以识别（0表示容易识别）。 • confidence：可选字段，标注目标的置信度，取值范围0-1之间。 • bndbox：必选字段，标注框的类型，可选值请参见**表7-19**。

表 7-19 标注框类型描述

type	形状	标注信息
point	点	点的坐标。 <x>100<x> <y>100<y>
line	线	各点坐标。 <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>200<y2>

type	形状	标注信息
bndbox	矩形框	左上和右下两个点坐标。 <xmin>100<xmin> <ymin>100<ymin> <xmax>200<xmax> <ymax>200<ymax>
polygon	多边形	各点坐标。 <x1>100<x1> <y1>100<y1> <x2>200<x2> <y2>100<y2> <x3>250<x3> <y3>150<y3> <x4>200<x4> <y4>200<y4> <x5>100<x5> <y5>200<y5> <x6>50<x6> <y6>150<y6>
circle	圆形	圆心坐标和半径。 <cx>100<cx> <cy>100<cy> <r>50<r>

示例：

```
<annotation>
  <folder>test_data</folder>
  <filename>260730932_t1.473722.jpg.jpg</filename>
  <size>
     <width>767</width>
     <height>959</height>
     <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
     <name>point</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
     <occluded>0</occluded>
     <difficult>0</difficult>
     <point>
        <x1>456</x1>
        <y1>596</y1>
     </point>
  </object>
  <object>
     <name>line</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
```

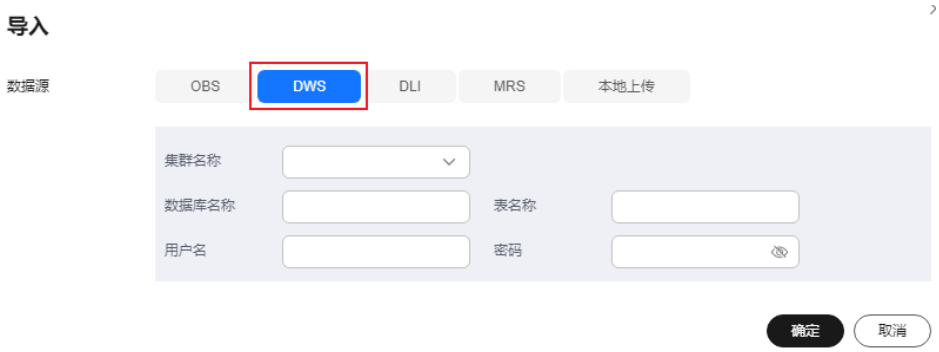


```
    <occluded>0</occluded>
    <difficult>0</difficult>
    <line>
      <x1>133</x1>
      <y1>651</y1>
      <x2>229</x2>
      <y2>561</y2>
    </line>
  </object>
  <object>
    <name>bag</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <bndbox>
      <xmin>108</xmin>
      <ymin>101</ymin>
      <xmax>251</xmax>
      <ymax>238</ymax>
    </bndbox>
  </object>
  <object>
    <name>boots</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <hard-coefficient>0.8</hard-coefficient>
    <polygon>
      <x1>373</x1>
      <y1>264</y1>
      <x2>500</x2>
      <y2>198</y2>
      <x3>437</x3>
      <y3>76</y3>
      <x4>310</x4>
      <y4>142</y4>
    </polygon>
  </object>
  <object>
    <name>circle</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <occluded>0</occluded>
    <difficult>0</difficult>
    <circle>
      <cx>405</cx>
      <cy>170</cy>
      <r>100<r>
    </circle>
  </object>
</annotation>
```

7.3.3 从 DWS 导入数据到 ModelArts 数据集

ModelArts支持从DWS导入表格数据，用户需要选择对应的DWS集群，并输入需要对应的数据库名、表名以及用户名和密码。所导入表的schema(列名和类型)需要跟数据集相同。

图 7-10 从 DWS 导入数据



- **集群名称**：系统自动将当前账号下的DWS集群展现在列表中，您可以在下拉框中选择您所需的DWS集群。
- **数据库名称**：根据选择的DWS集群，填写数据所在的数据库名称。
- **表名称**：根据选择的数据库，填写数据所在的表。
- **用户名**：输入DWS集群管理员用户的用户名。
- **密码**：输入DWS集群管理员用户的密码。

说明

从DWS导入数据，需要借助DLI的功能，如果用户没有访问DLI服务的权限，需根据页面提示创建DLI的委托。

7.3.4 从 DLI 导入数据到 ModelArts 数据集

表格数据集支持从DLI导入数据。

从DLI导入数据，用户需要选择DLI队列、数据库和表名称。所选择的表的schema(列名和类型)需与数据集一致，支持自动获取所选择表的schema。

图 7-11 DLI 导入数据

- 队列名称：系统自动将当前账号下的DLI队列展现在列表中，用户可以在下拉框中选择需要的队列。
- 数据库名称：根据选择的队列展现所有的数据库，请在下拉框中选择您所需的数据库。

- 表名称：根据选择的数据库展现此数据库中的所有表。请在下拉框中选择您所需的表。

说明

DLI的default队列只用作体验，不同账号间可能会出现抢占的情况，需进行资源排队，不能保证每次都可以得到资源执行相关操作。

DLI支持schema映射的功能，即导入的表的schema的字段名称可以不和数据集相同，但类型要保持一致。

7.3.5 从 MRS 导入数据到 ModelArts 数据集

ModelArts支持从MRS服务中导入存储在HDFS上的csv格式的数据，首先需要选择已有的MRS集群，并从HDFS文件列表选择文件名称或所在目录，导入文件的列数需与数据集schema一致。

图 7-12 从 MRS 导入数据

- **集群名称**：系统自动将当前账号下的MRS集群展现在此列表中，但是流式集群不支持导入操作。请在下拉框中选择您所需的集群。
- **文件路径**：根据选择的集群，输入对应的文件路径，此文件路径为HDFS路径。
- **导入是否包含表头**：开启表示导入时将表头同时导入。

7.3.6 从本地上传数据到 ModelArts 数据集

前提条件

- 已存在创建完成的数据集。
- 创建一个空的OBS桶，OBS桶与ModelArts在同一区域，并确保用户具有OBS桶的操作权限。

本地上传

文件型和表格型数据均支持从本地上传。从本地上传的数据存储在OBS目录中，请先提前创建OBS桶。

从本地上传的数据单次最多支持100个文件同时上传，总大小不超过5GB。

不同类型的数据集，导入操作界面的示意图存在区别，请参考界面信息了解当前类型数据集的示意图。当前操作指导以图像分类的数据集为例。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理 >数据集”，进入“数据集”管理页面。
2. 在数据集所在行，单击操作列的“导入”。

   或者，您可以单击数据集名称，进入数据集“概览”页，在页面右上角单击“导入”。
3. 在“导入”对话框中，参考如下说明填写参数，然后单击“确定”。
   - “数据来源”：“本地上传”
   - “上传数据存储路径”：数据存储的OBS路径。
   - “上传数据”：单击“文件上传”，上传本地的数据，单击“确定”。

图 7-13 从本地上传数据

7.4 标注 ModelArts 数据集中的数据

7.4.1 数据标注场景介绍

由于模型训练过程需要大量有标签的数据，因此在模型训练之前需对没有标签的数据添加标签。您可以通过创建单人标注作业或团队标注作业对数据进行手工标注，或对任务启动智能标注添加标签，快速完成对图片的标注操作，也可以对已标注图片修改或删除标签进行重新标注。

ModelArts为用户提供了标注数据的能力：

- 人工标注：用户创建单人标注作业，对数据进行手工标注。
- 智能标注：在标注一定量的数据情况下，用户可以通过启动智能标注任务对数据进行自动标注，提高标注的效率。
- 团队标注：对于大批量的数据，用户可以通过创建团队标注作业，进行多人协同标注。

人工标注

对于不同类型的数据，用户可以选择不同的标注类型。当前ModelArts支持如下类型的标注作业：

- 图片
  - 图像分类：识别一张图片中是否包含某种物体。
  - 物体检测：识别出图片中每个物体的位置及类别。
  - 图像分割：根据图片中的物体划分出不同区域。
- 音频
  - 声音分类：对声音进行分类。
  - 语音内容：对语音内容进行标注。
  - 语音分割：对语音进行分段标注。
- 文本
  - 文本分类：对文本的内容按照标签进行分类处理。
  - 命名实体：针对文本中的实体片段进行标注，如“时间”、“地点”等。
  - 文本三元组：针对文本中的实体片段和实体之间的关系进行标注。
- 视频

  视频标注：识别出视频中每个物体的位置及分类。目前仅支持mp4格式。

智能标注

除了人工标注外，ModelArts还提供了智能标注功能，快速完成数据标注，为您节省70%以上的标注时间。智能标注是指基于当前标注阶段的标签及图片学习训练，选中系统中已有的模型进行智能标注，快速完成剩余图片的标注操作。

目前只有“图像分类”和“物体检测”类型的数据集支持智能标注功能。

团队标注

数据标注任务中，一般由一个人完成，但是针对数据集较大时，需要多人协助完成。ModelArts提供了团队标注功能，可以由多人组成一个标注团队，针对同一个数据集进行标注管理。

团队标注功能当前仅支持“图像分类”、“物体检测”、“文本分类”、“命名实体”、“文本三元组”、“语音分割”类型的数据集。

不同类型数据集支持的功能列表

其中，不同类型的数据集，支持不同的功能，详细信息请参见**表7-20**。

表 7-20 不同类型数据集支持的功能

数据集类型	标注类型	人工标注	智能标注	团队标注
图片	图像分类	支持	支持	支持
	物体检测	支持	支持	支持
	图像分割	支持	-	-
音频	声音分类	支持	-	-
	语音内容	支持	-	-

数据集类型	标注类型	人工标注	智能标注	团队标注
	语音分割	支持	-	支持
文本	文本分类	支持	-	支持
	命名实体	支持	-	支持
	文本三元组	支持	-	支持
视频	视频标注	支持	-	-
自由格式	-	-	-	-
表格	-	-	-	-

7.4.2 通过人工标注方式标注数据

7.4.2.1 创建 ModelArts 人工标注作业

由于模型训练过程需要大量有标签的数据，因此在模型训练之前需对没有标签的数据添加标签。您可以通过创建单人标注作业或团队标注作业对数据进行手工标注，或对任务启动智能标注添加标签，快速完成对图片的标注操作，也可以对已标注图片修改或删除标签进行重新标注。

说明

数据标注功能仅在以下Region支持：华北-北京四、西南-贵阳一、中国-香港、亚太-新加坡、亚太-曼谷、亚太-雅加达、拉美-圣地亚哥、拉美-圣保罗一、拉美-墨西哥城二。

标注作业支持的数据类型

对于不同类型的数据集，用户可以选择不同的标注任务，当前ModelArts支持如下类型的标注任务。

- 图片
  - 图像分类：识别一张图片中是否包含某种物体。
  - 物体检测：识别出图片中每个物体的位置及类别。
  - 图像分割：根据图片中的物体划分出不同区域。
- 音频
  - 声音分类：对声音进行分类。
  - 语音内容：对语音内容进行标注。
  - 语音分割：对语音进行分段标注。
- 文本
  - 文本分类：对文本的内容按照标签进行分类处理。
  - 命名实体：针对文本中的实体片段进行标注，如“时间”、“地点”等。

  - 文本三元组：针对文本中的实体片段和实体之间的关系进行标注。
- 视频

  视频标注：识别出视频中每个物体的位置及分类。目前仅支持mp4格式。

前提条件

在进行数据标注前，需要创建相应类型的数据集。具体步骤参考**创建数据集**。

操作步骤

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“数据准备> 数据标注”，进入“数据标注”管理页面。
2. 在数据标注管理页面，单击页面右上角“创建标注作业”，进入“创建标注作业”页面，根据需求创建不同类型的标注作业。
   a. 填写标注作业基本信息，标注作业的“名称”和“描述”。
   b. 根据您的需求，选择“标注场景”和“标注类型”。

      图 7-14 选择标注场景和标注类型

   c. 针对不同类型的标注作业，需填写参数不同，请参考如下类型标注作业对应的参数介绍。
      - **图片（图像分类、物体检测、图像分割）**
      - **音频（声音分类、语音内容、语音分割）**
      - **文本（文本分类、命名实体、文本三元组）**
      - **视频**
   d. 参数填写无误后，单击页面右下角“创建”。

      标注作业创建完成后，系统自动跳转至数据标注管理页面，针对创建好的标注作业，您可以执行智能标注、发布、修改和删除等操作。

图片（图像分类、物体检测、图像分割）

图 7-15 图像分类和物体检测类型的参数

表 7-21 图片类型标注作业的详细参数

参数名称	说明
数据集名称	选择支持当前标注类型的数据集。
添加标签集	• **设置标签名称**：在标签名称文本框中，输入标签名称。长度为1～1024字符。 • **添加标签**：单击“添加标签”可增加多个标签。 • **设置标签颜色**：“物体检测”和“图像分割”类型标注作业需设置此参数。在每个标签右侧的标签颜色区域下，可在色板中选择颜色，或者直接输入十六进制颜色码进行设置。 • **设置标签属性**：针对“物体检测”类型标注作业，在设置完标签颜色后，可在右侧单击加号，增加对应的标签属性。标签属性用于区分同一标签物体的不同属性。例如，黄色小猫、黑色小猫。标签为cat，颜色为不同的标签属性。
启用团队标注	选择是否启用团队标注。图像分割暂不支持团队标注，当选择图像分割类型时，界面不显示此参数。 启用团队标注功能，需填写对应的团队标注任务“类型”，同时选择对应的“标注团队”及参与标注的“团队成员”。参数详细介绍请参见**创建团队标注任务**。 在启用“团队标注”前，需确保您已经在“标注团队”管理页面，添加相应的团队以及成员。如果没有标注团队，可直接从界面链接跳转至“标注团队”页面，添加您的团队并为其添加成员。详细指导请参见**创建和管理团队**。 启用团队标注功能的数据集，在创建完成后，可以在“标注类型”中看到“团队标注”的标识。

音频（声音分类、语音内容、语音分割）

图 7-16 声音分类、语音内容、语音分割类型的参数

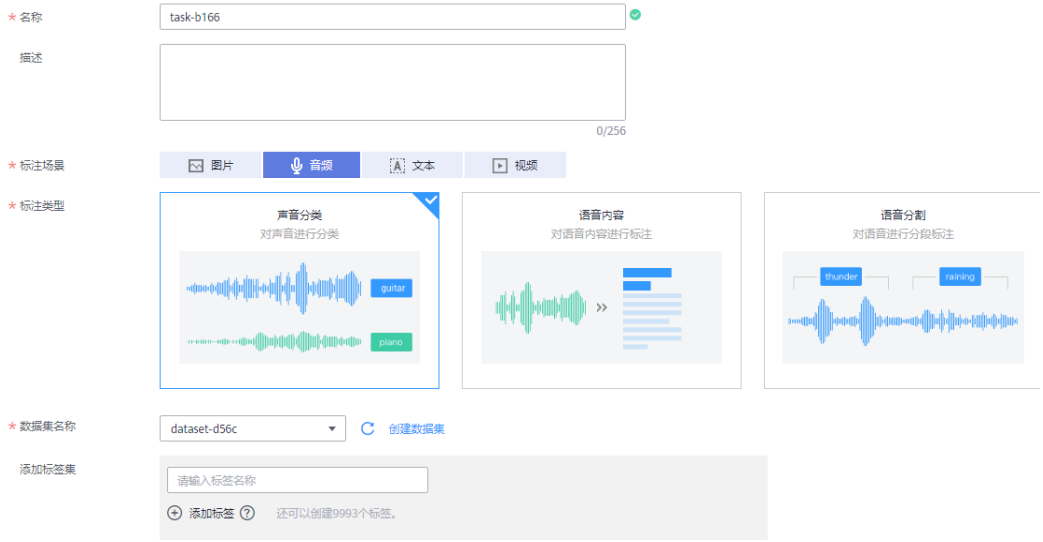


表 7-22 音频类型标注作业的详细参数

参数名称	说明
数据集名称	选择支持当前标注类型的数据集。
添加标签集（声音分类）	“声音分类”类型的标注作业可以添加标签集。 • **设置标签名称**：在标签名称文本框中，长度为1～1024字符。 • **添加标签**：单击“添加标签”可增加多个标签。
标签管理（语音分割）	“语音分割”类型的标注作业，支持标签管理。 • **单标签** 单标签适用于一段音频标注只有一种类别的音频，通常标注一个标签。 – **设置标签名称**：在“标签名”列输入标签名称。长度为1～1024字符。 – **设置标签颜色**：在“标签颜色”列设置标签颜色。可在色板中选择颜色，或者直接输入十六进制颜色码进行设置。 • **多标签** 多标签适用于多维度标注，例如在一段音频标注噪音与人说话的声音两种类别，其中说话的声音还可以标注为不同人的声音。单击“新建标签类别”可添加多个标签类别，一个标签类别可以包含多个标签。“标签类别”和“标签名”只能是中文、字母、数字、英文句号、下划线或中划线组成的合法字符串。长度为1～256字符。 – **设置标签类别**：在“标签类别”输入标签类别的名称。 – **设置标签名称**：在“标签名”输入标签名称。 – **添加标签**：单击“添加标签”可增加多个标签。

参数名称	说明
启用语音内容标注（语音分割）	仅“语音分割”类型数据集支持设置，默认关闭。如果启用此功能，支持针对语音内容进行标注。
启用团队标注（语音分割）	仅“语音分割”类型支持团队标注，因此选择创建语音分割类型时，支持设置是否启用团队标注。 启用团队标注功能，需填写对应的团队标注任务“类型”，同时选择对应的“标注团队”及参与标注的“团队成员”。参数详细介绍请参见**创建团队标注任务**。 在启用“团队标注”前，需确保您已经在“标注团队”管理页面，添加相应的团队以及成员。如果没有标注团队，可直接从界面链接跳转至“标注团队”页面，添加您的团队并为其添加成员。详细指导请参见**创建和管理团队**。 启用团队标注功能的数据集，在创建完成后，可以在“标注类型”中看到“团队标注”的标识。

文本（文本分类、命名实体、文本三元组）

图 7-17 文本分类、命名实体、文本三元组类型的参数

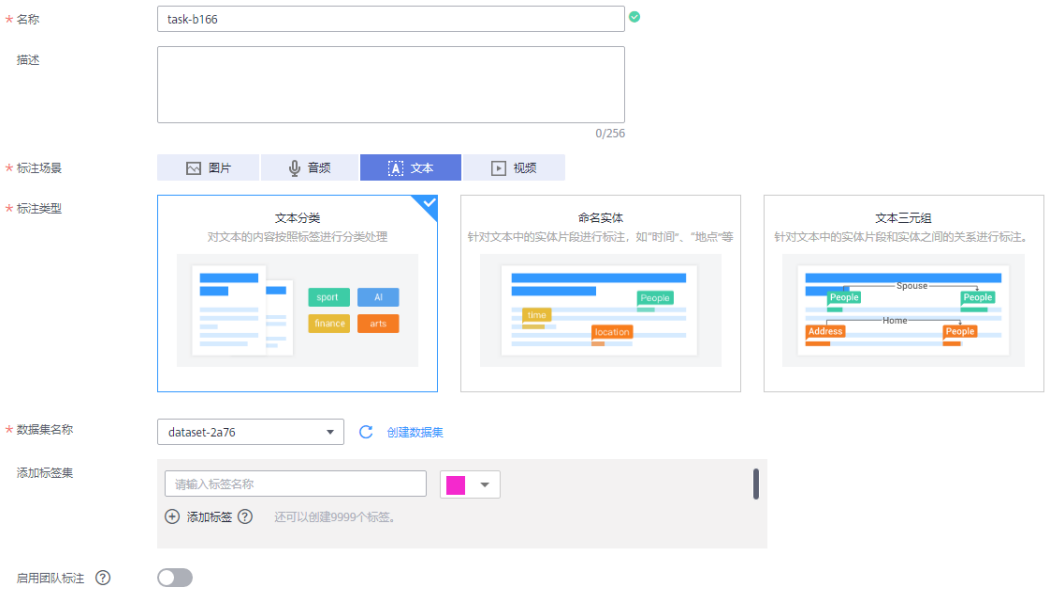
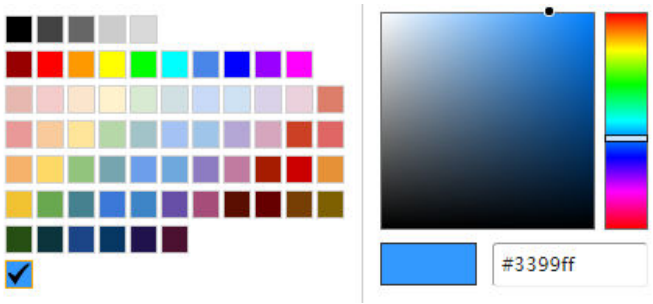
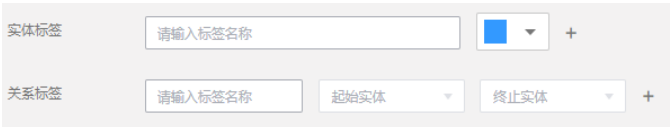


表 7-23 文本类型标注作业的详细参数

参数名称	说明
数据集名称	选择支持当前标注类型的数据集。

参数名称	说明
添加标签集（文本分类、命名实体）	• **设置标签名称**：在标签名称文本框中，输入标签名称。长度为1～1024字符。 • **添加标签**：单击“添加标签”可增加多个标签。 • **设置标签颜色**：在每个标签右侧的标签颜色区域下，可在色板中选择颜色，或者直接输入十六进制颜色码进行设置。 
添加标签集（文本三元组）	针对“文本三元组”类型的数据集，需要设置实体标签和关系标签。 • **实体标签**：需设置标签名以及标签颜色。可在颜色区域右侧单击加号增加多个标签。 • **关系标签**：关系标签为两个实体之间的关系。需设置起始实体和终止实体，您需要先添加至少2个实体标签后，再添加关系标签。 
启用团队标注	选择是否启用团队标注。 启用团队标注功能，需填写对应的团队标注任务“类型”，同时选择对应的“标注团队”及参与标注的“团队成员”。参数详细介绍请参见**创建团队标注任务**。 在启用“团队标注”前，需确保您已经在“标注团队”管理页面，添加相应的团队以及成员。如果没有标注团队，可直接从界面链接跳转至“标注团队”页面，添加您的团队并为其添加成员。详细指导请参见**创建和管理团队**。 启用团队标注功能的数据集，在创建完成后，可以在“标注类型”中看到“团队标注”的标识。

视频

图 7-18 视频类型的参数

表 7-24 视频类型标注作业的详细参数

参数名称	说明
数据集名称	选择支持当前标注类型的数据集。
添加标签集	• **设置标签名称**：在标签名称文本框中，输入标签名称。长度为1～1024字符。 • **添加标签**：单击“添加标签”可增加多个标签。 • **设置标签颜色**：在每个标签右侧的标签颜色区域下，可在色板中选择颜色，或者直接输入十六进制颜色码进行设置。

7.4.2.2 人工标注图片数据

由于模型训练过程需要大量有标签的图片数据，因此在模型训练之前需对没有标签的图片添加标签。您可以通过手工标注或智能一键标注的方式添加标签，快速完成对图片的标注操作，也可以对已标注图片修改或删除标签进行重新标注。

针对图像分类场景，开始标注前，您需要了解：

- 图片标注支持多标签，即一张图片可添加多个标签。

- 标签名是由中文、大小写字母、数字、中划线或下划线组成，且不超过1024位的字符串。

针对物体检测场景，开始标注前，您需要了解：

- 图片中所有目标物体都要标注。
- 目标物体清晰无遮挡的，必须画框。
- 画框仅包含整个物体。框内包含整个物体的全部，画框边缘不可与待标注的物体的边缘轮廓相交，在此基础之上确保边缘和待标注物体间不要留着空隙，避免背景对模型训练造成干扰。

针对图像分割场景，开始标注前，您需要了解：

- 图片中需要提取轮廓的物体都要标注。
- 支持使用多边形标注。
  - 多边形标注，根据目标物体的轮廓绘制多边形。
- 多边形标注或极点标注时，标注框必须在图片范围内，超出图片将导致后续作业异常。

开始标注

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“数据准备> 数据标注”，进入“数据标注”管理页面。
2. 在标注作业列表右侧“所有类型”页签下拉选择标注类型。基于“标注类型”选择需要进行标注的标注作业，单击标注作业名称进入标注作业标注详情页。

   图 7-19 下拉选择标注类型

3. 在标注作业标注详情中，展示此标注作业下全部数据。

标注图片（图像分类）

在标注作业详情页中，展示了此数据集中“全部”、“未标注”和“已标注”的图片，默认显示“未标注”的图片列表。单击图片，即可进行图片的预览，对于已标注图片，预览页面下方会显示该图片的标签信息。

1. 在“未标注”页签，勾选需进行标注的图片。
   - 手工点选：在图片列表中，单击勾选图片左上角的选择框，进入选择模式，表示图片已勾选。可勾选同类别的多个图片，一起添加标签。
   - 批量选中：如果图片列表的当前页，所有图片属于一种类型，可以在图片列表的右上角单击“选择当前页”，则当前页面所有的图片将选中。
2. 为选中图片添加标签。
   a. 在右侧的“添加标签”区域中，单击“标签名”右侧的文本框中设置标签。
      单击“标签名”右侧的文本框，然后从下拉列表中选择已有的标签。如果已有标签无法满足要求时，直接在文本框中添加新标签。
   b. 单击“确定”。此时，选中的图片将被自动移动至“已标注”页签，且在“未标注”和“全部”页签中，标签的信息也将随着标注步骤进行更新，如增加的标签名称、各标签对应的图片数量。

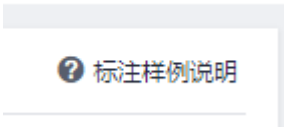
图 7-20 添加标签

说明

如果您还不太清楚如何进行标注，可参考数据集详情页面的“标注样例说明”完成标注。

1. 登录ModelArts管理控制台，选择“数据准备 > 数据标注”进入数据标注页。
2. 在“我创建的”或“我参与的”页签下，找到您需要标注的数据集。
3. 单击数据集名称，进入标注详情页。（默认直接进入“未标注”页签）。
4. 在标注详情页右上角单击“标注样例说明”。

图 7-21 标注样例说明



标注图片（物体检测）

标注作业详情页中，展示了此数据集中“全部”“未标注”和“已标注”的图片，默认显示“未标注”的图片列表。

1. 在“未标注”页签图片列表中，单击图片，自动跳转到标注页面。在标注页面，常用按钮的使用可参见表7-26。
2. 在页面上方工具栏选择合适的标注图形，系统默认的标注图形为矩形。本示例使用矩形工具进行标注。

说明

页面左侧可以选择多种形状对图片进行标注。标注第一张图片时，一旦选择其中一种，其他图片默认使用此形状进行标注，用户可以根据自己需求再进行切换。

表 7-25 支持的标注框

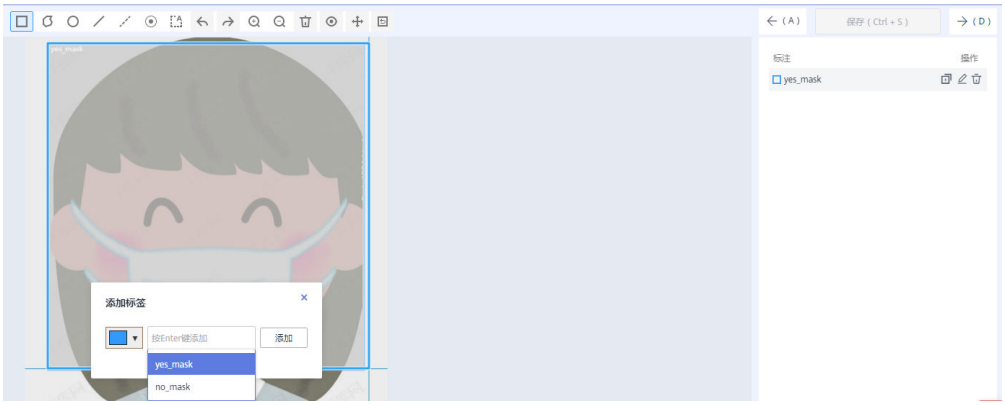
图标	使用说明
	矩形。也可使用快捷键【1】。鼠标单击标注对象左上角边缘位置，界面将出现矩形框，移动鼠标使得矩形框覆盖标注对象，然后单击完成标注。
	多边形。也可使用快捷键【2】。在标注对象所在范围内，鼠标左键单击完成一个点的标注，沿着物体的形状边缘，通过鼠标指定多个点，最终单击到第一个点的位置，由所有的点组成一个多边形形状。使得需标注的对象在此标注框内。
	圆形。也可使用快捷键【3】。在标注对象中，选择物体的中心点位置，单击鼠标确定圆心，然后移动鼠标，使得圆形框覆盖标注对象，然后再单击鼠标完成标注。
	直线。也可使用快捷键【4】。在标注对象中，选择物体的起始点，单击鼠标确定直线的起始点，然后使得直线覆盖标注对象，然后再单击鼠标完成标注。
	虚线。也可使用快捷键【5】。在标注对象中，选择物体的起始点，单击鼠标确定虚线的起始点，然后使得虚线覆盖标注对象，然后再单击鼠标完成标注。

图标	使用说明
	点。也可使用快捷键【6】。单击图片中的物体所在位置，即可完成点的标注。

3. 在弹出的添加标签文本框中，直接输入新的标签名，在文本框前面选中标签颜色，然后单击“添加”。如果已存在标签，从下拉列表中选择已有的标签，单击“添加”。

   逐步标注图片中所有物体所在位置，一张图片可添加多个标签。完成一张图片标注后，可单击图片右上角来切换下一张（也可使用快捷键【D】直接切换），快速选中其他未标注的图片，然后在标注页面中执行标注操作。

   图 7-22 添加物体检测标签

4. 单击页面上方“返回数据标注预览”查看标注信息，在弹框中单击“确定”保存当前标注并离开标注页面。

   选中的图片被自动移动至“已标注”页签，且在“未标注”和“全部”页签中，标签的信息也将随着标注步骤进行更新，如增加的标签名称、标签对应的图片数量。

表 7-26 标注界面的常用按钮

按钮图标	功能说明
	撤销上一个操作。也可使用快捷键【Ctrl+Z】
	重做上一个操作。也可使用快捷键【Ctrl+Shift+Z】
	放大图片。也可以使用滚轮进行放大。
	缩小图片。也可以使用滚轮进行缩小。
	删除当前图片中的所有标注框。也可使用快捷键【Shift+Delete】
	显示或隐藏标注框。只有在已标注图片中可使用此操作。也可使用快捷键【Shift+H】

按钮图标	功能说明
	拖动，可将标注好的框拖动至其他位置，也可以选择框的边缘，更改框的大小。也可使用【X+鼠标左键】
	复位，与上方拖动为同组操作，当执行了拖动后，可以单击复位按钮快速将标注框恢复为拖动前的形状和位置。也可使用快捷键【Esc】

标注图片（图像分割）

标注作业详情页中，展示了此标注作业中**“全部”**、“未标注”和“已标注”的图片，默认显示“未标注”的图片列表。

1. 在“未标注”页签图片列表中，单击图片，自动跳转到标注页面。在标注页面，常用按钮的使用可参见**表7-28**。
2. 选择标注方式。

   在标注页面，上方工具栏提供了常用的**表7-27**及**表7-28**，系统默认的标注方式为多边形标注。选择多边形标注或极点标注。

   说明

   标注第一张图片时，一旦选择其中一种，其他所有图片都需要使用此方式进行标注。

表 7-27 标注方式

图标	使用说明
	多边形。在标注对象所在范围内，鼠标左键单击完成一个点的标注，沿着物体的形状边缘，通过鼠标指定多个点，最终单击到第一个点的位置，由所有的点组成一个多边形形状。使得需标注的对象在此标注框内。

表 7-28 工具栏常用按钮

按钮图标	功能说明
	撤销上一个操作。
	重做上一个操作。
	放大图片。
	缩小图片。
	删除当前图片中的所有标注框。
	显示或隐藏标注框。只有在已标注图片中可使用此操作。

按钮图标	功能说明
	拖动，可将标注好的框拖动至其他位置，也可以选择框的边缘，更改框的大小。
	复位，与上方拖动为同组操作，当执行了拖动后，可以单击复位按钮快速将标注框恢复为拖动前的形状和位置。
	全屏显示标注的图片。

3. 标注物体。

   识别图片中的物体，单击左键分别定位物体的最上、最左、最下、最右的位置点。确定位置后，单击标注区域，将弹出对话框，填入标签名称，单击“确定”。完成一张图片标注后，可单击图片下方  展开缩略图，查看图片列表，快速选中其他未标注的图片，然后在标注页面中执行标注操作。

4. 单击页面上方“返回数据标注预览”查看标注信息，在弹框中单击“确定”保存当前标注并离开标注页面。

   选中的图片被自动移动至“已标注”页签，且在“未标注”和“全部”页签中，标签的信息也将随着标注步骤进行更新，如增加的标签名称、标签对应的图片数量。

快速复核

当前的标注作业无法实现批量复核，如果有某一样本的标签修改或者删除，只能进入到标注页面详情进行，操作繁琐。为了简化用户操作，实现此功能，用户可以批量进行标注信息的审核或者修改，提升用户效率。

1. 登录ModelArts管理控制台，在总览页选择“数据准备>数据标注”，进入“我创建的”页签，在右上方的作业类型中下拉选择对应类型的标注作业。（仅物体检测与图像分割支持快速复核功能）
2. 在物体检测类型的标注作业列表，单击标注作业名称，进入标注详情页。
3. 单击“已标注”页签的“快速复核”，进入复核页面，对标注结果进行确认。

图 7-23 进入快速复核

4. 快速复核，支持您按照标签批量复核。

   a. 在复核页面，单击“按照标签过滤”，选择需要复核的标签类型图片。

   b. 在当前页面，您可以选择对当前的标签类型的图片，按照标注面积排序，或按照宽高比排序。

   c. 依次单击需要复核的图片，在标注页面拖动图片的标注框，即可重新完成标注。（修改后的图片会带有“已修改”的信息。）

d. 您也可以选中需要删除标签的图片，单击右上方的 ，删除原始的标注信息。（删除后的图片会带有“已删除”的信息）

图 7-24 已修改

图 7-25 已删除

e. 您也可以对当前已标注的图片标签信息进行修改。

i. 选中待复核的图片，单击右侧的“全部标签”区域的 按钮。
ii. 输入新的标签，单击“确定”。

图 7-26 全部标签

图 7-27 添加标签

5. 标注页面和标签都修改完成后，单击“应用所有修改”，在弹出的对话框单击“确定”，自动返回至标注概览页，同时会覆盖原始的标注数据。

图 7-28 应用所有修改

6. 如果您对修改后的数据不满意，也可以单击“放弃修改”选择放弃本次修改，保持原有的标注数据。

图 7-29 放弃修改

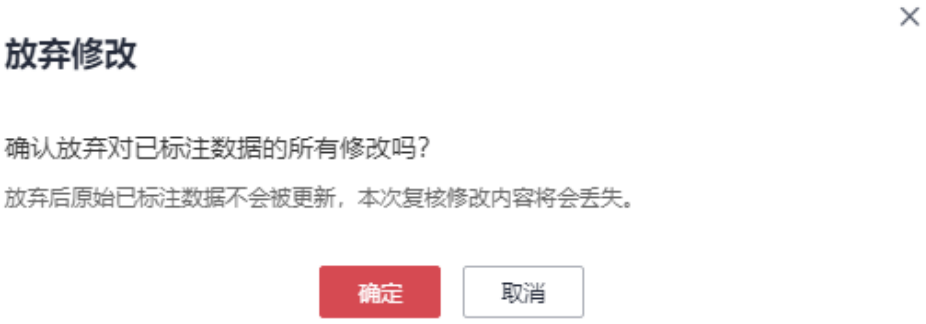


表 7-29 快速复核界面的常用按钮

按钮图标	功能说明
	删除原有的标注数据，删除后可重新标注。
	还原本页所有操作至未复核页面。
	撤销上一步操作。
	重做上一步操作。

7.4.2.3 人工标注文本数据

由于模型训练过程需要大量有标签的数据，因此在模型训练之前需对没有标签的文本添加标签。您也可以对已标注文本进行修改、删除和重新标注。

针对文本分类场景，是对文本的内容按照标签进行分类处理，开始标注前，您需要了解：

- 文本标注支持多标签，即一个标注对象可添加多个标签。
- 标签名是由中文、大小写字母、数字、中划线、下划线或特殊符号组成，且不超过1024位的字符串。

命名实体场景，是针对文本中的实体片段进行标注，如“时间”、“地点”等。开始标注前，您需要了解：

- 实体命名标签名是由中文、大小写字母、数字、中划线、下划线或特殊符号组成，且不超过1024位的字符串。

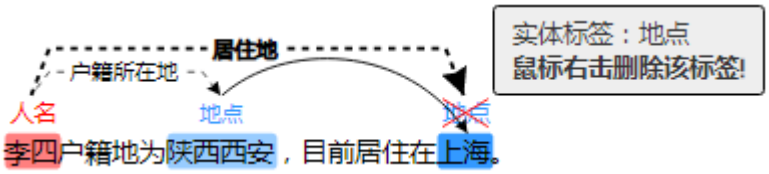
三元组标注适用于标注出语句当中形如（主语/Subject，谓词/Predicate，宾语/Object）结构化知识的场景，标注时不但可以标注出语句当中的实体，还可以标注出实体之间的关系，其在依存句法分析、信息抽取等自然语言处理任务中经常用到。在开始标注之前，您需要了解：

- 标注作业对应的“实体标签”和“关系标签”已定义好。“关系标签”需设置对应的“起始实体”和“终止实体”。“关系标签”只能添加至其设置好的“起始实体”和“终止实体”之间。
- 支持设置多个“实体标签”和“关系标签”。一个文本数据中，也可以标注多个“实体标签”和“关系标签”
- 创建数据集时定义的“实体标签”，不支持删除。

例如，如**图7-30**所示，当两个文本都被标注为“地点”，那么针对这两个实体，无法添加本示例中的任意一个关系标签。当无法添加某个关系标签时，界面将显示一个红色的叉号，如**图7-31**所示。

图 7-30 实体标签和关系标签的示例

图 7-31 无法添加关系标签



开始标注

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“数据准备> 数据标注”，进入“数据标注”管理页面。
2. 在标注作业列表右侧“所有类型”页签下拉选择标注类型，基于“标注类型”选择需要进行标注的标注作业，单击标注作业名称进入标注作业标注详情页。

图 7-32 下拉选择标注类型

3. 在标注作业标注详情中，展示此标注作业下全部数据。

标注文本（文本分类）

标注作业详情页中，展示了此标注作业中“未标注”和“已标注”的文本，默认显示“未标注”的文本列表。

1. 在“未标注”页签文本列表中，页面左侧罗列“标注对象列表”。在列表中单击需标注的文本对象，选择右侧“标签集”中的标签进行标注。一个标注对象可添加多个标签。

   以此类推，不断选中标注对象，并为其添加标签。

   图 7-33 文本分类标注

2. 当所有的标注对象都已完成标注，单击页面下方“保存当前页”，完成“未标注”列表的文本标注。

添加标签（文本分类）

- 在“未标注”页签添加：单击页面中标签集右侧的加号，然后在弹出的“新增标签”页中，添加标签名称，选择标签颜色，单击“确定”完成标签的新增。

  图 7-34 添加标签（1）

- 在“已标注”页签添加：单击页面中标签集右侧的加号，然后在弹出的“新增标签”页中，添加标签名称，选择标签颜色，单击“确定”完成标签的新增。

  图 7-35 添加标签（2）

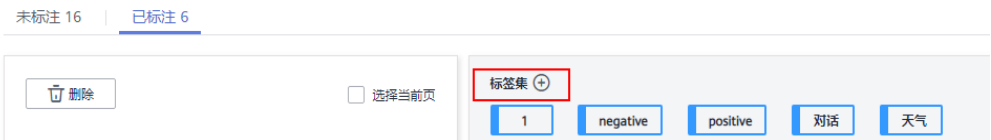
  

图 7-36 新增标签

标注文本（命名实体）

标注作业详情页中，展示了此标注作业中“未标注”和“已标注”的文本，默认显示“未标注”的文本列表。

1. 在“未标注”页签文本列表中，页面左侧罗列“标注对象列表”。在列表中单击需标注的文本对象，在右侧标签集下显示的文本内容中选中需要标注的部分，然后选择右侧“标签集”中的标签进行标注。

   以此类推，不断选中标注对象，并为其添加标签。

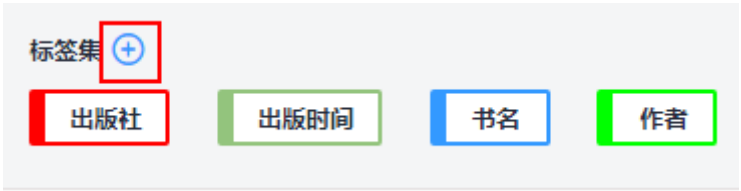
图 7-37 命名实体标注

2. 单击页面下方“保存当前页”完成文本标注。

添加标签（命名实体）

- 在“未标注”页签添加：单击页面中标签集右侧的加号，然后在弹出的“新增标签”页中，添加标签名称，选择标签颜色，单击“确定”完成标签的新增。

图 7-38 添加命名实体标签（1）



- 在“已标注”页签添加：单击页面中标签集右侧的加号，然后在弹出的“新增标签”页中，添加标签名称，选择标签颜色，单击“确定”完成标签的新增。

图 7-39 添加命名实体标签（2）

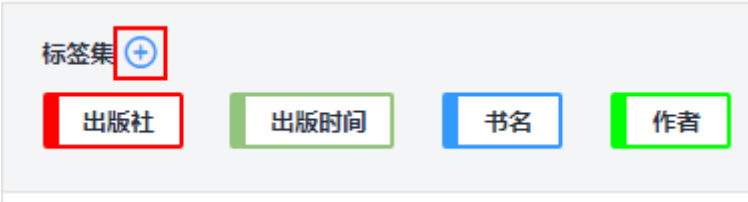


图 7-40 新增命名实体标签

标注文本（文本三元组）

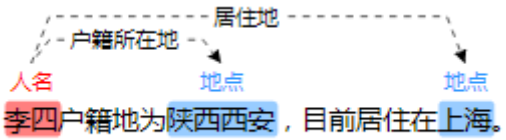
标注作业详情页中，展示了此标注作业中“未标注”和“已标注”的文本，默认显示“未标注”的文本列表。

1. 在“未标注”页签文本列表中，页面左侧罗列“标注对象列表”。在列表中单击需标注的文本对象，选中相应文本内容，在页面呈现的实体类型列表中选择实体名称，完成实体标注。

图 7-41 实体标注

2. 在完成多个实体标注后，鼠标左键依次单击起始实体和终止实体，在呈现的关系类型列表中选择一个对应的关系类型，完成关系标注。

图 7-42 关系标注



3. 当所有的标注对象都已完成标注，单击页面下方“保存当前页”完成“未标注”列表的文本标注。

说明

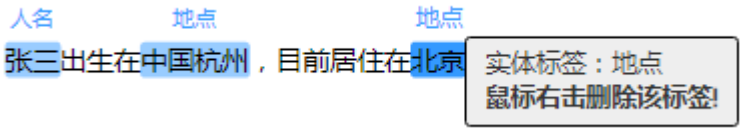
“文本三元组”类型的数据集，不支持在标注页面修改标签，需要进入“标签管理”页面，修改“实体标签”和“关系标签”。

修改标签（文本三元组）

当数据完成标注后，您还可以进入已标注页签，对已标注的数据进行修改。

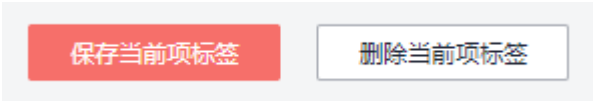
在标注作业详情页，单击“已标注”页签，在左侧文本列表中选中一行文本，右侧区域显示具体的标注信息。将鼠标移动至对应的实体标签或关系类型，单击鼠标右键，可删除此标注。单击鼠标左键，依次单击连接起始实体和终止实体，可增加关系类型，增加关系标注。

图 7-43 在文本中修改标签



您也可以在单击页面下方的“删除当前项标签”按钮，删除选中文本对象中的所有标签。

图 7-44 删除当前项标签



7.4.2.4 人工标注音频数据

由于模型训练过程需要大量有标签的音频数据，因此在模型训练之前需对没有标签的音频添加标签。通过ModelArts您可对音频进行一键式批量添加标签，快速完成对音频的标注操作，也可以对已标注音频修改或删除标签进行重新标注。音频标注涉及到的标注标签和声音内容只支持中文和英文，不支持小语种。

声音分类是对声音进行分类。语音内容是对语音内容进行标注。语音分割是对语音进行分段标注。

开始标注

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“数据准备> 数据标注”，进入“数据标注”管理页面。
2. 在标注作业列表右侧“所有类型”页签下拉选择标注类型，基于“标注类型”选择需要进行标注的标注作业，单击标注作业名称进入标注作业标注详情页。

图 7-45 下拉选择标注类型

3. 在标注作业标注详情中，展示此标注作业下全部数据。

标注音频（声音分类）

标注作业详情页中，展示了此标注作业中“未标注”和“已标注”的音频，默认显示“未标注”的音频列表。单击音频左侧，即可进行音频的试听。

1. 在“未标注”页签，勾选需进行标注的音频。
   - 手工点选：在音频列表中，单击音频，当右上角出现蓝色勾选框时，表示已勾选。可勾选同类别的多个音频，一起添加标签。
   - 批量选中：如果音频列表的当前页，所有音频属于一种类型，可以在列表的右上角单击“选择当前页”，则当前页面所有的音频将选中。
2. 添加标签。
   a. 在右侧的“添加标签”区域中，单击“标签”下侧的文本框设置标签。

      方式一（已存在标签）：单击“标签”下方的文本框，在快捷键下拉列表中选择快捷键，然后在标签文本输入框中选择已有的标签名称，然后单击“确定”。

      方式二（新增标签）：在“标签”下方的文本框中，在快捷键下拉列表中选择快捷键，然后在标签文本输入框中输入新的标签名称，然后单击“确定”。

   b. 选中的音频将被自动移动至“已标注”页签，且在“未标注”页签中，标签的信息也将随着标注步骤进行更新，如增加的标签名称、各标签对应的音频数量。

说明

快捷键的使用说明：为标签指定快捷键后，当您选择一段音频后，在键盘中按快捷键，即可为此音频增加为此快捷键对应的标签。例如“aa”标签对应的快捷键是“1”，在数据标注过程中，选中1个或多个文件，按“1”，界面将提示是否需要将此文件标注为“aa”标签，单击确认即可完成标注。

快捷键对应的是标签，1个标签对应1个快捷键。不同的标签，不能指定为同一个快捷键。快捷键的使用，可以提升标注效率。

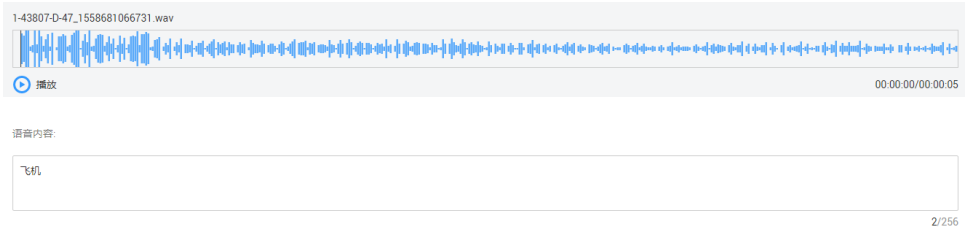
图 7-46 添加音频标签

标注音频（语音内容）

标注作业详情页中，展示了此数据集中“未标注”和“已标注”的音频，默认显示“未标注”的音频列表。

1. 在“未标注”页签左侧音频列表中，单击目标音频文件，在右侧的区域中出现音频，单击音频下方，即可进行音频播放。
2. 根据播放内容，在下方“语音内容”文本框中填写音频内容。
3. 输入内容后单击下方的“确认标注”按钮完成标注。音频将被自动移动至“已标注”页签。

图 7-47 语音内容音频标注



标注音频（语音分割）

标注作业详情页中，展示了此标注作业中“未标注”和“已标注”的音频，默认显示“未标注”的音频列表。

1. 在“未标注”页签左侧音频列表中，单击目标音频文件，在右侧的区域中出现音频，单击音频下方，即可进行音频播放。
2. 根据播放内容，选取合适的音频段，在下方“语音内容”文本框中填写音频标签和内容。

图 7-48 语音标签音频标注

3. 输入内容后单击下方的“确认标注”按钮完成标注。音频将被自动移动至“已标注”页签。

7.4.2.5 人工标注视频数据

由于模型训练过程需要大量有标签的视频数据，因此在模型训练之前需对没有标签的视频添加标签。通过ModelArts您可对视频添加标签，快速完成对视频的标注操作，也可以对已标注视频修改或删除标签进行重新标注。

说明

视频标注仅针对视频帧进行标注。

开始标注

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“数据准备> 数据标注”，进入“数据标注”管理页面。
2. 在标注作业列表右侧“所有类型”页签下拉选择标注类型，基于“标注类型”选择需要进行标注的标注作业，单击标注作业名称进入标注作业标注详情页。

图 7-49 下拉选择标注类型

3. 在标注作业标注详情中，展示此标注作业下全部数据。

标注视频

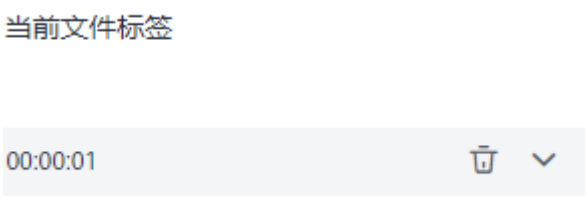
标注作业详情页中，展示了此数据集中“未标注”、“**已标注**”和“全部”的视频。

1. 在“未标注”页签左侧视频列表中，单击目标视频文件，打开标注页面。
2. 在标注页面中，播放视频，当视频播放至待标注时间时，单击进度条左侧的暂停按钮，将视频暂停至某一帧对应的画面。
3. 在上方区域选择标注框，默认为矩形框。使用鼠标在视频画面中框出目标，然后在弹出的添加标签文本框中，直接输入新的标签名，在文本框前面选中标签颜色，单击“添加”完成1个物体的标注。如果已存在标签，从下拉列表中选择已有的标签，然后单击“添加”完成标注。逐步此画面中所有物体所在位置，一帧对应的画面可添加多个标签。

   支持的标注框与“物体检测”类型一致，详细描述请参见物体检测章节的**表7-26**。
4. 上一帧对应的画面标注完成后，在进度条处单击播放按钮继续播放，在需要标注处暂停，然后重复执行步骤**3**完成整个视频的标注。

   单击界面右上角的“标注列表”，在“当前文件标签”的详情页将呈现当前视频带标注的时间点。

   图 7-50 当前文件标签信息

5. 单击页面左上角“返回数据标注预览”，页面将自动返回标注作业详情页面，同时，标注好的视频将呈现在“已标注”页签下。

常见问题

Q：视频数据集无法显示或者无法播放视频?

A：如果无法显示和播放视频，请检查视频格式类型，目前只支持MP4格式。

7.4.2.6 管理标注数据

同步新数据

ModelArts会自动将数据集中新增的数据同步至标注作业，包含数据及当前标注作业支持的标注信息。

为了快速获取数据集中最新数据，可在标注作业详情页的“全部”、“未标注”或“已标注”页签中，单击“同步新数据”，快速将数据集中的数据添加到标注作业中。

说明

问题现象：

将已标注好的数据上传至OBS，同步数据后，显示为未标注。

原因分析：

可能是OBS桶设置了自动加密导致此问题。

解决方法：

需要新建OBS桶重新上传数据，或者取消桶加密后，重新上传数据。

筛选数据

在标注作业详情页面，默认展示作业中全部未标注的数据，您可以在“全部”、“未标注”或“已标注”页签下，在筛选条件区域，单击 ，添加筛选条件，快速过滤出您想要查看的数据。

支持的筛选条件如下所示，您可以设置一个或多个选项进行筛选。

- 难例集：难例或非难例。
- 标签：您可以选择全部标签，或者基于您指定的标签，选中其中一个或多个。
- 文件名或目录：根据文件名称或者文件存储目录筛选。
- 标注人：选择执行标注操作的账号名称。
- 样本属性：表示自动分组生成的属性。只有启用了自动分组任务后才可使用此筛选条件。
- 数据属性：筛选数据的来源，选择“全部”或“推理”。

图 7-51 筛选条件

查看已标注图片

在标注任务详情页，单击“已标注”页签，您可以查看已完成标注的图片列表。图片缩略图下方默认呈现其对应的标签，您也可以勾选图片，在右侧的“选中文件标签”中了解当前图片的标签信息。

查看已标注文本

在数据集详情页，单击“已标注”页签，您可以查看已完成标注的文本列表。您也可以在右侧的“全部标签”中了解当前数据集支持的所有标签信息。

修改标注

当数据完成标注后，您还可以进入已标注页签，对已标注的数据进行修改。

- **基于图片修改**

  在标注作业详情页面，单击“已标注”页签，然后在图片列表中选中待修改的图片（选择一个或多个）。在右侧标签信息区域中对图片信息进行修改。

  修改标签：在“选中文件标签”区域中，单击操作列的编辑图标，然后在文本框中输入正确的标签名，然后单击确定图标完成修改。

  删除标签：在“选中文件标签”区域中，单击操作列的删除图标删除该标签。此操作仅删除选中图片中的标签。

  图 7-52 编辑标签

- **基于标签修改**
  - 在标注作业详情页，单击右侧区域的“标签管理”，显示全部标签列表。
    - 修改标签：单击操作列的“修改”，然后在弹出的对话框中输入修改后的标签名，然后单击“确定”完成修改。修改后，之前添加了此标签的图片，都将被标注为新的标签名称。
    - 删除标签：单击操作列“删除”，之前添加了此标签的图片，都将删除此标签。

    图 7-53 标签管理

图 7-54 全部标签的信息

- 单击标注作业操作列的“标签”，可跳转至标签管理页。
  - 单击操作列的“修改”，即可完成标签的修改。
  - 单击操作列的“删除”，即可删除该标签。

当数据完成标注后，您还可以进入已标注页签，对已标注的数据进行修改。

- **基于文本修改**

  在标注作业详情页，单击“已标注”页签，然后在文本列表中选中待修改的文本。

  在文本列表中，单击文本，当文本背景变为蓝色时，表示已选择。当文本有多个标签时，可以单击文本标签上方的×删除单个标签。

- **基于标签修改**

  在标注作业详情页，单击“已标注”页签，在图片列表右侧，显示全部标签的信息。

  - 批量修改：在“全部标签”区域中，单击操作列的编辑图标，然后在文本框中修改标签名称，选择标签颜色，单击“确定”完成修改。
  - 批量删除：在“全部标签”区域中，单击操作列的删除图标，在弹出对话框中，可选择“仅删除标签”或“删除标签及仅包含此标签的标注对象”，然后单击“确定”。

当数据完成标注后，您还可以进入“已标注”页签，对已标注的数据进行修改。

- **基于音频修改**

  在标注作业详情页面，单击“已标注”页签，然后在音频列表中选中待修改的音频（选择一个或多个）。在右侧标签信息区域中对标签进行修改。

  - 修改标签：在“选中文件标签”区域中，单击操作列的编辑图标，然后在文本框中输入正确的标签名，然后单击确定图标完成修改。
  - 删除标签：在“选中文件标签”区域中，单击操作列的删除图标删除该标签。

- **基于标签修改**

  在标注作业详情页面，单击“已标注”页签，在音频列表右侧，显示全部标签的信息。

图 7-55 全部标签信息

- 修改标签：单击操作列的编辑图标，然后在弹出的对话框中输入修改后的标签名，然后单击“确定”完成修改。修改后，之前添加了此标签的音频，都将被标注为新的标签名称。
- 删除标签：单击操作列的删除图标，在弹出的对话框中，根据提示框选择需要删除的对象，然后单击“确定”完成删除。

修改标注信息

当数据完成标注后，您还可以进入已标注页签，对已标注的数据进行修改。

在数据标注详情页面，单击“已标注”页签，然后在图片列表中选中待修改的图片，单击图片跳转到标注页面，在右侧标签信息区域中单击此图片已添加的标注信息。

- 修改标签：“标注”区域中，单击编辑图标，在弹出框中输入正确的标签名或标签颜色，然后单击  完成修改。也可以单击标签，在图片标注区域，调整标注框的位置和大小，完成调整后，单击其他标签即可保存修改。
- 删除标签：在“标注”区域中，单击删除图标即可删除此图片中的标签。图片的标签全部删除后，该图片会重新回到“未标注”页签。

标注信息修改后，单击页面左上角的“返回数据标注预览”离开标注页面，在弹出对话框中单击“确定”保存修改。

添加数据

除了同步数据集中的新数据外，您还可以在标注作业中，直接添加图片，用于数据标注。添加的数据将先导入至标注任务关联的数据集中，然后标注任务会自动同步数据集中最新的数据。

1. 在标注作业详情页面，单击“全部”、“已标注”或“未标注”页签，然后单击左上角“添加数据”，选择添加数据。

图 7-56 添加数据

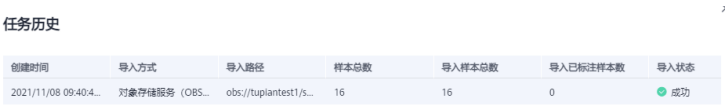
2. 在弹出的导入对话框中，选择数据来源和导入方式，选择导入的数据路径和数据标注状态。

图 7-57 添加图片

3. 在导入对话框中，单击“确定”，完成添加数据的操作。

   您添加的图片将自动呈现在“全部”的图片列表中，也可单击“添加数据>查看历史记录”，进入“任务历史”界面，可查看相应的导入历史。

图 7-58 查看历史数据



删除数据

通过数据删除操作，可将需要丢弃的数据快速删除。

在“全部”、“未标注”或“已标注”页面中，依次选中需要删除的内容，或者选择“选择当前页”选中该页面所有内容，然后单击“删除”。在弹出的对话框中，根据实际情况选择是否勾选“同时删除OBS源文件”，确认信息无误后，单击“确定”完成删除操作。

图 7-59 删除图片

其中，被选中的内容，其左上角将显示为勾选状态。如果当前页面无选中内容时，“删除”按钮为灰色，无法执行删除操作。

说明

如果勾选了“同时删除OBS源文件”，删除图片操作将删除对应OBS目录下存储的图片，此操作可能会影响已使用此源文件的其他数据集或数据集版本，有可能导致展示异常或训练/推理异常。删除后，数据将无法恢复，请谨慎操作。

7.4.3 通过智能标注方式标注数据

7.4.3.1 创建智能标注作业

除了人工标注外，ModelArts还提供了智能标注功能，快速完成数据标注，为您节省70%以上的标注时间。智能标注是指基于当前标注阶段的标签及图片学习训练，选中系统中已有的模型进行智能标注，快速完成剩余图片的标注操作。

说明

数据标注功能仅在以下Region支持：华北-北京四、华北-北京一、华东-上海一、华南-广州、西南-贵阳一、中国-香港、亚太-新加坡、亚太-曼谷、亚太-雅加达、拉美-圣地亚哥、拉美-圣保罗一、拉美-墨西哥城二。

背景信息

- 目前只有“图像分类”和“物体检测”类型的标注作业支持智能标注功能。
- 启动智能标注时，需标注作业存在至少2种标签，且每种标签已标注的图片不少于5张。
- 启动智能标注时，必须存在未标注图片。
- 启动智能标注前，保证当前系统中不存在正在进行中的智能标注任务。

- 检查用于标注的图片数据，确保您的图片数据中，不存在RGBA四通道图片。如果存在四通道图片，智能标注任务将运行失败，因此，请从数据集中删除四通道图片后，再启动智能标注。

启动智能标注作业

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“数据准备 > 数据标注”，进入“数据标注”管理页面。
2. 在标注作业列表中，选择“物体检测”或“图像分类”类型的标注作业，单击操作列的“智能标注”启动智能标注作业。
3. 在弹出的“启动智能标注”对话框中，选择智能标注类型，可选“主动学习”或者“预标注”，详见**表7-30**和**表7-31**。

表 7-30 主动学习

参数	说明
智能标注类型	“主动学习”。“主动学习”表示系统将自动使用半监督学习、难例筛选等多种手段进行智能标注，降低人工标注量，帮助用户找到难例。
算法类型	针对“图像分类”类型的数据集，您需要选择以下参数。 “快速型”：仅使用已标注的样本进行训练。 “精准型”：会额外使用未标注的样本做半监督训练，使得模型精度更高。
计算节点规格	即智能标注任务使用的资源规格。 **说明** 智能标注创建时免费，但OBS存储会按需收费，请参考**计费详情**。为保证您的资源不浪费，标注作业与后续任务完成后，请及时清理您的OBS桶。
计算节点个数	默认为1，表示单机模式。目前仅支持此参数值。

表 7-31 预标注

参数	说明
智能标注类型	“预标注”。“预标注”表示选择用户模型管理里面的模型，选择模型时需要注意模型类型和数据集的标注类型相匹配。预标注结束后，如果标注结果符合平台定义的标准标注格式，系统将进行难例筛选，该步骤不影响预标注结果。
选择模型及版本	• “我的模型”。您可以根据实际需求选择您的模型。您需要在目标模型的左侧单击下拉三角标，选择合适的版本。您的模型导入参见**创建模型**。
计算节点规格	在下拉框中，您可以选择目前ModelArts支持的节点规格选项。
计算节点个数	默认为1。您可以根据您的实际情况选择，最大为5。

说明

针对“物体检测”类型的标注作业，选择“主动学习”时，只支持识别和标注矩形框。

图 7-60 启动智能标注（图像分类）

图 7-61 启动智能标注（物体检测）

图 7-62 启动智能标注（预标注）

4. 完成参数设置后，单击“提交”，即可启动智能标注。
5. 在标注作业列表中，单击标注作业名称进入“标注作业详情”页。
6. 在“数据集概览页标注作业详情页”，选择“标注”页签，单击“待确认”页签，即可查看智能标注进度。

   您也可以在该页签，“启动智能标注”或者查看“智能标注历史”

图 7-63 标注进度

说明

当系统中智能标注任务过多时，因免费资源有限，可能会出现排队的情况，导致作业一直处于“标注中”的状态。请您耐心等待，为确保您的标注作业能顺利进行，建议您避开高峰期使用。

7. 智能标注完成后，“待确认”页面将呈现所有标注后的图片列表。
   - 图像分类标注作业

     在“待确认”页面查看标签是否准确，勾选标注准确的图片，然后单击“确认”完成智能标注结果的确认。确认完成后的图片将被归类至“已标注”页面下。

针对标为“难例”的图片，您可以根据实际情况判断，手工修正标签。详细操作及示例请参见**•针对“图像分类”数据集**。

- 物体检测标注作业

  在“待确认”页面，单击图片查看标注详情，查看标签及目标框是否准确，针对标注准确的图片单击“确认标注”完成智能标注结果的确认。确认完成后的图片将被归类至“已标注”页面下。

  针对标为“难例”的图片，您可以根据实际情况判断，手工修正标签或目标框。详细操作及示例请参见**•针对“物体检测”数据集**。

相关问题

- **智能标注失败，如何处理?**

  当前智能标注为免费使用阶段，当系统的标注任务过多时，因免费资源有限，导致任务失败，请您重新创建智能标注任务或建议您避开高峰期使用。

- **智能标注时间过长，如何处理?**

  当前智能标注为免费使用阶段，当系统的标注任务过多时，因免费资源有限，需要排队，您的标注任务会长时间处于“标注中”状态，请您耐心等待。或建议您避开高峰期使用。

7.4.3.2 确认智能标注作业的数据难例

在数据量很大的标注任务中，标注初期由于已标注图片不足，智能标注的结果无法直接用于训练。如果对所有的未标注数据一一进行调整确认仍然需要较大的人力和时间成本。为了更快地完成标注任务，在对未标注数据进行智能标注的任务中，ModelArts嵌入了自动难例发现功能。该功能会对剩余未标注图片的标注优先级给出建议。因为标注优先级高的图片的智能标注结果未达到预期，所以称之为难例。

ModelArts平台提供的自动难例发现功能，在智能标注以及数据采集筛选过程中，将自动标注出难例，建议对难例数据进一步确认标注，然后将其加入训练数据集中，使用此数据集训练模型，可得到精度更高的模型。首先，针对智能标注和采集筛选任务，难例的发现操作是系统自动执行的，无需人工介入，仅需针对标注后的数据进行确认和修改即可，提升数据管理和标注效率。其次，您可以基于难例的情况，补充类似数据，提升数据集的丰富性，进一步提升模型训练的精度。

在数据集管理中，对难例的管理有如下场景。

- **智能标注后，确认难例**
- **将数据集中的数据标注为难例**

说明

目前只有“图像分类”和“物体检测”类型的数据集支持难例发现功能。

智能标注后，确认难例

“智能标注”任务执行过程中，ModelArts将自动识别难例，并完成标注。当智能标注结束后，难例标注结果将呈现在“待确认”页签，建议您对难例数据进行人工修正，然后确认标注。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“数据准备>数据标注”，单击“我创建的”页签可查看所有作业列表。
2. 在标注作业列表中，选择“物体检测”或“图像分类”类型的标注作业，单击标注作业名称进入“标注作业详情”。

3. 在“标注作业详情页”，选择“待确认”页签，查看并确认难例。

   说明

   只有当智能标注任务完成后，待确认页签才会显示标注数据。否则，此页签内容为空。智能标注操作请参见**创建智能标注作业**。

   - 针对“物体检测”标注作业

     在“待确认”页签中，单击图片展开标注详情，查看图片数据的标注情况，如标签是否准确、目标框位置添加是否准确。如果智能标注结果不准确，建议手工调整标签或目标框，然后单击“确认标注”。完成确认后，重新标注的数据将呈现在“已标注”页签下。

   - 针对“图像分类”标注作业

     在“待确认”页签中，查看标注难例的图片，其添加的标签是否准确。勾选标注不准确的图片，删除错误标签，然后在右侧“标签名”处添加准确标签。单击“确认”，勾选的图片及其标注情况，将呈现在“已标注”页签下。

     选中的图片为标注错误图片，在右侧删除错误标签，然后在标签名处添加“狗”的标签，然后单击“确认”，完成难例确认。

将数据集中的数据标注为难例

针对标注作业中，已标注或未标注数据，也可以将图片数据标注为难例。标注为难例的数据，对后续模型训练中，通过内置规则提升模型精度。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“数据准备>数据标注”，单击“我创建的”页签可查看所有作业列表。
2. 在标注作业列表中，选择“物体检测”或“图像分类”类型的标注作业，单击数据集名称进入“标注作业详情页”。
3. 在标注作业详情页，选择“标注”页签”，单击“已标注”、“未标注”或“全部”页签，勾选需标注为难例的图片，然后单击“难例批处理 > 确认为难例”。完成标注后，图片预览时，其右上角将显示为“难例”。

7.4.3.3 使用自动分组智能标注作业

为了提升智能标注算法精度，可以均衡标注多个类别，有助于提升智能标注算法精度。ModelArts内置了分组算法，您可以针对您选中的数据，执行自动分组，提升您的数据标注效率。

自动分组可以理解为数据标注的预处理，先使用聚类算法对未标注图片进行聚类，再根据聚类结果进行处理，可以分组打标或者清洗图片。

例如，用户通过搜索引擎搜索XX，将相关图片下载并上传到数据集，然后再使用自动分组，可以将XX图片分类，比如论文、宣传海报、确认为XX的图片、其他。用户可以根据分组结果，快速剔除掉不想要的，或者将某一类直接全选后添加标签。

说明

目前只有“图像分类”、“物体检测”和“图像分割”类型的数据集支持自动分组功能。

启动自动分组任务

1. 登录**ModelArts管理控制台**，在左侧菜单栏中选择“数据准备>数据标注”，进入“数据标注”管理页面。

2. 在标注作业列表中，选择“物体检测”或“图像分类”类型的标注作业，单击标注作业名称进入“标注作业详情页”。
3. 在数据集详情页的“全部”页签中，单击“自动分组 > 启动任务”。

   说明

   只能在“全部”页签下启动自动分组任务或查看任务历史。
4. 在弹出的“自动分组”对话框中，填写参数信息，然后单击“确定”。
   - “分组数”：填写2~200之间的整数，指将图片分为多少组。
   - “结果处理方式”：“更新属性到当前样本中”，或者“保存到对象存储服务（OBS）”。
   - “属性名称”：当选择“更新属性到当前样本中”时，需输入一个属性名称。
   - “结果存储目录”：当选择“保存到对象存储服务（OBS）”时，需指定一个用于存储的OBS路径。
   - “高级特征选项”：启用此功能后，可选择“清晰度”、“亮度”、“图像色彩”等维度为自动分组功能增加选项，使得分组着重于图片亮度、色彩和清晰度等特征进行分组。支持多选。

   图 7-64 自动分组

5. 启动任务提交成功后，界面右上角显示此任务的进度。等待任务执行完成后，您可以查看自动分组任务的历史记录，了解任务状态。

查看自动分组结果

在数据集详情页面的“全部”页签中，展开“筛选条件”，将“样本属性”设置为自动分组任务中的“属性名称”，并通过设置样本属性值，筛选出分组结果。

图 7-65 查看自动分组结果

查看自动分组的历史任务

在数据集详情页面的“全部”页签中，单击“自动分组 > 任务历史”。在弹出的“任务历史”对话框中，展示当前数据集之前执行的自动分组任务的基本信息。

图 7-66 自动分组任务历史

7.4.4 通过团队标注方式标注数据

7.4.4.1 团队标注使用说明

数据标注任务中，一般由一个人完成，但是针对数据集较大时，需要多人协助完成。ModelArts提供了团队标注功能，可以由多人组成一个标注团队，针对同一个数据集进行标注管理。

说明

团队标注功能仅在以下Region支持：华北-北京四、西南-贵阳一、中国-香港、亚太-新加坡、亚太-曼谷。

团队标注功能当前仅支持“图像分类”、“物体检测”、“文本分类”、“命名实体”、“文本三元组”、“语音分割”类型的数据集。

针对启用团队标注功能的数据标注任务，支持创建团队标注任务，将标注任务指派给不同的团队，由多人完成标注任务。同时，在成员进行数据标注过程中，支持发起验收、继续验收以及查看验收报告等功能。

团队标注功能是以团队为单位进行管理，数据集启用团队标注功能时，必须指定一个团队。一个团队可以添加多个成员。

- 一个账号最多可添加10个团队。
- 如果数据集需要启用团队标注功能，当前账号至少拥有一个团队。如果没有，请执行**添加团队**操作添加。

7.4.4.2 创建和管理团队

团队标注功能是以团队为单位进行管理，数据集启用团队标注功能时，必须指定一个团队。一个团队可以添加多个成员。新添加的团队，其成员列表为空。您需要根据实际情况添加即将参与标注任务的成员信息。

- 一个账号最多可添加10个团队。一个团队最多支持添加100个成员，当超过100时，建议分为多个团队进行管理。
- 如果数据集需要启用团队标注功能，当前账号至少拥有一个团队。如果没有，请执行**添加团队**操作添加。

添加团队

1. 在ModelArts管理控制台左侧导航栏中，选择“数据准备>标注团队”，进入“标注团队”管理页面。
2. 在“标注团队”管理页面，单击“添加团队”。
3. 在弹出的“添加团队”对话框中，填写团队“名称”和“描述”，然后单击“确定”。完成标注团队的添加。

   团队添加完成后，“标注团队”管理页面呈现新添加的团队，在页面右侧区域，可以查看团队详情。新添加的团队，其成员列表为空，请参考**添加成员**操作，为您的团队添加成员。

添加成员

1. 在ModelArts管理控制台左侧导航栏中，选择“数据准备>标注团队”，进入“标注团队”管理页面。
2. 在“标注团队”管理页面，从左侧团队列表中选择一个团队，单击团队，其右侧区域将呈现“团队详情”。
3. 在“团队详情”区域，单击“添加成员”。
4. 邮箱作为团队管理中的唯一标识，不同成员不能使用同一个邮箱。您填写的邮箱地址将被记录并保存在ModelArts中，仅用于ModelArts团队标注功能，当成员删除后，其填写的邮箱信息也将被一并删除。

   其中，“角色”支持“Labeler”、“Reviewer”和“Team Manager”，“Team Manager”只能设置为一个人。

   需要注意的是：目前不支持从标注任务中删除labeler。labeler的标注必须通过审核后，才能同步到最终结果，不支持单独分离操作。

   成员添加完成后，团队详情区域中将呈现此成员的信息。

7.4.4.3 创建团队标注任务

如果您在创建标注作业时，即启用团队标注，且指派了某一团队负责标注，系统将默认基于此团队创建一个标注任务。您可以在创建数据标注任务后，在“我创建的”页面查看此任务。

您还可以重新创建一个团队标注任务，指派给同一团队的不同成员，或者指派给其他标注团队。

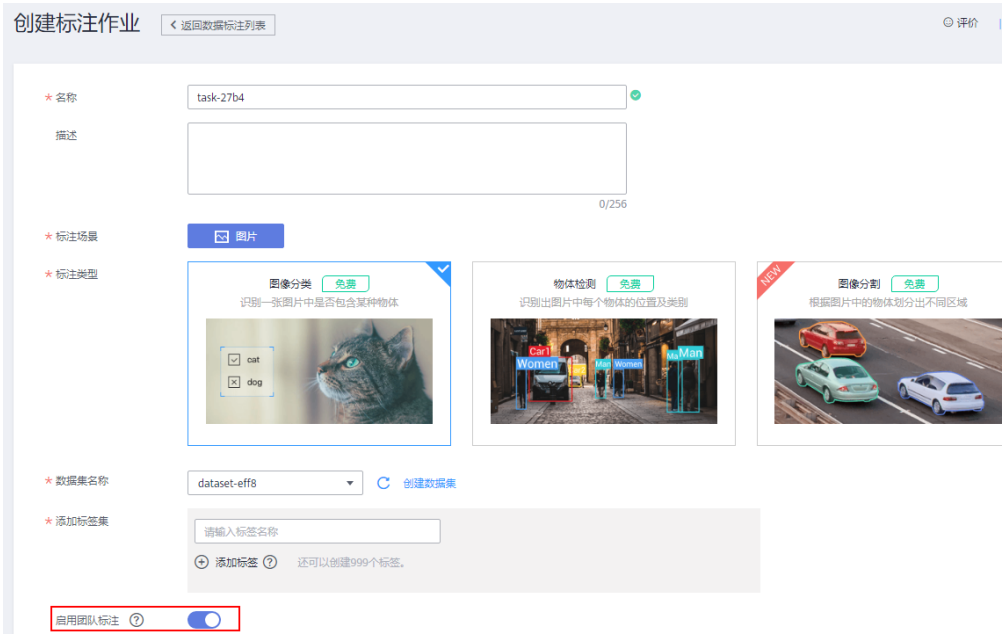
团队标注作业的创建方式

- 从控制台的“数据准备 >数据标注”页面进入，创建标注作业时，打开“启用团队标注”开关，同时指定一个标注团队，或者指定标注管理员。

  图 7-67 创建团队标注作业

- 从控制台的“资产管理 >数据集”进入数据集页面，在需要进行团队标注的数据集的操作列，单击“标注”，进入创建标注作业页面，打开“启用团队标注”开关。对于同一个数据集，可以创建多个团队标注任务。

图 7-68 打开启用团队标注



说明

- 只有当创建团队标注任务时，标注人员才会收到邮件。创建标注团队及添加标注团队的成员并不会发送邮件。此外，当所有样本都是已标注状态时，创建团队标注任务也不会收到邮件。
- 标注任务创建完成后，会将所有未标注状态的样本分配给标注人员。分配采用随机均分的策略，不支持重复分配。

创建团队标注任务

同一个数据集，支持创建多个团队标注作业，指派给同一团队的不同成员，或者指派给其他标注团队。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理 >数据集”，打开数据集列表。
2. 在数据集列表中，选择支持团队标注的数据集，单击数据集名称进入数据集概览页。
3. 在数据集概览页页面，右侧的“标注任务”区域，可查看此数据集已有的标注任务。单击“新建标注任务”开始创建新任务。

图 7-69 标注任务

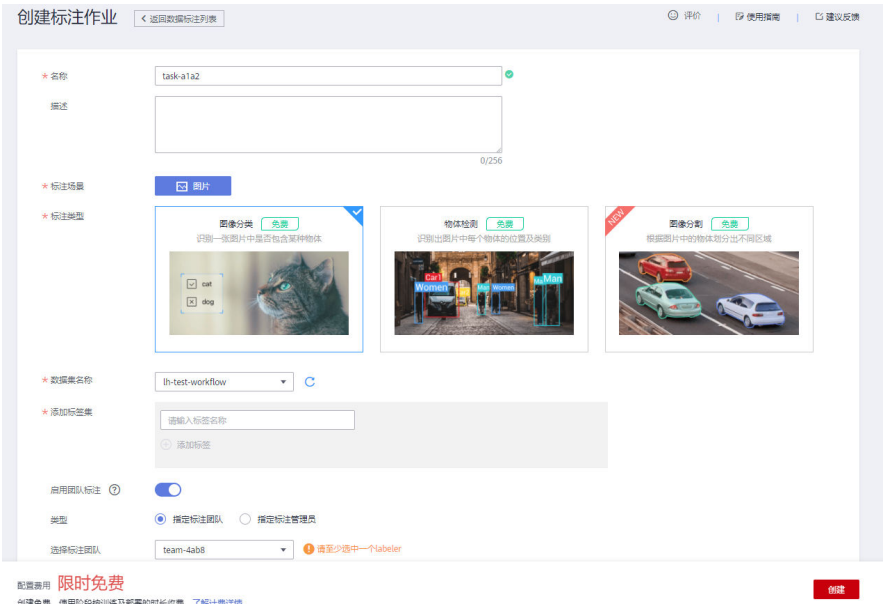
或者也可以从“数据准备 >数据标注”页面进入，单击“创建标注作业”进入创建标注作业页面。

4. 在弹出的“创建标注作业”页面中，填写相关参数，然后单击“确定”，完成任务创建。
   - “名称”：设置此任务的名称。
   - **“标注场景”**：选择标注作业的任务类型。
   - “标签集”：展示当前数据集已有的标签及标签属性。
   - **“启用团队标注”**：选择打开，并配置如下团队标注相关参数。
     - “类型”：设置任务类型，支持“指定标注团队”或“指定标注管理员”。
     - “选择标注团队”：任务类型设置为“指定标注团队”，需在此参数中指定一个团队，同时勾选此团队中某几个成员负责标注。下拉框中将罗列当前账号下创建的标注团队及其成员。
     - “选择标注接口人”：任务类型设置为“指定标注管理员”，需在所有团队的“Team Manager”中选择一人作为管理员。
     - “自动将新增图片同步给标注团队”：根据需要选择是否将任务中新增的数据自动同步给标注人员。
     - “团队标注的图片自动加载智能标注结果”：根据需要选择是否将任务中智能标注待确认的结果自动同步给标注人员。

       说明

       团队标注加载智能标注结果的处理步骤：
       - 如果类型选择"指定标注团队"，需要先创建团队标注任务，然后执行智能标注任务。
       - 如果类型选择"指定标注管理员"，在“我参与的”页签下选择团队标注任务，单击"分配任务"。

图 7-70 创建团队标注任务



任务创建完成后，您可以在“我创建的”页签下看到新建的任务。

登录 ModelArts-Console

在ModelArts中，一般用户使用数据标注功能，直接是在“数据标注”模块操作，此模块包含数据标注、数据导入导出、智能标注、团队标注和管理等。团队标注任务创建成功后，团队成员登录ModelArts-Console查看相关任务。

1. 团队标注任务创建成功后，团队成员收到标注任务的邮件。
2. 单击任务邮件中的标注任务地址，跳转至ModelArts控制台的“数据准备>数据标注 > 我参与的”页面。如果未登录控制台，请先登录。
3. 在“我参与的”页签下，可查看您的标注任务。

图 7-71 标注任务

数据标注访问地址可以查看如下表格获取，另外，如果团队成员绑定了邮箱，可以收到任务通知邮件，成员也可以通过邮件中给出的地址访问ModelArts-Console标注地址。

表 7-32 ModelArts 标注地址

局点	ModelArts-Console访问地址
中国-香港	https://console-intl.huaweicloud.com/modelarts/?region=ap-southeast-1&locale=zh-cn#/dataLabel?tabActive=labelConsole
亚太-新加坡	https://console-intl.huaweicloud.com/modelarts/?region=ap-southeast-3&locale=zh-cn#/dataLabel?tabActive=labelConsole
亚太-曼谷	https://console-intl.huaweicloud.com/modelarts/?region=ap-southeast-2&locale=zh-cn#/dataLabel?tabActive=labelConsole

登录后，仅显示当前用户（此邮箱用户）相关的团队标注任务及其相关数据。

启动团队标注任务

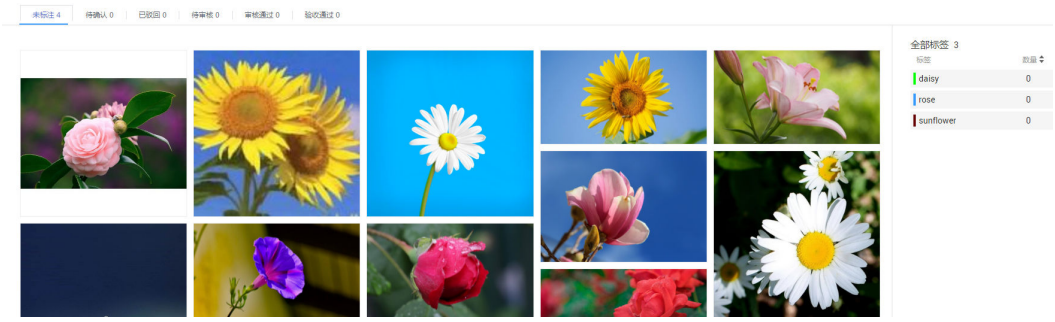
登录到console标注页面后在“我参与的”页签下，可查看到分配的标注任务，单击任务名称，可进入标注页面。不同类型的标注作业，标注方式不同，详细请参见：

- 标注图片（图像分类）
- 标注图片（物体检测）
- 标注文本（文本分类）
- 标注文本（命名实体）
- 标注文本（文本三元组）
- 标注音频（语音分割）

在标注页面中，每个成员可查看“未标注”、“待确认”、“已驳回”、“待审核”、“审核通过”、“验收通过”的图片信息。请及时关注管理员驳回以及待修正的图片。

当团队标注任务中，分配了Reviewer角色，则需要对标注结果进行审核，审核完成后，再提交给管理员验收。

图 7-72 成员标注平台



7.4.4.4 审核并验收团队标注任务结果

审核团队标注任务结果

团队标注成员完成后，团队审核者可以对标注结果进行审核。

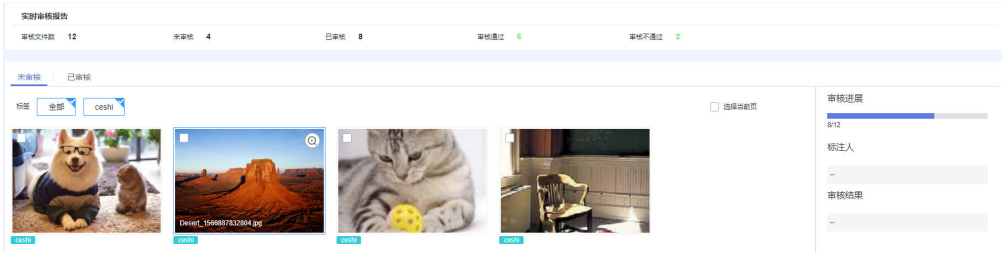
1. 登录ModelArts管理控制台，左侧菜单栏选择“数据准备>数据标注”，在数据标注页面选择“我参与的”，在任务列表“操作”列单击“审核”，发起审核。

   图 7-73 发起审核

2. 在审核页面中，审核人员可以查看“未审核”、“已审核”、“审核通过”、“审核不通过”的样本。

图 7-74 标注结果审核



3. 审核人员可以在审核页面的右侧选择“审核结果”（“通过”或“不通过”）。

   当选择审核结果为“通过”时，需设置“验收评分”（分“A”、“B”、“C”、“D”四个选项，“A”表示最高分）。当选择审核结果为“不通过”时，可以在文本框中写明驳回原因。

任务验收（管理员）

- **发起验收**

  当团队的成员已完成数据标注，标注作业的创建者可发起验收，对标注结果进行抽验。只有当标注成员存在标注完成的数据时，才可以发起验收，否则发起验收按钮为灰色。

  a. 录ModelArts管理控制台，在左侧菜单栏中选择“数据准备 >数据标注”，打开数据标注管理页。

  b. 选择“我参与的”，选择团队标注作业，单击作业名称进入“标注作业详情页”，单击右上角“团队标注>验收”，发起验收。

  图 7-75 发起验收

  c. 在弹出的对话框中，设置“抽样策略”，可设置为“按百分比”，也可以设置为“按数量”。设置好参数值后，单击“确定”启动验收。

  “按百分比”：按待验收图片总数的一定比例进行抽样验收。

  “按数量”：按一定数量进行抽样验收。

图 7-76 发起验收

d. 验收启动后，界面将展示实时验收报告，您可以在右侧选择“验收结果”（“通过”或“不通过”）。

当选择验收结果为“通过”时，需设置“验收评分”（分“A”、“B”、“C”、“D”四个选项，“A”表示最高分），如图7-78所示。当选择验收结果为“不通过”时，可以在文本框中写明驳回原因，如图7-79所示。

图 7-77 查看实时验收报告

图 7-78 设置验收结果为“通过”

图 7-79 设置验收结果为“不通过”

- **继续验收**

  针对未完成验收的任务，可以继续验收。针对未发起过验收流程的任务，不支持“继续验收”，按钮为灰色。

  在“任务统计>标注进展”页签中，针对需继续验收的任务，单击“继续验收”。系统直接进入“实时验收报告”页面，您可以继续验收未验收的图片，设置其“验收结果”。

- **完成验收**

  继续验收完成后，单击右上角“完成验收”在完成验收窗口，您可以查看本标注作业的验收情况，如抽样文件数等，同时设置如下参数，然后进行验收。只有完成验收，标注信息才会同步到标注作业的已标注页面中。

  一旦标注数据完成验收，团队成员无法再修改标注信息，只有数据集创建者可修改。

表 7-33 完成验收的参数设置

参数	说明
对已标注数据修改	• 不覆盖：针对同一个数据，不使用当前团队标注的结果覆盖已有数据。 • 覆盖：针对同一个数据，使用当前团队标注的结果覆盖已有数据。覆盖后无法恢复，请谨慎操作。

参数	说明
验收范围	• 全部通过：被驳回的样本，也会通过。 • 全部驳回：已经通过的样本，需要重新标注，下次验收时重新进行审核。 • 剩余全部通过：已经驳回的会驳回，其余会自动验收通过。 • 剩余全部驳回：样本抽中的通过的，不需要标注了，未通过和样本未抽中的需要重新标注验收。

图 7-80 完成验收

查看验收报告

针对进行中或已完成的标注任务，都可以查看其验收报告。登录管理控制台，选择“数据准备>数据标注”，在数据标注页选择“我创建的”，并单击某条团队标注的任务名称，进入标注详情页。在右上角单击“验收报告”，即可在弹出的“验收报告”对话框中查看详情。

图 7-81 查看验收报告

删除标注任务

验收结束后，针对不再使用的标注任务，您可单击任务所在行的删除。任务删除后，未验收的标注详情将丢失，请谨慎操作。但是数据集中的原始数据以及完成验收的标注数据仍然存储在对应的OBS桶中。

7.4.4.5 管理团队和团队成员

修改成员信息

团队中的成员，当其信息发生变化时，可以编辑其基本情况。

1. 在“团队详情”区域，选择需修改的成员。
2. 在成员所在行的“操作”列，单击“修改”。在弹出的对话框中，修改其“描述”或“角色”。

   成员的“邮箱”无法修改，如果需要修改邮箱地址，建议先删除此成员，然后再基于新的邮箱地址添加新成员。

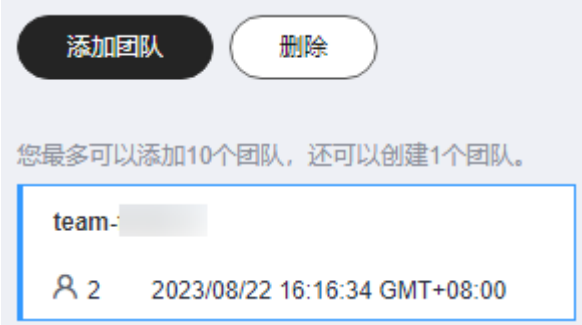
   “角色”支持“Labeler”、“Reviewer”和“Team Manager”，“Team Manager”只能设置为一个人。

删除团队

当已有的团队不再使用，您可以执行删除操作。

在“标注团队”管理页面中，选中需删除的团队，然后单击“删除”。在弹出的对话框中，确认信息无误后，单击“确定”完成团队删除。

图 7-82 删除团队



删除成员

- **删除单个成员**

  在“团队详情”区域，选择需要删除的成员， 单击“操作”列的“删除”。在弹出的对话框中，确认信息无误后，单击“确定”完成删除操作。

- **批量删除**

  在“团队详情”区域，勾选需删除的成员，然后单击“删除”。在弹出的对话框中，确认信息无误后，单击“确定”完成多个成员的删除操作。

图 7-83 批量删除

标注人员管理

如果您创建的标注作业，开启了团队标注，“标注人员管理”页面中可查看团队标注作业的标注详情。添加、修改或删除标注成员。

1. 登录“数据准备>数据标注”，在“我创建的”页签下可查看所有的标注作业列表。
2. 在作业列表的“名称”列，根据标注作业名称找到对应的团队标注作业。（团队标注作业的名称后带有 标识。）
3. 单击作业操作列的“更多>标注人员管理”。或单击作业名称进入作业详情，继续单击右上角“团队标注>标注人员管理”，进入成员管理页面。

图 7-84 进入标注人员管理页（1）

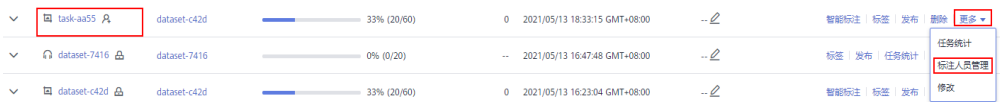


图 7-85 进入标注人员管理页（2）

- 添加成员：
  单击页面“添加成员”，选择成员名称，单击确定。
  在操作列，选择“发送邮件”，可将该标注任务以邮件的方式发送至该标注成员。
- 修改成员信息：
  单击操作列的“修改”，可修改该成员的角色。
- 删除标注成员：
  单击操作列的“删除”可删除该标注成员的所有信息。

7.4.5 管理标注作业

查看标注作业

在ModelArts数据标注页面可查看用户自己创建的标注作业。

1. 登录ModelArts管理控制台，在左侧菜单栏选择“数据准备>数据标注”，进入数据标注页面。
2. 在“我创建的”页签，可查看自己创建的标注作业。用户可查看自己创建的标注作业的相关信息。

图 7-86 我创建的

3. 在“我参与的”页签，可查看参与过标注的标注作业。用户可查看标注作业详细信息，包括标注团队的成员、标注进展等。

复制标注作业

1. 登录ModelArts管理控制台，在左侧菜单栏选择“数据准备>数据标注”，进入数据标注页面。
2. 在数据标注列表页，“我创建的”页签下，选择需要复制的标注任务。

3. 单击作业操作列的“更多>复制”。
4. 在标注任务复制的弹窗中，填写作业描述，作业名称task-xxxx-copy-***xxxx***，其中***xxxx***为系统生成的随机码，用来区分新作业与被复制作业。也可以修改新生成的作业名称。单击“确定”。

5. 复制完成后，在标注作业列表页即可查询新的标注任务，复制标注作业信息包含标注任务的样本、标签、团队标注信息。

通过条件筛选数据

在数据概览页中，默认展示数据集的概览情况。在界面右上方，单击“开始标注”，进入数据集的详细数据页面，默认展示数据集中全部数据。在“全部”、“未标注”或“已标注”页签下，您可以在筛选条件区域，添加筛选条件，快速过滤出您想要查看的数据。

支持的筛选条件如下所示，您可以设置一个或多个选项进行筛选。

- 难例集：难例或非难例。
- 标签：您可以选择全部标签，或者基于您指定的标签，选中其中一个或多个。
- 样本创建时间：1个月内、1天内或自定义，如果选择自定义，可以在时间框中指定明确时间范围。
- 文件名或目录：根据文件名称或者文件存储目录筛选。
- 标注人：选择执行标注操作的账号名称。
- 样本属性：表示自动分组生成的属性。只有启用了**自动分组**任务后才可使用此筛选条件。
- 数据属性：暂不支持。

图 7-87 筛选条件

7.5 发布 ModelArts 数据集中的数据版本

ModelArts在数据准备过程中，针对同一数据源的数据，对不同时间处理或标注后的数据，按照版本进行区分方便后续模型构建和开发时选择对应的数据集版本进行使用。

关于数据集版本

- 针对刚创建的数据集（未发布前），无数据集版本信息，必须执行发布操作后，才能应用于模型开发或训练。
- 数据集版本，默认按V001、V002递增规则进行命名，您也可以在发布时自定义设置。
- 您可以将任意一个版本设置为当前目录，即表示数据集列表中进入的数据集详情，为此版本的数据集标注信息。
- 针对每一个数据集版本，您可以通过“存储路径”参数，获得此版本对应的Manifest文件格式的数据集。可用于导入数据或难例筛选操作。
- 表格数据集暂不支持切换版本。

发布数据集版本

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理> 数据集”，进入“数据集”管理页面
2. 在数据集列表中，单击操作列的“发布”。或者，您可以单击数据集名称，进入数据集“概览”页，在页面右上角单击“发布”。
3. 在“发布新版本”弹出框中，填写发布数据集的相关参数，然后单击“确定”。

表 7-34 发布数据集的参数说明

参数	描述
“版本名称”	默认按V001、V002递增规则进行命名，您也可以自定义版本名称。版本名称只能包含字母、数字、中划线或下划线。

参数	描述
“版本格式”	仅“表格”类型数据集支持设置版本格式，支持“CSV”和“CarbonData”两种。 **说明** 如果导出的CSV文件中存在以“=”“+”“-”和“@”开头的命令时，为了安全考虑，ModelArts会自动加上Tab键，并对双引号进行转义处理。
“数据切分”	仅“图像分类”、“物体检测”、“文本分类”和“声音分类”类型数据集支持进行数据切分功能。 默认不启用。启用后，需设置对应的训练验证比例。 输入“训练集比例”，数值只能是0~1区间内的数。设置好“训练集比例”后，“验证集比例”自动填充。“训练集比例”加“验证集比例”等于1。 **说明** 为确保训练模型的精度，建议将训练集比例设置为0.8或者0.9。 “训练集比例”即用于训练模型的样本数据比例；“验证集比例”即用于验证模型的样本数据比例。“训练验证比例”会影响训练模板的性能。
“描述”	针对当前发布的数据集版本的描述信息。
“开启难例属性”	仅“图像分类”和“物体检测”类型数据集支持难例属性。 默认不开启。启用后，会将此数据集的难例属性等信息写入对应的Manifest文件中。

数据集版本文件目录结构

由于数据集是基于OBS目录管理的，发布为新版本后，对应的数据集输出位置，也将基于新版本生成目录。

以图像分类为例，数据集发布后，对应OBS路径下生成，其相关文件的目录如下所示。

```
|-- user-specified-output-path
    |-- DatasetName-datasetId
        |-- annotation
            |-- VersionMame1
                |-- VersionMame1.manifest
            |-- VersionMame2
                ...
            |-- ...
```

以物体检测为例，如果数据集导入的是Manifest文件，在数据集发布后，其相关文件的目录结构如下。

```
|-- user-specified-output-path
    |-- DatasetName-datasetId
        |-- annotation
            |-- VersionMame1
                |-- VersionMame1.manifest
                |-- annotation
                    |-- file1.xml
            |-- VersionMame2
                ...
            |-- ...
```

以视频标注为例，在数据集发布后，标注结果将标注结果文件（XML）存放在数据集输出目录下。

```
|-- user-specified-output-path
    |-- DatasetName-datasetId
        |-- annotation
            |-- VersionMame1
                |-- VersionMame1.manifest
                |-- annotations
                  |-- images
                      |-- videoName1
                          |-- videoName1.timestamp.xml
                       |-- videoName2
                          |-- videoName2.timestamp.xml
          |-- VersionMame2
              ...
          |-- ...
```

视频标注的关键帧存在数据集的输入目录下。

```
|-- user-specified-input-path
    |-- images
      |-- videoName1
          |-- videoName1.timestamp.jpg
       |-- videoName2
          |-- videoName2.timestamp.jpg
```

查看数据集演进过程

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理>数据集”，进入“数据集”管理页面。
2. 在数据集列表中，单击操作列的“更多 > 版本管理”，进入数据集“版本管理”页面。

   您可以查看数据集的基本信息，并在左侧查看版本演进信息及其发布时间。

设置当前版本

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理>数据集”，进入“数据集”管理页面。
2. 在数据集列表中，单击操作列的“更多 > 版本管理”，进入数据集“版本管理”页面。
3. 在“版本管理”页面中，选择对应的数据集版本，在数据集版本基本信息区域，单击“设置为当前版本”。设置完成后，版本名称右侧将显示为“当前版本”。

图 7-88 设置当前版本

说明

只有状态为“正常”的版本，才能被设置为当前版本。

删除数据集版本

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理>数据集”，进入“数据集”管理页面。
2. 在数据集列表中，单击操作列的“更多 > 版本管理”，进入数据集“版本管理”页面。
3. 选择需删除的版本所在行，单击操作列的“删除”。在弹出的对话框中确认信息，然后单击“确定”完成删除操作。

说明

删除数据集版本不会删除原始数据，数据及其标注信息仍存在于对应的OBS目录下。但是，执行删除操作后，无法在ModelArts管理控制台清晰的管理数据集版本，请谨慎操作。

7.6 分析 ModelArts 数据集中的数据特征

基于图片或目标框对图片的各项特征，如模糊度、亮度进行分析，并绘制可视化曲线，帮助处理数据集。

您还可以选择数据集的多个版本，查看其可视化曲线，进行对比分析。

背景信息

- 只有“图片”的数据集，且版本标注类型为“物体检测”和“图像分类”的数据集版本支持数据特征分析。
- 只有发布后的数据集支持数据特征分析。发布后的Default格式数据集版本支持数据特征分析。
- 数据特征分析的数据范围，不同类型的数据集，选取范围不同：
  - 对于标注任务类型为“物体检测”的数据集版本，当已标注样本数为0时，发布版本后，数据特征页签版本置灰不可选，无法显示数据特征。否则，显示已标注的图片的数据特征。

  - 对于标注任务类型为“图像分类”的数据集版本，当已标注样本数为0时，发布版本后，数据特征页签版本置灰不可选，无法显示数据特征。否则，显示全部的图片的数据特征。
- 数据集中的图片数量要达到一定量级才会具有意义，一般来说，需要有大约1000+的图片。
- “图像分类”支持分析指标有：“分辨率”、“图片高宽比”、“图片亮度”、“图片饱和度”、“清晰度”和“图像色彩的丰富程度”。“物体检测”支持所有的分析指标。目前ModelArts支持的所有分析指标请参见**支持分析指标及其说明**。

数据特征分析

1. 登录**ModelArts管理控制台**，在左侧菜单栏中选择“资产管理>数据集”，进入“数据集”管理页面。
2. 选择对应的数据集，单击操作列的“更多 > 数据特征”，进入数据集概览页的数据特征页面。

   您也可以在单击数据集名称进入数据集概览页后，单击“数据特征”页签进入。
3. 由于发布后的数据集不会默认启动数据特征分析，针对数据集的各个版本，需手动启动特征分析任务。在数据特征页签下，单击“启动特征分析”。
4. 在弹出的对话框中配置需要进行特征分析的数据集版本，然后单击“确定”启动分析。

   “版本选择”，即选择当前数据集的已发布版本。

   图 7-89 启动数据特征分析任务

5. 数据特征分析任务启动后，需执行一段时间，根据数据量不同等待时间不同，请耐心等待。当您选择分析的版本出现在“版本选择”列表下，且可选择时，即表示分析已完成。
6. 查看数据特征分析结果。

   “版本选择”：在右侧下拉框中选择进行对比的版本。也可以只选择一个版本。

   “类型”：选择需要分析的类型。支持“all”、“train”、“eval”和“inference”。

   “数据特征指标”：在右侧下拉框中勾选需要展示的指标。详细指标说明请参见**支持分析指标及其说明**。

   选择完成后，页面将自动呈现您选择对应版本及其指标数据，您可以根据呈现的图表了解数据分布情况，帮助您更好的处理您的数据。
7. 查看分析任务的历史记录。

   在数据特征分析后，您可以在“数据特征”页签下，单击右侧“任务历史”，可在弹出对话框中查看历史分析任务及其状态。

支持分析指标及其说明

表 7-35 分析指标列表

名称	说明	分析说明
分辨率 Resolution	图像分辨率。此处使用面积值作为统计值。	通过指标分析结果查看是否有偏移点。如果存在偏移点，可以对偏移点做resize操作或直接删除。
图片高宽比 Aspect Ratio	图像高宽比，即图片的高度/图片的宽度。	一般呈正态分布，一般用于比较训练集和真实场景数据集的差异。
图片亮度 Brightness	图片亮度，值越大代表观感上亮度越高。	一般呈正态分布，可根据分布中心判断数据集整体偏亮还是偏暗。可根据使用场景调整，比如使用场景是夜晚，图片整体应该偏暗。
图片饱和度 Saturation	图片的色彩饱和度，值越大表示图片整体色彩越容易分辨。	一般呈正态分布，一般用于比较训练集和真实场景数据集的差异。
清晰度 Clarity	图片清晰程度，使用拉普拉斯算子计算所得，值越大代表边缘越清晰，图片整体越清晰。	可根据使用场景判断清晰度是否满足需要。比如使用场景的数据采集来自高清摄像头，那么清晰度对应的需要高一些。可通过对数据集做锐化或模糊操作，添加噪声对清晰度做调整。
图像色彩的丰富程度 Colorfulness	横坐标：图像的色彩丰富程度，值越大代表色彩越丰富。 纵坐标：图片数量。	是观感上的色彩丰富程度，一般用于比较训练集和真实场景数据集的差异。
按单张图片中框的个数统计图片分布 Bounding Box Quantity	横坐标：单张图片中框的个数。 纵坐标：图片数量。	对模型而言一张图片的框个数越多越难检测，需要越多的这种数据用作训练。
按单张图片中框的面积标准差统计图片分布 Standard Deviation of Bounding Boxes Per Image	横坐标：单张图片中框的标准差。单张图片只有一个框时，标准差为0。标准差的值越大，表示图片中框大小不一程度越高。 纵坐标：图片数量。	对模型而言一张图中框如果比较多且大小不一，是比较难检测的，可以根据场景添加数据用作训练，或者实际使用没有这种场景可直接删除。

名称	说明	分析说明
按高宽比统计框数量的分布 Aspect Ratio of Bounding Boxes	横坐标：目标框的高宽比。 纵坐标：框数量（统计所有图片中的框）。	一般呈泊松分布，但与使用场景强相关。多用于比较训练集和验证集的差异，如训练集都是长方形框的情况下，验证集如果是接近正方形的框会有比较大影响。
按面积占比统计框数量的分布 Area Ratio of Bounding Boxes	横坐标：目标框的面积占比，即目标框的面积占整个图片面积的比例，越大表示物体在图片中的占比越大。 纵坐标：框数量（统计所有图片中的框）。	主要判断模型中使用的anchor的分布，如果目标框普遍较大，anchor就可以选择较大。
按边缘化程度统计框数量的分布 Marginalization Value of Bounding Boxes	横坐标：边缘化程度，即目标框中心点距离图片中心点的距离占图片总距离的比值，值越大表示物体越靠近边缘。（图片总距离表示以图片中心点为起点画一条经过标注框中心点的射线，该射线与图片边界交点到图片中心点的距离）。 纵坐标：框数量（统计所有图片中的框）。	一般呈正态分布。用于判断物体是否处于图片边缘，有一些只露出一部分的边缘物体，可根据需要添加数据集或不标注。
按堆叠度统计框数量的分布 Overlap Score of Bounding Boxes	横坐标：堆叠度，单个框被其他的框重叠的部分，取值范围为0～1，值越大表示被其他框覆盖的越多。 纵坐标：框数量（统计所有图片中的框）。	主要用于判断待检测物体的堆叠程度，堆叠物体一般对于检测难度较高，可根据实际使用需要添加数据集或不标注部分物体。
按亮度统计框数量的分布 Brightness of Bounding Boxes	横坐标：目标框的图片亮度，值越大表示越亮。 纵坐标：框数量（统计所有图片中的框）。	一般呈正态分布。主要用于判断待检测物体的亮度。在一些特殊场景中只有物体的部分亮度较暗，可以看是否满足要求。
按清晰度统计框数量的分布 Clarity of Bounding Boxes	横坐标：目标框的清晰度，值越大表示越清晰。 纵坐标：框数量（统计所有图片中的框）。	主要用于判断待检测物体是否存在模糊的情况。比如运动中的物体在采集中可能变得模糊，需要重新采集。

7.7 导出 ModelArts 数据集中的数据

针对数据集中的数据，用户可以选中部分数据或者通过条件筛选出需要的数据，导出成新的数据集。用户可以通过任务历史查看数据导出的历史记录。

目前只有“图像分类”、“物体检测”、“图像分割”类型的数据集支持导出功能。

- “图像分类”只支持导出txt格式的标注文件。
- “物体检测”只支持导出Pascal VOC格式的XML标注文件。
- “图像分割”只支持导出Pascal VOC格式的XML标注文件以及Mask图像。

导出数据为新数据集

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理>数据集”，进入“数据集”管理页面。
2. 在数据集列表中，选择“图片”类型的数据集，单击数据集名称进入“数据集概览页”。
3. 在“数据集概览页”，单击右上角“导出 ”。在弹出的“导出”对话框中，填写相关信息，然后单击“确定”，开始执行导出操作。

   “数据来源”：选择新数据集。

   “名称”：新数据集名称。

   “保存路径”：表示新数据集的输入路径，即当前数据导出后存储的OBS路径。

   “输出路径”：表示新数据集的输出路径，即新数据集在完成标注后输出的路径。“输出路径”不能与“保存路径”为同一路径，且“输出路径”不能是“保存路径”的子目录。

   图 7-90 导出新数据集

4. 数据导出成功后，您可以前往您设置的保存路径，查看到存储的数据。当导出方式选择为新数据集时，在导出成功后，您可以前往“数据集”列表中，查看到新的数据集。
5. 在“数据集概览页”，单击右上角“导出历史”，在弹出的“任务历史”对话框中，可以查看该数据集之前的导出任务历史。

7.7.1 导出 ModelArts 数据集中的数据到 OBS

针对数据集中的数据，用户可以选中部分数据或者通过条件筛选出需要的数据，当需要将数据集中的数据存储至OBS用于后续导出使用时，可通过此种方式导出成新的数据集。用户可以通过任务历史查看数据导出的历史记录。

目前只有“图像分类”、“物体检测”、“图像分割”类型的数据集支持导出功能。

- “图像分类”只支持导出txt格式的标注文件。
- “物体检测”只支持导出Pascal VOC格式的XML标注文件。
- “图像分割”只支持导出Pascal VOC格式的XML标注文件以及Mask图像。

导出数据到 OBS

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理>数据集”，进入“数据集”管理页面。
2. 在数据集列表中，选择“图片”类型的数据集，单击数据集名称进入“数据集概览页”。
3. 在“数据集概览页”，单击右上角“导出 ”。在弹出的“导出”对话框中，填写相关信息，然后单击“确定”，开始执行导出操作。

   “数据来源”：选择OBS。

   “保存路径”：即导出数据存储的路径。建议不要将数据存储至当前数据集所在的输入路径或输出路径。

   图 7-91 导出到 OBS

4. 数据导出成功后，您可以前往您设置的保存路径，查看到存储的数据。
5. 在“数据集概览页”，单击右上角“导出历史”，在弹出的“任务历史”对话框中，可以查看该数据集之前的导出任务历史。

   图 7-92 任务历史

7.7.2 导出 ModelArts 数据集中的数据为新数据集

针对数据集中的数据，用户可以选中部分数据或者通过条件筛选出需要的数据，导出成新的数据集。用户可以通过任务历史查看数据导出的历史记录。本章主要介绍将ModelArts数据集中的数据为新数据集的方式，新导出的数据集可直接在ModelArts控制台数据集列表中显示。

目前只有“图像分类”、“物体检测”、“图像分割”类型的数据集支持导出功能。

- “图像分类”只支持导出txt格式的标注文件。

- “物体检测”只支持导出Pascal VOC格式的XML标注文件。
- “图像分割”只支持导出Pascal VOC格式的XML标注文件以及Mask图像。

导出数据为新数据集

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理>数据集”，进入“数据集”管理页面。
2. 在数据集列表中，选择“图片”类型的数据集，单击数据集名称进入“数据集概览页”。
3. 在“数据集概览页”，单击右上角“导出 ”。在弹出的“导出”对话框中，填写相关信息，然后单击“确定”，开始执行导出操作。

   “数据来源”：选择新数据集。

   “名称”：新数据集名称。

   “保存路径”：表示新数据集的输入路径，即当前数据导出后存储的OBS路径。

   “输出路径”：表示新数据集的输出路径，即新数据集在完成标注后输出的路径。“输出路径”不能与“保存路径”为同一路径，且“输出路径”不能是“保存路径”的子目录。

   图 7-93 导出新数据集

4. 数据导出成功后，您可以前往您设置的保存路径，查看到存储的数据。当导出方式选择为新数据集时，在导出成功后，您可以前往“数据集”列表中，查看到新的数据集。
5. 在“数据集概览页”，单击右上角“导出历史”，在弹出的“任务历史”对话框中，可以查看该数据集之前的导出任务历史。

7.8 入门案例：快速创建一个物体检测的数据集

本节以准备训练物体检测模型的数据为例，介绍如何针对样例数据，进行数据分析、数据标注等操作，完成数据准备工作。在实际业务开发过程中，可以根据业务需求选择数据管理的一种或多种功能完成数据准备。此次操作分为以下流程：

- 准备工作

- 创建数据集
- 数据分析
- 数据标注
- 数据发布
- 数据导出

准备工作

在使用ModelArts数据管理的功能前，需要先完成以下准备工作。

用户在使用数据管理的过程中，ModelArts需要访问用户的OBS等依赖服务，需要用户进行在“权限管理”页面中进行委托授权。具体操作参考使用委托授权（推荐）。

说明

数据标注功能仅在以下Region支持：华北-北京四、西南-贵阳一、中国-香港、亚太-新加坡、亚太-曼谷、亚太-雅加达、拉美-圣地亚哥、拉美-圣保罗一、拉美-墨西哥城二。

创建数据集

本示例使用OBS中的数据作为数据集的输入目录创建数据集。参考如下操作创建一个物体检测类型的数据集，并将数据导入到数据集中。

步骤1 登录ModelArts管理控制台，在左侧菜单栏中选择“资产管理 > 数据集”，进入“数据集”管理页面。

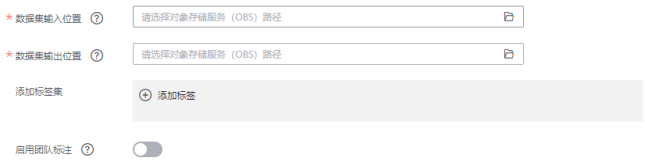
步骤2 单击“创建数据集”，进入“创建数据集”页面，根据数据类型以及数据标注要求，选择创建不同类型的数据集。

1. 填写数据集基本信息，数据集的“名称”和“描述”。
2. 选择“标注场景”和“标注类型”，本案例中分别选择“图片”和“物体检测”。

图 7-94 数据集标注场景和标注类型

3. 选择OBS中的数据目录作为“数据集输入位置”，选择不同的OBS目录作为“数据集输出位置”。

图 7-95 数据集的输入位置和输出位置



4. 参数填写无误后，单击页面右下角“创建”，即可完成数据集的创建。

----结束

数据分析

数据集创建完成后，可以基于图片各项特征，如模糊度、亮度等进行分析，帮助用户更好的分析数据集的数据质量，判断数据集是否满足自己的算法和模型要求。

1. 创建特征分析任务
   a. 在执行特征分析前，需先发布一个数据集版本。在“数据集概览”页单击右上角的“发布”，为数据集发布一个新版本。
   b. 版本发布完成后，进入数据集概览页。选择“数据特征”页签，单击“特征分析”，在弹窗中选择刚才发布的数据集版本，并单击“确定”，启动特征分析任务。

   图 7-96 启动特征分析

   c. 查看任务进度

   任务执行过程中，可以单击“任务历史”，查看任务进度。当任务状态变为“成功”时，表示任务执行完成。

   图 7-97 特征分析任务进度

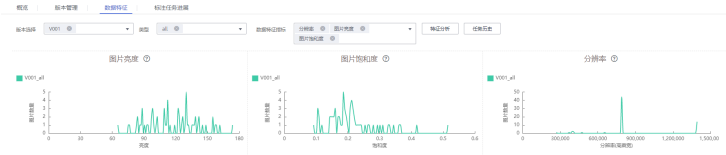
数据集版本	任务ID	创建时间	运行时间（hh:...	状态
V001	QFnWQpFyqi...	2021/08/25 2...	00:01:02	运行中

2. 查看特征分析结果

   特征分析任务执行完成后，可以在“数据特征”页签下，选择“数据集版本”、“类型”和“数据特征指标”，页面将自动呈现您选择对应版本及其指标数据，您可以根据呈现的图表了解数据分布情况，帮助您更好的理解您的数据。

   - “版本选择”：根据实际情况选择已执行过特征任务的版本，可以选多个进行对比，也可以只选择一个。
   - “类型”：根据需要分析的类型选择。支持“all”、“train”、“eval”和“inference”。分别表示所有、训练、评估和推理类型。
   - “数据特征指标”：选择您需要展示的指标。详细指标解释，可参见**特征分析指标列表**。

图 7-98 查看特征分析结果



在特征分析结果中，例如图片亮度指标，数据分布中，分布不均匀，缺少某一种亮度的图片，而此指标对模型训练非常关键。此时可选择增加对应亮度的图片，让数据更均衡，为后续模型构建做准备。

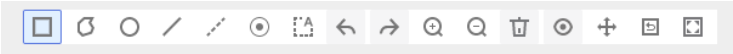
数据标注

- **人工标注**

  a. 在“未标注”页签图片列表中，单击图片，自动跳转到标注页面。

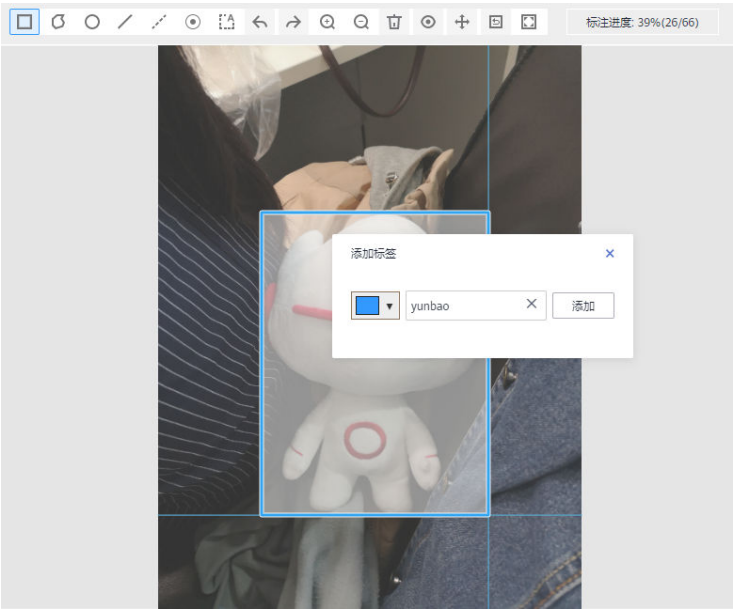
  b. 在标注页面的工具栏中选择合适的标注工具，本示例使用矩形框进行标注。

  图 7-99 标注工具

  c. 使用标注工具选中目标区域，在弹出的标签文本框中，直接输入新的标签名。如果已存在标签，从下拉列表中选择已有的标签。单击“添加”完成标注。

  图 7-100 添加物体检测标签

  d. 单击页面上方“返回数据标注预览”查看标注信息，在弹框中单击“确定”保存当前标注并离开标注页面。选中的图片被自动移动至“已标注”页签，且在“未标注”和“全部”页签中，标签的信息也将随着标注步骤进行更新，如增加的标签名称、标签对应的图片数量。

- **智能标注**

通过人工标注完成少量数据标注后，可以通过智能标注对剩下的数据进行自动标注，提高标注的效率。

a. 在数据集详情页面，单击右上角“启动智能标注”。

b. 在“启动智能标注”窗口中，填写如下参数，然后单击“提交”。

- **智能标注类型**：主动学习
- **算法类型**：快速型

其他参数采用默认值。

图 7-101 启动智能标注任务

c. 查看智能标注任务进度

智能标注任务启动后，可以在“待确认”页签下查看智能标注任务进度。当任务完成后，即可在“待确认”页签下查看自动标注好的数据。

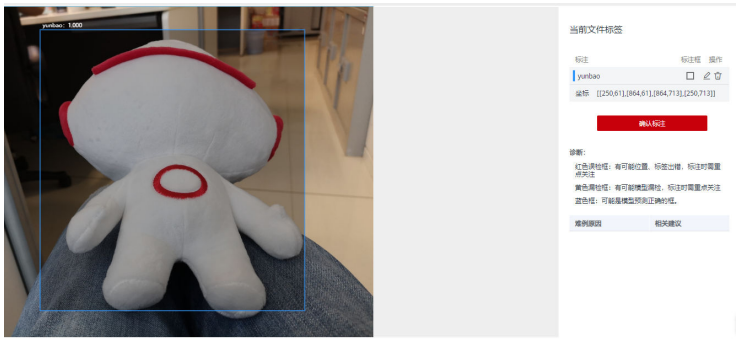
图 7-102 查看智能标注任务进度

d. 确认智能标注结果

在智能标注任务完成后，在“待确认”页签下，单击具体图片进入标注详情页面，可以查看或修改智能标注的结果。

如果智能标注的数据无误，可单击右侧的“确认标注”完成标注，如果标注信息有误，可直接删除错误标注框，然后重新标注，以纠正标注信息。针对物体检测任务，需一张一张确认。确保所有图片已完成确认，然后执行下一步操作。

图 7-103 确认智能标注结果



数据发布

ModelArts训练管理模块支持通过ModelArts数据集或者OBS目录中的文件创建训练作业。如果选择通过数据集作为训练作业的数据源，则需要指定数据集及特定的版本。因此，用户需要为准备好的数据发布一个版本，具体操作参考**发布ModelArts数据集中的数据版本**。

说明

为了便于后期的模型构建和开发，对同一数据源来说，将其不同时间对数据的处理和标注按照版本来进行区分，按照需求选择指定的版本使用。

数据导出

ModelArts训练管理模块支持通过ModelArts数据集或者OBS目录中的文件创建训练作业。如果选择通过OBS目录的方式创建训练作业，用户需要将数据集中准备好的数据导出到OBS中。

1. 导出数据到OBS
   a. 在数据集详情页面中，选中需要导出的数据或筛选出需要导出的数据，然后单击右上角“导出”。
   b. 导出方式选择“OBS”，填写相关信息，然后单击“确定”，开始执行导出操作。
      “保存路径”：即导出数据存储的路径。建议不要将数据存储至当前数据集所在的输入路径或输出路径。

图 7-104 导出至 OBS

c. 数据导出成功后，您可以前往您设置的保存路径，查看到存储的数据。

2. 查看任务历史

当您导出数据后，可以通过任务历史查看导出任务明细。

a. 在数据集详情页面中，单击右上角“任务历史 ”。

b. 在弹出的“任务历史”对话框中，可以查看该数据集之前的导出任务历史。包括“任务ID”、“创建时间”、“导出方式”、“导出路径”、“导出样本总数”和“导出状态”。

图 7-105 导出任务历史

8 使用 ModelArts Standard 训练模型

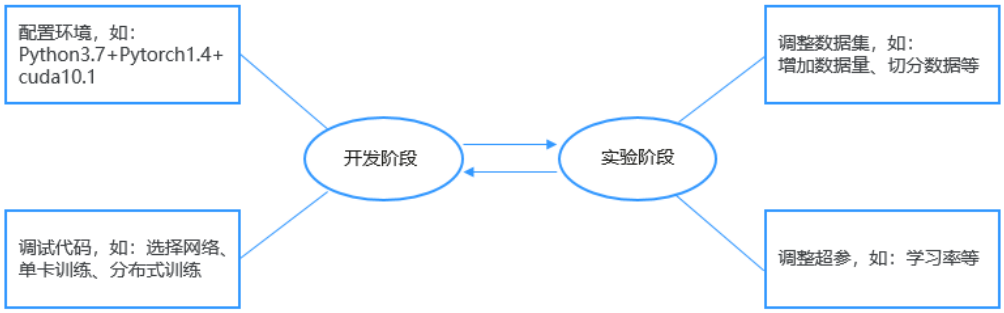
8.1 模型训练使用流程

AI模型开发的过程，称之为Modeling，一般包含两个阶段：

- 开发阶段：准备并配置环境，调试代码，使代码能够开始进行深度学习训练，推荐在ModelArts开发环境中调试。
- 实验阶段：调整数据集、调整超参等，通过多轮实验，训练出理想的模型，推荐在ModelArts训练中进行实验。

两个过程可以相互转换。如开发阶段代码稳定后，则会进入实验阶段，通过不断尝试调整超参来迭代模型；或在实验阶段，有一个可以优化训练的性能的想法，则会回到开发阶段，重新优化代码。

图 8-1 模型开发过程



ModelArts提供了模型训练的功能，方便您查看训练情况并不断调整您的模型参数。您还可以基于不同的数据，选择不同规格的资源池用于模型训练。

请参考以下指导在ModelArts Standard上训练模型。

图 8-2 ModelArts Standard 模型训练流程

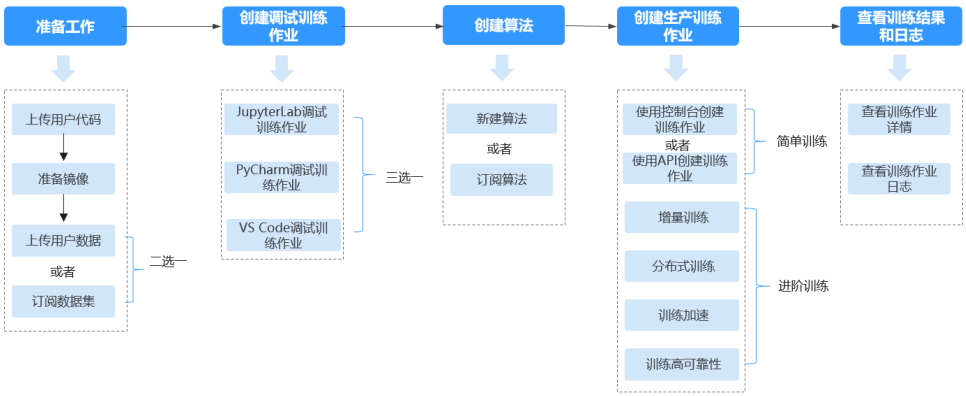


表 8-1 Standard 模型训练流程

操作任务	子任务	说明
准备工作	准备训练代码	模型训练必备要素包括训练代码、训练框架、训练数据。 训练代码包含训练作业的启动文件或启动命令、训练依赖包等内容。 • 当使用预置框架创建训练作业时，训练代码的开发规范可以参考**开发用于预置框架训练的代码**。 • 当使用自定义镜像创建训练作业时，训练代码的开发规范可以参考**开发用于自定义镜像训练的代码**。

操作任务	子任务	说明
	准备训练框架（即训练镜像）	模型训练有多种训练框架来源，具体可以参考**准备模型训练镜像**。 • ModelArts Standard平台提供了模型训练常用的预置框架，可以直接使用。 • 当预置框架不满足训练要求时，支持用户构建自定义镜像用于训练。
	准备训练数据	训练数据除了训练数据集，也可以是预测模型。在创建训练作业前，需要先准备好训练数据。 • 当训练数据可以直接使用，无需二次处理时，可以直接将数据上传至OBS桶。在创建训练作业时，训练的输入参数位置可以直接填写OBS桶路径。 • 当训练数据集的数据未标注或者需要进一步的数据预处理，可以先将数据导入ModelArts数据管理模块进行数据预处理。在创建训练作业时，训练的输入参数位置可以选择数据管理模块的数据集。
创建调试训练作业	调试训练作业	模型训练前，一般会先对代码进行调试，ModelArts提供多种方式创建调试训练作业。 • ModelArts提供了云化版本的JupyterLab，无需关注安装配置，即开即用。 • ModelArts也提供了本地IDE的方式开发模型，通过开启SSH远程开发，本地IDE可以远程连接到调试训练作业中，进行调试和运行代码。本地IDE方式不影响用户的编码习惯，并且调试完成的代码可以零成本直接创建生产训练作业。支持的本地IDE请参考**使用PyCharm ToolKit创建并调试训练作业**。
创建算法	创建算法	创建生产训练作业之前，需要先准备算法，可以是用户自己准备的算法，也可以使用从AI Gallery订阅的算法。
创建生产训练作业	训练作业基础功能	• ModelArts Standard支持通过Console控制台的可视化界面创建训练作业，创建时基于算法来源和训练框架又区分多种创建方式，具体请参见**表8-2**。 • ModelArts Standard也支持通过调用API接口创建训练作业，请参见**以PyTorch框架创建训练作业**。
	训练作业进阶功能	ModelArts Standard还支持以下训练进阶功能，例如： • 增量训练 • 分布式训练 • 训练加速 • 训练高可靠性
查看训练结果和日志	查看训练作业详情	训练作业运行中或运行结束后，可以在训练作业详情页面查看训练作业的参数设置，训练作业事件等。
	查看训练作业日志	训练日志用于记录训练作业运行过程和异常信息，可以通过查看训练作业日志定位作业运行中出现的问题。

表 8-2 训练作业的创建方式介绍

创建方式	适用场景
使用预置框架创建训练作业	如果您已在本地使用一些常用框架完成算法开发，您可以选择常用框架，创建训练作业来构建模型
使用自定义镜像创建训练作业	如果您开发算法时使用的框架并不是常用框架，您可以将算法构建为一个自定义镜像，通过自定义镜像创建训练作业。
使用已有算法创建训练作业	算法管理中，管理了用户自己创建的算法和AI Gallery订阅的算法，您可以使用算法管理中的算法，快速创建训练作业，构建模型。
使用订阅算法创建训练作业	AI Gallery中提供了现成的算法，供用户使用，您可以直接订阅AI Gallery中的算法，快速创建训练作业，构建模型。

8.2 准备模型训练代码

8.2.1 预置框架启动文件的启动流程说明

ModelArts Standard训练服务预置了多种AI框架，并对不同的框架提供了针对性适配，用户在使用这些预置框架进行模型训练时，训练的启动命令也需要做相应适配。

本章节详细介绍基于不同的预置框架创建训练作业时，如何修改训练的启动文件。

Ascend-Powered-Engine 框架启动原理

在ModelArts创建训练作业界面选择AI框架时，有一个AI框架是“Ascend-Powered-Engine”，它既不是一个AI框架（如：PyTorch、TensorFlow）也不是一个并行执行框架（如：MPI），而是适配加速芯片Ascend的一组AI框架+运行环境+启动方式的集合。

由于主流的Snt9系列Ascend加速卡都跑在ARM CPU规格的机器上，因此上层docker镜像也都是ARM镜像。相对于GPU场景的镜像中安装了与GPU驱动适配的CUDA（由英伟达推出的统一计算架构）计算库，Ascend-Powered-Engine引擎的镜像中安装了与Ascend驱动适配的CANN（华为针对AI场景推出的异构计算架构）计算库。

提交训练作业后，ModelArts Standard平台会自动运行训练作业的启动文件。

Ascend-Powered-Engine框架的启动文件的默认启动方式如下：

每个训练作业的启动文件的运行次数取决于任务卡数，即在训练作业运行时，有N个任务卡数训练作业内就会运行N次启动文件。例如，单机1卡，则worker-0任务的启动文件会被运行1次；单机8卡，则worker-0任务的启动文件会被运行8次。因此需要避免在启动文件中进行端口监听。

启动文件会被自动设置如下环境变量：

- RANK_TABLE_FILE：rank table file（RTF）文件路径。
- ASCEND_DEVICE_ID：逻辑device_id，例如单卡训练，该值始终为 0。
- RANK_ID：可以理解为训练作业级的device逻辑（顺序）编号。
- RANK_SIZE：根据RTF中device的数目设置该值，例如“4 * snt9b”，则该值即为4。

当需要启动文件仍然在逻辑上仅运行1次时，则可以在启动文件中判断“ASCEND_DEVICE_ID”的值，当值为“0”则执行逻辑，当值为非0则直接退出。

Ascend-Powered-Engine框架对应的代码示例“mindspore-verification.py”，请参见**训练mindspore-verification.py文件**。

Ascend-Powered-Engine框架单机启动命令和分布式启动命令无区别。

Ascend-Powered-Engine框架支持多种启动方式来启动“启动文件”，默认是基于“RANK_TABLE_FILE”启动，也可以通过配置“MA_RUN_METHOD”环境变量使用其他方式来启动。MA_RUN_METHOD环境变量支持torchrun和msrun。

- 当“MA_RUN_METHOD=torchrun”时，表示ModelArts Standard平台使用torchrun命令启动训练作业的“启动文件”。

  说明

  要求PyTorch版本大于等于1.11.0。

  - 单机时，ModelArts Standard平台使用如下命令启动训练作业的“启动文件”。
    ```
    torchrun --standalone --nnodes=${MA_NUM_HOSTS} --nproc_per_node=${MA_NUM_GPUS} ${MA_EXTRA_TORCHRUN_PARAMS} "启动文件" {arg1} {arg2} ...
    ```
  - 多机时，ModelArts Standard平台使用如下命令启动训练作业的“启动文件”。
    ```
    torchrun --nnodes=${MA_NUM_HOSTS} --nproc_per_node=${MA_NUM_GPUS} --node_rank=${VC_TASK_INDEX} --master_addr={master_addr} --master_port=${MA_TORCHRUN_MASTER_PORT} --rdzv_id={ma_job_name} --rdzv_backend=static ${MA_EXTRA_TORCHRUN_PARAMS} "启动文件" {arg1} {arg2} ...
    ```

  参数说明如下：

  - standalone：标识为单任务实例作业。
  - nnodes：任务实例个数。
  - nproc_per_node：每个任务实例启动的主进程数，设置为任务分配的NPU数相同。
  - node_rank：任务rank，用于多任务分布式训练。
  - master_addr：主任务（rank 0）的地址，设置为任务worker-0的通信域名。
  - master_port：在主任务（rank 0）上，用于分布式训练期间通信的端口。默认设置为18888端口。当遇到master_port冲突问题时，可通过设置MA_TORCHRUN_MASTER_PORT环境变量值修改端口配置。
  - rdzv_id：Rendezvous标识，设置为带有训练作业ID的值。
  - rdzv_backend：Rendezvous后端，固定设置为static，即不使用Rendezvous，而是使用master_addr和master_port配置。另外，可通过设置MA_EXTRA_TORCHRUN_PARAMS环境变量值，以增加额外的torchrun命令参数，或是覆盖预设的torchrun命令参数。例如配置torchrun命令中rdzv_conf参数的训练作业API环境变量的部分示例如下：
    ```
    "environments": {
    "MA_RUN_METHOD": "torchrun",
    "MA_EXTRA_TORCHRUN_PARAMS": "--rdzv_conf=timeout=7200"
    }
    ```


说明

如果在torchrun初始化分布式一致性协商阶段出现“RuntimeError：Socket Timeout”错误时，可以通过增加如下环境变量再次创建训练作业以查看torchrun初始化阶段的详细信息，进一步排查问题。

- LOGLEVEL=INFO
- TORCH_CPP_LOG_LEVEL=INFO
- TORCH_DISTRIBUTED_DEBUG=DETAIL

出现“RuntimeError: Socket Timeout”错误，一般是因为不同任务执行torchrun命令的时机差距过大导致的。torchrun命令执行时机差距过大，大多是因为在torchrun命令被执行之前任务还有一些初始化动作，例如下载训练数据集、CKPT等。这些初始化动作执行耗时差距过大会直接导致出现Socket Timeout错误。所以遇到Socket Timeout问题时首先需要排查的是各个任务执行torchrun的时间点差距是否在合理范围内，如果时间点差距过大，需要优化执行torchrun命令之前的初始化动作，使其时间点差距在合理范围内。

- 当“MA_RUN_METHOD=msrun”时，表示ModelArts Standard平台使用msrun命令启动训练作业的“启动文件”。

  说明

  要求MindSpore版本大于等于2.3.0。

  该方案支持动态组网和基于rank table file文件组网两种方式。当配置了环境变量MS_RANKTABLE_ENABLE="True"，则msrun会读取rank table file文件内容进行组网。否则默认使用动态组网。

  msrun使用如下命令启动训练作业的“启动文件”。

  ```
  msrun --worker_num=${msrun_worker_num} --local_worker_num=${MA_NUM_GPUS} --master_addr=${msrun_master_addr} --node_rank=${VC_TASK_INDEX} --master_port=${msrun_master_port} --log_dir=${msrun_log_dir} --join=True --cluster_time_out=${MSRUN_CLUSTER_TIME_OUT} --rank_table_file=${msrun_rank_table_file} "启动文件" {arg1} {arg2} ...
  ```

  参数说明如下：

  - worker_num：所有进程个数。因为一个卡起一个进程，所以也表示使用总卡数。
  - local_worker_num：当前节点进程个数，即当前节点使用的卡数。
  - master_addr：msrun组网调度进程所在节点的IP地址，单机场景无需配置。
  - master_port：msrun组网调度进程的端口。
  - node_rank：当前节点的编号。
  - log_dir：msrun组网和各个进程的日志输出地址。
  - join：训练进程拉起后，msrun进程是否仍存在，默认配置为“True”，等待所有进程退出后再退出。
  - cluster_time_out：集群组网超时时间，默认是“600s”，可通过环境变量“MSRUN_CLUSTER_TIME_OUT”控制。
  - rank_table_file：rank table file文件地址，如果配置了环境变量“MS_RANKTABLE_ENABLE="True"”，启动时会增加该参数。

PyTorch-GPU 框架启动原理

单机多卡场景下平台会为启动文件额外拼接 --init_method "tcp://<ip>:<port>" 参数。

多机多卡场景下平台会为启动文件额外拼接 --init_method "tcp://<ip>:<port>" --rank <rank_id> --world_size <node_num>参数。

启动文件需要解析上述参数。

PyTorch-GPU框架的代码示例，请参见**示例：创建DDP分布式训练（PyTorch+GPU）**中的方式一

TensorFlow-GPU 框架启动原理

单机场景下（即选择的实例数为1），ModelArts只会在一个节点上启动一个训练容器，该训练容器独享节点规格的可使用资源。

多机场景下（即选择的实例数大于1），ModelArts会优先在相同节点上启动一个parameter server（以下简称ps）和一个worker，平台会自动一比一分配ps与worker任务。例如，双机场景会分配2个ps和2个worker任务，并为启动文件额外注入如下参数。

```
--task_index <VC_TASK_INDEX> --ps_hosts <TF_PS_HOSTS> --worker_hosts <TF_WORKER_HOSTS> --job_name <MA_TASK_NAME>
```

启动文件需要解析如下参数。

- VC_TASK_INDEX：task序号，如0、1、2。
- TF_PS_HOSTS ： ps节点地址数组，如“[xx-ps-0.xx:TCP_PORT,xx-ps-1.xx:TCP_PORT]”，TCP_PORT是一个在5000~10000的随机端口。
- TF_WORKER_HOSTS：worker节点地址数组，如“[xx-worker-0.xx:TCP_PORT,xx-worker-1.xx:TCP_PORT]”，TCP_PORT是一个在5000~10000的随机端口。
- MA_TASK_NAME：任务名称，取值是ps或worker。

具体示例请参见：**TensorFlow-GPU框架的代码示例mnist.py（单机）**。

Horovod/MPI/MindSpore-GPU

使用Horovod/MPI/MindSpore-GPU预置框架来运行的启动文件，平台自动以mpirun命令启动之。使用ModelArts Standard训练相应预置引擎，用户仅需关注启动文件（即训练脚本）的编写；mpirun命令和训练作业集群的构建都由平台自动完成。平台不会为启动文件额外拼接参数。

“pytorch_synthetic_benchmark.py”文件示例如下：

```
import argparse
import torch.backends.cudnn as cudnn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data.distributed
from torchvision import models
import horovod.torch as hvd
import timeit
import numpy as np

# Benchmark settings
parser = argparse.ArgumentParser(description='PyTorch Synthetic Benchmark',
                                 formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--fp16-allreduce', action='store_true', default=False,
                    help='use fp16 compression during allreduce')

parser.add_argument('--model', type=str, default='resnet50',
                    help='model to benchmark')
parser.add_argument('--batch-size', type=int, default=32,
                    help='input batch size')

parser.add_argument('--num-warmup-batches', type=int, default=10,
                    help='number of warm-up batches that don\'t count towards benchmark')
parser.add_argument('--num-batches-per-iter', type=int, default=10,
```

```
                    help='number of batches per benchmark iteration')
parser.add_argument('--num-iters', type=int, default=10,
                    help='number of benchmark iterations')

parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')

parser.add_argument('--use-adasum', action='store_true', default=False,
                    help='use adasum algorithm to do reduction')

args = parser.parse_args()
args.cuda = not args.no_cuda and torch.cuda.is_available()

hvd.init()

if args.cuda:
    # Horovod: pin GPU to local rank.
    torch.cuda.set_device(hvd.local_rank())

cudnn.benchmark = True

# Set up standard model.
model = getattr(models, args.model)()

# By default, Adasum doesn't need scaling up learning rate.
lr_scaler = hvd.size() if not args.use_adasum else 1

if args.cuda:
    # Move model to GPU.
    model.cuda()
    # If using GPU Adasum allreduce, scale learning rate by local_size.
    if args.use_adasum and hvd.nccl_built():
        lr_scaler = hvd.local_size()

optimizer = optim.SGD(model.parameters(), lr=0.01 * lr_scaler)

# Horovod: (optional) compression algorithm.
compression = hvd.Compression.fp16 if args.fp16_allreduce else hvd.Compression.none

# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(optimizer,
                                     named_parameters=model.named_parameters(),
                                     compression=compression,
                                     op=hvd.Adasum if args.use_adasum else hvd.Average)

# Horovod: broadcast parameters & optimizer state.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)

# Set up fixed fake data
data = torch.randn(args.batch_size, 3, 224, 224)
target = torch.LongTensor(args.batch_size).random_() % 1000
if args.cuda:
    data, target = data.cuda(), target.cuda()


def benchmark_step():
    optimizer.zero_grad()
    output = model(data)
    loss = F.cross_entropy(output, target)
    loss.backward()
    optimizer.step()


def log(s, nl=True):
    if hvd.rank() != 0:
        return
    print(s, end='\n' if nl else '')
```



```
log('Model: %s' % args.model)
log('Batch size: %d' % args.batch_size)
device = 'GPU' if args.cuda else 'CPU'
log('Number of %ss: %d' % (device, hvd.size()))

# Warm-up
log('Running warmup...')
timeit.timeit(benchmark_step, number=args.num_warmup_batches)

# Benchmark
log('Running benchmark...')
img_secs = []
for x in range(args.num_iters):
    time = timeit.timeit(benchmark_step, number=args.num_batches_per_iter)
    img_sec = args.batch_size * args.num_batches_per_iter / time
    log('Iter #%d: %.1f img/sec per %s' % (x, img_sec, device))
    img_secs.append(img_sec)

# Results
img_sec_mean = np.mean(img_secs)
img_sec_conf = 1.96 * np.std(img_secs)
log('Img/sec per %s: %.1f +-%.1f' % (device, img_sec_mean, img_sec_conf))
log('Total img/sec on %d %s(s): %.1f +-%.1f' %
    (hvd.size(), device, hvd.size() * img_sec_mean, hvd.size() * img_sec_conf))
```

run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

MY_MPI_BTL_TCP_IF=${MY_MPI_BTL_TCP_IF:-"eth0,bond0"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
```

```
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-p $
{MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=${MY_MPI_BTL_TCP_IF} -x
NCCL_SOCKET_FAMILY=AF_INET \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?

    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    else
        echo "[run_mpi] exec command successfully, exited with $RET_CODE"
    fi

    # stop 1...N worker by killing the sleep proc
    sed -i '1d' ${MY_HOME}/hostfile
    if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
        echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

        sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
        printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

        mpirun \
        --hostfile ${MY_HOME}/hostfile \
        --mca btl_tcp_if_include ${MY_MPI_BTL_TCP_IF} \
        --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
```



```
        -x PATH -x LD_LIBRARY_PATH \
        pkill sleep \
        > /dev/null 2>&1
    fi

    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

8.2.2 开发用于预置框架训练的代码

当您使用ModelArts Standard提供的预置框架创建算法时，您需要提前完成算法的代码开发。本章详细介绍如何改造本地代码以适配ModelArts上的训练。

创建算法时，您需要在创建页面提供代码目录路径、代码目录路径中的启动文件、训练输入路径参数和训练输出路径参数。这四种输入搭建了用户代码和ModelArts Standard后台交互的桥梁。

- 代码目录路径

  您需要在OBS桶中指定代码目录，并将训练代码、依赖安装包或者预生成模型等训练所需文件上传至该代码目录下。训练作业创建完成后，ModelArts会将代码目录及其子目录下载至后台容器中。

  例如：OBS路径“obs://obs-bucket/training-test/demo-code”作为代码目录，OBS路径下的内容会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。

  请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。建议训练代码目录大小小于或等于50MB。

- 代码目录路径中的启动文件

  代码目录路径中的启动文件作为训练启动的入口，当前只支持python格式。预置框架启动文件的启动流程说明请参见**预置框架启动文件的启动流程说明**。

- 训练输入路径参数

  训练数据需上传至OBS桶或者存储至**数据集**中。在训练代码中，用户需解析**输入路径参数**。系统后台会自动下载输入参数路径中的训练数据至训练容器的本地目录。请保证您设置的桶路径有读取权限。在训练作业启动后，ModelArts会挂载硬盘至“/cache”目录，用户可以使用此目录来存储临时文件。“/cache”目录大小请参考**训练环境中不同规格资源“/cache”目录的大小**。

- 训练输出路径参数

  建议设置一个空目录为训练输出路径。在训练代码中，您需要解析**输出路径参数**。系统后台会自动上传训练输出至指定的训练输出路径，请保证您设置的桶路径有写入权限和读取权限。

在ModelArts中，训练代码需包含以下步骤：

（可选）引入依赖

1. 当您使用自定义脚本创建算法的时候，如果您的模型引用了其他依赖，您需要在“算法管理 > 创建算法”的“代码目录”下放置相应的文件或安装包。

- 安装python依赖包请参考**模型中引用依赖包时，如何创建训练作业？**
- 安装C++的依赖库请参考**如何安装C++的依赖库？**
- 在预训练模型中加载参数请参考**如何在训练中加载部分训练好的参数？**

解析输入路径参数、输出路径参数

运行在ModelArts Standard的训练作业会读取存储在OBS服务的数据，或者输出训练结果至OBS服务指定路径，输入和输出数据需要配置2个地方：

1. 训练代码中需解析输入路径参数和输出路径参数。ModelArts Standard推荐以下方式实现参数解析。

```
import argparse
# 创建解析
parser = argparse.ArgumentParser(description='train mnist')

# 添加参数
parser.add_argument('--data_url', type=str, default="./Data/mnist.npz", help='path where the dataset is saved')
parser.add_argument('--train_url', type=str, default="./Model", help='path where the model is saved')

# 解析参数
args = parser.parse_args()
```

   完成参数解析后，用户使用“data_url”、“train_url”代替算法中数据来源和数据输出所需的路径。

2. 在创建训练作业时，填写输入路径和输出路径。

   训练输入选择对应的OBS路径或者数据集路径；训练输出选择对应的OBS路径。

训练代码完整示例

训练代码示例中涉及的代码与您使用的AI引擎密切相关，以下案例以Tensorflow框架为例。案例中使用到的“mnist.npz”文件需要提前**下载**并上传至OBS桶中，训练输入为“mnist.npz”所在OBS路径。

以下训练代码样例中包含了保存模型代码。

```
import os
import argparse
import tensorflow as tf

parser = argparse.ArgumentParser(description='train mnist')
parser.add_argument('--data_url', type=str, default="./Data/mnist.npz", help='path where the dataset is saved')
parser.add_argument('--train_url', type=str, default="./Model", help='path where the model is saved')
args = parser.parse_args()

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data(args.data_url)
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
])

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
```

```
model.save(os.path.join(args.train_url, 'model'))
```

8.2.3 开发用于自定义镜像训练的代码

当ModelArts Standard提供的预置框架不满足您的诉求时，ModelArts Standard支持用户构建自定义镜像用于模型训练。

自定义镜像的制作要求用户对容器相关知识有比较深刻的了解，除非订阅算法和预置框架无法满足需求，否则不推荐使用。自定义镜像需上传至容器镜像服务（SWR），才能用于ModelArts Standard上训练。

自定义镜像的启动命令规范

用户遵循ModelArts镜像的规范要求制作镜像，选择自己的镜像，并且通过指定代码目录（可选）和启动命令的方式来创建的训练作业。

图 8-3 创建训练作业选择自定义方式

说明

当使用完全自定义镜像创建训练作业时，“启动命令”必须在“/home/ma-user”目录下执行，否则训练作业可能会运行异常。

在完全使用自定义镜像创建训练作业时，通过指定的“conda env”启动训练。由于训练作业运行时不是shell环境，因此无法直接使用“conda activate”命令激活指定的“conda env”，需要使用其他方式以达成使用指定“conda env”来启动训练的效果。假设您的自定义镜像中的“conda”安装于“/home/ma-user/anaconda3”目录“conda env”为“python-3.7.10”，训练脚本位于“/home/ma-user/modelarts/user-job-dir/code/train.py”。可通过以下方式使用指定的“conda env”启动训练：

- 方式一：为镜像设置正确的“DEFAULT_CONDA_ENV_NAME”环境变量与“ANACONDA_DIR”环境变量。

  ```
  ANACONDA_DIR=/home/ma-user/anaconda3
  DEFAULT_CONDA_ENV_NAME=python-3.7.10
  ```

  您可以使用Python命令启动训练脚本。启动命令示例如下：

  ```
  python /home/ma-user/modelarts/user-job-dir/code/train.py
  ```

- 方式二：使用“conda env python”的绝对路径。

  您可以使用“/home/ma-user/anaconda3/envs/python-3.7.10/bin/python”命令启动训练脚本。启动命令示例如下：

```
/home/ma-user/anaconda3/envs/python-3.7.10/bin/python /home/ma-user/modelarts/user-job-dir/code/train.py
```

- 方式三：设置PATH环境变量。

  您可以将指定的“conda env bin”目录配置到PATH环境变量中。您可以使用Python命令启动训练脚本。启动命令示例如下：

  ```
  export PATH=/home/ma-user/anaconda3/envs/python-3.7.10/bin:$PATH; python /home/ma-user/modelarts/user-job-dir/code/train.py
  ```

- 方式四：使用“conda run -n”命令。

  您可以使用“/home/ma-user/anaconda3/bin/conda run -n python-3.7.10”命令来执行训练命令，启动命令示例如下：

  ```
  /home/ma-user/anaconda3/bin/conda run -n python-3.7.10 python /home/ma-user/modelarts/user-job-dir/code/train.py
  ```

说明

如果在训练时发生找不到“$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib ”目录下“.so”文件的相关报错，可以尝试将该目录加入到“LD_LIBRARY_PATH”，将以下命令放在上述启动方式命令前：

```
export LD_LIBRARY_PATH=$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib:$LD_LIBRARY_PATH;
```

例如，方式一的启动命令示例此时变为：

```
export LD_LIBRARY_PATH=$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib:$LD_LIBRARY_PATH; python /home/ma-user/modelarts/user-job-dir/code/train.py
```

使用 Ascend 自定义镜像训练时的训练代码适配规范

使用NPU资源创建训练作业时，系统会在训练容器里自动生成Ascend HCCL RANK_TABLE_FILE文件。当使用预置框架创建训练作业时，在训练过程中预置框架会自动解析Ascend HCCL RANK_TABLE_FILE文件，当使用自定义镜像创建训练作业时，就要适配训练代码使得训练过程中在代码里读取解析Ascend HCCL RANK_TABLE_FILE文件。

Ascend HCCL RANK_TABLE_FILE文件说明

Ascend HCCL RANK_TABLE_FILE文件提供Ascend分布式训练作业的集群信息，用于Ascend芯片分布式通信，可以被HCCL集合通信库解析。该文件格式有模板一和模板二两个版本。

- ModelArts提供的是模板二格式。ModelArts训练环境的Ascend HCCL RANK_TABLE_FILE文件名为jobstart_hccl.json，获取方式可以通过预置的RANK_TABLE_FILE环境变量实现。

  表 8-3 RANK_TABLE_FILE 环境变量说明

环境变量	说明
RANK_TABLE_FILE	该环境变量指示Ascend HCCL RANK_TABLE_FILE文件所在目录，值为/user/config。 算法开发者可通过 “${RANK_TABLE_FILE}/jobstart_hccl.json”，路径获取该文件。

  ModelArts训练环境jobstart_hccl.json文件内容（模板二）示例：

  ```
  {
     "group_count": "1",
  ```



```
  "group_list": [{
    "device_count": "1",
    "group_name": "job-trainjob",
    "instance_count": "1",
    "instance_list": [{
      "devices": [{
        "device_id": "4",
        "device_ip": "192.1.10.254"
      }],
      "pod_name": "jobxxxxxxxx-job-trainjob-0",
      "server_id": "192.168.0.25"
    }]
  }],
  "status": "completed"
}
```

jobstart_hccl.json文件中的status字段的值在训练脚本启动时，并不一定为completed状态。因此需要训练脚本等待status字段的值等于completed之后，再去读取文件的剩余内容。

- 通过训练脚本，可以使用模板一格式的jobstart_hccl.json文件，在等待status字段的值等于completed之后，将模板二格式jobstart_hccl.json文件转换为模板一格式的jobstart_hccl.json文件。

  转换后的jobstart_hccl.json文件格式（模板一）示例：

```
{
  "server_count": "1",
  "server_list": [{
    "device": [{
      "device_id": "4",
      "device_ip": "192.1.10.254",
      "rank_id": "0"
    }],
    "server_id": "192.168.0.25"
  }],
  "status": "completed",
  "version": "1.0"
}
```

训练作业在容器中的挂载点说明

使用自定义镜像训练模型时，训练作业在容器中的挂载点参考如表8-4所示。

表 8-4 训练作业挂载点介绍

挂载点	是否只读	备注
/xxx	否	专属池使用SFS盘挂载的目录，路径由客户自己指定。
/home/ma-user/modelarts	否	空文件夹，建议用户主要用这个目录。
/cache	否	裸机规格支持，挂载宿主机NVMe的硬盘。
/dev/shm	否	用于PyTorch引擎加速。
/usr/local/nvidia	是	宿主机的nvidia库。

8.2.4 自定义镜像训练作业配置节点间 SSH 免密互信

当用户使用基于MPI和Horovod框架的自定义镜像进行分布式训练时，需配置训练作业节点间SSH免密互信，否则训练会失败。

配置节点间SSH免密互信涉及代码适配和训练作业参数配置，本文提供了一个操作示例。

1. 准备一个预装OpenSSH的自定义镜像，使用的训练框架是MPI或Horovod。
2. 准备一个sshd启动脚本文件“start_sshd.sh”。
   ```
   MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}
   mkdir -p /home/ma-user/etc
   ssh-keygen -f /home/ma-user/etc/ssh_host_rsa_key0 -N '' -t rsa > /dev/null
   /usr/sbin/sshd -p $MY_SSHD_PORT -h /home/ma-user/etc/ssh_host_rsa_key0
   ```
3. 将准备好的sshd启动脚本文件上传至OBS的训练代码目录下。
4. 创建自定义镜像训练作业。
   - “代码目录”选择存有sshd启动脚本文件的OBS地址。
   - “启动命令”需要适配sshd启动脚本，如下所示：
     ```
     bash ${MA_JOB_DIR}/demo-code/start_sshd.sh && your custom command
     ```
     命令中的“your custom command”表示训练作业中需要执行的其他自定义命令。
   - “环境变量”增加“MY_SSHD_PORT = 38888”。
   - “配置节点间SSH免密互信”开关打开，并设置“SSH密钥目录”，一般保持默认值。该配置会在下发训练作业后，自动在训练容器的“/home/ma-user/.ssh”目录下生成SSH密钥文件和配置文件“authorized_keys config id_rsa id_rsa.pub”。
5. 提交创建训练作业后，训练过程中，训练作业的节点可通过域名+端口的方式SSH连接到其他节点，示例代码如下所示：
   ```
   ssh modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-1 -p $MY_SSHD_PORT
   ```

8.3 准备模型训练镜像

ModelArts平台提供了Tensorflow，PyTorch，MindSpore等常用深度学习任务的基础镜像，镜像里已经安装好运行任务所需软件。当基础镜像里的软件无法满足您的程序运行需求时，您还可以基于这些基础镜像制作一个新的镜像并进行训练。

训练作业的预置框架介绍

ModelArts中预置的训练基础镜像如下表所示。

表 8-5 ModelArts 训练基础镜像列表

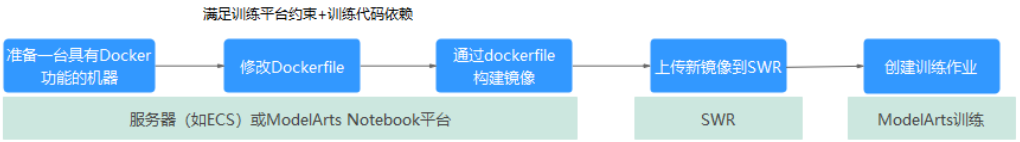
引擎类型	版本名称
PyTorch	pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
TensorFlow	tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64
Horovod	horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

引擎类型	版本名称
	horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
MPI	mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

构建自定义训练镜像

当基础镜像里的软件无法满足您的程序运行需求时，您还可以基于这些基础镜像制作一个新的镜像并进行训练。镜像制作流程如图8-4所示。

图 8-4 训练作业的自定义镜像制作流程



场景一：预置镜像满足ModelArts训练平台约束，但不满足代码依赖的要求，需要额外安装软件包。

具体案例参考**使用预置镜像制作自定义镜像用于训练模型**。

场景二：已有本地镜像满足代码依赖的要求，但是不满足ModelArts训练平台约束，需要适配。

具体案例参考**已有镜像迁移至ModelArts用于训练模型**。

场景三： 当前无可使用的镜像，需要从0制作镜像（既需要安装代码依赖，又需要制作出的镜像满足ModelArts平台约束）。具体案例参考：

- **从0制作自定义镜像用于创建训练作业（PyTorch+CPU/GPU）**
- **从0制作自定义镜像用于创建训练作业（MPI+CPU/GPU）**
- **从0制作自定义镜像用于创建训练作业（Tensorflow+GPU）**

8.4 创建调试训练作业

8.4.1 使用 PyCharm ToolKit 创建并调试训练作业

由于AI开发者会使用PyCharm工具开发算法或模型，为方便快速将本地代码提交到ModelArts的训练环境，ModelArts提供了一个PyCharm插件工具PyCharm ToolKit，协助用户完成SSH远程连接Notebook、代码上传、提交训练作业、将训练日志获取到本地展示等，用户只需要专注于本地的代码开发即可。

本章节介绍如何使用PyCharm ToolKit插件创建训练作业并调试。

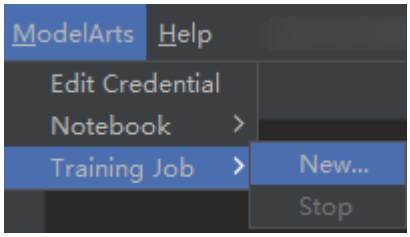
前提条件

- **Step1 下载并安装PyCharm ToolKit**。
- 在本地PyCharm中已有训练代码工程。
- 已在OBS中创建桶和文件夹，用于存放数据集和训练输出模型。 例如：创建命名为“test-modelarts2”的桶，创建文件夹“dataset-mnist”和“mnist-output”。训练作业使用的数据已上传至OBS，且OBS与ModelArts在同一区域。

配置训练作业参数

1. 在PyCharm中，打开训练代码工程和训练启动文件，然后在菜单栏中选择“ModelArts > Training Job > New...”。

   图 8-5 选择作业配置

2. 在弹出的对话框中，设置训练作业相关参数，详细参数说明请参见**表8-6**。

   表 8-6 训练作业配置参数说明

参数	说明
Job Name	训练作业的名称。 系统会自动生成一个名称，您可以根据业务需求重新命名，命名规则如下： • 支持1~64位字符。 • 并包含大小写字母、数字、中划线（-）或下划线（_）。
Job Description	训练作业的简要描述。
Algorithm Source	训练算法来源，分为“常用框架”和“自定义镜像”两种，二者选一项即可。 常用框架指使用ModelArts训练管理中支持的常用AI引擎，当前支持的引擎列表请参见**ModelArts支持的预置镜像列表**。 如果您使用的AI引擎为支持列表之外的，建议使用自定义镜像的方式创建训练作业。
AI Engine	选择代码使用的AI引擎及其版本。支持的AI引擎与ModelArts管理控制台里**ModelArts支持的预置镜像列表**一致。
Boot File Path	训练启动文件，所选启动文件必须是当前PyCharm训练工程中的文件。当“Algorithm source”选“Frequently-used”时，显示此参数。

参数	说明
Code Directory	训练代码目录，系统会自动填写为训练启动文件所在的目录，用户可根据需要修改，所选目录必须是当前工程中的目录且包含启动文件。 当算法来源为自定义镜像，训练代码已预置在镜像中时，该参数可以为空。
Image Path(optional)	SWR镜像的URL地址。
Boot Command	启动本次训练作业的运行命令。例如“bash /home/work/run_train.sh python {python启动文件及参数}”。当“Algorithm source”选“Custom”时，显示此参数。 当用户输入的命令中不包含“--data_url”和“--train_url”参数时，工具在提交训练作业时会在命令后面自动添加这两个参数，分别对应存储训练数据的OBS路径和存放训练输出的OBS路径。
Data OBS Path	设置为存储训练数据的OBS路径，例如“/test-modelarts2/mnist/dataset-mnist/”，其中“test-modelarts2”为桶名称。
Training OBS Path	设置OBS路径，该路径下会自动创建用于存放训练输出模型和训练日志的目录。
Running Parameters	运行参数。如果您的代码需要添加一些运行参数，可以在此处添加，多个运行参数使用英文分号隔开， 例如"key1=value1;key2=value2"。此参数也可以不设置，即保持为空。
Specifications	训练使用资源类型。目前支持公共资源池和专属资源池两种类型。 专属资源池规格以“Dedicated Resource Pool”标识。只有购买了专属资源池的用户才会显示专属资源池规格。
Compute Nodes	计算资源节点个数。数量设置为1时，表示单机运行；数量设置大于1时，表示后台的计算模式为分布式。
Available/Total Nodes	当“Specifications”选择专属资源池规格时，显示专属资源池的可用实例数和总实例数，用户选择“Compute Nodes”的个数不要超过可用实例数。

图 8-6 配置训练作业参数（公共资源池）

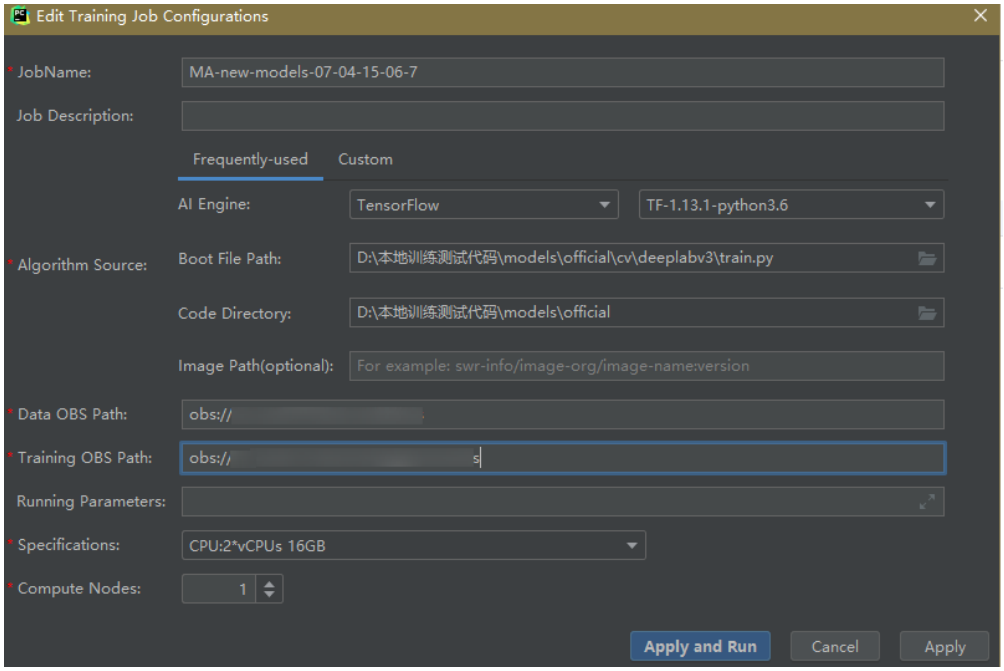


图 8-7 配置训练作业参数（专属资源池）

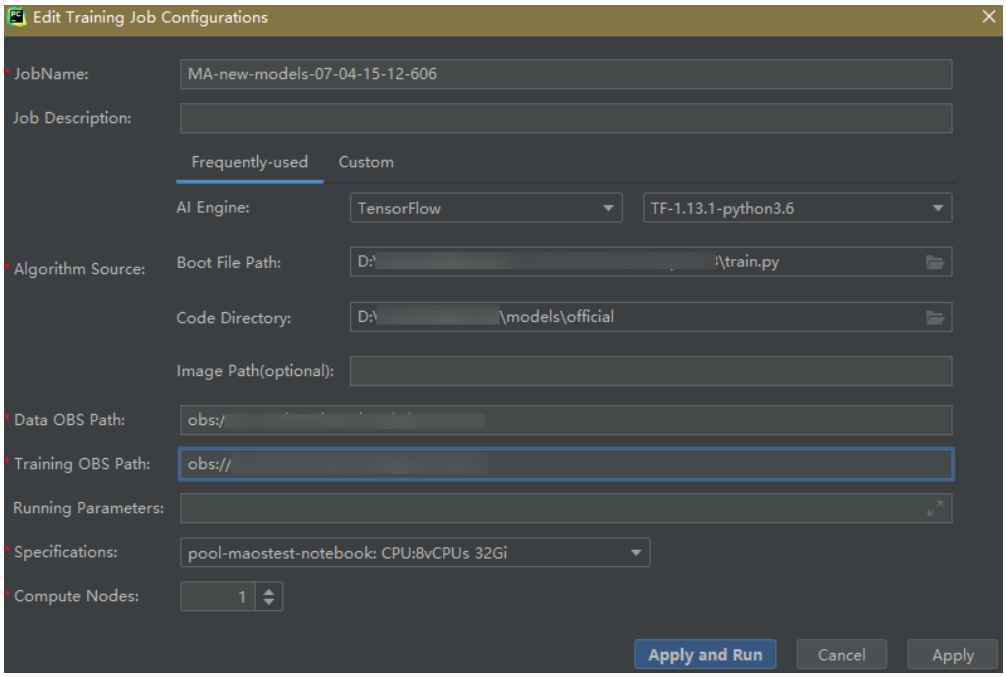
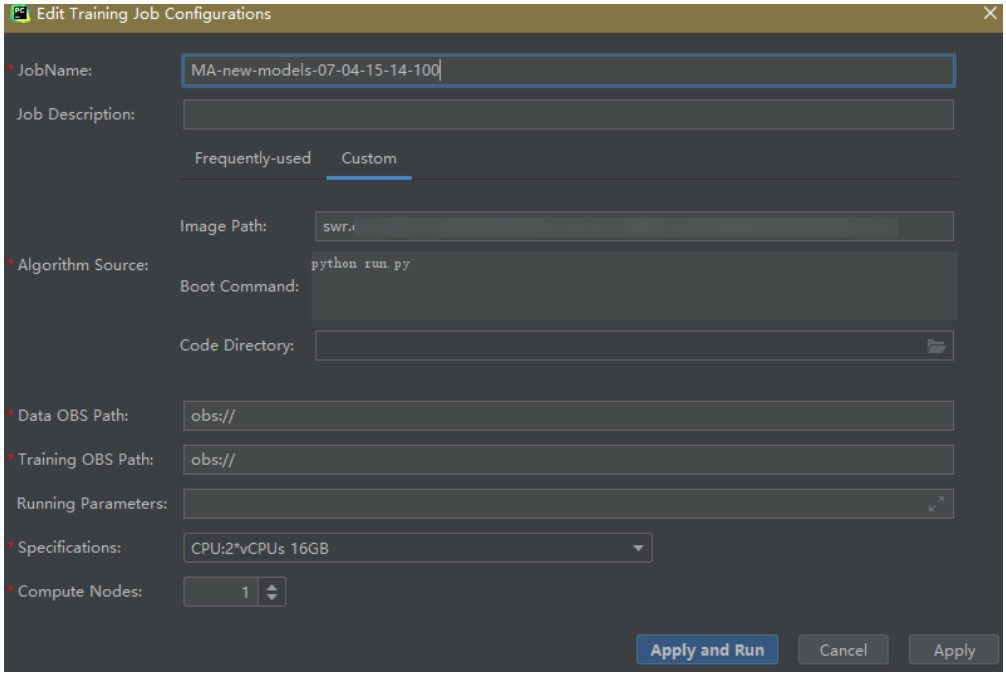


图 8-8 配置训练作业参数（自定义镜像）

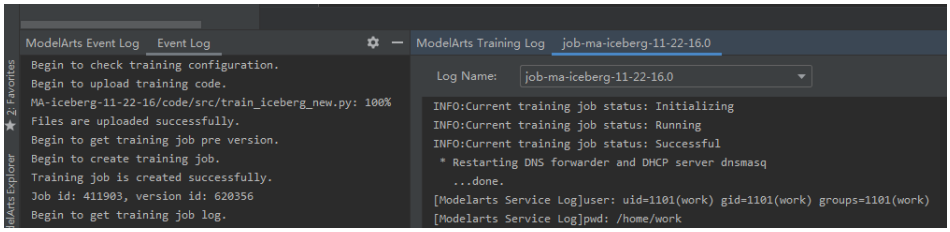


3. 参数填写完成后，单击“Apply and Run”，即自动上传本地代码至云端并启动训练，在工具下方的Training Log区域，会实时展示训练作业运行情况。当训练日志中出现“Current training job status: Successful”类似信息时，表示训练作业运行成功。

   说明

   - 在单击“Apply and Run”按钮后，系统将自动开始执行训练作业。如果您想停止此作业，可以选择菜单栏中的“ModelArts > Training Job >Stop”停止此作业。
   - 如果单击“Apply”，不会直接启动运行，只是保存训练作业的设置，如果需要启动作业，可以单击“Apply and Run”。

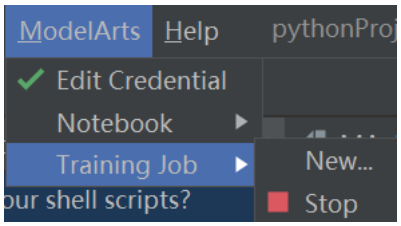
图 8-9 训练日志展示样例



停止作业

当训练作业在运行过程中时，您可以在PyCharm菜单栏中，选择“ModelArts >Training Job > Stop ”停止训练作业。

图 8-10 停止作业



查看训练日志

查看训练日志有2种方式，在OBS查看和在PyCharm ToolKit工具中查看。

- **在OBS查看训练日志**

  提交训练作业时，系统将自动在您配置的OBS Path中，使用作业名称创建一个新的文件夹，用于存储训练输出的模型、日志和代码。

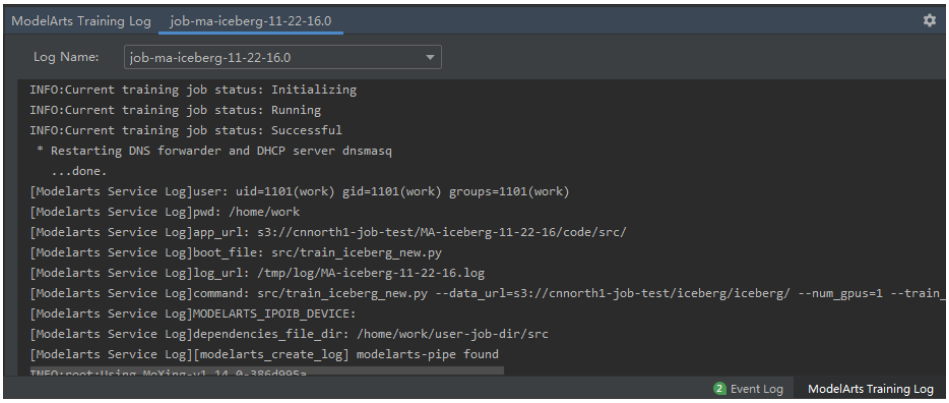
  例如“train-job-01”作业，提交作业时会在“test-modelarts2”桶下创建一个命名为“train-job-01”的文件夹，且此文件夹下分别新建了三个文件夹“output”、“log”、“code”，分别用于存储输出模型、日志和训练代码。“output”文件夹还会根据您的训练作业版本再创建子文件夹，结构示例如下。

```
test-modelarts2
  |---train-job-01
      |---output
      |---log
      |---code
```

- 在PyCharm ToolKit工具中查看训练日志

  在PyCharm ToolKit工具中，单击页面右下角的ModelArts Training Log，展示训练日志。

图 8-11 查看训练日志



8.5 创建算法

机器学习从有限的观测数据中学习一般性的规律，并利用这些规律对未知的数据进行预测。为了获取更准确的预测结果，用户需要选择一个合适的算法来训练模型。针对不同的场景，ModelArts提供大量的算法样例。以下章节提供了关于业务场景、算法学习方式、算法实现方式的指导。

选择算法的实现方式

ModelArts提供如下方式实现模型训练前的算法准备。

- 使用订阅算法

  ModelArts的AI Gallery提供了可以直接订阅的算法，不需要进行代码开发，即可使用现成的算法进行模型构建。

- 使用预置框架

  如果您需要使用自己开发的算法，可以选择使用ModelArts预置框架。ModelArts支持了大多数主流的AI引擎，详细请参见**预置训练引擎**。这些预置引擎预加载了一些额外的python包，例如numpy等；也支持您通过在代码目录中使用“requirements.txt”文件安装依赖包。使用预置框架创建训练作业请参考**开发用于预置框架训练的代码**指导。

- 使用预置框架 + 自定义镜像：

  如果先前基于预置框架且通过指定代码目录和启动文件的方式来创建的算法；但是随着业务逻辑的逐渐复杂，您期望可以基于预置框架修改或增加一些软件依赖的时候，此时您可以使用预置框架 + 自定义镜像的功能，即选择预置框架名称后，在预置框架版本下拉列表中选择“自定义”。

  此功能与直接基于预置框架创建算法的区别仅在于，镜像是由用户自行选择的。用户可以基于预置框架制作自定义镜像。

- 完全自定义镜像：

  订阅算法和预置框架涵盖了大部分的训练场景。针对特殊场景，ModelArts支持用户构建自定义镜像用于模型训练。用户遵循ModelArts镜像的规范要求制作镜像，选择自己的镜像，并且通过指定代码目录（可选）和启动命令的方式来创建的训练作业。

  自定义镜像需上传至容器镜像服务（SWR），才能用于ModelArts上训练，请参考**使用自定义镜像训练模型**。由于自定义镜像的制作要求用户对容器相关知识有比较深刻的了解，除非订阅算法和预置引擎无法满足需求，否则不推荐使用。

  说明

  当使用完全自定义镜像创建训练作业时，“启动命令”必须在“/home/ma-user”目录下执行，否则训练作业可能会运行异常。

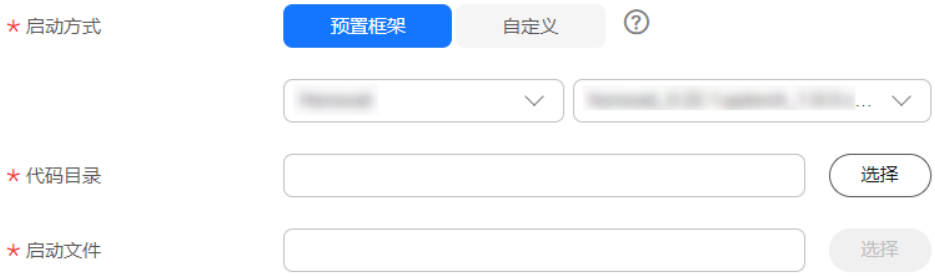
创建算法

您在本地或使用其他工具开发的算法，支持上传至ModelArts中统一管理。

1. 创建算法的准备工作。
   - 完成数据准备：已在ModelArts中创建可用的数据集，或者您已将用于训练的数据集上传至OBS目录。
   - 准备训练脚本，并上传至OBS目录。训练脚本开发指导参见**开发用于预置框架训练的代码**或**开发用于自定义镜像训练的代码**。
   - 在OBS创建至少1个空的文件夹，用于存储训练输出的内容。
   - 确保您使用的OBS目录与ModelArts在同一区域。
2. 进入算法创建页面。
   a. 登录ModelArts管理控制台，单击左侧菜单栏的“资产管理 > 算法管理”。
   b. 在“我的算法”管理页面，单击“创建”，进入“创建算法”页面。填写算法的基本信息，包含“名称”和“描述”。

3. 设置算法启动方式，有以下三种方式可以选择。
   - 设置算法启动方式（预置框架）

图 8-12 使用预置框架创建算法



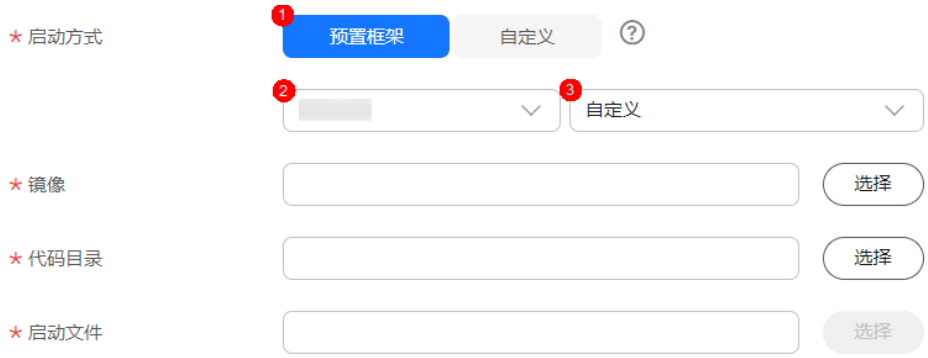
需根据实际算法代码情况设置“代码目录”和“启动文件”。选择的预置框架和编写算法代码时选择的框架必须一致。例如编写算法代码使用的是TensorFlow，则在创建算法时也要选择TensorFlow。

表 8-7 使用预置框架创建算法

参数	说明
“启动方式”	选择“预置框架”。 选择算法使用的预置框架引擎和引擎版本。
“代码目录”	算法代码存储的OBS路径。训练代码、依赖安装包或者预生成模型等训练所需文件上传至该代码目录下。 请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。 训练作业创建完成后，ModelArts会将代码目录及其子目录下载至训练后台容器中。 例如：OBS路径“obs://obs-bucket/training-test/demo-code”作为代码目录，OBS路径下的内容会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。 **说明** • 编程语言不限。 • 文件数（含文件、文件夹数量）小于或等于1000个。 • 文件总大小小于或等于5GB。
“启动文件”	必须为“代码目录”下的文件，且以“.py”结尾，即ModelArts目前只支持使用Python语言编写的启动文件。 代码目录路径中的启动文件为训练启动的入口。

   - 设置算法启动方式（预置框架+自定义）

图 8-13 使用预置框架+自定义镜像创建算法



需根据实际算法代码情况设置“镜像”、“代码目录”和“启动文件”。选择的预置框架和编写算法代码时选择的框架必须一致。例如编写算法代码使用的是TensorFlow，则在创建算法时也要选择TensorFlow。

表 8-8 使用预置框架+自定义镜像创建算法

参数	说明
“启动方式”	选择“预置框架”。 预置框架的引擎版本选择“自定义”。
“镜像”	用户制作的镜像需要提前上传到SWR，才可以在这里选择。制作镜像的方式请参见**训练作业的自定义镜像制作流程**。
“代码目录”	算法代码存储的OBS路径。训练代码、依赖安装包或者预生成模型等训练所需文件上传至该代码目录下。 请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。 训练作业启动时，ModelArts会将训练代码目录及其子目录下载至训练后台容器中。 例如：OBS路径“obs://obs-bucket/training-test/demo-code”作为代码目录，OBS路径下的内容会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。 **说明** • 训练代码编程语言不限。训练启动文件必须为Python语言。 • 文件数（含文件、文件夹数量）小于或等于1000个。 • 文件总大小要小于或等于5GB。 • 文件深度要小于或等于32
“启动文件”	必须为“代码目录”下的文件，且以“.py”结尾，即ModelArts目前只支持使用Python语言编写的启动文件。 代码目录路径中的启动文件为训练启动的入口。

选择预置框架+自定义时，该功能的后台行为与直接基于预置框架运行训练作业相同，例如：

- 系统将会自动注入一系列环境变量
  ```
  PATH=${MA_HOME}/anaconda/bin:${PATH}
  LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
  PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
  ```
- 您选择的启动文件将会被系统自动以python命令直接启动，因此请确保镜像中的Python命令为您预期的Python环境。注意到系统自动注入的PATH环境变量，您可以参考下述命令确认训练作业最终使用的Python版本：
  ```
  export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
  docker run --rm {image} $(which python) -V
  ```
- 系统将会自动添加预置框架关联的超参

– 设置算法启动方式（自定义）

图 8-14 完全使用自定义镜像创建算法

表 8-9 完全使用自定义镜像创建算法

参数	说明
“启动方式”	选择“自定义”。
“镜像”	用户制作的镜像需要提前上传到SWR，才可以在这里选择。制作镜像的方式请参见**训练作业的自定义镜像制作流程**。

参数	说明
“代码目录”	算法代码存储的OBS路径。训练代码、依赖安装包或者预生成模型等训练所需文件上传至该代码目录下。如果自定义镜像中不含训练代码则需要配置该参数，如果自定义镜像中已包含训练代码则不需要配置。 请注意不要将训练数据放在代码目录路径下。训练数据比较大，训练代码目录在训练作业启动后会下载至后台，可能会有下载失败的风险。 训练作业启动时，ModelArts会将训练代码目录及其子目录下载至训练后台容器中。 例如：OBS路径“obs://obs-bucket/training-test/demo-code”作为代码目录，OBS路径下的内容会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，demo-code为OBS存放代码路径的最后一级目录，用户可以根据实际修改。 **说明** • 训练代码编程语言不限。训练启动文件必须为Python语言。 • 文件数（含文件、文件夹数量）小于或等于1000个。 • 文件总大小要小于或等于5GB。 • 文件深度要小于或等于32
“启动命令”	必填，镜像的启动命令。 运行训练作业时，当“代码目录”下载完成后，“启动命令”会被自动执行。 • 如果训练启动脚本用的是py文件，例如“train.py”，则启动命令如下所示。 `python ${MA_JOB_DIR}/demo-code/train.py` • 如果训练启动脚本用的是sh文件，例如“main.sh”，则启动命令如下所示。 `bash ${MA_JOB_DIR}/demo-code/main.sh` 启动命令支持使用“;”和“&&”拼接多条命令，命令中的“demo-code”为存放代码目录的最后一级OBS目录，以实际情况为准。

训练支持的自定义镜像使用说明请参考**自定义镜像的启动命令规范**。

4. 输入输出管道设置。

   训练过程中，算法需要从OBS桶或者数据集中获取数据进行模型训练，训练产生的输出结果也需要存储至OBS桶中。用户的算法代码中需解析输入输出参数实现ModelArts后台与OBS的数据交互，用户可以参考**准备模型训练代码**完成适配ModelArts训练的代码开发。

   - 输入配置

表 8-10 输入配置

参数	参数说明
参数名称	根据实际代码中的输入数据参数定义此处的名称。此处设置的代码路径参数必须与算法代码中解析的训练输入数据参数保持一致，否则您的算法代码无法获取正确的输入数据。 例如，算法代码中使用argparse解析的data_url作为输入数据的参数，那么创建算法时就需要配置输入数据的参数名称为“data_url”。
描述	输入参数的说明，用户可以自定义描述。
获取方式	输入参数的获取方式，默认使用“超参”，也可以选择“环境变量”。
输入约束	开启后，用户可以根据实际情况限制数据输入来源。输入来源可以选择“数据存储位置”或者“ModelArts数据集”。 如果用户选择数据来源为ModelArts数据集，还可以约束以下三种： • 标注类型。数据类型请参考**标注数据**。 • 数据格式。可选“Default”和“CarbonData”，支持多选。其中“Default”代表Manifest格式。 • 数据切分。仅“图像分类”、“物体检测”、“文本分类”和“声音分类”类型数据集支持进行数据切分功能。可选“仅支持切分的数据集”、“仅支持未切分数据集”和“无限制”。数据切分详细内容可参考**发布数据版本**。
添加	用户可以根据实际算法添加多个输入数据来源。

- 输出配置

表 8-11 输出配置

参数	参数说明
参数名称	根据实际代码中的训练输出参数定义此处的名称。此处设置的代码路径参数必须与算法代码中解析的训练输出参数保持一致，否则您的算法代码无法获取正确的输出路径。 例如，算法代码中使用argparse解析的train_url作为训练输出数据的参数，那么创建算法时就需要配置输出数据的参数名称为“train_url”。
描述	输出参数的说明，用户可以自定义描述。
获取方式	输出参数的获取方式，默认使用“超参”，也可以选择“环境变量”。
添加	用户可以根据实际算法添加多个输出数据路径。

5. 定义超参。

创建算法时，ModelArts支持用户自定义超参，方便用户查阅或修改。定义超参后会体现在启动命令中，以命令行参数的形式传入您的启动文件中。

a. 单击“增加超参”手动添加超参。

b. 编辑超参。

说明

为保证数据安全，请勿输入敏感信息，例如明文密码。

表 8-12 超参编辑参数

参数	说明
名称	填入超参名称。 超参名称支持64个以内字符，仅支持大小写字母、数字、下划线和中划线。
类型	填入超参的数据类型。支持String、Integer、Float和Boolean。
默认值	填入超参的默认值。创建训练作业时，默认使用该值进行训练。
约束	单击“约束”。在弹出对话框中，支持用户设置默认值的取值范围或者枚举值范围。
必须	选择是或否。 • 选择否，则在使用该算法创建训练作业时，支持在创建训练作业页面删除该超参。 • 选择是，则在使用该算法创建训练作业时，不支持在创建训练作业页面删除该超参。
描述	填入超参的描述说明。 超参描述支持大小写字母、中文、数字、空格、中划线、下划线、中英文逗号和中英文句号。

6. 支持的策略。

   ModelArts支持用户使用自动化搜索功能。自动化搜索功能在零代码修改的前提下，自动找到最合适的超参，有助于提高模型精度和收敛速度。详细的参数配置请参考**创建自动模型优化的训练作业**。

   自动搜索目前仅支持“tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64”和“pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64”镜像

7. 添加训练约束。

   用户可以根据实际情况定义此算法的训练约束。

   - 资源类型：选择适用的资源类型，支持多选。
   - 多卡训练：选择是否支持多卡训练。
   - 分布式训练：选择是否支持分布式训练。

8. 当创建算法的参数配置完成后，单击“提交”，返回算法管理列表。

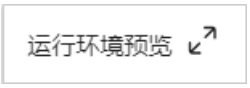
   在“我的算法”列表，单击算法名称进入详情页，可以查看算法详细信息。

   - 选择“基本信息”页签可以查看算法信息。

     “基本信息”页签，单击“编辑”，支持修改除名称和ID之外的算法信息。修改完成，单击“保存”即可完成修改。

- 选择“训练列表”页签可以查看使用该算法的训练作业信息，例如训练作业名称、状态。

运行环境预览

创建算法时，可以打开创建页面右下方的运行环境预览窗口 

，辅助您了解代码目录、启动文件、输入输出等数据配置在训练容器中的路径。

删除算法

须知

删除后，创建的算法资产会被删除，且无法恢复，请谨慎操作。

删除我的算法：在“资产管理 > 算法管理 > 我的算法”页面，“删除”运行结束的训练作业。您可以单击“操作”列的“删除”，在弹出的提示框中单击“确定”，删除对应的算法。

删除订阅算法：前往AI Gallery，在“我的资产 > 算法”中，单击我的订阅，对需要删除的算法单击“取消订阅”，在弹出的提示框中单击“确定”即可。

8.6 创建生产训练作业

模型训练是一个不断迭代和优化模型权重的过程。ModelArts的训练模块支持创建训练作业、查看训练情况以及管理训练版本。通过模型训练试验模型结构、数据和超参的各种组合，便于找到最佳的模型结构和权重。

创建生产环境的训练作业有2种方式：

- 通过ModelArts Standard控制台的方式创建生产环境的训练作业，详细操作请参考本章节以下内容。
- 通过ModelArts提供的API接口创建生产环境的训练作业，详细操作请参见**使用API创建训练作业**。

前提条件

- 已经将用于训练作业的数据上传至OBS目录。
- 已经在OBS目录下创建了至少1个空的文件夹，用于存储训练输出的内容。

  说明

  ModelArts不支持加密的OBS桶，创建OBS桶时，请勿开启桶加密。
- 由于训练作业运行需消耗资源，为了避免训练失败请确保账户未欠费。
- 确保使用的OBS目录与ModelArts在同一区域。
- 检查是否配置了访问授权。如果未配置，请参见**配置ModelArts Standard访问授权**完成操作。
- 已经准备好训练算法，具体操作请参见**创建算法**。

操作流程介绍

创建训练作业的操作步骤如下所示。

步骤1 **进入创建训练作业页面**。

步骤2 **配置训练作业基本信息**。

步骤3 根据不同的算法来源，选择不同的训练作业创建方式。

- 使用已有算法创建训练作业：**选择创建方式（使用我的算法）**
- 使用预置镜像创建训练作业：**选择创建方式（自定义算法）**
- 使用自定义镜像创建训练作：**选择创建方式（使用自定义镜像）**

步骤4 **配置训练参数**：配置训练作业的输入、输出、超参、环境变量等参数。

步骤5 根据需要选择不同的资源池用于训练作业，推荐使用专属资源池，两者的差异说明请参见**专属资源池和公共资源池的能力差异**。

- **配置资源池（公共资源池）**
- **配置资源池（专属资源池）**

步骤6 **（可选）选择训练模式**：当训练作业的算法框架选用的是预置框架的MindSpore类引擎、资源池类型选用的是Ascend资源时，则支持选择训练模式。

步骤7 **（可选）设置标签**：如果需要对训练作业进行资源分组管理，可以设置标签。

步骤8 **后续操作**。

----结束

进入创建训练作业页面

1. 登录ModelArts管理控制台。
2. 在左侧导航栏中，选择“模型训练 > 训练作业”进入训练作业列表。
3. 单击“创建训练作业”，进入创建训练作业页面。

配置训练作业基本信息

在创建训练作业页面填写训练作业基本信息。

表 8-13 创建训练作业的基本信息

参数名称	说明
名称	必填，训练作业的名称。 系统会自动生成一个名称，可以根据业务需求重新命名，命名规则如下： • 支持1~64位字符。 • 可以包含大小写字母、数字、中划线（-）或下划线（_）。
描述	训练作业的简介，便于在训练作业列表了解作业信息。

选择创建方式（使用我的算法）

如果选择使用已有算法创建训练作业，则“创建方式”选择“我的算法”，在算法列表中选择算法。如果没有满足条件的算法，也可以新建算法，具体操作请参见**创建算法**。

选择创建方式（自定义算法）

如果在算法管理中已经创建算法，此处建议选择“我的算法”页签中已经准备好的算法。如果没有已经准备好的算法，此处可以选择“自定义算法”创建方式。如果选择使用自定义算法创建训练作业，则参考**表8-14**选择训练作业的创建方式。

表 8-14 创建训练作业的创建方式（使用自定义算法）

参数名称	说明
创建方式	必选，选择“自定义算法”。
启动方式	必选，选择“预置框架”，并选择训练作业要使用的预置框架引擎和引擎版本。 如果引擎版本选择“自定义”，则需要配置“镜像”参数，选择自定义镜像用于训练作业。
镜像	仅当预置框架的引擎版本选择“自定义”时才显示该参数，且是必填参数。 容器镜像地址的填写支持如下方式。 • 选择自有镜像或他人共享的镜像：单击右边的“选择”，从容器镜像中选择用于训练的容器镜像。所需镜像需要提前上传到SWR服务中。 • 选择公开镜像：直接输入SWR服务中公开镜像的地址。地址直接填写“组织名称/镜像名称:版本名称”，不需要带域名信息，系统会自动拼接域名地址。
代码来源	选择训练代码来源。 • 对象OBS存储：如果训练代码存放在OBS中，则选择“对象OBS存储”。 • 文件存储：如果训练代码存放在文件存储中，则选择“文件存储”。
代码目录	仅当“代码来源”选择“对象OBS存储”时才显示该参数。 必填，选择训练代码文件所在的OBS目录。 • 需要提前将代码上传至OBS桶中，目录内文件总大小要小于或等于5GB，文件数要小于或等于1000个，文件深度要小于或等于32。 • 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“${MA_JOB_DIR}/code”目录中。

参数名称	说明
启动文件	必填，选择代码目录中训练作业的Python启动脚本。 ModelArts只支持使用Python语言编写的启动文件，因此启动文件必须以“.py”结尾。
本地代码目录	仅当“代码来源”选择“对象OBS存储”时才显示该参数。 指定训练容器的本地目录，启动训练时系统会将代码目录下载至此目录。 此参数可选，默认本地代码目录为“/home/ma-user/modelarts/user-job-dir”。
工作目录	训练时，系统会自动cd到此目录下执行启动文件。

选择预置框架+自定义时，该功能的后台行为与直接基于预置框架运行训练作业相同，例如：

- 系统将会自动注入一系列环境变量
  ```
  PATH=${MA_HOME}/anaconda/bin:${PATH}
  LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
  PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
  ```
- 您选择的启动文件将会被系统自动以python命令直接启动，因此请确保镜像中的Python命令为您预期的Python环境。注意到系统自动注入的PATH环境变量，您可以参考下述命令确认训练作业最终使用的Python版本：
  ```
  export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
  docker run --rm {image} $(which python) -V
  ```
- 系统将会自动添加预置框架关联的超参

选择创建方式（使用自定义镜像）

如果选择使用自定义镜像创建训练作业，则参考**表8-15**选择训练作业的创建方式。

表 8-15 创建训练作业的创建方式（使用自定义镜像）

参数名称	说明
创建方式	必选，选择“自定义算法”。
启动方式	必选，选择“自定义”。
镜像	必填，填写容器镜像的地址。 容器镜像地址的填写支持如下方式。 ● 选择自有镜像或他人共享的镜像：单击右边的“选择”，从容器镜像中选择用于训练的容器镜像。所需镜像需要提前上传到SWR服务中。 ● 选择公开镜像：直接输入SWR服务中公开镜像的地址。地址直接填写“组织名称/镜像名称:版本名称”，不需要带域名信息，系统会自动拼接域名地址。

参数名称	说明
代码目录	选择训练代码文件所在的OBS目录。如果自定义镜像中不含训练代码则需要配置该参数，如果自定义镜像中已包含训练代码则不需要配置。 ● 需要提前将代码上传至OBS桶中，目录内文件总大小要小于或等于5GB，文件数要小于或等于1000个，文件深度要小于或等于32。 ● 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“${MA_JOB_DIR}/code”目录中。
运行用户ID	容器运行时的用户ID，该参数为选填参数，建议使用默认值1000。 如果需要指定uid，则uid数值需要在规定范围内，不同资源池的uid范围如下： ● 公共资源池：1000-65535 ● 专属资源池：0-65535
启动命令	必填，镜像的启动命令。 运行训练作业时，当“代码目录”下载完成后，“启动命令”会被自动执行。 ● 如果训练启动脚本用的是py文件，例如“train.py”，则启动命令如下所示。 `python ${MA_JOB_DIR}/demo-code/train.py` ● 如果训练启动脚本用的是sh文件，例如“main.sh”，则启动命令如下所示。 `bash ${MA_JOB_DIR}/demo-code/main.sh` 启动命令支持使用“;”和“&&”拼接多条命令，命令中的“demo-code”为存放代码目录的最后一级OBS目录，以实际情况为准。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。
本地代码目录	仅当“代码来源”选择“对象OBS存储”时才显示该参数。 指定训练容器的本地目录，启动训练时系统会将代码目录下载至此目录。 此参数可选，默认本地代码目录为“/home/ma-user/modelarts/user-job-dir”。
工作目录	训练时，系统会自动cd到此目录下执行启动文件。

训练支持的自定义镜像使用说明请参考**自定义镜像的启动命令规范**。

配置训练参数

训练过程中可以从OBS桶或者数据集中获取输入数据进行模型训练，训练输出的结果也支持存储至OBS桶中。创建训练作业时可以参考表8-16配置输入、输出、超参、环境变量等参数。

说明

创建训练作业时选择的创建方式不同，训练作业的输入、输出和超参显示不同。如果参数值置灰，即表示该参数已经在算法代码中配置了且不支持修改。

表 8-16 配置训练参数

参数名称	子参数	说明
输入	参数名称	算法代码需要通过“输入”的“参数名称”去读取训练的输入数据。 建议设置为“data_url”。训练输入参数要与所选算法的“输入”参数匹配，请参见创建算法时的表8-10。
	数据集	单击“数据集”，在ModelArts数据集列表中勾选目标数据集并选择对应的版本。 训练启动时，系统将自动下载输入路径中的数据到训练运行容器。 **说明** ModelArts数据管理模块在重构升级中，对未使用过数据管理的用户不可见。建议新用户将训练数据存放至OBS桶中使用。
	数据存储位置	单击“数据存储位置”，从OBS桶中选择训练输入数据的存储位置。文件总大小要小于或等于10GB，文件数要小于或等于1000个，单个文件大小要小于或等于1GB。 训练启动时，系统将自动下载输入路径中的数据到训练运行容器。
	获取方式	以参数名称为“data_path”的训练输入为例，说明获取方式的作用。 • 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 `import argparse` `parser = argparse.ArgumentParser()` `parser.add_argument('--data_path')` `args, unknown = parser.parse_known_args()` `data_path = args.data_path` • 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 `import os` `data_path = os.getenv("data_path", "")`
输出	参数名称	算法代码需要通过“输出”的“参数名称”去读取训练的输出目录。 建议设置为“train_url”。训练输出参数要与所选算法的“输出”参数匹配，请参见创建算法时的表8-11。

参数名称	子参数	说明
	数据存储位置	单击“数据存储位置”，从OBS桶中选择训练输出数据的存储位置。文件总大小要小于或等于1GB，文件数要小于或等于128个，单个文件大小要小于或等于128MB。 训练过程中，系统将自动从训练容器的本地代码目录下同步文件到数据存储位置。 **说明** 数据存储位置仅支持OBS路径。为避免数据存储冲突，建议选择一个空目录用作“数据存储位置”。
	获取方式	以参数名称为“train_url”的训练输出为例，说明获取方式的作用。 ● 当参数的“获取方式”为“超参”时，可以参考如下代码来读取数据。 `import argparse` `parser = argparse.ArgumentParser()` `parser.add_argument('--train_url')` `args, unknown = parser.parse_known_args()` `train_url = args.train_url` ● 当参数的“获取方式”为“环境变量”时，可以参考如下代码来读取数据。 `import os` `train_url = os.getenv("train_url", "")`
	预下载至本地目录	选择是否将输出目录下的文件预下载至本地目录。 ● 不下载：表示启动训练作业时不会将输出数据的存储位置中的文件下载到训练容器的本地代码目录中。 ● 下载：表示系统会在启动训练作业时自动将输出数据的存储位置中的所有文件下载到训练容器的本地代码目录中。下载时间会随着文件变大而变长，为了防止训练时间过长，请及时清理训练容器的本地代码目录中的无用文件。如果要使用**设置断点续训练**，则必须选择“下载”。
超参	-	超参用于训练调优。此参数由选择的算法决定，如果在算法中已经定义了超参，则此处会显示算法中所有的超参。 超参支持修改和删除，状态取决于算法中的超参“约束”设置，详情请参见**表8-12**。 单击“本地上传”可以本地批量导入超参，需要按模板填写超参且总数不能超过100条，否则会导入失败。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。
环境变量	-	根据业务需求增加环境变量。训练容器中预置的环境变量请参见**管理训练容器环境变量**。 单击“本地上传”可以本地批量导入环境变量，需要按模板填写环境变量且总数不能超过100条，否则会导入失败。 **说明** 为保证数据安全，请勿输入敏感信息，例如明文密码。

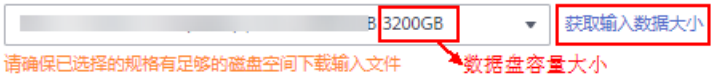
参数名称	子参数	说明
自动重启	-	打开开关后，可以设置重启次数和是否启用无条件自动重启。 打开自动重启开关后，当由于环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力，开启此功能前请确认代码已适配断点续训，操作指导请参见**设置断点续训练**。 “重启次数”的取值范围是1~128，缺省值为3。创建训练后不支持修改重启次数，请合理设置次数。 勾选“无条件自动重启”后，只要系统检测到训练异常，就无条件重启训练作业。为了避免无效重启浪费算力资源，系统最多只支持连续无条件重启3次。 系统支持自动监控作业进程的状态和资源利用率来判定作业是否卡死，开启“作业卡死重启”开关后，支持将标记为卡死的作业进行进程级自动重启，以提高资源使用率。因系统无法核实代码逻辑且检测存在周期性，卡死检测存在一定的误报概率，开启开关即表示接受误报率。为了避免无效重启浪费算力资源，系统最多只支持连续作业卡死重启3次。 当训练过程中触发了自动重启，则系统会记录重启信息，在训练作业详情页可以查看故障恢复详情，具体请参见**训练作业重调度**。

配置资源池（公共资源池）

如果使用公共资源池创建训练作业，则参考**表8-17**配置公共资源池。

表 8-17 创建训练作业的公共资源池

参数名称	说明
资源池	必选，选择“公共资源池”。
资源类型	必选，选择训练需要的资源类型。当训练代码中已定义资源类型时，则根据算法的约束条件选择合适的资源类型。例如，训练代码中定义的资源类型为CPU，这里选择其他类型时会导致训练失败。如果部分资源类型不可见或不可选，表示不支持。

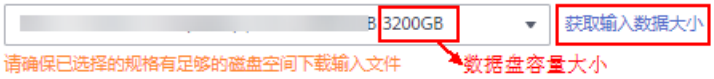
参数名称	说明
实例规格	必选，根据不同的资源类型，选择所需的资源规格。 当“输入”参数选择“数据存储位置”时，在选择资源池规格时可以单击右侧的“获取输入数据大小”，检查输入数据的大小是否超出数据盘的容量限制，避免训练过程中出现内存不足的情况。  **须知** 资源规格为“GPU:n*tnt004”（n表示具体数字）的资源不支持多进程的训练作业。
实例数	必填，根据需要选择实例数的个数。默认值为“1”。 ● 当“实例数 = 1”时，创建的是单机训练作业，ModelArts只会在一个节点上启动一个训练容器，该训练容器独享所选规格的计算资源。 ● 当“实例数 > 1”时，创建的是分布式训练作业，更多分布式训练配置请参见**分布式训练功能介绍**。
永久保存日志	选择CPU或者GPU资源时，支持选择是否关闭“永久保存日志”开关。 ● 开关打开（默认打开）：表示永久保存日志，此时必须配置“作业日志路径”，系统会将训练日志永久保存至指定的OBS路径。 ● 开关关闭：表示不永久保存日志，则训练日志会在30天后会被清理。可以在作业详情页下载全部日志至本地。
作业日志路径	打开“永久保存日志”开关时，必须配置“作业日志路径”，用于存放训练作业产生的日志文件。 建议选择一个空的OBS文件目录存放运行中产生的日志文件，同时需要OBS文件目录的读写权限。

参数名称	说明
事件通知	选择是否打开“事件通知”开关。 ● 开关关闭（默认关闭）：表示不启用消息通知服务。 ● 开关打开：表示订阅消息通知服务，当训练作业发生特定事件（如作业状态变化或疑似卡死）时会发送通知。此时必须配置“主题名”和“事件”。 – “主题名”：事件通知的主题名称。单击“创建主题”，前往消息通知服务中创建主题。 – “事件”：选择要订阅的事件类型。例如“作业开始”、“作业结束”、“作业失败”、“作业终止”、“作业疑似卡死”等。 **说明** ● 需要为消息通知服务中创建的主题添加订阅，当订阅状态为“已确认”后，方可收到事件通知。订阅主题的详细操作请参见**添加订阅**。 ● 使用消息通知服务会产生相关服务费用，详细信息请参见**计费说明**。 ● 只有资源类型为GPU或NPU的训练作业才支持通知“作业疑似卡死”的事件。
自动停止	当使用付费资源时，可以选择是否打开“自动停止”开关。 ● 开关关闭（默认关闭）：表示训练作业将一直运行直至训练完成。 ● 开关打开：表示启用自动停止功能，此时必须配置自动停止时间，支持设置为“1小时”、“2小时”、“4小时”、6小时或“自定义”，自定义时间取值范围为1~720小时。启用该参数并设置时间后，运行时长到期后将会自动终止训练，准备排队等状态不扣除运行时长。
配置节点间SSH免密互信	选择是否打开“配置节点间SSH免密互信”开关。 ● 开关关闭（默认关闭）：表示不配置训练作业节点间SSH免密互信。 ● 开关打开：表示配置训练作业节点间SSH免密互信，此时必须配置“SSH密钥目录”，即自动生成的SSH密钥文件在训练容器中所在的目录，默认值为“/home/ma-user/.ssh”。

配置资源池（专属资源池）

如果使用专属资源池创建训练作业，则参考**表8-18**配置专属资源池。

表 8-18 创建训练作业的专属资源池

参数名称	说明
资源池	必选，选择“专属资源池”并选择要使用的资源池。 选择专属资源池时，支持查看当前资源池的状态、节点规格、空闲/碎片节点数、可用节点/总节点数以及卡数信息。单击“空闲/碎片节点数”列的“查看”可以查看碎片详情，确认资源池是否满足训练需求。
实例规格	必选，根据不同的资源类型，选择所需的资源规格。 当“输入”参数选择“数据存储位置”时，在选择资源池规格时可以单击右侧的“获取输入数据大小”，检查输入数据的大小是否超出数据盘的容量限制，避免训练过程中出现内存不足的情况。  **须知** 资源规格为“GPU:n*tnt004”（n表示具体数字）的资源不支持多进程的训练作业。
实例数	必填，根据需要选择实例的个数。默认值为“1”。 ● 当“实例数 = 1”时，创建的是单机训练作业，ModelArts只会在一个节点上启动一个训练容器，该训练容器独享所选规格的计算资源。 ● 当“实例数> 1”时，创建的是分布式训练作业，更多分布式训练配置请参见**分布式训练功能介绍**。
作业优先级	使用专属资源池创建训练作业时，支持设置训练作业的优先级。取值为1~3，默认优先级为1，最高优先级为3。 ● 默认用户权限可选择优先级1和2，配置了“**设置作业为高优先级权限**”的用户可选择优先级1~3。 ● 如果训练作业长时间处于“等待中”的状态，则可以通过修改作业优先级来减少排队时长，请参见**修改训练作业优先级**。

参数名称	说明
SFS Turbo	当ModelArts和SFS Turbo间网络直通时，训练作业支持挂载多个SFS Turbo存放训练数据。单击“增加挂载配置”，填写如下参数。 • “文件系统”：选择一个SFS Turbo。 • “云上挂载路径”：输入SFS Turbo对应在训练容器内的云上挂载路径。 • “存储位置”：选择SFS Turbo的存储位置。如果用户配置了文件夹控制权限，请选择存储位置；如果用户未配置文件夹控制权限，可以保持默认值“/”或者自定义位置。 • “挂载方式”：显示挂载SFS Turbo的权限。根据SFS Turbo存储位置的权限显示“读写”或“只读”，如果用户未配置文件夹控制权限，则该参数不可见。 • “挂载参数”：支持配置SFS挂载参数实现训练加速优化，具体参数说明请参见**设置极速文件存储挂载参数**。不设置时，默认配置如下参数： `mountOptions:` `- vers=3` `- timeo=600` `- nolock` `- hard` **说明** • 文件系统支持重复挂载，但是挂载路径不可重复。最多可以挂载5个盘。 • 云上挂载路径有如下限制：不能为 / 目录，不能为 /cache、 /home/ma-user/modelarts等系统已经默认挂载的路径。 • 如果需要设置SFS Turbo的文件夹权限，请参考**权限管理**文档配置。
永久保存日志	选择CPU或者GPU资源时，支持选择是否关闭“永久保存日志”开关。 • 开关打开（默认打开）：表示永久保存日志，此时必须配置“作业日志路径”，系统会将训练日志永久保存至指定的OBS路径。 • 开关关闭：表示不永久保存日志，则训练日志会在30天后会被清理。可以在作业详情页下载全部日志至本地。
作业日志路径	打开“永久保存日志”开关时，必须配置“作业日志路径”，用于存放训练作业产生的日志文件。 建议选择一个空的OBS文件目录存放运行中产生的日志文件，同时需要OBS文件目录的读写权限。

参数名称	说明
事件通知	选择是否打开“事件通知”开关。 ● 开关关闭（默认关闭）：表示不启用消息通知服务。 ● 开关打开：表示订阅消息通知服务，当训练作业发生特定事件（如作业状态变化或疑似卡死）时会发送通知。此时必须配置“主题名”和“事件”。 – “主题名”：事件通知的主题名称。单击“创建主题”，前往消息通知服务中创建主题。 – “事件”：选择要订阅的事件类型。例如“作业开始”、“作业结束”、“作业失败”、“作业终止”、“作业疑似卡死”等。 **说明** ● 需要为消息通知服务中创建的主题添加订阅，当订阅状态为“已确认”后，方可收到事件通知。订阅主题的详细操作请参见**添加订阅**。 ● 使用消息通知服务会产生相关服务费用，详细信息请参见**计费说明**。 ● 只有资源类型为GPU或NPU的训练作业才支持通知“作业疑似卡死”的事件。
自动停止	当使用付费资源时，可以选择是否打开“自动停止”开关。 ● 开关关闭（默认关闭）：表示训练作业将一直运行直至训练完成。 ● 开关打开：表示启用自动停止功能，此时必须配置自动停止时间，支持设置为“1小时”、“2小时”、“4小时”、6小时或“自定义”，自定义时间取值范围为1~720小时。启用该参数并设置时间后，运行时长到期后将会自动终止训练，准备排队等状态不扣除运行时长。
配置节点间SSH免密互信	选择是否打开“配置节点间SSH免密互信”开关。 ● 开关关闭（默认关闭）：表示不配置训练作业节点间SSH免密互信。 ● 开关打开：表示配置训练作业节点间SSH免密互信，此时必须配置“SSH密钥目录”，即自动生成的SSH密钥文件在训练容器中所在的目录，默认值为“/home/ma-user/.ssh”。

（可选）选择训练模式

当训练作业的算法框架选用的是预置框架的MindSpore类引擎、资源池类型选用的是Ascend资源时，则支持选择训练模式。ModelArts提供了3种训练模式供用户选择，支持根据实际场景获取不同的诊断信息。

- 普通模式：默认训练场景。
- 高性能模式：最小化调测信息，可以提升运行速度，适合于网络稳定并追求高性能的场景。

- 故障诊断模式：收集更多的信息用于定位，适合于执行出现问题需要收集故障信息进行定位的场景。此模式提供故障诊断，用户可以根据实际需求选择诊断类别。

（可选）设置标签

如果需要通过标签实现资源分组管理，可以在“高级选项”处勾选“现在配置”，可以设置训练作业的“标签”。标签详细用法请参见**使用TMS标签实现资源分组管理**。

后续操作

当创建训练作业的参数配置完成后，单击“提交”，在信息确认页面单击“确定”，提交创建训练作业任务。

训练作业一般需要运行一段时间，前往训练作业列表，可以查看训练作业的基本情况。

- 在训练作业列表中，刚创建的训练作业状态为“等待中”。
- 当训练作业的状态变为“已完成”时，表示训练作业运行结束，其生成的模型将存储至对应的“输出”目录中。
- 当训练作业的状态变为“运行失败”或“异常”时，可以单击训练作业的名称进入详情页面，通过查看日志等手段处理问题。

说明

训练作业运行过程中将按照选择的资源进行计费。

8.7 分布式模型训练

8.7.1 分布式训练功能介绍

ModelArts提供了如下能力：

- 丰富的官方预置镜像，满足用户的需求。
- 支持基于预置镜像自定义制作专属开发环境，并保存使用。
- 丰富的教程，帮助用户快速适配分布式训练，使用分布式训练极大减少训练时间。
- 分布式训练调测的能力，可在PyCharm/VSCode/JupyterLab等开发工具中调试分布式训练。

约束限制

- 如果切换了Notebook的规格，那么只能在Notebook进行单机调测，不能进行分布式调测，也不能提交远程训练作业。
- 当前仅支持PyTorch和MindSpore AI框架，如果MindSpore要进行多机分布式训练调试，则每台机器上都必须有8张卡。
- 本文档提供的调测代码中涉及到的OBS路径，请用户替换为自己的实际OBS路径。
- 本文档提供的调测代码是以PyTorch为例编写的，不同的AI框架之间，整体流程是完全相同的，只需要修改个别的参数即可。

DataParallel 进行单机多卡训练的优缺点

- 代码简单：仅需修改一行代码。
- 通信瓶颈 ： 负责reducer的GPU更新模型参数后分发到不同的GPU，因此有较大的通信开销。
- GPU负载不均衡：负责reducer的GPU需要负责汇总输出、计算损失和更新权重，因此显存和使用率相比其他GPU都会更高。

DistributedDataParallel 进行多机多卡训练的优缺点

- **通信更快**：相比于DP，通信速度更快
- **负载相对均衡**：相比于DP，GPU负载相对更均衡
- **运行速度快**：因为通信时间更短，效率更高，能更快速地完成训练作业。

相关章节

- **创建单机多卡的分布式训练（DataParallel）**：介绍单机多卡数据并行分布式训练原理和代码改造点。
- **创建多机多卡的分布式训练（DistributedDataParallel）**：介绍多机多卡数据并行分布式训练原理和代码改造点。
- **示例：创建DDP分布式训练（PyTorch+GPU）**：提供了分布式训练调测具体的代码适配操作过程和代码示例。
- **示例：创建DDP分布式训练（PyTorch+NPU）**：针对Resnet18在cifar10数据集上的分类任务，给出了分布式训练改造(DDP)的完整代码示例，供用户学习参考。
- **基于开发环境使用SDK调测训练作业**：介绍如何在ModelArts的开发环境中，使用SDK调测单机和多机分布式训练作业。

8.7.2 创建单机多卡的分布式训练（DataParallel）

本章节介绍基于PyTorch引擎的单机多卡数据并行训练。

MindSpore引擎的分布式训练参见**MindSpore官网**。

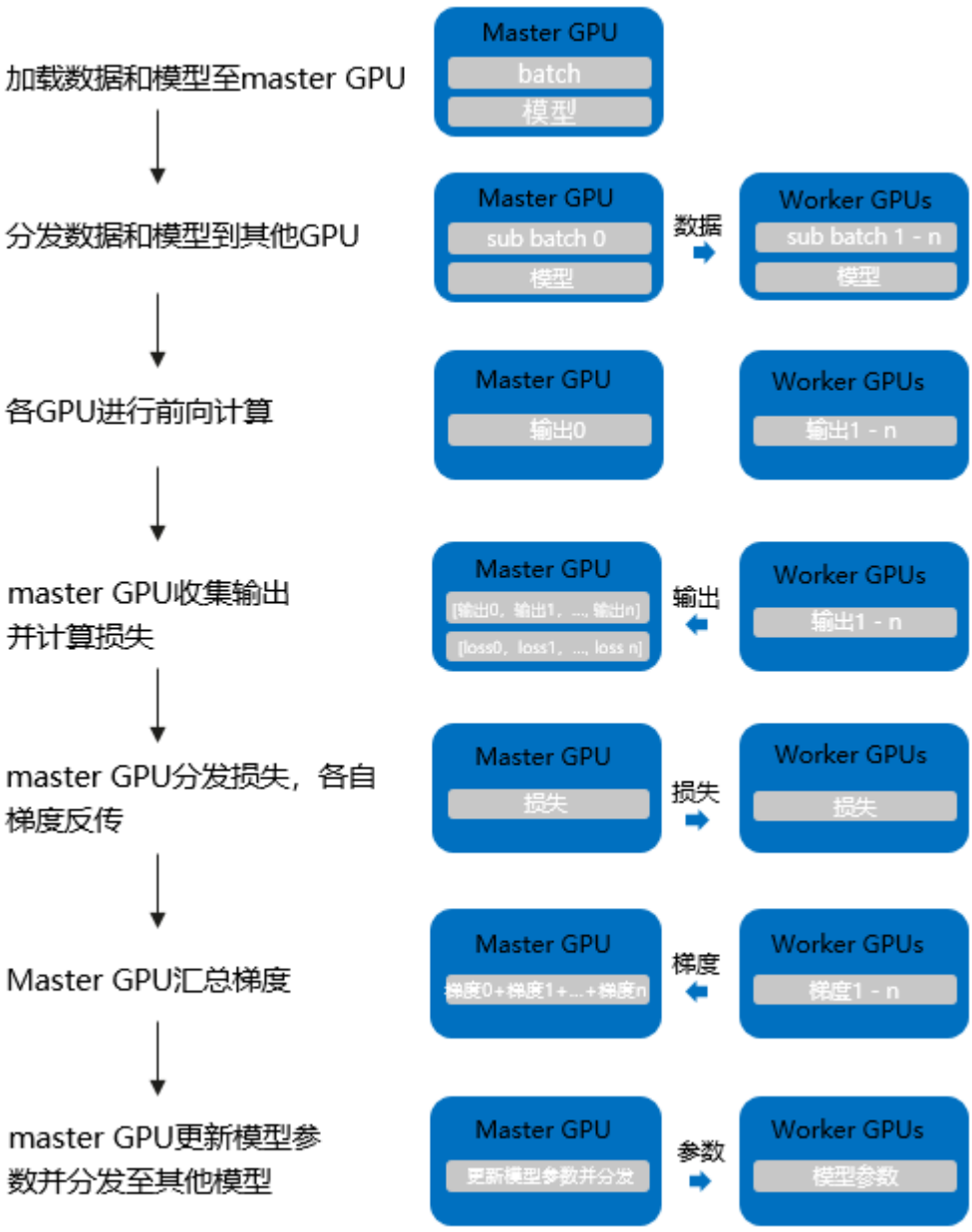
训练流程简述

单机多卡数据并行训练流程介绍如下：

1. 将模型复制到多个GPU上
2. 将一个Batch的数据均分到每一个GPU上
3. 各GPU上的模型进行前向传播，得到输出
4. 主GPU（逻辑序号为0）收集各GPU的输出，汇总后计算损失
5. 分发损失，各GPU各自反向传播梯度
6. 主GPU收集梯度并更新参数，将更新后的模型参数分发到各GPU

具体流程图如下：

图 8-15 单机多卡数据并行训练



代码改造点

模型分发：DataParallel(model)

完整代码由于代码变动较少，此处进行简略介绍。

```
import torch
class Net(torch.nn.Module):
    pass

model = Net().cuda()

### DataParallel Begin ###
model = torch.nn.DataParallel(Net().cuda())
### DataParallel End ###
```

8.7.3 创建多机多卡的分布式训练（DistributedDataParallel）

本章节介绍基于PyTorch引擎的多机多卡数据并行训练。并提供了分布式训练调测具体的代码适配操作过程和代码示例。同时还针对Resnet18在cifar10数据集上的分类任务，给出了分布式训练改造(DDP)的完整代码示例，供用户学习参考。

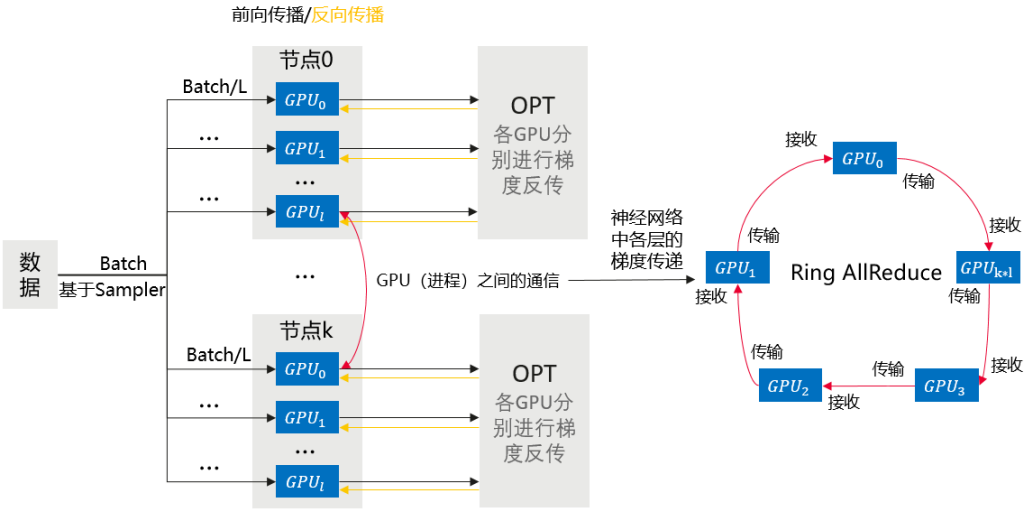
训练流程简述

相比于DP，DDP能够启动多进程进行运算，从而大幅度提升计算资源的利用率。可以基于torch.distributed实现真正的分布式计算，具体的原理此处不再赘述。大致的流程如下：

1. 初始化进程组。
2. 创建分布式并行模型，每个进程都会有相同的模型和参数。
3. 创建数据分发Sampler，使每个进程加载一个mini batch中不同部分的数据。
4. 网络中相邻参数分桶，一般为神经网络模型中需要进行参数更新的每一层网络。
5. 每个进程前向传播并各自计算梯度。
6. 模型某一层的参数得到梯度后会马上进行通讯并进行梯度平均。
7. 各GPU更新模型参数。

具体流程图如下：

图 8-16 多机多卡数据并行训练



代码改造点

- 引入多进程启动机制：初始化进程
- 引入几个变量：tcp协议，rank进程序号，worldsize开启的进程数量
- 分发数据：DataLoader中多了一个Sampler参数，避免不同进程数据重复
- 模型分发：DistributedDataParallel(model)
- 模型保存：在序号为0的进程下保存模型

```
import torch
class Net(torch.nn.Module):
    pass
```

```
model = Net().cuda()

### DistributedDataParallel Begin ###
model = torch.nn.parallel.DistributedDataParallel(Net().cuda())
### DistributedDataParallel End ###
```

多节点分布式调测适配及代码示例

在DistributedDataParallel中，不同进程分别从原始数据中加载batch的数据，最终将各个进程的梯度进行平均作为最终梯度，由于样本量更大，因此计算出的梯度更加可靠，可以适当增大学习率。

以下对resnet18在cifar10数据集上的分类任务，给出了单机训练和分布式训练改造(DDP)的代码。直接执行代码为多节点分布式训练且支持CPU分布式和GPU分布式，将代码中的分布式改造点注释掉后即可进行单节点单卡训练。

训练代码中包含三部分入参，分别为训练基础参数、分布式参数和数据相关参数。其中分布式参数由平台自动入参，无需自行定义。数据相关参数中的custom_data表示是否使用自定义数据进行训练，该参数为“true”时使用基于torch自定义的随机数据进行训练和验证。

cifar10数据集

在Notebook中，无法直接使用默认版本的torchvision获取数据集，因此示例代码中提供了三种训练数据加载方式。

cifar-10数据集**下载链接**，单击“CIFAR-10 python version”。

- 尝试基于torchvision获取cifar10数据集。
- 基于数据链接下载数据并解压，放置在指定目录下，训练集和测试集的大小分别为(50000，3，32，32)和(10000，3，32，32)。
- 考虑到下载cifar10数据集较慢，基于torch生成类似cifar10的随机数据集，训练集和测试集的大小分别为(5000，3，32，32)和(1000，3，32，32)，标签仍为10类，指定custom_data = 'true'后可直接进行训练作业，无需加载数据。

训练代码

以下代码中以“### 分布式改造，... ###”注释的代码即为多节点分布式训练需要适配的代码改造点。

不对示例代码进行任何修改，适配数据路径后即可在ModelArts上完成多节点分布式训练。

注释掉分布式代码改造点，即可完成单节点单卡训练。完整代码见**分布式训练完整代码示例**。

- **导入依赖包**

```
import datetime
import inspect
import os
import pickle
import random

import argparse
import numpy as np
import torch
import torch.distributed as dist
from torch import nn, optim
from torch.utils.data import TensorDataset, DataLoader
from torch.utils.data.distributed import DistributedSampler
from sklearn.metrics import accuracy_score
```

- **定义加载数据的方法和随机数**，由于加载数据部分代码较多，此处省略

```
def setup_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
    torch.backends.cudnn.deterministic = True

def get_data(path):
    pass
```

- **定义网络结构**

```
class Block(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.residual_function(x) + self.shortcut(x)
        return nn.ReLU(inplace=True)(out)


class ResNet(nn.Module):

    def __init__(self, block, num_classes=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2 = self.make_layer(block, 64, 64, 2, 1)
        self.conv3 = self.make_layer(block, 64, 128, 2, 2)
        self.conv4 = self.make_layer(block, 128, 256, 2, 2)
        self.conv5 = self.make_layer(block, 256, 512, 2, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.dense_layer = nn.Linear(512, num_classes)

    def make_layer(self, block, in_channels, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(in_channels, out_channels, stride))
            in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.dense_layer(out)
        return out
```


- **进行训练和验证**

```
def main():
    file_dir = os.path.dirname(inspect.getframeinfo(inspect.currentframe()).filename)

    seed = datetime.datetime.now().year
    setup_seed(seed)

    parser = argparse.ArgumentParser(description='Pytorch distribute training',
                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--enable_gpu', default='true')
    parser.add_argument('--lr', default='0.01', help='learning rate')
    parser.add_argument('--epochs', default='100', help='training iteration')

    parser.add_argument('--init_method', default=None, help='tcp_port')
    parser.add_argument('--rank', type=int, default=0, help='index of current task')
    parser.add_argument('--world_size', type=int, default=1, help='total number of tasks')

    parser.add_argument('--custom_data', default='false')
    parser.add_argument('--data_url', type=str, default=os.path.join(file_dir, 'input_dir'))
    parser.add_argument('--output_dir', type=str, default=os.path.join(file_dir, 'output_dir'))
    args, unknown = parser.parse_known_args()

    args.enable_gpu = args.enable_gpu == 'true'
    args.custom_data = args.custom_data == 'true'
    args.lr = float(args.lr)
    args.epochs = int(args.epochs)

    if args.custom_data:
        print('[warning] you are training on custom random dataset, '
              'validation accuracy may range from 0.4 to 0.6.')

    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###
    dist.init_process_group(init_method=args.init_method, backend="nccl", world_size=args.world_size,
rank=args.rank)
    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###

    tr_set, val_set = get_data(args.data_url, custom_data=args.custom_data)

    batch_per_gpu = 128
    gpus_per_node = torch.cuda.device_count() if args.enable_gpu else 1
    batch = batch_per_gpu * gpus_per_node

    tr_loader = DataLoader(tr_set, batch_size=batch, shuffle=False)

    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###
    tr_sampler = DistributedSampler(tr_set, num_replicas=args.world_size, rank=args.rank)
    tr_loader = DataLoader(tr_set, batch_size=batch, sampler=tr_sampler, shuffle=False, drop_last=True)
    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###

    val_loader = DataLoader(val_set, batch_size=batch, shuffle=False)

    lr = args.lr * gpus_per_node
    max_epoch = args.epochs
    model = ResNet(Block).cuda() if args.enable_gpu else ResNet(Block)

    ### 分布式改造，构建DDP分布式模型 ###
    model = nn.parallel.DistributedDataParallel(model)
    ### 分布式改造，构建DDP分布式模型 ###

    optimizer = optim.Adam(model.parameters(), lr=lr)
    loss_func = torch.nn.CrossEntropyLoss()

    os.makedirs(args.output_dir, exist_ok=True)

    for epoch in range(1, max_epoch + 1):
        model.train()
        train_loss = 0

        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###
```

```
        tr_sampler.set_epoch(epoch)
        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###

        for step, (tr_x, tr_y) in enumerate(tr_loader):
            if args.enable_gpu:
                tr_x, tr_y = tr_x.cuda(), tr_y.cuda()
            out = model(tr_x)
            loss = loss_func(out, tr_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
        print('train | epoch: %d | loss: %.4f' % (epoch, train_loss / len(tr_loader)))

        val_loss = 0
        pred_record = []
        real_record = []
        model.eval()
        with torch.no_grad():
            for step, (val_x, val_y) in enumerate(val_loader):
                if args.enable_gpu:
                    val_x, val_y = val_x.cuda(), val_y.cuda()
                out = model(val_x)
                pred_record += list(np.argmax(out.cpu().numpy(), axis=1))
                real_record += list(val_y.cpu().numpy())
                val_loss += loss_func(out, val_y).item()
        val_accu = accuracy_score(real_record, pred_record)
        print('val | epoch: %d | loss: %.4f | accuracy: %.4f' % (epoch, val_loss / len(val_loader), val_accu),
'\n')

        if args.rank == 0:
            # save ckpt every epoch
            torch.save(model.state_dict(), os.path.join(args.output_dir, f'epoch_{epoch}.pth'))


if __name__ == '__main__':
    main()
```

- **结果对比**

  分别以单机单卡和两节点16卡两种资源类型完成100epoch的cifar-10数据集训练，训练时长和测试集准确率如下。

  表 8-19 训练结果对比

资源类型	单机单卡	两节点16卡
耗时	60分钟	20分钟
准确率	80+	80+

分布式训练完整代码示例

以下对resnet18在cifar10数据集上的分类任务，给出了分布式训练改造(DDP)的完整代码示例。

训练启动文件main.py内容如下（如果需要执行单机单卡训练作业，则将分布式改造的代码删除）：

```
import datetime
import inspect
import os
import pickle
import random
```



```
import logging

import argparse
import numpy as np
from sklearn.metrics import accuracy_score
import torch
from torch import nn, optim
import torch.distributed as dist
from torch.utils.data import TensorDataset, DataLoader
from torch.utils.data.distributed import DistributedSampler

file_dir = os.path.dirname(inspect.getframeinfo(inspect.currentframe()).filename)


def load_pickle_data(path):
    with open(path, 'rb') as file:
        data = pickle.load(file, encoding='bytes')
    return data


def _load_data(file_path):
    raw_data = load_pickle_data(file_path)
    labels = raw_data[b'labels']
    data = raw_data[b'data']
    filenames = raw_data[b'filenames']

    data = data.reshape(10000, 3, 32, 32) / 255
    return data, labels, filenames


def load_cifar_data(root_path):
    train_root_path = os.path.join(root_path, 'cifar-10-batches-py/data_batch_')
    train_data_record = []
    train_labels = []
    train_filenames = []
    for i in range(1, 6):
        train_file_path = train_root_path + str(i)
        data, labels, filenames = _load_data(train_file_path)
        train_data_record.append(data)
        train_labels += labels
        train_filenames += filenames
    train_data = np.concatenate(train_data_record, axis=0)
    train_labels = np.array(train_labels)

    val_file_path = os.path.join(root_path, 'cifar-10-batches-py/test_batch')
    val_data, val_labels, val_filenames = _load_data(val_file_path)
    val_labels = np.array(val_labels)

    tr_data = torch.from_numpy(train_data).float()
    tr_labels = torch.from_numpy(train_labels).long()
    val_data = torch.from_numpy(val_data).float()
    val_labels = torch.from_numpy(val_labels).long()
    return tr_data, tr_labels, val_data, val_labels


def get_data(root_path, custom_data=False):
    if custom_data:
        train_samples, test_samples, img_size = 5000, 1000, 32
        tr_label = [1] * int(train_samples / 2) + [0] * int(train_samples / 2)
        val_label = [1] * int(test_samples / 2) + [0] * int(test_samples / 2)
        random.seed(2021)
        random.shuffle(tr_label)
        random.shuffle(val_label)
        tr_data, tr_labels = torch.randn((train_samples, 3, img_size, img_size)).float(),
torch.tensor(tr_label).long()
        val_data, val_labels = torch.randn((test_samples, 3, img_size, img_size)).float(), torch.tensor(
            val_label).long()
        tr_set = TensorDataset(tr_data, tr_labels)
        val_set = TensorDataset(val_data, val_labels)
```



```
        return tr_set, val_set
    elif os.path.exists(os.path.join(root_path, 'cifar-10-batches-py')):
        tr_data, tr_labels, val_data, val_labels = load_cifar_data(root_path)
        tr_set = TensorDataset(tr_data, tr_labels)
        val_set = TensorDataset(val_data, val_labels)
        return tr_set, val_set
    else:
        try:
            import torchvision
            from torchvision import transforms
            tr_set = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                  download=True, transform=transforms)
            val_set = torchvision.datasets.CIFAR10(root='./data', train=False,
                                                   download=True, transform=transforms)
            return tr_set, val_set
        except Exception as e:
            raise Exception(
                f"{e}, you can download and unzip cifar-10 dataset manually, "
                "the data url is http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz")


class Block(nn.Module):

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.residual_function(x) + self.shortcut(x)
        return nn.ReLU(inplace=True)(out)


class ResNet(nn.Module):

    def __init__(self, block, num_classes=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2 = self.make_layer(block, 64, 64, 2, 1)
        self.conv3 = self.make_layer(block, 64, 128, 2, 2)
        self.conv4 = self.make_layer(block, 128, 256, 2, 2)
        self.conv5 = self.make_layer(block, 256, 512, 2, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.dense_layer = nn.Linear(512, num_classes)

    def make_layer(self, block, in_channels, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(in_channels, out_channels, stride))
            in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
```



```
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.conv5(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.dense_layer(out)
        return out


def setup_seed(seed):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    random.seed(seed)
    torch.backends.cudnn.deterministic = True


def obs_transfer(src_path, dst_path):
    import moxing as mox
    mox.file.copy_parallel(src_path, dst_path)
    logging.info(f"end copy data from {src_path} to {dst_path}")


def main():
    seed = datetime.datetime.now().year
    setup_seed(seed)

    parser = argparse.ArgumentParser(description='Pytorch distribute training',
                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--enable_gpu', default='true')
    parser.add_argument('--lr', default='0.01', help='learning rate')
    parser.add_argument('--epochs', default='100', help='training iteration')

    parser.add_argument('--init_method', default=None, help='tcp_port')
    parser.add_argument('--rank', type=int, default=0, help='index of current task')
    parser.add_argument('--world_size', type=int, default=1, help='total number of tasks')

    parser.add_argument('--custom_data', default='false')
    parser.add_argument('--data_url', type=str, default=os.path.join(file_dir, 'input_dir'))
    parser.add_argument('--output_dir', type=str, default=os.path.join(file_dir, 'output_dir'))
    args, unknown = parser.parse_known_args()

    args.enable_gpu = args.enable_gpu == 'true'
    args.custom_data = args.custom_data == 'true'
    args.lr = float(args.lr)
    args.epochs = int(args.epochs)

    if args.custom_data:
        logging.warning('you are training on custom random dataset, '
            'validation accuracy may range from 0.4 to 0.6.')

    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###
    dist.init_process_group(init_method=args.init_method, backend="nccl", world_size=args.world_size,
rank=args.rank)
    ### 分布式改造，DDP初始化进程，其中init_method, rank和world_size参数均由平台自动入参 ###

    tr_set, val_set = get_data(args.data_url, custom_data=args.custom_data)

    batch_per_gpu = 128
    gpus_per_node = torch.cuda.device_count() if args.enable_gpu else 1
    batch = batch_per_gpu * gpus_per_node

    tr_loader = DataLoader(tr_set, batch_size=batch, shuffle=False)

    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###
    tr_sampler = DistributedSampler(tr_set, num_replicas=args.world_size, rank=args.rank)
    tr_loader = DataLoader(tr_set, batch_size=batch, sampler=tr_sampler, shuffle=False, drop_last=True)
```



```
    ### 分布式改造，构建DDP分布式数据sampler，确保不同进程加载到不同的数据 ###

    val_loader = DataLoader(val_set, batch_size=batch, shuffle=False)

    lr = args.lr * gpus_per_node * args.world_size
    max_epoch = args.epochs
    model = ResNet(Block).cuda() if args.enable_gpu else ResNet(Block)

    ### 分布式改造，构建DDP分布式模型 ###
    model = nn.parallel.DistributedDataParallel(model)
    ### 分布式改造，构建DDP分布式模型 ###

    optimizer = optim.Adam(model.parameters(), lr=lr)
    loss_func = torch.nn.CrossEntropyLoss()

    os.makedirs(args.output_dir, exist_ok=True)

    for epoch in range(1, max_epoch + 1):
        model.train()
        train_loss = 0

        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###
        tr_sampler.set_epoch(epoch)
        ### 分布式改造，DDP sampler, 基于当前的epoch为其设置随机数，避免加载到重复数据 ###

        for step, (tr_x, tr_y) in enumerate(tr_loader):
            if args.enable_gpu:
                tr_x, tr_y = tr_x.cuda(), tr_y.cuda()
            out = model(tr_x)
            loss = loss_func(out, tr_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_loss += loss.item()
        print('train | epoch: %d | loss: %.4f' % (epoch, train_loss / len(tr_loader)))

        val_loss = 0
        pred_record = []
        real_record = []
        model.eval()
        with torch.no_grad():
            for step, (val_x, val_y) in enumerate(val_loader):
                if args.enable_gpu:
                    val_x, val_y = val_x.cuda(), val_y.cuda()
                out = model(val_x)
                pred_record += list(np.argmax(out.cpu().numpy(), axis=1))
                real_record += list(val_y.cpu().numpy())
                val_loss += loss_func(out, val_y).item()
        val_accu = accuracy_score(real_record, pred_record)
        print('val | epoch: %d | loss: %.4f | accuracy: %.4f' % (epoch, val_loss / len(val_loader), val_accu), '\n')

        if args.rank == 0:
            # save ckpt every epoch
            torch.save(model.state_dict(), os.path.join(args.output_dir, f'epoch_{epoch}.pth'))


if __name__ == '__main__':
    main()
```

常见问题

1、示例代码中如何使用不同的数据集?

- 上述代码如果使用cifar10数据集，则将数据集下载并解压后，上传至OBS桶中，文件目录结构如下：

```
DDP
|--- main.py
|--- input_dir
|------ cifar-10-batches-py
```



```
|-------- data_batch_1
|-------- data_batch_2
|-------- ...
```

其中“DDP”为创建训练作业时的“代码目录”，“main.py”为上文代码示例（即创建训练作业时的“启动文件”），“cifar-10-batches-py”为解压后的数据集文件夹（放在input_dir文件夹下）。

- 如果使用自定义的随机数据，则将代码示例中的参数“custom_data”改为“true”，修改后内容如下：

```
parser.add_argument('--custom_data', default='true')
```

然后直接运行代码示例“main.py”即可，创建训练作业的参数与上图相同。

2、为什么DDP可以不输入主节点ip?

“parser.add_argument('--init_method', default=None, help='tcp_port')”中的init method参数值会包含主节点的ip和端口，由平台自动入参，不需要用户输入主节点的ip和端口。

8.7.4 示例：创建 DDP 分布式训练（PyTorch+GPU）

本文介绍三种使用训练作业来启动PyTorch DDP训练的方法及对应代码示例。

- 使用PyTorch预置框架功能，通过mp.spawn命令启动
- 使用自定义镜像功能
  - 通过torch.distributed.launch命令启动
  - 通过torch.distributed.run命令启动

创建训练作业

- 方式一：使用PyTorch预置框架功能，通过mp.spawn命令启动训练作业。

  创建训练作业的关键参数如**表8-20**所示。

  表 8-20 创建训练作业（预置框架）

参数名称	说明
创建方式	选择“自定义算法”。
启动方式	选择“预置框架”，引擎选择“PyTorch”，PyTorch版本根据训练要求选择。
代码目录	选择OBS桶中训练code文件夹所在路径，例如“obs://test-modelarts/code/”。
启动文件	选择代码目录中训练作业的Python启动脚本。例如“obs://test-modelarts/code/main.py”。
超参	当资源规格为单机多卡时，需要指定超参world_size和rank。 当资源规格为多机时（即实例数大于 1），无需设置超参world_size和rank，超参会由平台自动注入。

- 方式二：使用自定义镜像功能，通过torch.distributed.launch命令启动训练作业。

  创建训练作业的关键参数如**表8-21**所示。

表 8-21 创建训练作业（自定义镜像+torch.distributed.launch 命令）

参数名称	说明
创建方式	选择“自定义算法”。
启动方式	选择“自定义”。
镜像	选择用于训练的PyTorch镜像。
代码目录	选择OBS桶中训练code文件夹所在路径，例如“obs://test-modelarts/code/”。
启动命令	输入镜像的Python启动命令，例如： `bash ${MA_JOB_DIR}/code/torchlaunch.sh`

- 方式三：使用自定义镜像功能，通过torch.distributed.run命令启动训练作业。
  创建训练作业的关键参数如表8-22所示。

表 8-22 创建训练作业（自定义镜像+torch.distributed.run 命令）

参数名称	说明
创建方式	选择“自定义算法”。
启动方式	选择“自定义”。
镜像	选择用于训练的PyTorch镜像。
代码目录	选择OBS桶中训练code文件夹所在路径，例如“obs://test-modelarts/code/”。
启动命令	输入镜像的Python启动命令，例如： `bash ${MA_JOB_DIR}/code/torchrun.sh`

代码示例

文件目录结构如下所示，将以下文件上传至OBS桶中：

```
code                    # 代码根目录
 └─torch_ddp.py           # PyTorch DDP训练代码文件
 └─main.py              # 使用PyTorch预置框架功能，通过mp.spawn命令启动训练的启动文件
 └─torchlaunch.sh         # 使用自定义镜像功能，通过torch.distributed.launch命令启动训练的启动文件
 └─torchrun.sh          # 使用自定义镜像功能，通过torch.distributed.run命令启动训练的启动文件
```

torch_ddp.py内容如下：

```
import os
import torch
import torch.distributed as dist
import torch.nn as nn
import torch.optim as optim
from torch.nn.parallel import DistributedDataParallel as DDP

# 用于通过 mp.spawn 启动
def init_from_arg(local_rank, base_rank, world_size, init_method):
    rank = base_rank + local_rank
    dist.init_process_group("nccl", rank=rank, init_method=init_method, world_size=world_size)
    ddp_train(local_rank)
```

```
# 用于通过 torch.distributed.launch 或 torch.distributed.run 启动
def init_from_env():
    dist.init_process_group(backend='nccl', init_method='env://')
    local_rank=int(os.environ["LOCAL_RANK"])
    ddp_train(local_rank)

def cleanup():
    dist.destroy_process_group()

class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = nn.Linear(10, 10)
        self.relu = nn.ReLU()
        self.net2 = nn.Linear(10, 5)
    def forward(self, x):
        return self.net2(self.relu(self.net1(x)))

def ddp_train(device_id):
    # create model and move it to GPU with id rank
    model = ToyModel().to(device_id)
    ddp_model = DDP(model, device_ids=[device_id])
    loss_fn = nn.MSELoss()
    optimizer = optim.SGD(ddp_model.parameters(), lr=0.001)
    optimizer.zero_grad()
    outputs = ddp_model(torch.randn(20, 10))
    labels = torch.randn(20, 5).to(device_id)
    loss_fn(outputs, labels).backward()
    optimizer.step()
    cleanup()

if __name__ == "__main__":
    init_from_env()
```

main.py内容如下：

```
import argparse
import torch
import torch.multiprocessing as mp

parser = argparse.ArgumentParser(description='ddp demo args')
parser.add_argument('--world_size', type=int, required=True)
parser.add_argument('--rank', type=int, required=True)
parser.add_argument('--init_method', type=str, required=True)
args, unknown = parser.parse_known_args()

if __name__ == "__main__":
    n_gpus = torch.cuda.device_count()
    world_size = n_gpus * args.world_size
    base_rank = n_gpus * args.rank
    # 调用 DDP 示例代码中的启动函数
    from torch_ddp import init_from_arg
    mp.spawn(init_from_arg,
        args=(base_rank, world_size, args.init_method),
        nprocs=n_gpus,
        join=True)
```

torchlaunch.sh内容如下：

```
#!/bin/bash
# 系统默认环境变量，不建议修改
MASTER_HOST="$VC_WORKER_HOSTS"
MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"
MASTER_PORT="6060"
JOB_ID="1234"
NNODES="$MA_NUM_HOSTS"
NODE_RANK="$VC_TASK_INDEX"
NGPUS_PER_NODE="$MA_NUM_GPUS"

# 自定义环境变量，指定python脚本和参数
```



```
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""

CMD="python -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --nproc_per_node=$NGPUS_PER_NODE \
    --master_addr $MASTER_ADDR \
    --master_port=$MASTER_PORT \
    --use_env \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
"
echo $CMD
$CMD
```

torchrun.sh内容如下：

须知

PyTorch 2.1版本需要将“rdzv_backend”参数设置为“static：--rdzv_backend=static”。

```
#!/bin/bash
# 系统默认环境变量，不建议修改
MASTER_HOST="$VC_WORKER_HOSTS"
MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"
MASTER_PORT="6060"
JOB_ID="1234"
NNODES="$MA_NUM_HOSTS"
NODE_RANK="$VC_TASK_INDEX"
NGPUS_PER_NODE="$MA_NUM_GPUS"

# 自定义环境变量，指定python脚本和参数
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""

if [[ $NODE_RANK == 0 ]]; then
    EXT_ARGS="--rdzv_conf=is_host=1"
else
    EXT_ARGS=""
fi

CMD="python -m torch.distributed.run \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    $EXT_ARGS \
    --nproc_per_node=$NGPUS_PER_NODE \
    --rdzv_id=$JOB_ID \
    --rdzv_backend=c10d \
    --rdzv_endpoint=$MASTER_ADDR:$MASTER_PORT \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
    "
echo $CMD
$CMD
```

8.7.5 示例：创建 DDP 分布式训练（PyTorch+NPU）

本文介绍了使用训练作业的自定义镜像+自定义启动命令来启动PyTorch DDP on Ascend加速卡训练。

前提条件

需要有Ascend加速卡资源池。

创建训练作业

本案例创建训练作业时，需要配置如下参数。

表 8-23 创建训练作业的配置说明

参数名称	说明
“创建方式”	选择“自定义算法”。
“启动方式”	选择“自定义”。
“镜像”	选择用于训练的自定义镜像。
“代码目录”	执行本次训练作业所需的代码目录。本文示例的代码目录为“obs://test-modelarts/ascend/code/”。
“启动命令”	镜像的Python启动命令。本文示例的启动命令为“bash ${MA_JOB_DIR}/code/run_torch_ddp_npu.sh”。其中，启动脚本的完整代码请参见**代码示例**。

（可选）启用 ranktable 动态路由

如果训练作业需要使用ranktable动态路由算法进行网络加速，则可以联系技术支持开启集群的cabinet调度权限。同时，训练作业要满足如下要求才能正常实现ranktable动态路由加速。

- 训练使用的Python版本是3.7或3.9。
- 训练作业的实例数要大于或等于3。
- 路由加速的原理是改变rank编号，所以代码中对rank的使用要统一。

将训练作业完成如下修改后，启动训练作业即可实现网络加速。

- 将训练启动脚本中的“NODE_RANK="$VC_TASK_INDEX"”修改为“NODE_RANK="$RANK_AFTER_ACC"”。
- 将训练启动脚本中的“MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"”修改为“MASTER_ADDR="${MA_VJ_NAME}-${MA_TASK_NAME}-${MA_MASTER_INDEX}.${MA_VJ_NAME}"”。
- 在创建训练作业页面配置环境变量“ROUTE_PLAN”，取值为“true”，具体操作请参见**管理训练容器环境变量**。

代码示例

训练作业的启动脚本示例如下。

说明

启动脚本中设置plog生成后存放在“/home/ma-user/modelarts/log/modelarts-job-{id}/worker-{index}/”目录，而“/home/ma-user/modelarts/log/”目录下的“*.log”文件将会被自动上传至ModelArts训练作业的日志目录（OBS）。如果本地相应目录没有生成大小>0的日志文件，则对应的父级目录也不会上传。因此，PyTorch NPU的plog日志是按worker存储的，而不是按rank id存储的（这是区别于MindSpore的）。目前，PyTorch NPU并不依赖rank table file。

```
#!/bin/bash

# MA preset envs
MASTER_HOST="$VC_WORKER_HOSTS"
MASTER_ADDR="${VC_WORKER_HOSTS%%,*}"
NNODES="$MA_NUM_HOSTS"
NODE_RANK="$VC_TASK_INDEX"
# also indicates NPU per node
NGPUS_PER_NODE="$MA_NUM_GPUS"

# self-define, it can be changed to >=10000 port
MASTER_PORT="38888"

# replace ${MA_JOB_DIR}/code/torch_ddp.py to the actutal training script
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""

export HCCL_WHITELIST_DISABLE=1

# set npu plog env
ma_vj_name=`echo ${MA_VJ_NAME} | sed 's:ma-job:modelarts-job:g'`
task_name="worker-${VC_TASK_INDEX}"
task_plog_path=${MA_LOG_DIR}/${ma_vj_name}/${task_name}

mkdir -p ${task_plog_path}
export ASCEND_PROCESS_LOG_PATH=${task_plog_path}

echo "plog path: ${ASCEND_PROCESS_LOG_PATH}"

# set hccl timeout time in seconds
export HCCL_CONNECT_TIMEOUT=1800

# replace ${ANACONDA_DIR}/envs/${ENV_NAME}/bin/python to the actual python
CMD="${ANACONDA_DIR}/envs/${ENV_NAME}/bin/python -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --nproc_per_node=$NGPUS_PER_NODE \
    --master_addr=$MASTER_ADDR \
    --master_port=$MASTER_PORT \
    --use_env \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
"
echo $CMD
$CMD
```

8.8 增量模型训练

什么是增量训练

增量训练（Incremental Learning）是机器学习领域中的一种训练方法，它允许人工智能（AI）模型在已经学习了一定知识的基础上，增加新的训练数据到当前训练流程中，扩展当前模型的知识和能力，而不需要从头开始。

增量训练不需要一次性存储所有的训练数据，缓解了存储资源有限的问题；另一方面，增量训练节约了重新训练中需要消耗大量算力、时间以及经济成本。

增量训练特别适用于以下情况：

- 数据流更新：在实际应用中，数据可能会持续更新，增量训练允许模型适应新的数据而不必重新训练。
- 资源限制：如果重新训练一个大型模型成本过高，增量训练可以是一个更经济的选择。
- 避免灾难性遗忘：在传统训练中，新数据可能会覆盖旧数据的知识，导致模型忘记之前学到的内容。增量训练通过保留旧知识的同时学习新知识来避免这个问题。

增量训练在很多领域都有应用，比如自然语言处理、计算机视觉和推荐系统等。它使得AI系统能够更加灵活和适应性强，更好地应对现实世界中不断变化的数据环境。

ModelArts Standard 中如何实现增量训练

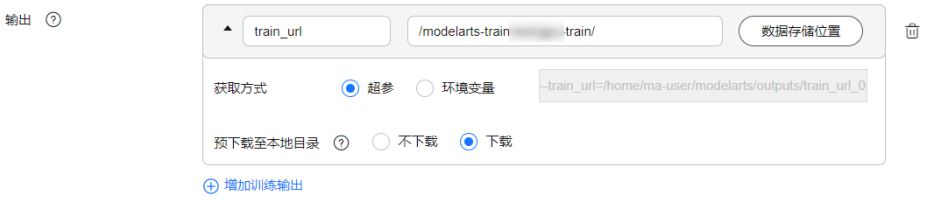
增量训练是通过Checkpoint机制实现。

Checkpoint的机制是：在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。当需要增加新的数据继续训练时，只需要加载Checkpoint，并用Checkpoint信息初始化训练状态即可。用户需要在代码里加上reload ckpt的代码，使能读取前一次训练保存的预训练模型。

在ModelArts训练中实现增量训练，建议使用“训练输出”功能。

在创建训练作业时，设置训练“输出”参数为“train_url”，在指定的训练输出的数据存储位置中保存**Checkpoint**，且**“预下载至本地目录”**选择“下载”。选择预下载至本地目录时，系统在训练作业启动前，自动将数据存储位置中的**Checkpoint**文件下载到训练容器的本地目录。

图 8-17 训练输出设置



PyTorch 版 reload ckpt

1. PyTorch模型保存有两种方式。
   - 仅保存模型参数
     ```
     state_dict = model.state_dict()
     torch.save(state_dict, path)
     ```
   - 保存整个Model（不推荐）
     ```
     torch.save(model, path)
     ```
2. 可根据step步数、时间等周期性保存模型的训练过程的产物。

   将模型训练过程中的网络权重、优化器权重、以及epoch进行保存，便于中断后继续训练恢复。

   ```
   checkpoint = {
         "net": model.state_dict(),
         "optimizer": optimizer.state_dict(),
         "epoch": epoch
   }
   ```

```
    if not os.path.isdir('model_save_dir'):
        os.makedirs('model_save_dir')
    torch.save(checkpoint,'model_save_dir/ckpt_{}.pth'.format(str(epoch)))
```

3. 完整代码示例。

```
import os
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--train_url", type=str)
args, unparsed = parser.parse_known_args()
args = parser.parse_known_args()
# train_url 将被赋值为"/home/ma-user/modelarts/outputs/train_url_0"
train_url = args.train_url

# 判断输出路径中是否有模型文件。如果无文件则默认从头训练，如果有模型文件，则加载epoch值最大的ckpt文件当做预训练模型。
if os.listdir(train_url):
    print('> load last ckpt and continue training!!')
    last_ckpt = sorted([file for file in os.listdir(train_url) if file.endswith(".pth")])[-1]
    local_ckpt_file = os.path.join(train_url, last_ckpt)
    print('last_ckpt:', last_ckpt)
    # 加载断点
    checkpoint = torch.load(local_ckpt_file)
    # 加载模型可学习参数
    model.load_state_dict(checkpoint['net'])
    # 加载优化器参数
    optimizer.load_state_dict(checkpoint['optimizer'])
    # 获取保存的epoch，模型会在此epoch的基础上继续训练
    start_epoch = checkpoint['epoch']
start = datetime.now()
total_step = len(train_loader)
for epoch in range(start_epoch + 1, args.epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.cuda(non_blocking=True)
        labels = labels.cuda(non_blocking=True)
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        ...

    # 保存模型训练过程中的网络权重、优化器权重、以及epoch
    checkpoint = {
          "net": model.state_dict(),
          "optimizer": optimizer.state_dict(),
          "epoch": epoch
        }
    if not os.path.isdir(train_url):
        os.makedirs(train_url)
        torch.save(checkpoint, os.path.join(train_url, 'ckpt_best_{}.pth'.format(epoch)))
```

8.9 自动模型优化（AutoSearch）

8.9.1 自动模型优化介绍

ModelArts训练支持超参搜索功能，自动实现模型超参搜索，为您的模型匹配最合适的超参。

在模型训练过程中，有很多超参需要根据任务进行调整，比如learning_rate、weight_decay等，这一工作往往需要一个有经验的算法工程师花费一定精力和大量时间进行手动调优。ModelArts支持的超参搜索功能，在无需算法工程师介入的情况下，即可自动进行超参的调优，在速度和精度上超过人工调优。

ModelArts支持以下三种超参搜索算法：

- 贝叶斯优化（SMAC）
- TPE算法
- 模拟退火算法（Anneal）

贝叶斯优化（SMAC）

贝叶斯优化假设超参和目标函数存在一个函数关系。基于已搜索超参的评估值，通过高斯过程回归来估计其他搜索点处目标函数值的均值和方差。根据均值和方差构造采集函数（Acquisition Function），下一个搜索点为采集函数的极大值点。相比网格搜索，贝叶斯优化会利用之前的评估结果，从而降低迭代次数、缩短搜索时间；缺点是不容易找到全局最优解。

表 8-24 贝叶斯优化的参数说明

参数	说明	取值参考
num_samples	搜索尝试的超参组数	int，一般在10-20之间，值越大，搜索时间越长，效果越好
kind	采集函数类型	string，默认为'ucb'，可能取值还有'ei'、'poi'，一般不建议用户修改
kappa	采集函数ucb的调节参数，可理解为上置信边界	float，一般不建议用户修改
xi	采集函数poi和ei的调节参数	float，一般不建议用户修改

TPE 算法

TPE算法全称Tree-structured Parzen Estimator，是一种利用高斯混合模型来学习超参模型的算法。在每次试验中，对于每个超参，TPE为与最佳目标值相关的超参维护一个高斯混合模型l(x)，为剩余的超参维护另一个高斯混合模型g(x)，选择l(x)/g(x)最大化时对应的超参作为下一组搜索值。

表 8-25 TPE 算法的参数说明

参数	说明	取值参考
num_samples	搜索尝试的超参组数	int，一般在10-20之间，值越大，搜索时间越长，效果越好
n_initial_points	采用TPE接近目标函数之前，对目标函数的随机评估数	int，一般不建议用户修改
gamma	TPE算法的一定分位数，用于划分l(x)和g(x)	float，范围(0,1)，一般不建议用户修改

模拟退火算法（Anneal）

模拟退火算法即Anneal算法，是随机搜索中一个简单但有效的变体，它利用了响应曲面中的平滑度。退火速率不自适应。Anneal算法从先前采样的一个试验点作为起点，然后从与先验分布相似的分布中采样每组超参数，但其密度更集中在选择的试验点周围。随着时间推移，算法会倾向于从越来越接近最佳点处采样。在采样过程中，算法可能绘制一个次佳试验作为最佳试验，以一定概率跳出局部最优解。

表 8-26 模拟退火算法的参数说明

参数	说明	取值参考
num_samples	搜索尝试的超参组数	int，一般在10-20之间，值越大，搜索时间越长，效果越好
avg_best_idx	要探索试验的几何分布平均，从按照分数排序的试验中选择	float，一般不建议用户修改
shrink_coef	随着更多的点被探索，邻域采样大小的减少率	float，一般不建议用户修改

8.9.2 创建自动模型优化的训练作业

背景信息

如果用户使用的AI引擎为pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64和tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64，并且优化的超参类型为float类型，ModelArts支持用户使用超参搜索功能。

在0代码修改的基础下，实现算法模型的超参搜索。需要完成以下步骤：

1. **准备工作**
2. **创建算法**
3. **创建训练作业**
4. **查看超参搜索作业详情**

准备工作

- 数据已完成准备：已在ModelArts中创建可用的数据集，或者您已将用于训练的数据集上传至OBS目录。
- 请准备好训练脚本，并上传至OBS目录。训练脚本开发指导参见**开发用于预置框架训练的代码**。
- 在训练代码中，用户需打印搜索指标参数。
- 已在OBS创建至少1个空的文件夹，用于存储训练输出的内容。
- 由于训练作业运行需消耗资源，确保账户未欠费。
- 确保您使用的OBS目录与ModelArts在同一区域。

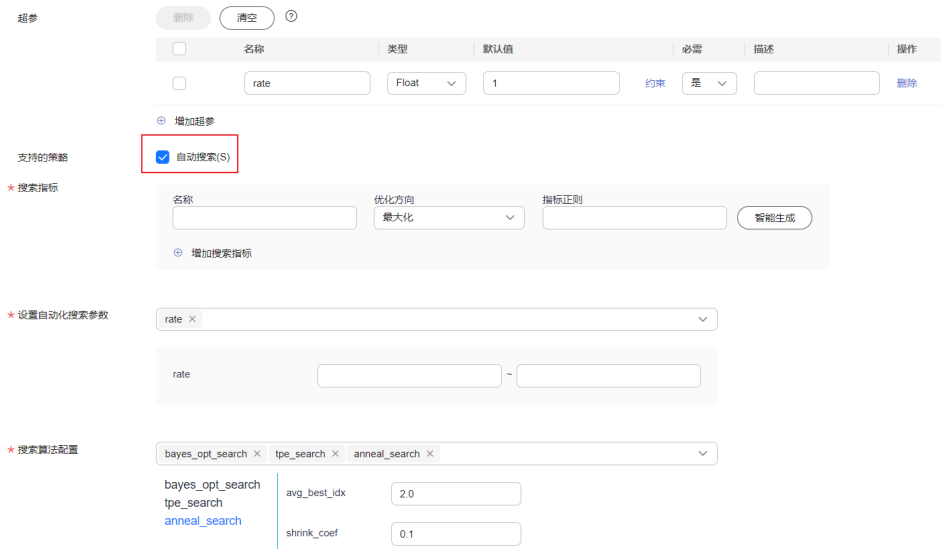
创建算法

进入ModelArts控制台，参考**创建算法**操作指导，创建自定义算法。镜像应该满足pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64或tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64引擎。

对于用户希望优化的超参，需在“超参”设置中定义，可以给定名称、类型、默认值、约束等，具体设置方法可以参考**表8-12**。

单击勾选“自动搜索”，为算法设置算法搜索功能。自动搜索作业运行过程中，ModelArts后台通过指标正则表达式获取搜索指标参数，朝指定的优化方向进行超参优化。用户需要在代码中打印搜索参数并在控制台配置以下参数。

图 8-18 设置算法搜索功能



- 搜索指标

  搜索指标为目标函数的值，通常可以设置为loss、accuracy等。通过优化搜索指标的目标值超优化方向收敛，找到最契合的超参，提高模型精度和收敛速度。

  表 8-27 搜索指标参数

参数	说明
名称	搜索指标的名称。需要与您在代码中打印的搜索指标参数保持一致。
优化方向	可选“最大化”或者“最小化”。
指标正则	填入正则表达式。您可以单击智能生成功能自动获取正则表达式。

- 设置自动化搜索参数

  从已设置的“超参”中选择可用于搜索优化的超参。优化的超参仅支持float类型，选中自动化搜索参数后，需设置取值范围。

- 搜索算法配置

ModelArts内置三种超参搜索算法，用户可以根据实际情况选择对应的算法，支持多选。对应的算法和参数解析请参考以下：

- bayes_opt_search：**贝叶斯优化（SMAC）**
- tpe_search：**TPE算法**
- anneal_search：**模拟退火算法（Anneal）**

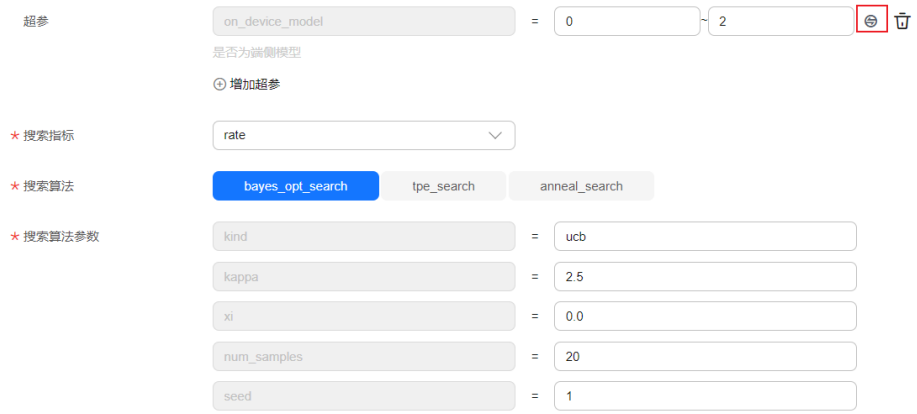
提交创建算法完成后即可执行下一步，创建训练作业。

创建训练作业

登录ModelArts控制台，参考**创建生产训练作业**操作指导，创建训练作业。用户需关注以下操作才能开启超参搜索。

当您选择支持超参搜索的算法，需单击超参的范围设置按钮才能开启超参搜索功能。

图 8-19 开启超参搜索功能



开启超参搜索功能后，用户可以设置搜索指标、搜索算法和搜索算法参数。三个参数显示的支持值与算法管理模块的超参设置对应。

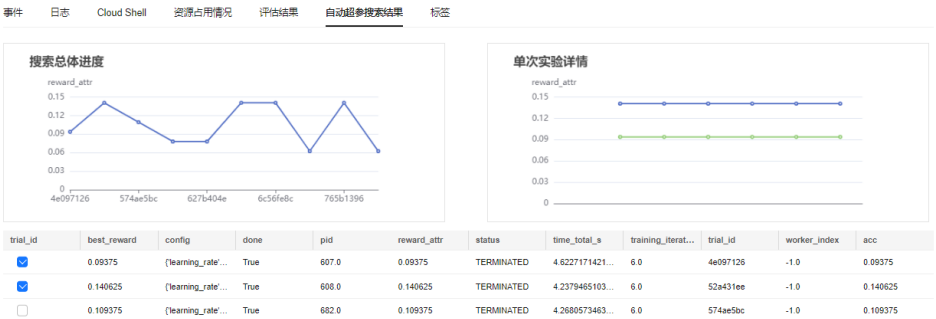
完成超参搜索作业的创建后，训练作业需要运行一段时间。

查看超参搜索作业详情

训练作业运行结束后，可以查看自动超参搜索结果判断此训练作业是否满意。

如果训练作业是超参搜索作业，进入训练作业详情页，选择“自动超参搜索结果”页签查看超参搜索结果。

图 8-20 超参搜索结果



8.10 模型训练高可靠性

8.10.1 训练作业容错检查

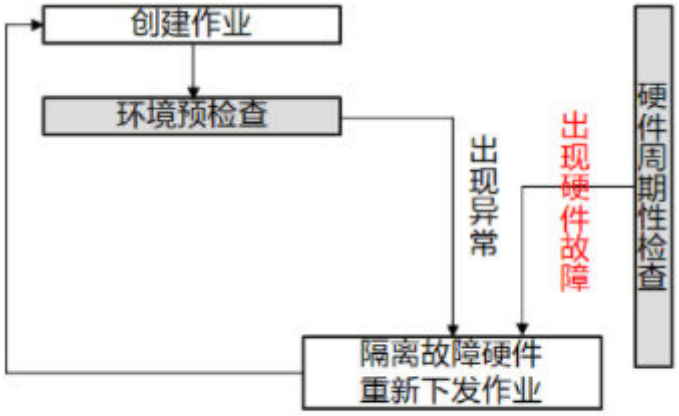
用户在训练模型过程中，存在因硬件故障而产生的训练失败场景。针对硬件故障场景，ModelArts提供容错检查功能，帮助用户隔离故障节点，优化用户训练体验。

容错检查包括两个检查项：环境预检测与硬件周期性检查。当环境预检查或者硬件周期性检查任一检查项出现故障时，隔离故障硬件并重新下发训练作业。针对于分布式场景，容错检查会检查本次训练作业的全部计算节点。

下图中有四个场景，其中场景四为正常训练作业失败场景，其他三个场景下可开启容错功能进行训练作业自动恢复。

- 场景一：环境预检测失败、硬件检测出现故障，系统隔离所有故障节点并重新下发训练作业。

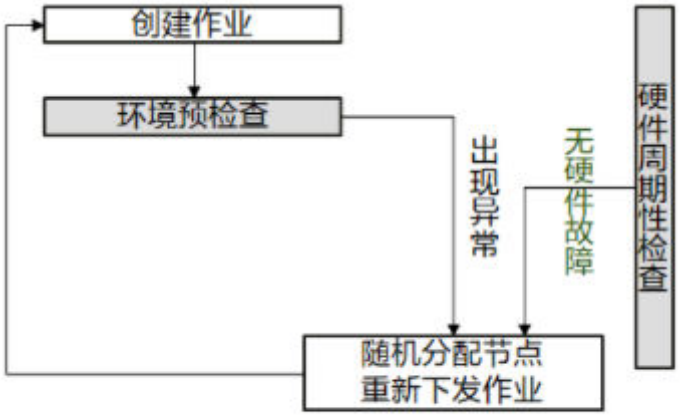
  图 8-21 预检失败&硬件故障



1. 预检失败&硬件故障

- 场景二：环境预检测失败、硬件无故障，系统随机再分配节点并重新下发训练作业。

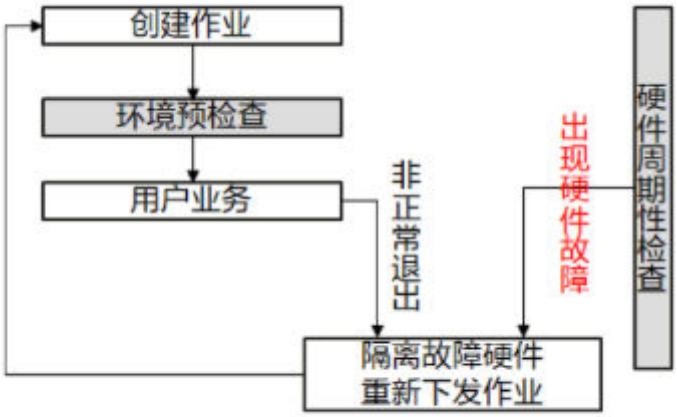
图 8-22 预检失败&硬件正常



2. 预检失败&硬件正常

- 场景三：环境预检测成功并进入用户业务阶段，硬件检测出现故障并且用户业务非正常退出，系统隔离所有故障节点并重新下发训练作业。

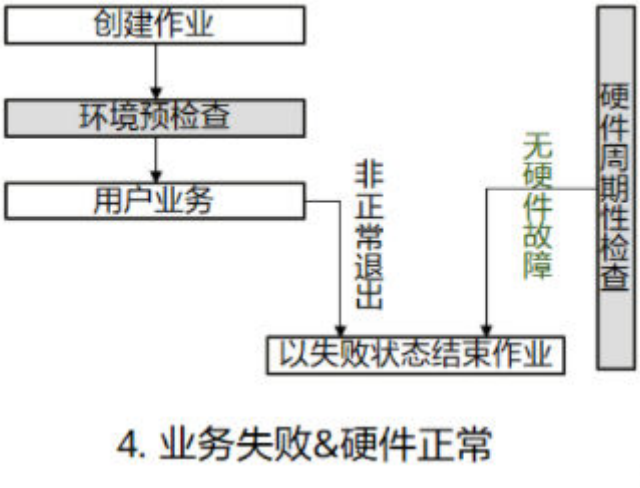
图 8-23 业务失败&硬件故障



3. 业务失败&硬件故障

- 场景四：环境预检测成功并进入用户业务阶段，硬件无故障，当用户业务异常时系统以失败状态结束作业。

图 8-24 业务失败&硬件正常



隔离故障节点后，系统会在新的计算节点上重新创建训练作业。如果资源池规格紧张，重新下发的训练作业会以第一优先级进行排队。如果排队时间超过30分钟，训练作业会自动退出。该现象表明资源池规格任务紧张，训练作业无法正常启动，推荐您购买专属资源池补充计算节点。

如果您使用专属资源池创建训练作业，容错检查识别的故障节点会被剔除。系统自动补充健康的计算节点至专属资源池。（该功能即将上线）

容错检查详细介绍请参考：

1. **开启容错检查**
2. **检测项目与执行条件**
3. **触发容错环境检测达到的效果**
4. 环境预检查通过后，如果发生硬件故障会导致用户业务中断。您可以在训练中补充reload ckpt的代码逻辑，使能读取训练中断前保存的预训练模型。指导请参考**设置断点续训练**。

开启容错检查

用户可以在创建训练作业时通过设置自动重启的方式开启容错检查。

- 使用ModelArts Standard控制台的创建训练作业页面设置自动重启：

  用户可以在控制台页面通过开关的方式开启自动重启。“自动重启”开关默认不开启，表示不做重新下发作业，也不会启用环境检测。打开开关后，允许设置重启次数为1~128次。

图 8-25 自动重启设置

- 使用API接口设置容错检查：

  用户可以通过API接口的方式开启自动重启。创建训练作业时，在“metadata”字段的“annotations”中传入“fault-tolerance/job-retry-num”字段。

  添加“fault-tolerance/job-retry-num”字段，视为开启自动重启，value的范围可以设置为1~128的整数。value值表示最大允许重新下发作业的次数。如果不传入则默认为0，表示不做重新下发作业，也不会启用环境检测。

  图 8-26 设置 API

```
{
    "kind": "job",
    "metadata": {
        "annotations": {
            "fault-tolerance/job-retry-num": "3"
        },
```

检测项目与执行条件

检测项目	item（日志关键字）	执行条件	检测成功要求
域名检测	dns	无	volcano容器的域名都解析成功（/etc/volcano下的“.host”文件中的域名解析成功）
磁盘空间-容器根目录	disk-size root	无	大于32GB
磁盘空间-/dev/shm目录	disk-size shm	无	大于1GB
磁盘空间-/cache目录	disk-size cache	无	大于32GB
ulimit检查	ulimit	使用IB网络时	• max locked memory > 16000 • open files > 1000000 • stack size > 8000 • max user processes > 1000000
gpu检查	gpu-check	使用gpu，且使用v2训练引擎时	检测到gpu

触发容错环境检测达到的效果

- 容错检查正常通过时，会打印检测项目的日志，表示具体涉及的检查项目成功。您可以通过在日志中搜索“item”关键字查看。当容错检查正常通过时，可以减少运行故障上报问题。

- 容错检查失败时，会打印检查失败的日志。您可以通过在日志中搜索“item”关键字查看失败信息。

  如果作业重启次数没有达到设定的次数，则会自动做重新下发作业。您可以通过搜索“error,exiting”关键字查找作业重启失败结束的日志。

使用 reload ckpt 恢复中断的训练

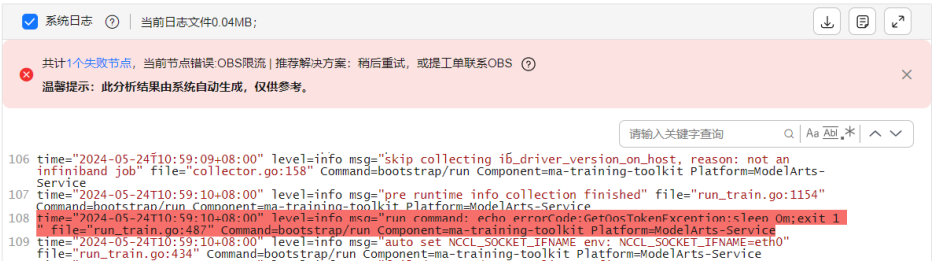
在容错机制下，如果因为硬件问题导致训练作业重启，用户可以在代码中读取预训练模型，恢复至重启前的训练状态。用户需要在代码里加上reload ckpt的代码，使能读取训练中断前保存的预训练模型。具体请参见**断点续训练**。

8.10.2 训练日志失败分析

在ModelArts Standard中训练作业遇到问题时，可首先查看日志，多数场景下的问题可以通过日志报错信息直接定位。

ModelArts Standard提供了训练作业失败定位与分析功能，如果训练作业运行失败，ModelArts会自动识别导致作业失败的原因，在训练日志界面上给出提示。提示包括三部分：失败的可能原因、推荐的解决方案以及对应的日志（底色标红部分）。

图 8-27 训练故障识别



ModelArts Standard会对部分常见训练错误给出分析建议，目前还不能识别所有错误，提供的失败可能原因仅供参考。针对分布式作业，只会显示当前节点的一个分析结果，作业的失败需要综合各个节点的失败原因做一个综合判断。

常见训练问题定位思路如下：

1. 根据日志界面提示中提供的分析建议解决。
   - 参考案例解决：会提供当前故障对应的指导文档链接，请参照文档中的解决方案修复问题。
   - 重建作业：建议重建作业进行重试，大概率能修复问题。
2. 上一步不能解决问题时，可以尝试分析日志中提示的错误信息，定位并解决问题。
3. 最后，如果以上均不能解决问题，可以提工单进行人工咨询。

8.10.3 训练作业卡死检测

什么是训练作业卡死检测

训练作业在运行中可能会因为某些未知原因导致作业卡死，如果不能及时发现，就会导致无法及时释放资源，从而造成极大的资源浪费。为了节省训练资源成本，提高使

用体验，ModelArts提供了卡死检测功能，能自动识别作业是否卡死，并在日志详情界面上展示，同时能配置通知及时提醒用户作业卡死。

检测规则

卡死检测主要是通过监控作业进程的状态和资源利用率来判定作业是否卡死。会启动一个进程来周期性地监控上述两个指标的变化情况。

- 进程状态：只要训练作业中存在进程IO有变化，进入下一个检测周期。如果在多个检测周期内，作业所有进程IO都没有变化，则进入资源利用率检测阶段。
- 资源利用率：在作业进程IO没有变化的情况下，采集一定时间段内的GPU利用率或NPU利用率，并根据这段时间内的GPU利用率或NPU利用率的方差和中位数来判断资源使用率是否有变化。如果没有变化，则判定作业卡死。

系统预置了卡死检测的环境变量“MA_HANG_DETECT_TIME=30”，表示30分钟内进程IO无变化则判定作业卡死。如果需要修改卡死检测时间，则可以修改环境变量“MA_HANG_DETECT_TIME”的值，具体操作指导请参见**管理训练容器环境变量**。

⚠ 注意

- 由于检测规则的局限性，当前卡死检测存在一定的误检率。如果是作业代码本身逻辑（如长时间sleep）导致的卡死，请忽略。

约束限制

卡死检测仅支持资源类型为GPU和NPU的训练作业。

操作步骤

卡死检测无需额外配置，作业运行中会自动执行检测。检测到作业卡死后会在训练作业详情页提示作业疑似卡死。如需检测到卡死后发送通知（短信、邮件等）请在作业创建页面配置事件通知。

常见案例：复制数据卡死

问题现象

调用mox.file.copy_parallel复制数据时卡死。

解决方案

- 复制文件和文件夹均可采用：
```
import moxing as mox
mox.file.set_auth(is_secure=False)
```
- 复制单个大文件5G以上时可采用：
```
from moxing.framework.file import file_io
```
查看当前moxing调用的接口版本：file_io._LARGE_FILE_METHOD，如果输出值为1则为V1版本，如果输出值为2，则为V2版本。

V1版本修改：file_io._NUMBER_OF_PROCESSES=1

V2版本修改：可以 file_io._LARGE_FILE_METHOD = 1，将模式设置成V1然后用V1的方式修改规避，也可以直接file_io._LARGE_FILE_TASK_NUM=1。

- 复制文件夹时可采用：
  ```
  mox.file.copy_parallel(threads=0,is_processing=False)
  ```

常见案例：训练前卡死

作业为多节点训练，且还未开始训练时发生卡死，可以在代码中加入os.environ["NCCL_DEBUG"] = "INFO"，查看NCCL DEBUG信息。

- **问题现象**1

  日志中还未出现NCCL DEBUG信息时已卡死。

  解决方案1

  检查代码，检查是否有参数中未传入“master_ip”和“rank”参数等问题。

- **问题现象2**

  分布式训练的日志中，发现有的节点含有GDR信息，而有的节点无GDR信息，导致卡死的原因可能为GDR。

  ```
  # 节点A日志
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1136:1191 [2] NCCL INFO Channel 00 : 3[5f000] -> 10[5b000] [receive] via NET/IB/0/GDRDMA
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1140:1196 [6] NCCL INFO Channel 00 : 14[e1000] -> 15[e9000] via P2P/IPC
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1141:1187 [7] NCCL INFO Channel 00 : 15[e9000] -> 11[5f000] via P2P/IPC
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1138:1189 [4] NCCL INFO Channel 00 : 12[b5000] -> 14[e1000] via P2P/IPC
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-1:1137:1197 [3] NCCL INFO Channel 00 : 11[5f000] -> 16[2d000] [send] via NET/IB/0/GDRDMA

  # 节点B日志
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1139:1198 [2] NCCL INFO Channel 00 : 18[5b000] -> 19[5f000] via P2P/IPC
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1144:1200 [7] NCCL INFO Channel 00 : 23[e9000] -> 20[b5000] via P2P/IPC
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1142:1196 [5] NCCL INFO Channel 00 : 21[be000] -> 17[32000] via P2P/IPC
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1143:1194 [6] NCCL INFO Channel 00 : 22[e1000] -> 21[be000] via P2P/IPC
  modelarts-job-a7305e27-d1cf-4c71-ae6e-a12da6761d5a-worker-2:1141:1191 [4] NCCL INFO Channel 00 : 20[b5000] -> 22[e1000] via P2P/IPC
  ```

  解决方案2

  在程序开头设置“os.environ["NCCL_NET_GDR_LEVEL"] = '0'”关闭使用GDR，或者寻找运维人员将机器添加GDR。

- **问题现象3**

  NCCL信息中报出Got completion with error 12, opcode 1, len 32478, vendor err 129等通信信息时，说明当前网络不是很稳定。

  解决方案3

  可加入3个环境变量。

  - NCCL_IB_GID_INDEX=3： 使用RoCE v2协议，默认使用RoCE v1，但是v1在交换机上没有拥塞控制，可能丢包，而且后面的交换机不会支持v1，就无法启动。
  - NCCL_IB_TC=128：数据包走交换机的队列4通道，这是RoCE协议标准。
  - NCCL_IB_TIMEOUT=22：把超时时间设置长一点，正常情况下网络不稳定会有5秒钟左右的间断，超过5秒就返回timeout了，改成22预计有二十秒左右，算法为4.096 μs * 2 ^ timeout。

常见案例：训练中途卡死

- **问题现象1**

  检测每个节点日志是否有报错信息，某个节点报错但作业未退出导致整个训练作业卡死。

  解决方案1

  查看报错原因，解决报错。

- **问题现象2**

  作业卡在sync-batch-norm中或者训练速度变慢。pytorch如果开了sync-batch-norm，多机会慢，因开了sync-batch-norm以后，每一个iter里面每个batch-norm层都要做同步，通信量很大，而且要所有节点同步。

  解决方案2

  关掉sync-batch-norm，或者升pytorch版本，升级pytorch到1.10。

- **问题现象3**

  作业卡在tensorboard中，出现报错：

  ```
  writer = Sumarywriter('./path)/to/log')
  ```

  解决方案3

  存储路径设为本地路径，如cache/tensorboard，不要使用OBS路径。

- **问题现象4**

  使用pytorch中的dataloader读数据时，作业卡在读数据过程中，日志停在训练的过程中并不再更新日志。

  解决方案4

  用dataloader读数据时，适当减小num_worker。

常见案例：训练最后一个 epoch 卡死

问题现象

通过日志查看数据切分是否对齐，如果未对齐，容易导致部分进程完成训练退出，而部分训练进程因未收到其他进程反馈卡死，如下图同一时间有的进程在epoch48，而有的进程在epoch49。

```
loss exit lane:0.12314446270465851
step loss is 0.29470521211624146
[2022-04-26 13:57:20,757][INFO][train_epoch]:Rank:2 Epoch:[48][20384/all] Data Time 0.000(0.000) Net Time 0.705(0.890) Loss 0.3403(0.3792)LR 0.00021887
[2022-04-26 13:57:20,757][INFO][train_epoch]:Rank:1 Epoch:[48][20384/all] Data Time 0.000(0.000) Net Time 0.705(0.891) Loss 0.3028(0.3466) LR 0.00021887
[2022-04-26 13:57:20,757][INFO][train_epoch]:Rank:4 Epoch:[49][20384/all] Data Time 0.000(0.147) Net Time 0.705(0.709) Loss 0.3364(0.3414)LR 0.00021887
[2022-04-26 13:57:20,758][INFO][train_epoch]:Rank:3 Epoch:[49][20384/all] Data Time 0.000 (0.115) Net Time 0.706(0.814) Loss 0.3345(0.3418) LR 0.00021887
[2022-04-26 13:57:20,758][INFO][train_epoch]:Rank:0 Epoch:[49][20384/all] Data Time 0.000(0.006) Net Time 0.704(0.885) Loss 0.2947(0.3566) LR 0.00021887
[2022-04-26 13:57:20,758][INFO][train_epoch]:Rank:7 Epoch:[49][20384/all] Data Time 0.001 (0.000) Net Time 0.706 (0.891) Loss 0.3782(0.3614) LR 0.00021887
[2022-04-26 13:57:20,759][INFO][train_epoch]:Rank:5 Epoch:[48][20384/all] Data Time 0.000(0.000) Net Time 0.706(0.891) Loss 0.5471(0.3642) LR 0.00021887
[2022-04-26 13:57:20,763][INFO][train_epoch]:Rank:6 Epoch:[49][20384/all] Data Time 0.000(0.000) Net Time 0.704(0.891) Loss 0.2643(0.3390)LR 0.00021887
stage 1 loss 0.4600560665130615 mul_cls_loss loss:0.01245919056236744 mul_offset_loss 0.44759687781333923 origin stage2_loss 0.048592399805784225
stage 1 loss:0.4600560665130615 stage 2 loss:0.048592399805784225 loss exit lane:0.10233864188194275
```

解决方案

使用tensor的切分操作对齐数据。

8.10.4 训练作业重调度

当训练作业发生故障恢复时（例如进程级恢复、POD级重调度、JOB级重调度等），作业详情页面中会出现“故障恢复详情”页签，里面记录了训练作业的启停情况。

1. 在ModelArts管理控制台的左侧导航栏中选择“模型训练 > 训练作业”。
2. 在训练作业列表中，单击作业名称进入训练作业详情页面。
3. 在训练作业详情页面，单击“故障恢复详情”页签查看故障恢复信息。

图 8-28 查看故障恢复详情

8.10.5 设置断点续训练

什么是断点续训练

断点续训练是指因为某些原因（例如容错重启、资源抢占、作业卡死等）导致训练作业还未完成就被中断，下一次训练可以在上一次的训练基础上继续进行。这种方式对于需要长时间训练的模型而言比较友好。

断点续训练是通过checkpoint机制实现。

checkpoint的机制是：在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

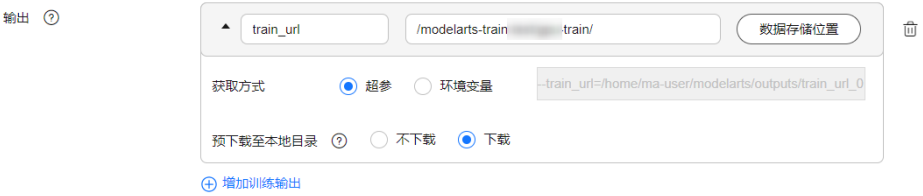
当需要从训练中断的位置接续训练，只需要加载checkpoint，并用checkpoint信息初始化训练状态即可。用户需要在代码里加上reload ckpt的代码，使能读取前一次训练保存的预训练模型。

ModelArts Standard 中如何实现断点续训练

在ModelArts Standard训练中实现断点续训练或增量训练，建议使用“训练输出”功能。

在创建训练作业时，设置训练“输出”参数为“train_url”，在指定的训练输出的数据存储位置中保存**Checkpoint**，且**“预下载至本地目录”**选择“下载”。选择预下载至本地目录时，系统在训练作业启动前，自动将数据存储位置中的**Checkpoint**文件下载到训练容器的本地目录。

图 8-29 训练输出设置



断点续训练建议和训练容错检查（即自动重启）功能同时使用。在创建训练作业页面，开启“自动重启”开关。训练环境预检测失败、或者训练容器硬件检测故障、或者训练作业失败时会自动重新下发并运行训练作业。

PyTorch 版 reload ckpt

- PyTorch模型保存有两种方式。
  - 仅保存模型参数
    ```
    state_dict = model.state_dict()
    torch.save(state_dict, path)
    ```
  - 保存整个Model（不推荐）
    ```
    torch.save(model, path)
    ```
- 可根据step步数、时间等周期性保存模型的训练过程的产物。

  将模型训练过程中的网络权重、优化器权重、以及epoch进行保存，便于中断后继续训练恢复。

  ```
  checkpoint = {
          "net": model.state_dict(),
          "optimizer": optimizer.state_dict(),
          "epoch": epoch
  }
  if not os.path.isdir('model_save_dir'):
      os.makedirs('model_save_dir')
  torch.save(checkpoint,'model_save_dir/ckpt_{}.pth'.format(str(epoch)))
  ```
- 完整代码示例。

  ```
  import os
  import argparse
  parser = argparse.ArgumentParser()
  parser.add_argument("--train_url", type=str)
  args, unparsed = parser.parse_known_args()
  args = parser.parse_known_args()
  # train_url 将被赋值为"/home/ma-user/modelarts/outputs/train_url_0"
  train_url = args.train_url

  # 判断输出路径中是否有模型文件。如果无文件则默认从头训练，如果有模型文件，则加载epoch值最大的ckpt文件当做预训练模型。
  if os.listdir(train_url):
      print('> load last ckpt and continue training!!')
      last_ckpt = sorted([file for file in os.listdir(train_url) if file.endswith(".pth")])[-1]
      local_ckpt_file = os.path.join(train_url, last_ckpt)
      print('last_ckpt:', last_ckpt)
      # 加载断点
      checkpoint = torch.load(local_ckpt_file)
      # 加载模型可学习参数
      model.load_state_dict(checkpoint['net'])
      # 加载优化器参数
      optimizer.load_state_dict(checkpoint['optimizer'])
      # 获取保存的epoch，模型会在此epoch的基础上继续训练
      start_epoch = checkpoint['epoch']
  start = datetime.now()
  total_step = len(train_loader)
  for epoch in range(start_epoch + 1, args.epochs):
      for i, (images, labels) in enumerate(train_loader):
          images = images.cuda(non_blocking=True)
          labels = labels.cuda(non_blocking=True)
          # Forward pass
          outputs = model(images)
          loss = criterion(outputs, labels)
          # Backward and optimize
          optimizer.zero_grad()
          loss.backward()
          optimizer.step()
          ...

      # 保存模型训练过程中的网络权重、优化器权重、以及epoch
  ```



```
checkpoint = {
      "net": model.state_dict(),
      "optimizer": optimizer.state_dict(),
      "epoch": epoch
    }
if not os.path.isdir(train_url):
    os.makedirs(train_url)
    torch.save(checkpoint, os.path.join(train_url, 'ckpt_best_{}.pth'.format(epoch)))
```

8.10.6 设置无条件自动重启

背景信息

训练过程中可能会碰到预期外的情况导致训练失败，且无法及时重启训练作业，导致训练周期长，而无条件自动重启可以避免这类问题。无条件自动重启是指当训练作业失败时，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。为了避免无效重启浪费算力资源，系统最多只支持连续无条件重启3次。

为了避免丢失训练进度、浪费算力，开启此功能前请确认代码已适配断点续训，操作指导请参见**设置断点续训练**。

当训练过程中触发了自动重启，则系统会记录重启信息，在训练作业详情页可以查看故障恢复详情，具体请参见**训练作业重调度**。

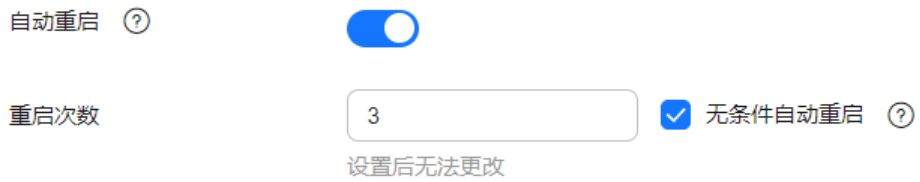
开启无条件自动重启

开启无条件自动重启有2种方式：控制台设置或API接口设置。

- 控制台设置

  在创建训练作业页面，开启“自动重启”开关，并勾选“无条件自动重启”，开启无条件自动重启。开启无条件自动重启后，只要系统检测到训练异常，就无条件重启训练作业。如果未勾选“无条件自动重启”只是打开了“自动重启”开关，则表示仅环境问题导致训练作业异常时才会自动重启，其他问题导致训练作业异常时会直接返回“运行失败”。

  图 8-30 开启无条件重启

- API接口设置

  通过API接口创建训练作业时，在“metadata”字段的“annotations”中传入“fault-tolerance/job-retry-num”和“fault-tolerance/job-unconditional-retry”字段。“fault-tolerance/job-retry-num”赋值为1~128表示开启自动重启，“fault-tolerance/job-unconditional-retry”赋值为“true”表示启用了无条件自动重启。

  ```
  {
    "kind": "job",
    "metadata": {
      "annotations": {
        "fault-tolerance/job-retry-num": "8",
        "fault-tolerance/job-unconditional-retry": "true"
      }
  ```

```
    }
}
```

8.11 管理模型训练作业

8.11.1 查看训练作业详情

1. 登录ModelArts管理控制台。
2. 在左侧导航栏中，选择“模型训练 > 训练作业”，进入“训练作业”列表。

   在作业列表，单击“导出”，可以将训练作业根据时间周期导出Excel表到本地。最多只支持导出前200行数据。
3. 在“训练作业”列表中，单击作业名称，进入训练作业详情页。
4. 在训练作业详情页的左侧，可以查看此次训练作业的基本信息和算法配置的相关信息。
   - **训练作业基本信息**

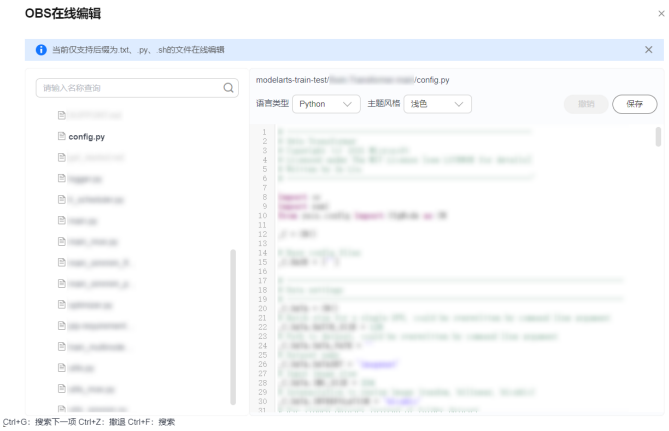
     表 8-28 训练作业基本信息

参数	说明
“作业ID”	训练作业唯一标识。
“作业状态”	训练作业状态。
“创建时间”	记录训练作业创建时间。
“运行时长”	记录训练作业运行时长。
“重启次数”	记录训练过程中时如果出现故障，作业自动重启的次数。仅当创建训练作业时开启“自动重启”功能时可见。
“描述”	训练作业的描述。 您可以单击编辑图标，更新训练作业的描述。
“作业优先级”	显示训练作业的优先级。

   - **训练作业参数**

     表 8-29 训练作业参数

参数	说明
“算法名称”	本次训练作业使用的算法。单击算法名称，可以跳转至算法详情页面。
“预置镜像”	本次训练作业使用的预置镜像框架。仅使用预置框架创建的训练作业才有该参数。

参数	说明
“自定义镜像”	本次训练作业使用的自定义镜像。仅使用自定义镜像创建的训练作业才有该参数。
“代码目录”	训练作业代码目录所在的OBS路径。 您可以单击代码目录后的“编辑代码”，在“OBS在线编辑”对话框中实时编辑训练脚本代码。当训练作业状态为“等待中”、“创建中”和“运行中”时，不支持“OBS在线编辑”功能。  **说明** 当您使用订阅算法创建训练作业时，不支持该参数。
“启动文件”	训练作业启动文件位置。 **说明** 当您使用订阅算法创建训练作业时，不支持该参数。
“运行用户ID”	容器运行时的用户ID。
“本地代码目录”	训练代码在训练容器中的存放路径。
“工作目录”	训练启动文件在训练容器中的路径。
“实例数”	本次训练作业设置的实例数。
“专属资源池”	专属资源池信息，仅当训练作业使用专属资源池时可见。
“实例规格”	本次训练作业使用的训练规格。
“输入-输入路径”	本次训练中，输入数据的OBS路径。
“输入-参数名称”	算法代码中，输入路径指代的参数。
“输入-获取方式”	本次训练作业的输入采用的获取方式。

参数	说明
“输入-本地路径（训练参数值）”	训练启动后，ModelArts将OBS路径中的数据下载至后台容器，本地路径指ModelArts后台容器中存储输入数据的路径。
“输出-输出路径”	本次训练中，输出数据的OBS路径。
“输出-参数名称”	算法代码中，输出路径指代的参数。
“输出-获取方式”	本次训练作业的输出采用的获取方式。
“输出-本地路径（训练参数值）”	ModelArts后台容器中存储训练输出的路径。
“超参”	本次训练作业使用的超参。
“环境变量”	本次训练作业设置的环境变量。

8.11.2 查看训练作业资源占用情况

约束限制

训练作业的资源占用情况系统会自动保存30天，过期会被清除。

如何查看训练作业资源使用详情

1. 在ModelArts管理控制台的左侧导航栏中选择“模型训练 > 训练作业”。
2. 在训练作业列表中，单击作业名称进入训练作业详情页面。
3. 在训练作业详情页面，单击“资源占用情况”页签查看计算节点的资源使用情况，最多可显示最近三天的数据。在“资源占用情况”窗口打开时，会定期向后台获取最新的资源使用率数据并刷新。

   操作一：如果训练作业使用多个计算节点，可以通过实例名称的下拉框切换节点。

   操作二：单击图例“cpuUsage”、“gpuMemUsage”、“gpuUtil”、“memUsage”“npuMemUsage”、“npuUtil”，可以添加或取消对应参数的使用情况图。

   操作三：鼠标悬浮在图片上的时间节点，可查看对应时间节点的占用率情况。

图 8-31 资源占用情况

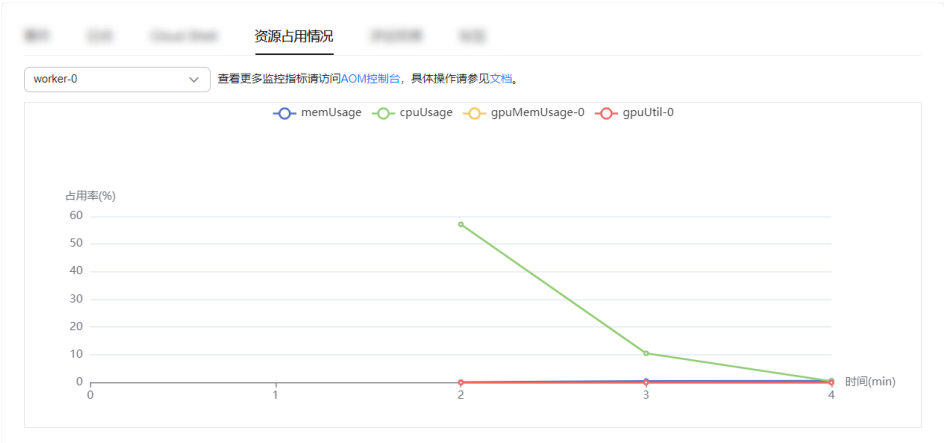


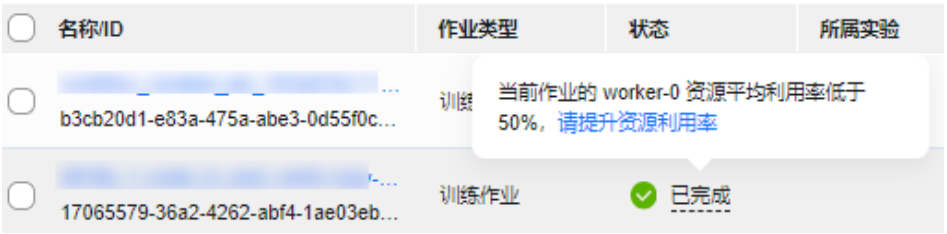
表 8-30 参数说明

参数	说明
cpuUsage	cpu使用率。
gpuMemUsage	gpu内存使用率。
gpuUtil	gpu使用情况。
memUsage	内存使用率。
npuMemUsage	npu内存使用率。
npuUtil	npu使用情况。

如何判断训练作业资源利用率高低

在模型训练的训练作业列表页可以查看作业资源利用率情况。当作业worker-0实例的GPU/NPU的平均利用率低于50%时，在训练作业列表中会进行告警提示。

图 8-32 作业列表显示作业资源利用率情况



此处的作业资源利用率只涉及GPU和NPU资源。作业worker-0实例的GPU/NPU平均利用率计算方法：将作业worker-0实例的各个GPU/NPU加速卡每个时间点的利用率汇总取平均值。

如何提高训练作业资源利用率

- 适当增大batch_size：较大的batch_size可以让GPU/NPU计算单元获得更高的利用率，但是也要根据实际情况来选择batch_size，防止batch_YLLsize过大导致内存溢出。
- 提升数据读取的效率：如果读取一个batch数据的时间要长于GPU/NPU计算一个batch的时间，就有可能出现GPU/NPU利用率上下浮动的情况。建议优化数据读取和数据增强的性能，例如将数据读取并行化，或者使用NVIDIA Data Loading Library（DALI）等工具提高数据增强的速度。
- 模型保存不要太频繁：模型保存操作一般会阻塞训练，如果模型较大，并且较频繁地进行保存，就会影响GPU/NPU利用率。同理，其他非GPU/NPU操作尽量不要阻塞训练主进程太多的时间，如日志打印，保存训练指标信息等。

8.11.3 查看模型评估结果

训练作业运行结束后，ModelArts可为您的模型进行评估，并且给出调优诊断和建议。

- 针对使用预置算法创建训练作业，无需任何配置，即可查看此评估结果（由于每个模型情况不同，系统将自动根据您的模型指标情况，给出一些调优建议，请仔细阅读界面中的建议和指导，对您的模型进行进一步的调优）。
- 针对用户自己编写训练脚本或自定义镜像方式创建的训练作业，则需要在您的训练代码中添加评估代码，才可以在训练作业结束后查看相应的评估诊断建议。

  说明

  - 只支持验证集的数据格式为图片。
  - 目前，仅如下常用框架的训练脚本支持添加评估代码。
    - TF-1.13.1-python3.6
    - TF-2.1.0-python3.6
    - PyTorch-1.4.0-python3.6

下文将介绍如何在训练中使用评估代码。对训练代码做一定的适配和修正，分为三个方面：**添加输出目录**、**复制数据集到本地**、**映射数据集路径到OBS**。

添加输出目录

添加输出目录的代码比较简单，即在代码中添加一个输出评估结果文件的目录，被称为train_url，也就是页面上的训练输出位置。并把train_url添加到使用的函数analysis中，使用save_path来获取train_url。示例代码如下所示：

```
FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('model_url', '', 'path to saved model')
tf.app.flags.DEFINE_string('data_url', '', 'path to output files')
tf.app.flags.DEFINE_string('train_url', '', 'path to output files')
tf.app.flags.DEFINE_string('adv_param_json',
                           '{"attack_method":"FGSM","eps":40}',
                           'params for adversarial attacks')
FLAGS(sys.argv, known_only=True)

...

# analyse
res = analyse(
    task_type=task_type,
    pred_list=pred_list,
    label_list=label_list,
    name_list=file_name_list,
```

```
    label_map_dict=label_dict,
    save_path=FLAGS.train_url)
```

复制数据集到本地

复制数据集到本地主要是为了防止长时间访问OBS容易导致OBS连接中断使得作业卡住，所以一般先将数据复制到本地再进行操作。

数据集复制有两种方式，推荐使用OBS路径复制。

- OBS路径（推荐）

  直接使用moxing的copy_parallel接口，复制对应的OBS路径。
- ModelArts数据管理中的数据集（即manifest文件格式）

  使用moxing的copy_manifest接口将文件复制到本地并获取新的manifest文件路径，然后使用SDK解析新的manifest文件。

说明

ModelArts数据管理模块在重构升级中，对未使用过数据管理的用户不可见。建议新用户将训练数据存放至OBS桶中使用。

```
if data_path.startswith('obs://'):
    if '.manifest' in data_path:
        new_manifest_path, _ = mox.file.copy_manifest(data_path, '/cache/data/')
        data_path = new_manifest_path
    else:
        mox.file.copy_parallel(data_path, '/cache/data/')
        data_path = '/cache/data/'
    print('------------- download dataset success ------------')
```

映射数据集路径到 OBS

由于最终JSON体中需要填写的是图片文件的真实路径，也就是OBS对应的路径，所以在复制到本地做完分析和评估操作后，需要将原来的本地数据集路径映射到OBS路径，然后将新的list送入analysis接口。

如果使用的是OBS路径作为输入的data_url，则只需要替换本地路径的字符串即可。

```
if FLAGS.data_url.startswith('obs://'):
    for idx, item in enumerate(file_name_list):
        file_name_list[idx] = item.replace(data_path, FLAGS.data_url)
```

如果使用manifest文件，需要再解析一遍原版的manifest文件获取list，然后再送入analysis接口。

```
if or FLAGS.data_url.startswith('obs://'):
    if 'manifest' in FLAGS.data_url:
            file_name_list = []
            manifest, _ = get_sample_list(
                manifest_path=FLAGS.data_url, task_type='image_classification')
            for item in manifest:
                if len(item[1]) != 0:
                    file_name_list.append(item[0])
```

完整的适配了训练作业创建的图像分类样例代码如下：

```
import json
import logging
import os
import sys
import tempfile

import h5py
```

```
import numpy as np
from PIL import Image

import moxing as mox
import tensorflow as tf
from deep_moxing.framework.manifest_api.manifest_api import get_sample_list
from deep_moxing.model_analysis.api import analyse, tmp_save
from deep_moxing.model_analysis.common.constant import TMP_FILE_NAME

logging.basicConfig(level=logging.DEBUG)

FLAGS = tf.app.flags.FLAGS
tf.app.flags.DEFINE_string('model_url', '', 'path to saved model')
tf.app.flags.DEFINE_string('data_url', '', 'path to output files')
tf.app.flags.DEFINE_string('train_url', '', 'path to output files')
tf.app.flags.DEFINE_string('adv_param_json',
                           '{"attack_method":"FGSM","eps":40}',
                           'params for adversarial attacks')
FLAGS(sys.argv, known_only=True)


def _preprocess(data_path):
    img = Image.open(data_path)
    img = img.convert('RGB')
    img = np.asarray(img, dtype=np.float32)
    img = img[np.newaxis, :, :, :]
    return img


def softmax(x):
    x = np.array(x)
    orig_shape = x.shape
    if len(x.shape) > 1:
        # Matrix
        x = np.apply_along_axis(lambda x: np.exp(x - np.max(x)), 1, x)
        denominator = np.apply_along_axis(lambda x: 1.0 / np.sum(x), 1, x)
        if len(denominator.shape) == 1:
            denominator = denominator.reshape((denominator.shape[0], 1))
        x = x * denominator
    else:
        # Vector
        x_max = np.max(x)
        x = x - x_max
        numerator = np.exp(x)
        denominator = 1.0 / np.sum(numerator)
        x = numerator.dot(denominator)
    assert x.shape == orig_shape
    return x


def get_dataset(data_path, label_map_dict):
    label_list = []
    img_name_list = []
    if 'manifest' in data_path:
        manifest, _ = get_sample_list(
            manifest_path=data_path, task_type='image_classification')
        for item in manifest:
            if len(item[1]) != 0:
                label_list.append(label_map_dict.get(item[1][0]))
                img_name_list.append(item[0])
            else:
                continue
    else:
        label_name_list = os.listdir(data_path)
        label_dict = {}
        for idx, item in enumerate(label_name_list):
            label_dict[str(idx)] = item
            sub_img_list = os.listdir(os.path.join(data_path, item))
            img_name_list += [
```

```
                os.path.join(data_path, item, img_name) for img_name in sub_img_list
            ]
            label_list += [label_map_dict.get(item)] * len(sub_img_list)
    return img_name_list, label_list


def deal_ckpt_and_data_with_obs():
    pb_dir = FLAGS.model_url
    data_path = FLAGS.data_url

    if pb_dir.startswith('obs://'):
        mox.file.copy_parallel(pb_dir, '/cache/ckpt/')
        pb_dir = '/cache/ckpt'
        print('------------- download success ------------')
    if data_path.startswith('obs://'):
        if '.manifest' in data_path:
            new_manifest_path, _ = mox.file.copy_manifest(data_path, '/cache/data/')
            data_path = new_manifest_path
        else:
            mox.file.copy_parallel(data_path, '/cache/data/')
            data_path = '/cache/data/'
        print('------------- download dataset success ------------')
    assert os.path.isdir(pb_dir), 'Error, pb_dir must be a directory'
    return pb_dir, data_path


def evalution():
    pb_dir, data_path = deal_ckpt_and_data_with_obs()
    index_file = os.path.join(pb_dir, 'index')
    try:
        label_file = h5py.File(index_file, 'r')
        label_array = label_file['labels_list'][:].tolist()
        label_array = [item.decode('utf-8') for item in label_array]
    except Exception as e:
        logging.warning(e)
        logging.warning('index file is not a h5 file, try json.')
        with open(index_file, 'r') as load_f:
            label_file = json.load(load_f)
        label_array = label_file['labels_list'][:]
    label_map_dict = {}
    label_dict = {}
    for idx, item in enumerate(label_array):
        label_map_dict[item] = idx
        label_dict[idx] = item
    print(label_map_dict)
    print(label_dict)

    data_file_list, label_list = get_dataset(data_path, label_map_dict)

    assert len(label_list) > 0, 'missing valid data'
    assert None not in label_list, 'dataset and model not match'

    pred_list = []
    file_name_list = []
    img_list = []

    for img_path in data_file_list:
        img = _preprocess(img_path)
        img_list.append(img)
        file_name_list.append(img_path)

    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    config.gpu_options.visible_device_list = '0'
    with tf.Session(graph=tf.Graph(), config=config) as sess:
        meta_graph_def = tf.saved_model.loader.load(
            sess, [tf.saved_model.tag_constants.SERVING], pb_dir)
        signature = meta_graph_def.signature_def
        signature_key = 'predict_object'
```

```
        input_key = 'images'
        output_key = 'logits'
        x_tensor_name = signature[signature_key].inputs[input_key].name
        y_tensor_name = signature[signature_key].outputs[output_key].name
        x = sess.graph.get_tensor_by_name(x_tensor_name)
        y = sess.graph.get_tensor_by_name(y_tensor_name)
        for img in img_list:
            pred_output = sess.run([y], {x: img})
            pred_output = softmax(pred_output[0])
            pred_list.append(pred_output[0].tolist())

    label_dict = json.dumps(label_dict)
    task_type = 'image_classification'

    if FLAGS.data_url.startswith('obs://'):
        if 'manifest' in FLAGS.data_url:
            file_name_list = []
            manifest, _ = get_sample_list(
                manifest_path=FLAGS.data_url, task_type='image_classification')
            for item in manifest:
                if len(item[1]) != 0:
                    file_name_list.append(item[0])
        for idx, item in enumerate(file_name_list):
            file_name_list[idx] = item.replace(data_path, FLAGS.data_url)
    # analyse
    res = analyse(
        task_type=task_type,
        pred_list=pred_list,
        label_list=label_list,
        name_list=file_name_list,
        label_map_dict=label_dict,
        save_path=FLAGS.train_url)

if __name__ == "__main__":
    evalution()
```

8.11.4 查看训练作业事件

训练作业的（从用户可看见训练作业开始）整个生命周期中，每一个关键事件点在系统后台均有记录，用户可随时在对应训练作业的详情页面进行查看。

方便用户更清楚的了解训练作业运行过程，遇到任务异常时，更加准确的排查定位问题。当前支持的作业事件如下所示：

- 训练作业创建成功
- 训练作业创建失败报错：
- 准备阶段超时。可能原因是跨区域算法同步或者创建共享存储超时
- 训练作业已排队，正在等待资源分配
- 训练作业排队失败
- 训练作业开始运行
- 训练作业运行成功
- 训练作业运行失败
- 训练作业被抢占
- 系统检测到您的作业疑似卡死，请及时前往作业详情界面查看并处理
- 训练作业已重启
- 训练作业已被手动终止
- 训练作业已被终止（最大运行时长：xh）

- 训练作业已被手动删除
- 计费信息同步结束
- [worker-0] 训练环境预检中
- [worker-0] [耗时： 秒] 预检完成
- [worker-0] [耗时： 秒] 检查失败。发现异常：
- [worker-0] [耗时： 秒] 检查失败。发现错误：
- [worker-0] 训练代码下载中
- [worker-0] [耗时： 秒] 训练代码下载完成
- [worker-0] [耗时： 秒] 训练代码下载失败，失败原因：
- [worker-0] 训练输入下载中
- [worker-0] [耗时： 秒] 训练输入（参数名称： ）下载完成
- [worker-0] [耗时： 秒] 训练输入（参数名称： ）下载失败，失败原因：
- [worker-0] 正在安装Python依赖包，导入文件：
- [worker-0] [耗时： 秒] Python依赖包安装完成，导入文件：
- [worker-0] 训练作业开始运行
- [worker-0] 训练作业运行结束，退出码
- [worker-0] 训练输入上传中
- [worker-0] [耗时： 秒] 训练输出（参数名称： ）上传完成

训练运行到结束的过程中，关键事件支持手动/自动刷新。

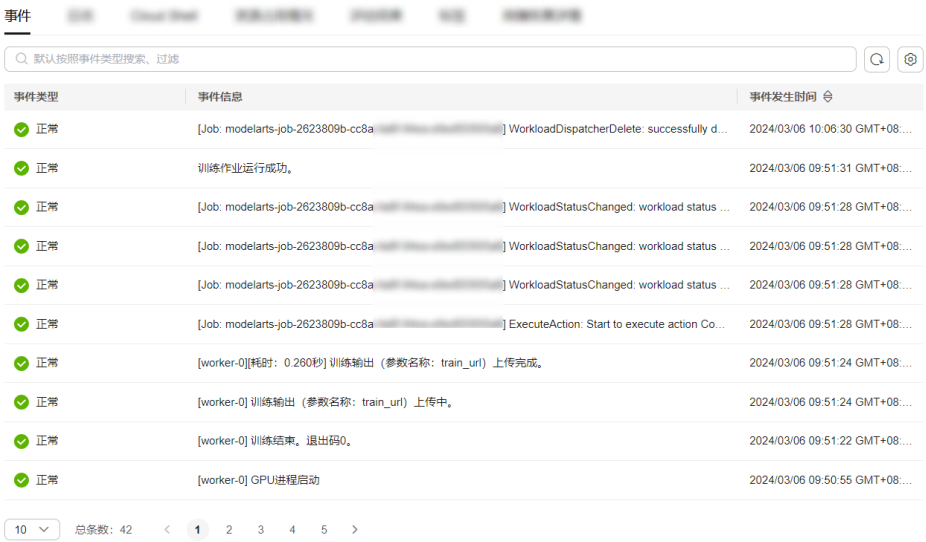
约束限制

训练作业的事件信息系统会自动保存30天，过期会被清除。

查看操作

1. 在ModelArts管理控制台的左侧导航栏中选择“模型训练 > 训练作业”。
2. 在训练作业列表中，单击作业名称进入训练作业详情页面。
3. 在训练作业详情页面，单击“事件”页签查看事件信息。

图 8-33 查看事件信息



8.11.5 查看训练作业日志

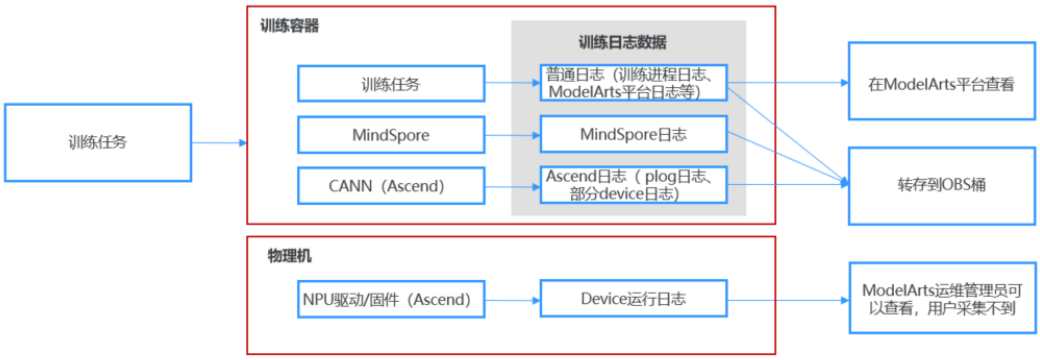
训练日志定义

训练日志用于记录训练作业运行过程和异常信息，为快速定位作业运行中出现的问题提供详细信息。用户代码中的标准输出、标准错误信息会在训练日志中呈现。在ModelArts中训练作业遇到问题时，可首先查看日志，多数场景下的问题可以通过日志报错信息直接定位。

训练日志包括普通训练日志和Ascend相关日志。

- **普通日志说明**：当使用Ascend之外的资源训练时仅产生普通训练日志，普通日志中包含训练进程日志、pip-requirement.txt安装日志和ModelArts平台日志。
- **Ascend场景日志说明**：使用Ascend资源训练时会产生device日志、plog日志、proc log单卡训练日志、MindSpore日志、普通日志。

图 8-34 ModelArts 训练日志



说明

只有MindSpore+Ascend训练场景下会产生单独的MindSpore日志。其他AI引擎的日志都包含在普通日志中，无法区分。

训练日志的时效性

从日志产生的时效性上可以分为以下3种情况：

- 实时日志：训练作业实时运行时产生，在ModelArts训练作业详情页面上可以查看。
- 历史日志：训练作业结束后，可以在ModelArts训练作业详情页面上查看历史日志，ModelArts系统自动保存30天。
- 永久日志：转存到OBS桶中的训练日志，在创建训练作业时，打开永久保存日志开关设置作业日志路径即可将日志转存至OBS路径。

图 8-35 开启永久保存日志开关

实时日志和历史日志都是标准日志输出，内容上没有区别。Ascend训练场景下，永久日志中会包含Ascend日志，这部分日志内容在ModelArts界面上看不到。

普通日志说明

普通日志中包含训练进程日志、pip-requirement.txt安装日志和ModelArts Standard平台日志。

表 8-31 普通日志类型

日志类型	说明
训练进程日志	用户训练代码的标准输出。
pip-requirement.txt安装日志	如果用户有定义pip-requirement.txt文件，会产生pip包安装日志。
ModelArts平台日志	ModelArts平台产生的系统日志，主要用于运维人员定位平台问题。

普通日志的文件格式如下，其中task id为训练作业中的节点id。

```
统一日志格式：modelarts-job-[job id]-[task id].log
样例：log/modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-worker-0.log
```

- 单机训练作业只会生成一个日志文件，单机作业的task id默认为worker-0。
- 分布式场景下有多个节点日志文件并存，通过task id区分不同节点，例如：worker-0，worker-1等。

训练进程日志、“pip-requirement.txt”安装日志和ModelArts平台日志都包含在普通日志文件“modelarts-job-[job id]-[task id].log”中。

ModelArts平台日志可以通过关键字在训练的普通日志文件“modelarts-job-[job id]-[task id].log”中筛查，筛查关键字有：“[ModelArts Service Log]”或“Platform=ModelArts-Service”。

- 类型一：[ModelArts Service Log] xxx
  ```
  [ModelArts Service Log][init] download code_url: s3://dgg-test-user/snt9-test-cases/mindspore/lenet/
  ```
- 类型二：time=“xxx” level=“xxx” msg=“xxx” file=“xxx” Command=xxx Component=xxx Platform=xxx
  ```
  time="2021-07-26T19:24:11+08:00" level=info msg="start the periodic upload task, upload period = 5 seconds " file="upload.go:46" Command=obs/upload Component=ma-training-toolkit Platform=ModelArts-Service
  ```

Ascend 场景日志说明

使用Ascend资源运行训练作业时，会产生Ascend相关日志。Ascend训练场景下会生成device日志、plog日志、proc log单卡训练日志、MindSpore日志、普通日志。

其中，Ascend训练场景下的普通日志包括训练进程日志、pip-requirement.txt安装日志、ModelArts平台日志、ma-pre-start日志和davincirun日志。

Ascend日志结构举例说明如下：

```
obs://dgg-test-user/snt9-test-cases/log-out/                    # 作业日志路径
├──modelarts-job-9ccf15f2-6610-42f9-ab99-059ba049a41e
   ├── ascend
```



```
    ├── process_log
        ├── rank_0
    ├── plog                                # plog日志
                ...
    ├── device-0                            # device日志
                ...
  ├── mindspore                                          # MindSpore日志
├──modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-worker-0.log              # 普通日志
├──modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-proc-rank-0-device-0.txt    # proc log单卡训练日志
```

表 8-32 Ascend 场景下日志说明

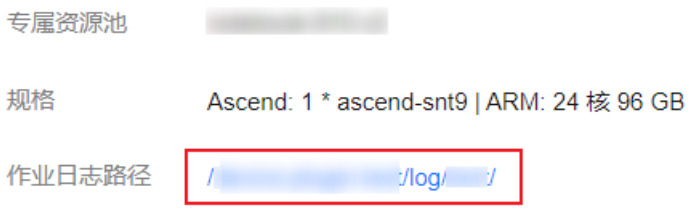
日志类型	日志说明	日志文件名
device日志	HOST侧用户进程，在DEVICE侧产生的AICPU、HCCP的日志，回传到HOST侧（训练容器）。 如果出现如下情况，则device日志会获取不到。 ● 节点异常重启 ● 被主动停止的节点 在训练进程结束后，该日志会生成到训练容器中。其中，使用MindSpore预置框架训练的device日志会自动上传到OBS，使用其他预置框架和自定义镜像训练的device日志如果需要自动上传到OBS，则需要在代码中配置ASCEND_PROCESS_LOG_PATH，具体请参考如下示例。 `# set npu plog env` `ma_vj_name=`echo ${MA_VJ_NAME} \| sed 's:ma-job:modelarts-job:g'`` `task_name="worker-${VC_TASK_INDEX}"` `task_plog_path=${MA_LOG_DIR}/${ma_vj_name}/${task_name}` `mkdir -p ${task_plog_path}` `export ASCEND_PROCESS_LOG_PATH=${task_plog_path}`	“~/ascend/log/device-{device-id}/device-{pid}_{timestamp}.log” 其中，pid是HOST侧用户进程号。 样例： device-166_20220718191853764.log

日志类型	日志说明	日志文件名
plog日志	HOST侧用户进程，在HOST侧产生的日志（例如:ACL / GE）。 plog日志会生成到训练容器中。其中，使用MindSpore预置框架训练的plog日志会自动上传到OBS，使用自定义镜像训练的plog日志如果需要自动上传到OBS，则需要在代码中配置ASCEND_PROCESS_LOG_PATH，具体请参考如下示例。 `# set npu plog env` `ma_vj_name=`echo ${MA_VJ_NAME} \| sed 's:ma-job:modelarts-job:g'`` `task_name="worker-${VC_TASK_INDEX}"` `task_plog_path=${MA_LOG_DIR}/${ma_vj_name}/${task_name}` `mkdir -p ${task_plog_path}` `export ASCEND_PROCESS_LOG_PATH=${task_plog_path}`	“~/ascend/log/plog/plog-{pid}_{timestamp}.log” 其中，pid是HOST侧用户进程号。 样例： plog-166_20220718191843620.log
proc log	proc log是单卡训练日志重定向文件，方便用户快速定位对应计算节点的日志。使用自定义镜像训练的作业不涉及proc log；使用预置框架训练的proc log日志会生成到训练容器中，且自动保存到OBS。	“[modelarts-job-uuid]-proc-rank-[rank id]-device-[device logic id].txt” ● device id为本次训练作业的NPU卡编号，取值单卡为0，8卡为0~7。 例如：Ascend规格为 8*Snt9时，device id取值为0~7；Ascend规格为 1*Snt9时，device id取值为0。 ● rank id为本次训练作业的全局NPU卡编号，取值为0~实例数*卡数-1，单个实例下，rank id与device id取值相同。 样例： modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-proc-rank-0-device-0.txt

日志类型	日志说明	日志文件名
MindSpore日志	使用MindSpore+Ascend训练时会产生单独的MindSpore日志。 MindSpore日志会生成到训练容器中。其中，使用MindSpore预置框架训练的plog日志会自动上传到OBS，使用自定义镜像训练的plog日志如果需要自动上传到OBS，则需要在代码中配置ASCEND_PROCESS_LOG_PATH，具体请参考如下示例。 `# set npu plog env` `ma_vj_name=`echo ${MA_VJ_NAME} \| sed 's:ma-job:modelarts-job:g'`` `task_name="worker-${VC_TASK_INDEX}"` `task_plog_path=${MA_LOG_DIR}/${ma_vj_name}/${task_name}` `mkdir -p ${task_plog_path}` `export ASCEND_PROCESS_LOG_PATH=${task_plog_path}`	MindSpore的日志介绍请参见**MindSpore官网**。
普通训练日志	普通训练日志会生成到训练容器的“/home/ma-user/modelarts/log”目录中，且自动上传到OBS。普通训练日志的类型如下所示。 • ma-pre-start日志（Ascend场景特有）：如果用户有定义ma-pre-start脚本，会产生该脚本执行日志。 • davincirun日志（Ascend场景特有）：Ascend训练进程通过davincirun.py文件启动，该启动文件产生的日志。 • 训练进程日志：用户训练代码的标准输出。 • pip-requirement.txt安装日志：如果用户有定义pip-requirement.txt文件，会产生pip包安装日志。 • ModelArts平台日志：ModelArts平台产生的系统日志，主要用于运维人员定位平台问题。	合并输出在日志文件modelarts-job-[job id]-[task id].log中。 task id表示实例ID，单节点时取值为worker-0，多节点时取值为worker-0、worker-1、...worker-{n-1}，n为实例数。 样例： modelarts-job-95f661bd-1527-41b8-971c-eca55e513254-worker-0.log

Ascend训练场景下，当训练进程退出后，ModelArts会上传训练容器中的日志文件至“作业日志路径” 参数设置的OBS目录中。在作业详情页可以获取“作业日志路径”，单击OBS地址可以直接跳转到OBS控制台查看日志。

图 8-36 日志存放路径



您可以通过ma-pre-start脚本修改默认环境变量配置。

```
ASCEND_GLOBAL_LOG_LEVEL=3     # 设置日志级别  debug级别为0;info级别为1;warning级别为2;error级别为3。
ASCEND_SLOG_PRINT_TO_STDOUT=1  # 设置plog日志是否在屏幕上显示，1表示默认设置在屏幕上显示日志。
ASCEND_GLOBAL_EVENT_ENABLE=1   # 设置事件级别  不开启Event日志级别为0；开启Event日志级别为1。
```

ma-pre-start脚本在与训练启动文件同级的目录下放置，命名为ma-pre-start.sh or ma-pre-start.py脚本。

在训练启动文件被执行前，系统会在 /home/work/user-job-dir/ 目录下执行上述ma-pre-start脚本，使用该机制可以更新容器镜像内安装的Ascend RUN包，或者设置一些训练运行时额外需要的全局环境变量。

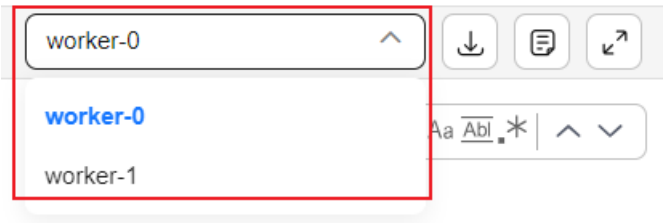
如何查看训练作业日志

在训练作业详情页，训练日志窗口提供日志预览、日志下载、日志中搜索关键字、系统日志过滤能力。

- 预览

  系统日志窗口提供训练日志预览功能，如果训练作业有多个节点，则支持查看不同计算节点的日志，通过右侧下拉框可以选择目标节点预览。

  图 8-37 查看不同计算节日志

  当日志文件过大时，系统日志窗口仅加载最新的部分日志，并在日志窗口上方提供全量日志访问链接。打开该链接可在新页面查看全部日志。

  图 8-38 查看全量日志

说明

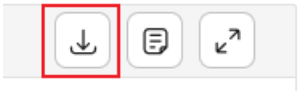
- 如果全部日志超过500M，可能会引起浏览页面卡顿，建议您直接下载日志查看。
- 预览链接在生成后的一小时内，支持任何人打开并查看。您可以分享链接至他人。
- **请注意日志中不能包含隐私内容，否则会造成信息泄露。**

- 下载

  训练日志仅保留30天，超过30天会被清理。如果用户需要永久保存日志，请单击系统日志窗口右上角下载按钮下载日志至本地保存，支持批量下载多节点日志。用户也可以在创建训练作业时打开永久保存日志按钮，保存训练日志至指定OBS路径。

  针对使用Ascend规格创建的训练作业，部分系统日志暂不支持直接在训练日志窗口下载，请在创建训练作业时指定OBS路径用于保存训练日志。

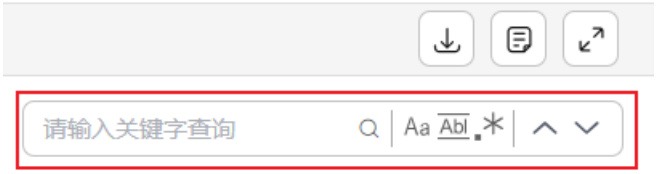
  图 8-39 下载日志

- 搜索关键字

  用户可以在系统日志右上角的搜索框搜索关键字，如图8-40所示。

  图 8-40 搜索关键字

  系统支持高亮关键字并实现搜索结果间的跳转。搜索功能仅支持搜索当前页面加载的日志，如果日志加载不全（请关注页面提示）则需要下载或者通过打开全量日志访问链接进行搜索。全量日志访问链接打开的新页面可以通过Ctrl+F进行搜索。

- 系统日志过滤

  图 8-41 系统日志复选框

  如果勾选了系统日志复选框，则日志中呈现系统日志和用户日志。如果去勾选，则只显示用户日志。

8.11.6 修改训练作业优先级

使用专属资源池训练作业时，支持在创建训练作业时设置任务优先级，也支持作业在长时间处于“等待中”的状态时调整优先级。如通过调整作业优先级可以减少作业的排队时长。

什么是训练作业优先级

在用户运行训练作业过程中，需要对训练作业做优先级划分。比如有一些任务是低优先级，可能是跑一些测试、也可能是跑一些简单的不重要的实验。在这类场景下，当有高优先级任务的时候，需要能比低优先级任务更快进入排队队列。

在资源使用高峰期，用户可以通过提供或降低训练作业的优先级，来动态调节作业的执行顺序，保障关键业务的及时运行。

约束限制

- 仅使用新版专属资源池训练时才支持设置训练作业优先级。公共资源池和旧版专属资源池均不支持设置训练作业优先级。
- 作业优先级取值为1~3，默认优先级为1，最高优先级为3。默认用户权限可选择优先级1和2，配置了“设置作业为高优先级权限”的用户可选择优先级1~3。

如何设置训练作业优先级

在创建训练作业页面可以设置训练的“作业优先级”。取值为1~3，默认优先级为1，最高优先级为3。

如何修改训练作业优先级

在训练作业列表页面，选择“状态”为“等待中”的训练作业，单击“作业优先级”列的，在弹窗中修改优先级后单击“确定”。

图 8-42 修改作业优先级

给子账号配置"设置作业为高优先级"权限

默认用户权限可选择优先级1和2，配置了"设置作业为高优先级"权限的用户可选择优先级1~3。

1. 使用主用户账号登录华为云的管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”按如下要求设置完成后单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：允许用户设置训练作业最高优先级。
   - “策略配置方式”：选择可视化视图。

- “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词“modelarts:trainJob:setHighPriority”并选中，所有资源选择默认值。

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限通过Cloud Shell登录运行中的训练作业容器。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

8.11.7 使用 Cloud Shell 调试生产训练作业

ModelArts Standard提供了Cloud Shell，可以登录运行中的容器，用于调试生产环境的训练作业。

约束限制

仅专属资源池支持使用Cloud Shell登录训练容器，且训练作业必须处于“运行中”状态。

前提条件：给子账号配置允许使用 Cloud Shell 的权限

1. 使用主用户账号登录华为云的管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”按如下要求设置完成后单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：允许通过Cloud Shell访问运行中的训练作业。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:trainJob:exec并选中，所有资源选择默认值。
3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限通过Cloud Shell登录运行中的训练作业容器。

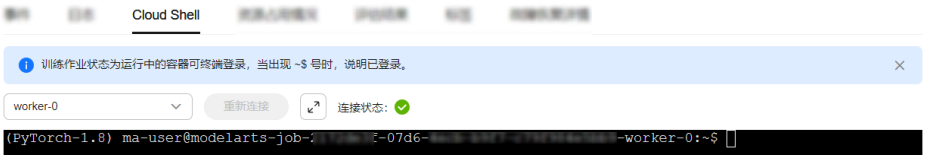
   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

使用 Cloud Shell

1. 参考**前提条件：给子账号配置允许使用Cloud Shell的权限**，完成配置。
2. 在ModelArts管理控制台的左侧导航栏中选择“模型训练 > 训练作业”。
3. 在训练作业列表中，单击作业名称进入训练作业详情页面。
4. 在训练作业详情页面，单击“Cloud Shell”页签，登录训练容器。

   连接成功后，Cloud Shell界面提示如下。

图 8-43 Cloud Shell 界面



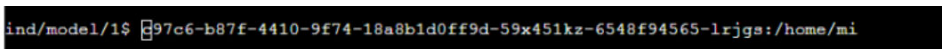
当作业处于非运行状态或权限不足时会导致无法使用Cloud Shell，请根据提示定位原因即可。

图 8-44 报错提示

说明

部分用户登录Cloud Shell界面时，可能会出现路径显示异常情况，此时在Cloud Shell中单击回车键即可恢复正常。

图 8-45 路径异常



如何使训练作业保持运行中状态

由于需要训练作业处于“运行中”状态才能登录Cloud Shell，因此本文介绍如何使训练作业保持运行中状态，方便您快速通过Cloud Shell登录运行中的训练容器。

通过Sleep命令使训练作业保持运行

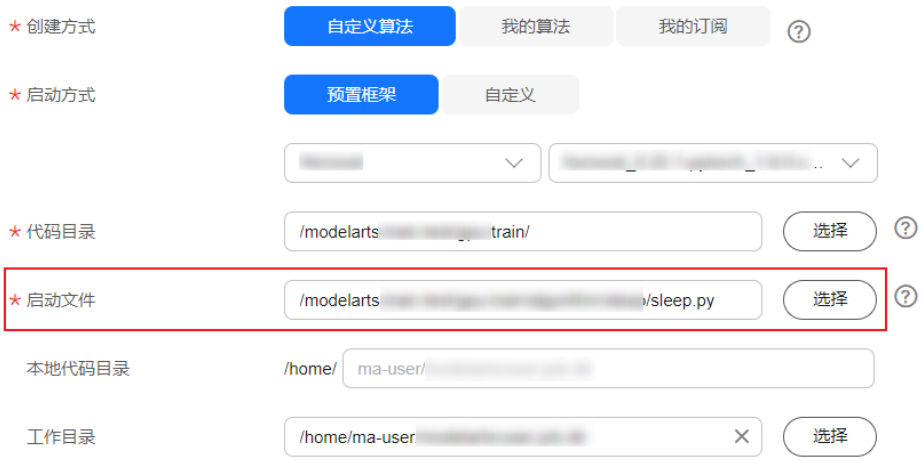
- 如果训练作业使用的是预置框架：

  在创建训练作业时，“创建方式”选择“自定义算法”，“启动方式”选择“预置框架”，代码目录中新增sleep.py并将此脚本作为“启动文件”。这样启动的作业将会持续运行60分钟。您可通过Cloud Shell进入容器进行调试。

  sleep.py示例：

  ```
  import os
  os.system('sleep 60m')
  ```

图 8-46 预置框架启动方式



- 如果训练作业使用的是自定义镜像

  在创建训练作业时，“创建方式”选择“自定义算法”，“启动方式”选择“自定义”，“启动命令”输入“sleep 60m”。这样启动的作业将会持续运行60分钟。您可通过Cloud Shell进入容器进行调试。

图 8-47 自定义启动方式

出错的任务如何卡在运行中状态

创建训练作业时，启动命令末尾新增"|| sleep 5h"，并启动训练作业，例如下方的cmd为您的启动命令：

```
cmd || sleep 5h
```

如果训练失败，则会执行sleep命令，此时可通过Cloud Shell登录容器镜像中调试。

说明

在Cloud Shell中调试多节点训练作业时，需要在Cloud Shell中切换work0、work1来实现对不同节点下发启动命令，否则任务会处于等待其他节点的状态。

如何防止 Cloud Shell 的 Session 断开

如果需要长时间运行某一个任务，为避免在期间连接断开导致任务失败，可通过使用screen命令使得任务在远程终端窗口运行。

1. 如果镜像中未安装screen，则执行“apt-get install screen”安装。
2. 创建screen终端。
```
# 使用 -S 创建一个叫name的screen终端
screen -S name
```
3. 显示已创建的screen终端。
```
screen -ls
There are screens on:
2433.pts-3.linux    (2013年10月20日 16时48分59秒)    (Detached)
2428.pts-3.linux    (2013年10月20日 16时48分05秒)    (Detached)
2284.pts-3.linux    (2013年10月20日 16时14分55秒)    (Detached)
2276.pts-3.linux    (2013年10月20日 16时13分18秒)    (Detached)
4 Sockets in /var/run/screen/S-root.
```
4. 连接“screen_id”为“2276”的screen终端。
```
screen -r 2276
```
5. 按下“Ctrl”+“a”+“d”键离开screen终端。离开后，screen会话仍将是活跃的，之后可以随时重新连接。

更多Screen使用说明可参考**Screen User's Manual**。

通过 py-spy 工具分析卡死进程的调用栈并结合代码分析定位卡死问题

本文指导用户通过py-spy工具分析卡死进程的调用栈并结合代码分析定位卡死问题。

步骤1 在ModelArts Standard控制台，选择“模型训练>训练作业”。

步骤2 在训练作业详情页面，选择Cloud Shell页签，登录训练容器（训练作业需处于运行中）。

步骤3 安装py-spy工具。
```
# 通过utils.sh脚本自动配置python环境
source /home/ma-user/modelarts/run/utils.sh

# 安装py-spy
pip install py-spy

# 如果超时提示connection broken by 'ProxyError('Cannot connect to proxy.')则表示用户设置了proxy，需要先关掉
export no_proxy=$no_proxy,repo.myhuaweicloud.com（此处需要替换成对应局点的pip源地址）
pip install py-spy
```

步骤4 查看堆栈。py-spy工具的具体使用方法可参考**py-spy官方文档**。
```
# 找到训练进程的PID
ps -ef

# 查看进程12345的进程堆栈
# 如果是8卡的训练作业，一般用此命令依次去查看主进程起的对应的8个进程的堆栈情况
py-spy dump --pid 12345
```

----结束

8.11.8 重建、停止或删除训练作业

另存为算法

当您需要修改训练作业的算法时，可以在训练作业详情页面右上角，单击“另存为算法”。

在“创建算法”页面中，会自动填充上一次训练作业的算法参数配置，您可以根据业务需求在原来算法配置基础上进行修改。

说明

订阅算法不支持另存为算法。

重建训练作业

当对创建的训练作业不满意时，您可以单击操作列的重建，重新创建训练作业。在重创训练作业页面，会自动填入上一次训练作业设置的参数，您仅需在原来的基础上进行修改即可重新创建训练作业。

停止训练作业

在训练作业列表中，针对“创建中”、“等待中”、“运行中”的训练作业，您可以单击“操作”列的“终止”，停止正在运行中的训练作业。

训练作业停止后，ModelArts将停止计费。

运行结束的训练作业，如“已完成”、“运行失败”、“已终止”、“异常”的作业，不涉及“终止”操作。

删除训练作业

如果不再需要使用此训练作业，建议清除相关资源，避免产生不必要的费用。

说明

请注意，删除训练作业后无法恢复，请谨慎操作。

- 在“训练作业”页面，删除运行结束的训练作业。您可以单击“操作”列的“删除”，在弹出的提示框中单击“确认”，删除对应的训练作业。
- 进入OBS，删除本训练作业使用的OBS桶及文件。

查找训练作业

当用户使用IAM账号登录时，训练作业列表会显示IAM账号下所有训练作业。ModelArts提供查找训练作业功能帮助用户快速查找训练作业。

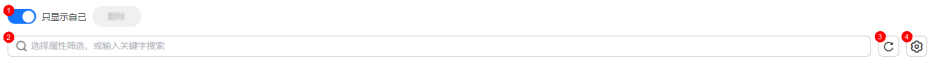
操作一：单击“只显示自己”按钮，训练作业列表仅显示当前子账号下创建的训练作业。

操作二：按照名称、ID、作业类型、状态、创建时间、算法、资源池等条件筛选的高级搜索。

操作三：单击作业列表右上角“刷新”图标，刷新作业列表。

操作四：自定义列功能设置。

图 8-48 查找训练作业



8.11.9 管理训练容器环境变量

什么是环境变量

本章节展示了训练容器环境中预置的环境变量，方便用户查看，主要包括以下类型。

- 路径相关环境变量
- 分布式训练作业环境变量
- NCCL（Nvidia Collective multi-GPU Communication Library）环境变量
- OBS环境变量
- PIP源环境变量
- API网关地址环境变量
- 作业元信息环境变量

约束限制

为了避免新设置的环境变量与系统环境变量冲突，而引起作业运行异常或失败，请在定义自定义环境变量时，不要使用“MA_”开头的名称。

如何修改环境变量

用户可以在创建训练作业页面增加新的环境变量，也可以设置新的取值覆盖当前训练容器中预置的环境变量值。

说明

为保证数据安全，请勿输入敏感信息，例如明文密码。

训练容器中预置的环境变量

训练容器中预置的环境变量如下面表格所示，包括表8-33、表8-34、表8-35、表8-36、表8-37、表8-38、表8-39。

此处的环境变量取值仅为示例，涉及不同规格、引擎、Region可能取值不一样，此处仅供参考。

表 8-33 路径相关环境变量

变量名	说明	示例
PATH	可执行文件路径，已包含常用的可执行文件路径。	“PATH=/usr/local/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin”

变量名	说明	示例
LD_LIBRARY_PATH	动态链接库路径，已包含常用的动态库路径。	“LD_LIBRARY_PATH=/usr/local/seccomponent/lib:/usr/local/cuda/lib64:/usr/local/cuda/compat:/root/miniconda3/lib:/usr/local/lib:/usr/local/nvidia/lib64”
LIBRARY_PATH	静态库路径，已包含常用的静态库路径。	“LIBRARY_PATH=/usr/local/cuda/lib64/stubs”
MA_HOME	训练作业的主目录。	“MA_HOME=/home/ma-user”
MA_JOB_DIR	训练算法文件夹所在的父目录。	“MA_JOB_DIR=/home/ma-user/modelarts/user-job-dir”
MA_MOUNT_PATH	ModelArts挂载至训练容器内的路径，用于临时存放训练算法、算法输入、算法输出、日志等文件。	“MA_MOUNT_PATH=/home/ma-user/modelarts”
MA_LOG_DIR	训练日志目录。	“MA_LOG_DIR=/home/ma-user/modelarts/log”
MA_SCRIPT_INTERPRETER	训练脚本解释器。	“MA_SCRIPT_INTERPRETER=”
WORKSPACE	训练算法目录。	“WORKSPACE=/home/ma-user/modelarts/user-job-dir/code”

表 8-34 分布式训练作业环境变量

变量名	说明	示例
MA_CURRENT_IP	作业容器IP。	“MA_CURRENT_IP=192.168.23.38”
MA_NUM_GPUS	作业容器的加速卡数量。	“MA_NUM_GPUS=8”
MA_TASK_NAME	作业容器的角色名，例如： • MindSpore、PyTorch为worker • 强化学习引擎为learner，worker • TensorFlow为ps，worker	“MA_TASK_NAME=worker”
MA_NUM_HOSTS	实例数。系统自动从资源参数的“实例数”中读取。	“MA_NUM_HOSTS=4”

变量名	说明	示例
VC_TASK_INDEX	当前容器索引，容器从0开始编号。单机训练的时候，该字段无意义。在多机作业中，用户可以根据这个值来确定当前容器运行的算法逻辑。	“VC_TASK_INDEX=0”
VC_WORKER_NUM	训练作业使用的实例数量。	“VC_WORKER_NUM=4”
VC_WORKER_HOSTS	多节点训练时，每个节点的域名地址，按顺序以英文逗号分隔，可以通过域名解析获取IP地址。	“VC_WORKER_HOSTS=modelarts-job-a0978141-1712-4f9b-8a83-00000000000000-worker-0.modelarts-job-a0978141-1712-4f9b-8a83-00000000000000,modelarts-job-a0978141-1712-4f9b-8a83-00000000000000-worker-1.ob-a0978141-1712-4f9b-8a83-00000000000000,modelarts-job-a0978141-1712-4f9b-8a83-00000000000000-worker-2.modelarts-job-a0978141-1712-4f9b-8a83-00000000000000,ob-a0978141-1712-4f9b-8a83-00000000000000-worker-3.modelarts-job-a0978141-1712-4f9b-8a83-00000000000000”
${MA_VJ_NAME}-${MA_TASK_NAME}-N.${MA_VJ_NAME}	表示不同节点的通信域名，例如0号节点的通信域名为 “${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}”。 N表示实例数。	例如，实例数为4时，此环境变量分别为 “${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}”、 “${MA_VJ_NAME}-${MA_TASK_NAME}-1.${MA_VJ_NAME}”、 “${MA_VJ_NAME}-${MA_TASK_NAME}-2.${MA_VJ_NAME}”、 “${MA_VJ_NAME}-${MA_TASK_NAME}-3.${MA_VJ_NAME}”。

表 8-35 NCCL 环境变量

变量名	说明	示例
NCCL_VERSION	NCCL版本。	“NCCL_VERSION=2.7.8”
NCCL_DEBUG	NCCL日志等级。	“NCCL_DEBUG=INFO”
NCCL_IB_HCA	指定NCCL使用的IB网卡。	“NCCL_IB_HCA=^mlx5_bond_0”
NCCL_SOCKET_IFNAME	指定NCCL使用的SOCKET网卡。	“NCCL_SOCKET_IFNAME=bond0,eth0”

表 8-36 OBS 环境变量

变量名	说明	示例
S3_ENDPOINT	OBS地址。	“-”
S3_VERIFY_SSL	访问OBS是否使用SSL。	“S3_VERIFY_SSL=0”
S3_USE_HTTPS	访问OBS是否使用HTTPS。	“S3_USE_HTTPS=1”

表 8-37 PIP 源和 API 网关地址环境变量

变量名	说明	示例
MA_PIP_HOST	PIP源域名。	“MA_PIP_HOST=repo.myhuaweicloud.com”
MA_PIP_URL	PIP源地址。	“MA_PIP_URL=http://repo.myhuaweicloud.com/repository/pypi/simple/”
MA_APIGW_ENDPOINT	ModelArts API网关地址。	“MA_APIGW_ENDPOINT=https://modelarts.region.cn-east-3.myhuaweicloud.com”

表 8-38 作业元信息环境变量

变量名	说明	示例
MA_CURRENT_INSTANCE_NAME	多节点训练时，当前节点的名称。	“MA_CURRENT_INSTANCE_NAME=modelarts-job-a0978141-1712-4f9b-8a83-000000000000-worker-1”

表 8-39 预检相关环境变量

变量名	说明	示例
MA_SKIP_IMAGE_DETECT	ModelArts预检是否开启。默认为1，1表示开启预检，0表示关闭预检。 推荐开启预检，预检可提前发现节点故障、驱动故障。	“1”

表 8-40 卡死检测相关环境变量

变量名	说明	示例
MA_HANG_DETECT_TIME	卡死检测时间。在这段时间内IO无变化则判定为任务卡死。 取值范围：10~720 单位：分钟 默认值：30	“30”

如何查看训练环境变量

在创建训练作业时，“启动命令”输入为“env”，其他参数保持不变。

当训练作业执行完成后，在训练作业详情页面中查看“日志”。日志中即为所有的环境变量信息。

图 8-49 查看日志

```
NV_LIBCUBLAS_DEV_VERSION=11.3.1.68-1
NV_CUDA_COMPAT_PACKAGE=cuda-compat-11-2
NV_CUDNN_PACKAGE_DEV=libcudnn8-dev=8.1.1.33-1+cuda11.2
LD_LIBRARY_PATH=/usr/local/nvidia/lib:/usr/local/nvidia/lib64
NV_LIBNCCL_DEV_PACKAGE=libnccl-dev=2.8.4-1+cuda11.2
MA_ENGINE_VERSION=
MA_NUM_HOSTS=1
VC_WORKER_HOSTS=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0.modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
VK_TASK_INDEX=0
_=/usr/bin/env
MA_SCRIPT_INTERPRETER=
NV_LIBNPP_DEV_PACKAGE=libnpp-dev-11-2=11.2.1.68-1
MA_MAX_BACKOFF=0
HOSTNAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0
MA_IAM_USER_ID=79098163dd814fd986eca7ef0325d086
MA_CURRENT_IP=10.0.0.62
NV_LIBNPP_VERSION=11.2.1.68-1
NV_NVPROF_DEV_PACKAGE=cuda-nvprof-11-2=11.2.67-1
MA_MOUNT_PATH=/home/ma-user/modelarts
NVIDIA_VISIBLE_DEVICES=all
MA_ENGINE_TYPE=
NV_NVPROF_VERSION=11.2.67-1
NV_LIBCUSPARSE_VERSION=11.3.1.68-1
MODELARTS_SCC_SERVICE_PORT=60687
MA_HOME=/home/ma-user
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.247.0.1
NV_LIBCUBLAS_DEV_PACKAGE=libcublas-dev-11-2=11.3.1.68-1
MA_VJ_NAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
MA_PIP_URL=http://repo.myhuaweicloud.com/repository/pypi/simple/
NCCL_VERSION=2.8.4-1
KUBERNETES_PORT=tcp://10.247.0.1:443
PWD=/
NVARCH=x86_64
HOME=/home/ma-user
```

8.11.10 查看训练作业标签

通过给训练作业添加标签，可以标识云资源，便于快速搜索训练作业。

1. 在ModelArts管理控制台的左侧导航栏中选择“模型训练 > 训练作业”。
2. 在训练作业列表中，单击作业名称进入训练作业详情页面。
3. 在训练作业详情页面，单击“标签”页签查看标签信息。
   支持添加、修改、删除标签。标签详细用法请参见**使用TMS标签实现资源分组管理**。

图 8-50 查看训练标签

说明

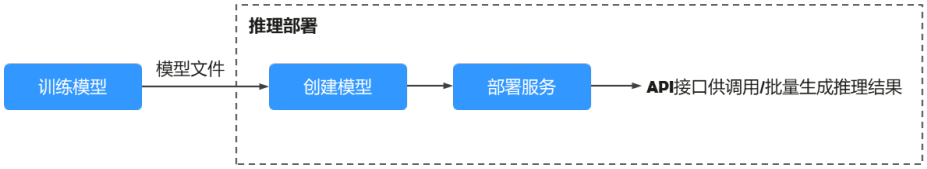
最多支持添加20个标签。

9 使用 ModelArts Standard 部署模型并推理预测

9.1 推理部署使用场景

AI模型开发完成后，在ModelArts服务中可以将AI模型创建为模型，将模型快速部署为推理服务，您可以通过调用API的方式把AI推理能力集成到自己的IT平台，或者批量生成推理结果。

图 9-1 推理简介



1. 训练模型：可以在ModelArts服务中进行，也可以在您的本地开发环境进行，本地开发的模型需要上传到华为云OBS服务。
2. 创建模型：把模型文件和推理文件导入到ModelArts的模型仓库中，进行版本化管理，并构建为可运行的模型。
3. 部署服务：模型构建完成后，根据您的业务场景，选择将模型部署成对应的服务类型。

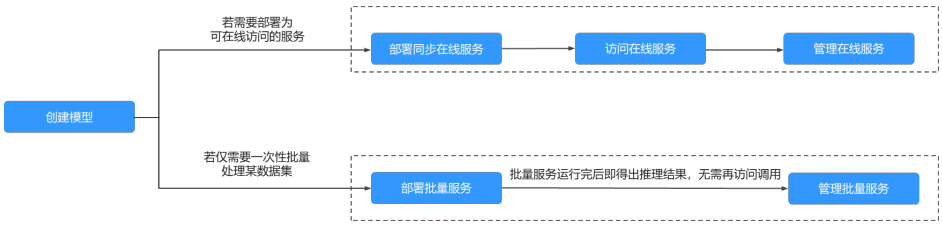
- **将模型部署为实时推理作业**

  将模型部署为一个Web Service，并且提供在线的测试UI与监控功能，部署成功的在线服务，将为用户提供一个可调用的API。

- **将模型部署为批量推理服务**

  批量服务可对批量数据进行推理，完成数据处理后自动停止。

图 9-2 不同类型的推理作业使用场景



9.2 创建模型

9.2.1 创建模型不同方式的场景介绍

AI开发和调优往往需要大量的迭代和调试，数据集、训练代码或参数的变化都可能会影响模型的质量，如不能统一管理开发流程元数据，可能会出现无法重现最优模型的现象。

ModelArts的模型可导入所有训练生成的元模型、上传至对象存储服务（OBS）中的元模型和容器镜像中的元模型，可对所有迭代和调试的模型进行统一管理。

约束与限制

- 自动学习项目中，在完成模型部署后，其生成的模型也将自动上传至模型列表中。但是自动学习生成的模型无法下载，只能用于部署上线。
- 创建模型、管理模型版本等功能目前是免费开放给所有用户，使用此功能不会产生费用。

创建模型的几种场景

- **从训练作业中导入模型文件创建模型**：在ModelArts中创建训练作业，并完成模型训练，在得到满意的模型后，可以将训练后得到的模型创建为模型，用于部署服务。
- **从OBS中导入模型文件创建模型**：如果您使用常用框架在本地完成模型开发和训练，可以将本地的模型按照模型包规范上传至OBS桶中，从OBS将模型导入至ModelArts中，创建为模型，直接用于部署服务。
- **从容器镜像中导入模型文件创建模型**：针对ModelArts目前不支持的AI引擎，可以通过自定义镜像的方式将编写的模型镜像导入ModelArts，创建为模型，用于部署服务。

推理支持的 AI 引擎

在ModelArts创建模型时，如果使用预置镜像“从OBS中选择”导入模型，则支持如下常用引擎及版本的模型包。

说明

- 统一镜像Runtime的命名规范：<AI引擎名字及版本> - <硬件及版本：cpu或cuda或cann> - <python版本> - <操作系统版本> - <CPU架构>
- 当前支持自定义模型启动命令，预置AI引擎都有默认的启动命令，如非必要无需改动

表 9-1 支持的常用引擎及其 Runtime 以及默认启动命令

模型使用的引擎类型	支持的运行环境（Runtime）	注意事项
TensorFlow	python3.6 python2.7（待下线） tf1.13-python3.6-gpu tf1.13-python3.6-cpu tf1.13-python3.7-cpu tf1.13-python3.7-gpu tf2.1-python3.7（待下线） tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64（推荐）	• python2.7、python3.6的运行环境搭载的TensorFlow版本为1.8.0。 • python3.6、python2.7、tf2.1-python3.7，表示该模型可同时在CPU或GPU运行。其他Runtime的值，如果后缀带cpu或gpu，表示该模型仅支持在CPU或GPU中运行。 • 默认使用的Runtime为python2.7。 • 默认启动命令：sh /home/mind/run.sh
Spark_MLlib	python2.7（待下线） python3.6（待下线）	• python2.7以及python3.6的运行环境搭载的Spark_MLlib版本为2.3.2。 • 默认使用的Runtime为python2.7。 • python2.7、python3.6只能用于运行适用于CPU的模型。 • 默认启动命令：bash /home/work/predict/bin/run.sh
Scikit_Learn	python2.7（待下线） python3.6（待下线）	• python2.7以及python3.6的运行环境搭载的Scikit_Learn版本为0.18.1。 • 默认使用的Runtime为python2.7。 • python2.7、python3.6只能用于运行适用于CPU的模型。 • 默认启动命令：bash /home/work/predict/bin/run.sh
XGBoost	python2.7（待下线） python3.6（待下线）	• python2.7以及python3.6的运行环境搭载的XGBoost版本为0.80。 • 默认使用的Runtime为python2.7。 • python2.7、python3.6只能用于运行适用于CPU的模型。 • 默认启动命令：bash /home/work/predict/bin/run.sh

模型使用的引擎类型	支持的运行环境（Runtime）	注意事项
PyTorch	python2.7（待下线） python3.6 python3.7 pytorch1.4-python3.7 pytorch1.5-python3.7（待下线） pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64（推荐）	• python2.7、python3.6、python3.7的运行环境搭载的PyTorch版本为1.0。 • python2.7、python3.6、python3.7、pytorch1.4-python3.7、pytorch1.5-python3.7，表示该模型可同时在CPU或GPU运行。 • 默认使用的Runtime为python2.7。 • 默认启动命令：sh /home/mind/run.sh
MindSpore	aarch64（推荐）	aarch64只能用于运行在Snt3芯片上。 • 默认启动命令：sh /home/mind/run.sh

9.2.2 从训练作业中导入模型文件创建模型

在ModelArts中创建训练作业，并完成模型训练，在得到满意的模型后，可以将训练后得到的模型导入至模型管理，方便统一管理，同时支持将模型快速部署上线为服务。

约束与限制

- 针对使用订阅算法的训练作业，无需推理代码和配置文件，其生成的模型可直接导入ModelArts。
- 使用容器化部署，导入的元模型有大小限制，详情请参见**导入模型对于镜像大小限制**。

前提条件

- 请确保训练作业已运行成功，且模型已存储至训练输出的OBS目录下（输入参数为train_url）。
- 针对使用常用框架或自定义镜像创建的训练作业，需根据**模型包结构介绍**，将推理代码和配置文件上传至模型的存储目录中。
- 确保您使用的OBS目录与ModelArts在同一区域。

创建模型操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型管理”，进入模型列表页面。
2. 单击左上角的“创建模型”，进入“创建模型”页面。
3. 在“创建模型”页面，填写相关参数。
   a. 填写模型基本信息，详细参数说明请参见**表9-2**。

表 9-2 模型基本信息参数说明

参数名称	说明
名称	模型名称。支持1～64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线。
版本	设置所创建模型的版本。第一次导入时，默认为0.0.1。 **说明** 模型创建完成后，可以通过**创建新版本**，导入不同的元模型进行调优。
描述	模型的简要描述。

b. 填写元模型来源及其相关参数。当“元模型来源”选择“从训练中选择”时，其相关的参数配置请参见**表9-3**。

图 9-3 从训练中选择元模型

表 9-3 元模型来源参数说明

参数	说明
“元模型来源”	选择“从训练中选择”。 • 在“选择训练作业”右侧下拉框中选择当前账号下已完成运行的训练作业。 • “动态加载”：用于实现快速部署和快速更新模型。如果勾选动态加载，则模型文件和运行时依赖仅在实际部署时拉取。当单个模型文件大小超过5GB时，必须配置“动态加载”。
“AI引擎”	元模型使用的推理引擎，选择训练作业后会自动匹配。
“运行时依赖”	罗列选中模型对环境的依赖。例如依赖“tensorflow”，安装方式为“pip”，其版本必须为1.8.0及以上版本。
“模型说明”	为了帮助其他模型开发者更好的理解及使用您的模型，建议您提供模型的说明文档。单击“添加模型说明”，设置“文档名称”及其“URL”。模型说明最多支持3条。

参数	说明
“部署类型”	选择此模型支持部署服务的类型，部署上线时只支持部署为此处选择的部署类型，例如此处只选择在线服务，那您导入后只能部署为在线服务。

c. 确认信息填写无误，单击“立即创建”，完成模型的创建。

在模型列表中，您可以查看刚创建的模型及其对应的版本。当模型状态变更为“正常”时，表示模型导入成功。在此页面，您还可以创建新版本、快速部署服务等操作。

后续操作

部署服务：在“模型列表”中，单击模型的操作列的“部署”，在对应版本所在行，单击“操作”列的部署按钮，可以将模型部署上线为创建模型时所选择的部署类型。

9.2.3 从 OBS 中导入模型文件创建模型

针对使用常用框架完成模型开发和训练的场景，可以将您的模型导入至ModelArts中，创建为模型，并进行统一管理。

约束与限制

- 针对创建模型的模型，需符合ModelArts的模型包规范，推理代码和配置文件也需遵循ModelArts的要求，详细说明请参见**模型包结构介绍**、**模型配置文件编写说明**、**模型推理代码编写说明**。
- 使用容器化部署，导入的元模型有大小限制，详情请参见**导入模型对于镜像大小限制**。

前提条件

- 已完成模型开发和训练，使用的AI引擎为ModelArts支持的类型和版本，详细请参见**推理支持的AI引擎**。
- 已完成训练的模型包，及其对应的推理代码和配置文件，且已上传至OBS目录中。
- 确保您使用的OBS与ModelArts在同一区域。

创建模型操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型管理”，进入模型列表页面。
2. 单击左上角的“创建模型”，进入“创建模型”页面。
3. 在“创建模型”页面，填写相关参数。
   a. 填写模型基本信息，详细参数说明请参见**表9-4**。

表 9-4 模型基本信息参数说明

参数名称	说明
名称	模型名称。支持1～64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线。
版本	设置所创建模型的版本。第一次导入时，默认为0.0.1。 **说明** 模型创建完成后，可以通过**创建新版本**，导入不同的元模型进行调优。
描述	模型的简要描述。

b. 填写元模型来源及其相关参数。当“元模型来源”选择“从对象存储服务（OBS）中选择”时，其相关的参数配置请参见**表9-5**。

针对从OBS导入的元模型，ModelArts要求根据**模型包规范**，编写推理代码和配置文件，并将推理代码和配置文件放置元模型存储的“model”文件夹下。如果您选择的目录下不符合模型包规范，将无法创建模型。

图 9-4 从 OBS 中选择元模型

表 9-5 元模型来源参数说明

参数	说明
“元模型来源”	选择“从对象存储服务（OBS）中选择”。
“选择元模型”	选择元模型存储的OBS路径。 OBS路径不能含有空格，否则创建模型会失败。
“AI引擎”	根据您选择的元模型存储路径，将自动关联出元模型使用的“AI引擎”。
“容器调用接口”	当“AI引擎”选择“Custom”时，才会显示该参数。 模型提供的推理接口所使用的协议和端口号，缺省值是HTTPS和8080，端口和协议需要根据模型实际定义的推理接口进行配置。

参数	说明
“健康检查”	用于指定模型的健康检查。使用Custom引擎时，会显示该参数。使用非Custom引擎时，选择了“AI引擎”和“运行环境”后，部分支持健康检查的引擎会显示该参数，请以实际界面显示为准。 当使用Custom引擎时，引擎包需要选择容器镜像，仅当容器镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致模型创建失败。 当前支持以下三种探针： • **启动探针：**用于检测应用实例是否已经启动。如果提供了启动探针(startup probe)，则禁用所有其他探针，直到它成功为止。如果启动探针失败，将会重启实例。如果没有提供启动探针，则默认状态为成功Success。 • **就绪探针：**用于检测应用实例是否已经准备好接收流量。如果就绪探针失败，即实例未准备好，会从服务负载均衡的池中剔除该实例，不会将流量路由到该实例，直到探测成功。 • **存活探针：**用于检测应用实例内应用程序的健康状态。如果存活探针失败，即应用程序不健康，将会自动重启实例。 3种探针的配置参数均为： • **检查方式：**仅支持“HTTP请求检查”。 • **健康检查URL**：健康检查的URL固定为“/health”。 • **健康检查周期（秒）**：填写1-2147483647之前的整数，单位为秒。 • **延迟时间（秒）：**实例启动后，延迟执行健康检查的时间。填写0-2147483647之间的整数，单位为秒，不能为空。 • **超时时间（秒）：** 每次检查的超时时间，填写0-2147483647之间的整数，单位为秒。 • **最大失败次数**：填写1-2147483647之间的整数。在服务启动阶段，当健康检查请求连续失败达到所填次数后，服务会进入异常状态；在服务运行阶段，当健康检查请求连续失败达到所填次数后，服务会进入告警状态。 **说明** 使用Custom引擎时需要符合自定义引擎规范，请参见**使用自定义引擎创建模型**。 当模型配置了健康检查，部署的服务在收到停止指令后，会延后3分钟才停止。
“动态加载”	用于实现快速部署和快速更新模型。如果勾选“动态加载”，则模型文件和运行时依赖仅在实际部署时拉取。单个模型文件大小超过5GB，需要配置“动态加载”。
“运行时依赖”	罗列选中模型对环境的依赖。例如依赖“tensorflow”，安装方式为“pip”，其版本必须为1.8.0及以上版本。

参数	说明
“模型说明”	为了帮助其他模型开发者更好的理解及使用您的模型，建议您提供模型的说明文档。单击“添加模型说明”，设置“文档名称”及其“URL”。模型说明支持增加3条。
“配置文件”	系统默认关联您存储在OBS中的配置文件。打开开关，您可以直接在当前界面查看或编辑模型配置文件。 **说明** 该功能即将下线，后续请根据“AI引擎”、“运行时依赖”和“apis定义”修改模型的配置信息。
“部署类型”	选择此模型支持部署服务的类型，部署上线时只支持部署为此处选择的部署类型，例如此处只选择在线服务，那您导入后只能部署为在线服务。
“启动命令”	选填参数，指定模型的启动命令，您可以自定义该命令。 如果使用预置的AI引擎，如果启动命令没有填写，会使用默认的启动命令，默认的启动命令见**表9-1**。如果填写了启动命令，新填写的启动命令覆盖默认启动命令。 **说明** 包含字符$，|，>，<，`，!，\n，\，?，-v，--volume，--mount，--tmpfs，--privileged，--cap-add的启动命令，在模型发布时将会置空。
“apis定义”	提供模型对外Restfull api数据定义，用于定义模型的输入、输出格式。apis定义填写规范请参见**模型配置文件编写说明**中的apis参数说明，示例代码请参见**apis参数代码示例**。

c. 确认信息填写无误，单击“立即创建”，完成模型创建。

在模型列表中，您可以查看刚创建的模型及其对应的版本。当模型状态变更为“正常”时，表示模型创建成功。在此页面，您还可以创建新版本、快速部署服务等操作。

后续操作

部署服务：在“模型列表”中，单击模型的操作列的“部署”，在对应版本所在行，单击“操作”列的部署按钮，可以将模型部署上线为创建模型时所选择的部署类型。

9.2.4 从容器镜像中导入模型文件创建模型

针对ModelArts目前不支持的AI引擎，您可以通过自定义镜像的方式将编写的模型导入ModelArts。

约束与限制

- 关于自定义镜像规范和说明，请参见**模型镜像规范**。
- 使用容器化部署，导入的元模型有大小限制，详情请参见**导入模型对于镜像大小限制**。

前提条件

确保您使用的OBS目录与ModelArts在同一区域。

创建模型操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型管理”，进入模型列表页面。
2. 单击左上角的“创建模型”，进入“创建模型”页面。
3. 在“创建应用”页面，填写相关参数。
   a. 填写模型基本信息，详细参数说明请参见**表9-6**。

表 9-6 模型基本信息参数说明

参数名称	说明
名称	模型名称。支持1～64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线。
版本	设置所创建模型的版本。第一次导入时，默认为0.0.1。 **说明** 模型创建完成后，可以通过**创建新版本**，导入不同的元模型进行调优。
描述	模型的简要描述。

   b. 填写元模型来源及其相关参数。当“元模型来源”选择“从容器镜像中选择”时，其相关的参数配置请参见**表9-7**。

图 9-5 从容器镜像中选择模型

表 9-7 元模型来源参数说明

参数	说明
“容器镜像所在的路径”	单击 从容器镜像中导入模型的镜像，其中，模型均为Image类型，且不再需要用配置文件中的“swr_location”来指定您的镜像位置。 制作自定义镜像的操作指导及规范要求，请参见**模型镜像规范**。 **说明** 您选择的模型镜像将共享给系统管理员，请确保具备共享该镜像的权限（不支持导入其他账户共享给您的镜像），部署上线时，ModelArts将使用该镜像部署成推理服务，请确保您的镜像能正常启动并提供推理接口。
“容器调用接口”	模型提供的推理接口所使用的协议和端口号，请根据模型实际定义的推理接口进行配置。
“镜像复制”	镜像复制开关，选择是否将容器镜像中的模型镜像复制到ModelArts中。 • 关闭时，表示不复制模型镜像，可极速创建模型，更改或删除SWR源目录中的镜像会影响服务部署。 • 开启时，表示复制模型镜像，无法极速创建模型，SWR源目录中的镜像更改或删除不影响服务部署。 **说明** 如果使用他人共享的镜像，需要开启镜像复制功能，否则会导致创建模型失败。

参数	说明
“健康检查”	用于指定模型的健康检查。仅当自定义镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致模型创建失败。当前支持以下三种探针： • **启动探针：**用于检测应用实例是否已经启动。如果提供了启动探针(startup probe)，则禁用所有其他探针，直到它成功为止。如果启动探针失败，将会重启实例。如果没有提供启动探针，则默认状态为成功Success。 • **就绪探针：**用于检测应用实例是否已经准备好接收流量。如果就绪探针失败，即实例未准备好，会从服务负载均衡的池中剔除该实例，不会将流量路由到该实例，直到探测成功。 • **存活探针：**用于检测应用实例内应用程序的健康状态。如果存活探针失败，即应用程序不健康，将会自动重启实例。 3种探针的配置参数均为： • **检查方式：**可以选择“HTTP请求检查”或者“执行命令检查”。 • **健康检查URL**：“检查方式”选择“HTTP请求检查”时显示，填写健康检查的URL，默认值为“/health”。 • **健康检查命令：**“检查方式”选择“执行命令检查”时显示，填写健康检查的命令。 • **健康检查周期（秒）**：填写1-2147483647之前的整数，单位为秒。 • **延迟时间（秒）：**实例启动后，延迟执行健康检查的时间。填写0-2147483647之间的整数，单位为秒，不能为空。 • **超时时间（秒）：** 每次检查的超时时间，填写0-2147483647之间的整数，单位为秒。 • **最大失败次数**：填写1-2147483647之间的整数。在服务启动阶段，当健康检查请求连续失败达到所填次数后，服务会进入异常状态；在服务运行阶段，当健康检查请求连续失败达到所填次数后，服务会进入告警状态。 **说明** 当模型配置了健康检查，部署的服务在收到停止指令后，会延后3分钟才停止。
“模型说明”	为了帮助其他模型开发者更好的理解及使用您的模型，建议您提供模型的说明文档。单击“添加模型说明”，设置“文档名称”及其“URL”。模型说明支持增加3条。
“部署类型”	选择此模型支持部署服务的类型，部署上线时只支持部署为此处选择的部署类型，例如此处只选择在线服务，那您导入后只能部署为在线服务。

参数	说明
“启动命令”	指定模型的启动命令，您可以自定义该命令。 **说明** 包含字符$，|，>，<，`，!，\n，\，?，-v，--volume，--mount，--tmpfs，--privileged，--cap-add的启动命令，在模型发布时将会置空。
“apis定义”	提供模型对外Restfull api数据定义，用于定义模型的输入、输出格式。apis定义填写规范请参见**模型配置文件编写说明**中的apis参数说明，示例代码请参见**apis参数代码示例**。

c. 确认信息填写无误，单击“立即创建”，完成模型创建。

在模型列表中，您可以查看刚创建的模型及其对应的版本。当模型状态变更为“正常”时，表示模型创建成功。在此页面，您还可以进行创建新版本、快速部署服务等操作。

后续操作

部署服务：在“模型列表”中，单击模型的操作列的“部署”，在对应版本所在行，单击“操作”列的部署按钮，可以将模型部署上线为创建模型时所选择的部署类型。

9.3 创建模型规范参考

9.3.1 模型包结构介绍

创建模型时，如果是从OBS中导入元模型，则需要符合一定的模型包规范。

说明

- 模型包规范适用于单模型场景，如果是多模型场景（例如含有多个模型文件）推荐使用自定义镜像方式。
- ModelArts推理平台不支持的AI引擎，推荐使用自定义镜像方式。
- 请参考**创建模型的自定义镜像规范**和**从0-1制作自定义镜像并创建模型**，制作自定义镜像。
- 更多的自定义脚本代码示例，请参考**自定义脚本代码示例**。

模型包里面必须包含“model”文件夹，“model”文件夹下面放置模型文件，模型配置文件，模型推理代码文件。

- **模型文件**：在不同模型包结构中模型文件的要求不同，具体请参见**模型包结构示例**。
- **模型配置文件**：模型配置文件必须存在，文件名固定为“config.json”，有且只有一个，模型配置文件编写请参见**模型配置文件编写说明**。
- **模型推理代码文件**：模型推理代码文件是必选的。文件名固定为“customize_service.py”，此文件有且只能有一个，模型推理代码编写请参见**模型推理代码编写说明**。
  - customize_service.py依赖的py文件可以直接放model目录下，推荐采用相对导入方式导入自定义包。

- customize_service.py依赖的其他文件可以直接放model目录下，需要采用绝对路径方式访问。绝对路径获取请参考**绝对路径如何获取**。

ModelArts针对多种引擎提供了样例及其示例代码，您可以参考样例编写您的配置文件和推理代码，详情请参见**ModelArts样例列表**。ModelArts也提供了常用AI引擎对应的自定义脚本示例，请参见**自定义脚本代码示例**。

如果您在导入元模型过程中遇到问题，可**联系华为云技术支持**协助解决故障。

模型包结构示例

- TensorFlow模型包结构

  发布该模型时只需要指定到“ocr”目录。

```
OBS桶/目录名
├── ocr
│   ├── model 必选： 固定子目录名称，用于放置模型相关文件
│   │   ├── <<自定义python包>> 可选：用户自有的Python包，在模型推理代码中可以直接引用
│   │   ├── saved_model.pb 必选: protocol buffer格式文件，包含该模型的图描述
│   │   ├── variables 对*.pb模型主文件而言必选；固定子目录名称，包含模型的权重偏差等信息
│   │   │   ├── variables.index 必选
│   │   │   ├── variables.data-00000-of-00001 必选
│   │   ├──config.json 必选：模型配置文件，文件名称固定为config.json, 只允许放置一个
│   │   ├──customize_service.py 必选：模型推理代码，文件名称固定为customize_service.py, 只允许放置
一个
customize_service.py依赖的文件可以直接放model目录下
```

- PyTorch模型包结构

  发布该模型时只需要指定到“resnet”目录。

```
OBS桶/目录名
├── resnet
│   ├── model 必选： 固定子目录名称，用于放置模型相关文件
│   │   ├── <<自定义Python包>> 可选：用户自有的Python包，在模型推理代码中可以直接引用
│   │   ├── resnet50.pth 必选，pytorch模型保存文件，保存为“state_dict”，存有权重变量等信息。
│   │   ├──config.json 必选：模型配置文件，文件名称固定为config.json, 只允许放置一个
│   │   ├──customize_service.py 必选：模型推理代码，文件名称固定为customize_service.py, 只允许放置
一个，customize_service.py依赖的文件可以直接放model目录下
```

- Custom模型包结构，与您自定义镜像中AI引擎有关。例如自定义镜像中的AI引擎为TensorFlow，则模型包采用TensorFlow模型包结构。

9.3.2 模型配置文件编写说明

模型开发者发布模型时需要编写配置文件config.json。模型配置文件描述模型用途、模型计算框架、模型精度、推理代码依赖包以及模型对外API接口。

配置文件格式说明

配置文件为JSON格式，参数说明如**表9-8**所示。

表 9-8 参数说明

参数	是否必选	参数类型	描述
model_algorithm	是	String	模型算法，表示该模型的用途，由模型开发者填写，以便使用者理解该模型的用途。只能以英文字母开头，不能包含中文以及&!'\"<>=，不超过36个字符。常见的模型算法有image_classification（图像分类）、object_detection（物体检测）、predict_analysis（预测分析）等。
model_type	是	String	模型AI引擎，表明模型使用的计算框架，支持常用AI框架和“Image”。 ● 可选的常用AI框架请参见**推理支持的AI引擎**。 ● 当model_type设置为Image，表示以自定义镜像方式创建模型，此时swr_location为必填参数。Image镜像制作规范可参见**创建模型的自定义镜像规范**。
runtime	否	String	模型运行时环境，系统默认使用python2.7。runtime可选值与model_type相关，当model_type设置为Image时，不需要设置runtime，当model_type设置为其他常用框架时，请选择您使用的引擎所对应的运行时环境。目前支持的运行时环境列表请参见**推理支持的AI引擎**。 需要注意的是，如果您的模型需指定CPU或GPU上运行时，请根据runtime的后缀信息选择，当runtime中未包含cpu或gpu信息时，请仔细阅读“推理支持的AI引擎”中每个runtime的说明信息。
metrics	否	object数据结构	模型的精度信息，包括平均数、召回率、精确率、准确率，metrics object数据结构说明如**表9-9**所示。 结果会显示在模型详情页面的“模型精度”模块。
apis	否	api数据结构数组	表示模型接收和返回的请求样式，为结构体数据。 即模型可对外提供的Restful API数组，API数据结构如**表9-10**所示。示例代码请参见**apis参数代码示例**。 ● “model_type”为“Image”时，即自定义镜像的模型场景，“apis”可根据镜像实际对外暴露的请求路径在“apis”中声明不同路径的API。 ● “model_type”不为“Image”时，“apis”只能声明一个请求路径为“/”的API，因为系统预置的AI引擎仅暴露一个请求路径为“/”的推理接口。

参数	是否必选	参数类型	描述
dependencies	否	dependency结构数组	表示模型推理代码需要依赖的包，为结构体数据。 模型开发者需要提供包名、安装方式、版本约束。目前只支持pip安装方式。dependency结构数组说明如表9-13所示。 如果模型包内没有推理代码customize_service.py文件，则该字段可不填。自定义镜像模型不支持安装依赖包。 **说明** “dependencies”参数支持多个“dependency”结构数组，以list格式填入，默认安装包存在先后依赖关系（即写在前面的先安装，写在后面的后安装），且支持线下wheel包安装（wheel包必须与模型文件放在同一目录）。示例请参考**导入模型时安装包依赖配置文件如何书写?**
health	否	health数据结构	镜像健康接口配置信息，只有“model_type”为“Image”时才需填写。 如果在滚动升级时要求不中断业务，那么必须提供健康检查的接口供ModelArts调用。health数据结构如表9-15所示。

表 9-9 metrics object 数据结构说明

参数	是否必选	参数类型	描述
f1	否	Number	平均数。精确到小数点后17位，超过17位时，取前17位数值。
recall	否	Number	召回率。精确到小数点后17位，超过17位时，取前17位数值。
precision	否	Number	精确率。精确到小数点后17位，超过17位时，取前17位数值。
accuracy	否	Number	准确率。精确到小数点后17位，超过17位时，取前17位数值。

表 9-10 api 数据结构说明

参数	是否必选	参数类型	描述
url	否	String	请求路径。默认值为“/”。自定义镜像的模型（即model_type为Image时）需要根据镜像内实际暴露的请求路径填写“url”。非自定义镜像模型（即model_type不为Image时）时，“url”只能为“/”。

参数	是否必选	参数类型	描述
method	否	String	请求方法。默认值为“POST”。
request	否	Object	请求体，request结构说明如表9-11所示。
response	否	Object	响应体，response结构说明如表9-12所示。

表 9-11 request 结构说明

参数	是否必选	参数类型	描述
Content-type	在线服务-非必选 批量服务-必选	String	data以指定内容类型发送。默认值为“application/json”。 一般情况包括如下两种内容类型： ● “application/json”，发送json数据。 ● “multipart/form-data”，上传文件。 **说明** 针对机器学习类模型，仅支持“application/json”
data	在线服务-非必选 批量服务-必选	String	请求体以json schema描述。参数说明请参考官方指导。

表 9-12 response 结构说明

参数	是否必选	参数类型	描述
Content-type	在线服务-非必选 批量服务-必选	String	data以指定内容类型发送。默认值为“application/json”。 **说明** 针对机器学习类模型，仅支持“application/json”
data	在线服务-非必选 批量服务-必选	String	响应体以json schema描述。参数说明请参考官方指导。

表 9-13 dependency 结构数组说明

参数	是否必选	参数类型	描述
installer	是	String	安装方式，当前只支持“pip”。
packages	是	package结构数组	依赖包集合，package结构数组说明如表9-14所示。

表 9-14 package 结构数组说明

参数	是否必选	参数类型	描述
package_name	是	String	依赖包名称。不能含有中文及特殊字符&!'"<>=。
package_version	否	String	依赖包版本，如果不强依赖于版本号，则该项不填。不能含有中文及特殊字符&!'"<>=。
restraint	否	String	版本限制条件，当且仅当“package_version”存在时必填，可选“EXACT/ATLEAST/ATMOST”。 ● “EXACT”表示安装给定版本。 ● “ATLEAST”表示安装版本不小于给定版本。 ● “ATMOST”表示安装包版本不大于给定版本。 **说明** ● 如果对版本有明确要求，优先使用“EXACT”；如果使用“EXACT”与系统安装包有冲突，可以选择“ATLEAST” ● 如果对版本没有明确要求，推荐不填写“restraint”、“package_version”，只保留“package_name”参数

表 9-15 health 数据结构说明

参数	是否必选	参数类型	描述
check_method	是	String	健康检查方式。可选“HTTP/EXEC”。 ● HTTP：HTTP请求检查 ● EXEC：执行命令检查。

参数	是否必选	参数类型	描述
command	否	String	健康检查命令。健康检查方式为EXEC时必选。
url	否	String	健康检查接口请求路径。健康检查方式为HTTP时必选。
protocol	否	String	健康检查接口请求协议，默认为http。健康检查方式为HTTP时必选。
initial_delay_seconds	否	String	健康检查初始化延迟时间。
timeout_seconds	否	String	健康检查超时时间。
period_seconds	是	String	健康检查周期。填写大于0且小于等于2147483647的整数，单位为秒。
failure_threshold	是	String	健康检查最大失败次数。填写大于0且小于等于2147483647的整数。

apis 参数代码示例

```
[{
    "url": "/",
    "method": "post",
    "request": {
        "Content-type": "multipart/form-data",
        "data": {
            "type": "object",
            "properties": {
                "images": {
                    "type": "file"
                }
            }
        }
    },
    "response": {
        "Content-type": "applicaton/json",
        "data": {
            "type": "object",
            "properties": {
                "mnist_result": {
                    "type": "array",
                    "item": [
                        {
                            "type": "string"
                        }
                    ]
                }
            }
        }
    }
}]
```

目标检测模型配置文件示例

如下代码以TensorFlow引擎为例，您可以根据实际使用的引擎类型修改model_type参数后使用。

- 模型输入

  key：images

  value：图片文件

- 模型输出

```
{
    "detection_classes": [
        "face",
        "arm"
    ],
    "detection_boxes": [
        [
            33.6,
            42.6,
            104.5,
            203.4
        ],
        [
            103.1,
            92.8,
            765.6,
            945.7
        ]
    ],
    "detection_scores": [0.99, 0.73]
}
```

- 配置文件

```
{
    "model_type": "TensorFlow",
    "model_algorithm": "object_detection",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [{
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "application/json",
            "data": {
                "type": "object",
                "properties": {
                    "detection_classes": {
                        "type": "array",
                        "items": [{
                            "type": "string"
                        }]
                    },
                    "detection_boxes": {
```



```
                    "type": "array",
                    "items": [{
                        "type": "array",
                        "minItems": 4,
                        "maxItems": 4,
                        "items": [{
                            "type": "number"
                        }]
                    }]
                },
                "detection_scores": {
                    "type": "array",
                    "items": [{
                        "type": "number"
                    }]
                }
            }
        }
    }
  }],
  "dependencies": [{
      "installer": "pip",
      "packages": [{
            "restraint": "EXACT",
            "package_version": "1.15.0",
            "package_name": "numpy"
        },
        {
            "restraint": "EXACT",
            "package_version": "5.2.0",
            "package_name": "Pillow"
        }
      ]
  }]
}
```

图像分类模型配置文件示例

如下代码以TensorFlow引擎为例，您可以根据实际使用的引擎类型修改model_type参数后使用。

- 模型输入

  key：images

  value：图片文件

- 模型输出

```
{
  "predicted_label": "flower",
  "scores": [
    ["rose", 0.99],
    ["begonia", 0.01]
  ]
}
```

- 配置文件

```
{
  "model_type": "TensorFlow",
  "model_algorithm": "image_classification",
  "metrics": {
      "f1": 0.345294,
      "accuracy": 0.462963,
      "precision": 0.338977,
      "recall": 0.351852
  },
  "apis": [{
      "url": "/",
      "method": "post",
```



```
    "request": {
        "Content-type": "multipart/form-data",
        "data": {
            "type": "object",
            "properties": {
                "images": {
                    "type": "file"
                }
            }
        }
    },
    "response": {
        "Content-type": "application/json",
        "data": {
            "type": "object",
            "properties": {
                "predicted_label": {
                    "type": "string"
                },
                "scores": {
                    "type": "array",
                    "items": [{
                        "type": "array",
                        "minItems": 2,
                        "maxItems": 2,
                        "items": [
                            {
                                "type": "string"
                            },
                            {
                                "type": "number"
                            }
                        ]
                    }]
                }
            }
        }
    }
  }],
  "dependencies": [{
      "installer": "pip",
      "packages": [{
            "restraint": "ATLEAST",
            "package_version": "1.15.0",
            "package_name": "numpy"
        },
        {
            "restraint": "",
            "package_version": "",
            "package_name": "Pillow"
        }
    ]
  }]
}
```

如下代码以MindSpore引擎为例，您可以根据实际使用的引擎类型修改model_type参数后使用。

- 模型输入

  key：images

  value：图片文件

- 模型输出

  ```
  "[[-2.404526   -3.0476532  -1.9888215   0.45013925 -1.7018927   0.40332815\n  -7.1861157
  11.290332   -1.5861531   5.7887416 ]]"
  ```

- 配置文件

  ```
  {
      "model_algorithm": "image_classification",
  ```



```
    "model_type": "MindSpore",
    "metrics": {
        "f1": 0.124555,
        "recall": 0.171875,
        "precision": 0.0023493892851938493,
        "accuracy": 0.00746268656716417
    },
    "apis": [{
            "url": "/",
            "method": "post",
            "request": {
                "Content-type": "multipart/form-data",
                "data": {
                    "type": "object",
                    "properties": {
                        "images": {
                            "type": "file"
                        }
                    }
                }
            },
            "response": {
                "Content-type": "applicaton/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "mnist_result": {
                            "type": "array",
                            "item": [{
                                "type": "string"
                            }]
                        }
                    }
                }
            }
        }
    ],
    "dependencies": []
}
```

预测分析模型配置文件示例

如下代码以TensorFlow引擎为例，您可以根据实际使用的引擎类型修改model_type参数后使用。

- 模型输入

```
{
    "data": {
        "req_data": [
            {
                "buying_price": "high",
                "maint_price": "high",
                "doors": "2",
                "persons": "2",
                "lug_boot": "small",
                "safety": "low",
                "acceptability": "acc"
            },
            {
                "buying_price": "high",
                "maint_price": "high",
                "doors": "2",
                "persons": "2",
                "lug_boot": "small",
                "safety": "low",
                "acceptability": "acc"
            }
        ]
```

```
    }
}
```

- 模型输出

```
{
    "data": {
        "resp_data": [
            {
                "predict_result": "unacc"
            },
            {
                "predict_result": "unacc"
            }
        ]
    }
}
```

- 配置文件

  说明

  代码中request结构和response结构中的data参数是json schema数据结构。data/properties里面的内容对应“模型输入”和“模型输出”。

```
{
    "model_type": "TensorFlow",
    "model_algorithm": "predict_analysis",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [
        {
            "url": "/",
            "method": "post",
            "request": {
                "Content-type": "application/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "data": {
                            "type": "object",
                            "properties": {
                                "req_data": {
                                    "items": [
                                        {
                                            "type": "object",
                                            "properties": {}
                                        }
                                    ],
                                    "type": "array"
                                }
                            }
                        }
                    }
                }
            },
            "response": {
                "Content-type": "application/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "data": {
                            "type": "object",
                            "properties": {
                                "resp_data": {
                                    "type": "array",
                                    "items": [
                                        {
```



```
                            "type": "object",
                            "properties": {}
                        }
                    ]
                }
            }
        }
    }
  ],
  "dependencies": [
    {
      "installer": "pip",
      "packages": [
        {
          "restraint": "EXACT",
          "package_version": "1.15.0",
          "package_name": "numpy"
        },
        {
          "restraint": "EXACT",
          "package_version": "5.2.0",
          "package_name": "Pillow"
        }
      ]
    }
  ]
}
```

自定义镜像类型的模型配置文件示例

模型输入和输出与**目标检测模型配置文件示例**类似。

- 模型预测输入为**图片类型**时，request请求示例如下：

  该示例表示模型预测接收一个参数名为images、参数类型为file的预测请求，在推理界面会显示文件上传按钮，以文件形式进行预测。

  ```
  {
    "Content-type": "multipart/form-data",
    "data": {
      "type": "object",
      "properties": {
        "images": {
          "type": "file"
        }
      }
    }
  }
  ```

- 模型预测输入为**json数据类型**时，request请求示例如下：

  该示例表示模型预测接收json请求体，只有一个参数名为input、参数类型为string的预测请求，在推理界面会显示文本输入框，用于填写预测请求。

  ```
  {
    "Content-type": "application/json",
    "data": {
      "type": "object",
      "properties": {
        "input": {
          "type": "string"
        }
      }
    }
  }
  ```

完整请求示例如下：

```
{
    "model_algorithm": "image_classification",
    "model_type": "Image",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [{
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "application/json",
            "data": {
                "type": "object",
                "required": [
                    "predicted_label",
                    "scores"
                ],
                "properties": {
                    "predicted_label": {
                        "type": "string"
                    },
                    "scores": {
                        "type": "array",
                        "items": [{
                            "type": "array",
                            "minItems": 2,
                            "maxItems": 2,
                            "items": [{
                                    "type": "string"
                                },
                                {
                                    "type": "number"
                                }
                            ]
                        }]
                    }
                }
            }
        }
    }]
}
```

机器学习类型的模型配置文件示例

以下代码以XGBoost为例。

- 模型输入：

```
{
    "req_data": [
        {
            "sepal_length": 5,
            "sepal_width": 3.3,
            "petal_length": 1.4,
            "petal_width": 0.2
```



```
    },
    {
        "sepal_length": 5,
        "sepal_width": 2,
        "petal_length": 3.5,
        "petal_width": 1
    },
    {
        "sepal_length": 6,
        "sepal_width": 2.2,
        "petal_length": 5,
        "petal_width": 1.5
    }
  ]
}
```

- 模型输出：

```
{
  "resp_data": [
    {
        "predict_result": "Iris-setosa"
    },
    {
        "predict_result": "Iris-versicolor"
    }
  ]
}
```

- 配置文件：

```
{
  "model_type": "XGBoost",
  "model_algorithm": "xgboost_iris_test",
  "runtime": "python2.7",
  "metrics": {
    "f1": 0.345294,
    "accuracy": 0.462963,
    "precision": 0.338977,
    "recall": 0.351852
  },
  "apis": [
    {
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "application/json",
            "data": {
                "type": "object",
                "properties": {
                    "req_data": {
                        "items": [
                            {
                                "type": "object",
                                "properties": {}
                            }
                        ],
                        "type": "array"
                    }
                }
            }
        },
        "response": {
            "Content-type": "applicaton/json",
            "data": {
                "type": "object",
                "properties": {
                    "resp_data": {
                        "type": "array",
                        "items": [
                            {
```



```
                        "type": "object",
                        "properties": {
                            "predict_result": {}
                        }
                    }
                ]
            }
        }
    }
  ]
}
```

使用自定义依赖包的模型配置文件示例

如下示例中，定义了1.16.4版本的numpy的依赖环境。

```
{
  "model_algorithm": "image_classification",
  "model_type": "TensorFlow",
  "runtime": "python3.6",
  "apis": [
    {
      "url": "/",
      "method": "post",
      "request": {
        "Content-type": "multipart/form-data",
        "data": {
          "type": "object",
          "properties": {
            "images": {
              "type": "file"
            }
          }
        }
      },
      "response": {
        "Content-type": "applicaton/json",
        "data": {
          "type": "object",
          "properties": {
            "mnist_result": {
              "type": "array",
              "item": [
                {
                  "type": "string"
                }
              ]
            }
          }
        }
      }
    }
  ],
  "metrics": {
    "f1": 0.124555,
    "recall": 0.171875,
    "precision": 0.00234938928519385,
    "accuracy": 0.00746268656716417
  },
  "dependencies": [
    {
      "installer": "pip",
      "packages": [
        {
          "restraint": "EXACT",
          "package_version": "1.16.4",
```

```
            "package_name": "numpy"
        }
      ]
    }
  ]
}
```

9.3.3 模型推理代码编写说明

本章节介绍了在ModelArts中模型推理代码编写的通用方法及说明，针对常用AI引擎的自定义脚本代码示例（包含推理代码示例），请参见**自定义脚本代码示例**。本文在编写说明下方提供了一个TensorFlow引擎的推理代码示例以及一个在推理脚本中自定义推理逻辑的示例。

ModelArts推理因API网关（APIG）的限制，模型单次预测的时间不能超过40S，模型推理代码编写需逻辑清晰，代码简洁，以此达到更好的推理效果。

推理代码编写指导

1. 在模型代码推理文件“customize_service.py”中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如**表9-16**所示。导入语句所涉及的Python包在ModelArts环境中已配置，用户无需自行安装。

表 9-16 各模型类型的父类名称和导入语句

模型类型	父类	导入语句
TensorFlow	TfServingBaseService	from model_service.tfserving_model_service import TfServingBaseService
PyTorch	PTServingBaseService	from model_service.pytorch_model_service import PTServingBaseService
MindSpore	SingleNodeService	from model_service.model_service import SingleNodeService

2. 可以重写的方法有以下几种。

表 9-17 重写方法

方法名	说明
__init__(self, model_name, model_path)	初始化方法，适用于深度学习框架模型。该方法内加载模型及标签等（pytorch和caffe类型模型必须重写，实现模型加载逻辑）。
__init__(self, model_path)	初始化方法，适用于机器学习框架模型。该方法内初始化模型的路径（self.model_path）。在Spark_MLlib中，该方法还会初始化SparkSession（self.spark）。
_preprocess(self, data)	预处理方法，在推理请求前调用，用于将API接口输入的用户原始请求数据转换为模型期望输入数据。
_inference(self, data)	实际推理请求方法（不建议重写，重写后会覆盖ModelArts内置的推理过程，运行自定义的推理逻辑）。

方法名	说明
_postprocess(self, data)	后处理方法，在推理请求完成后调用，用于将模型输出转换为API接口输出。

说明

- 用户可以选择重写preprocess和postprocess方法，以实现API输入数据的预处理和推理输出结果的后处理。
- 重写模型父类的初始化方法init可能导致模型“运行异常”。

3. 可以使用的属性为模型所在的本地路径，属性名为“self.model_path”。另外pyspark模型在“customize_service.py”中可以使用“self.spark”获取SparkSession对象。

说明

推理代码中，需要通过绝对路径读取文件。模型所在的本地路径可以通过self.model_path属性获得。

- 当使用TensorFlow、Caffe、MXNet时，self.model_path为模型文件目录路径，读取文件示例如下：

```
# model目录下放置label.json文件，此处读取
with open(os.path.join(self.model_path, 'label.json')) as f:
    self.label = json.load(f)
```

- 当使用PyTorch、Scikit_Learn、pyspark时，self.model_path为模型文件路径，读取文件示例如下：

```
# model目录下放置label.json文件，此处读取
dir_path = os.path.dirname(os.path.realpath(self.model_path))
with open(os.path.join(dir_path, 'label.json')) as f:
    self.label = json.load(f)
```

4. 预处理方法、实际推理请求方法和后处理方法中的接口传入“data”当前支持两种content-type，即“multipart/form-data”和“application/json”。
   - “multipart/form-data”请求

```
curl -X POST \
  <modelarts-inference-endpoint> \
  -F image1=@cat.jpg \
  -F image2=@horse.jpg
```

   对应的传入data为

```
[
  {
    "image1":{
      "cat.jpg":"<cat.jpg file io>"
    }
  },
  {
    "image2":{
      "horse.jpg":"<horse.jpg file io>"
    }
  }
]
```

   - “application/json”请求

```
curl -X POST \
  <modelarts-inference-endpoint> \
  -d '{
   "images":"base64 encode image"
   }'
```

   对应的传入data为python dict


```
{
  "images":"base64 encode image"
}
```

TensorFlow 的推理脚本示例

TensorFlow MnistService示例如下。更多TensorFlow推理代码示例请参考**Tensorflow**、**Tensorflow2.1**。

- 推理代码

```
from PIL import Image
import numpy as np
from model_service.tfserving_model_service import TfServingBaseService

class MnistService(TfServingBaseService):

    def _preprocess(self, data):
        preprocessed_data = {}

        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1, 784))
                preprocessed_data[k] = image1

        return preprocessed_data

    def _postprocess(self, data):

        infer_output = {}

        for output_name, result in data.items():

            infer_output["mnist_result"] = result[0].index(max(result[0]))

        return infer_output
```

- 请求

```
curl -X POST \ 在线服务地址 \ -F images=@test.jpg
```

- 返回

```
{"mnist_result": 7}
```

在上面的代码示例中，完成了将用户表单输入的图片的大小调整，转换为可以适配模型输入的shape。首先通过Pillow库读取“32×32”的图片，调整图片大小为“1×784”以匹配模型输入。在后续处理中，转换模型输出为列表，用于Restful接口输出展示。

自定义推理逻辑的推理脚本示例

首先，需要在配置文件中，定义自己的依赖包，详细示例请参见**使用自定义依赖包的模型配置文件示例**。然后通过如下示例代码，实现了“saved_model”格式模型的加载推理。

说明

当前推理基础镜像使用的python的logging模块，采用的是默认的日志级别Warning，即当前只有warning级别的日志可以默认查询出来。如果想要指定INFO等级的日志能够查询出来，需要在代码中指定logging的输出日志等级为INFO级别。

```
# -*- coding: utf-8 -*-
import json
import os
import threading
```

```
import numpy as np
import tensorflow as tf
from PIL import Image
from model_service.tfserving_model_service import TfServingBaseService
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class MnistService(TfServingBaseService):
    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model_inputs = {}
        self.model_outputs = {}

        # label文件可以在这里加载,在后处理函数里使用
        # label.txt放在OBS和模型包的目录

        # with open(os.path.join(self.model_path, 'label.txt')) as f:
        #     self.label = json.load(f)

        # 非阻塞方式加载saved_model模型，防止阻塞超时
        thread = threading.Thread(target=self.get_tf_sess)
        thread.start()

    def get_tf_sess(self):
        # 加载saved_model格式的模型
        # session要重用，建议不要用with语句
        sess = tf.Session(graph=tf.Graph())
        meta_graph_def = tf.saved_model.loader.load(sess, [tf.saved_model.tag_constants.SERVING],
self.model_path)
        signature_defs = meta_graph_def.signature_def
        self.sess = sess
        signature = []

        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)
        if len(signature) == 1:
            model_signature = signature[0]
        else:
            logger.warning("signatures more than one, use serving_default signature")
            model_signature = tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY

        logger.info("model signature: %s", model_signature)

        for signature_name in meta_graph_def.signature_def[model_signature].inputs:
            tensorinfo = meta_graph_def.signature_def[model_signature].inputs[signature_name]
            name = tensorinfo.name
            op = self.sess.graph.get_tensor_by_name(name)
            self.model_inputs[signature_name] = op

        logger.info("model inputs: %s", self.model_inputs)

        for signature_name in meta_graph_def.signature_def[model_signature].outputs:
            tensorinfo = meta_graph_def.signature_def[model_signature].outputs[signature_name]
            name = tensorinfo.name
            op = self.sess.graph.get_tensor_by_name(name)
            self.model_outputs[signature_name] = op

        logger.info("model outputs: %s", self.model_outputs)

    def _preprocess(self, data):
        # https两种请求形式
        # 1. form-data文件格式的请求对应：data = {"请求key值":{"文件名":<文件io>}}
        # 2. json格式对应：data = json.loads("接口传入的json体")
        preprocessed_data = {}

        for k, v in data.items():
```



```
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1, 28, 28))
                preprocessed_data[k] = image1

        return preprocessed_data

    def _inference(self, data):
        feed_dict = {}
        for k, v in data.items():
            if k not in self.model_inputs.keys():
                logger.error("input key %s is not in model inputs %s", k, list(self.model_inputs.keys()))
                raise Exception("input key %s is not in model inputs %s" % (k, list(self.model_inputs.keys())))
            feed_dict[self.model_inputs[k]] = v

        result = self.sess.run(self.model_outputs, feed_dict=feed_dict)
        logger.info('predict result : ' + str(result))
        return result

    def _postprocess(self, data):
        infer_output = {"mnist_result": []}
        for output_name, results in data.items():

            for result in results:
                infer_output["mnist_result"].append(np.argmax(result))

        return infer_output

    def __del__(self):
        self.sess.close()
```

说明

对于ModelArts不支持的结构模型或者多模型加载，需要__init__方法中自己指定模型加载的路径。示例代码如下：

```
# -*- coding: utf-8 -*-
import os
from model_service.tfserving_model_service import TfServingBaseService

class MnistService(TfServingBaseService):
    def __init__(self, model_name, model_path):
        # 获取程序当前运行路径，即model文件夹所在的路径
        root = os.path.dirname(os.path.abspath(__file__))
        # test.onnx为待加载模型文件的名称，需要放在model文件夹下
        self.model_path = os.path.join(root, test.onnx)

        # 多模型加载，例如：test2.onnx
        # self.model_path2 = os.path.join(root, test2.onnx)
```

9.3.4 自定义引擎创建模型规范

使用自定义引擎创建模型，用户可以通过选择自己存储在SWR服务中的镜像作为模型的引擎，指定预先存储于OBS服务中的文件目录路径作为模型包来创建模型，轻松地应对ModelArts平台预置引擎无法满足个性化诉求的场景。

ModelArts将自定义引擎类型的模型部署为服务时，会先将模型相关的SWR镜像下载至集群中，用“uid=1000, gid=100”的用户启动SWR镜像为容器，然后将OBS文件下载到容器中的“/home/mind/model”目录下，最后执行SWR镜像中预置的启动命令。ModelArts平台会在APIG上注册一个预测接口提供给用户使用，用户可以通过平台提供的预测接口访问服务。

自定义引擎创建模型的规范

使用自定义引擎创建模型，用户的SWR镜像、OBS模型包和文件大小需要满足以下规范：

- SWR镜像规范：
  - 镜像必须内置一个用户名为“ma-user”，组名为“ma-group”的普通用户，且必须确保该用户的uid=1000、gid=100。内置用户的dockerfile指令如下：
    ```
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
    ```
  - 明确设置镜像的启动命令。在dockerfile文件中指定cmd，dockerfile指令示例如下：
    ```
    CMD sh /home/mind/run.sh
    ```
    启动入口文件run.sh需要自定义。示例如下：
    ```
    #!/bin/bash

    # 自定义脚本内容
    ...

    # run.sh调用app.py启动服务器，app.py请参考https示例
    python app.py
    ```
    说明

    除了按上述要求设置启动命令，您也可以在镜像中自定义启动命令，在创建模型时填写与您镜像中相同的启动命令。
  - 提供的服务可使用HTTPS/HTTP协议和监听的容器端口，使用的协议和端口号请根据模型实际定义的推理接口进行配置。HTTPS协议的示例可参考**https示例**。
  - （可选）服务对外提供的端口，提供URL路径为“/health”的健康检查服务（健康检查的URL路径必须为“/health”）。
- OBS模型包规范

  模型包的名字必须为model。模型包规范请参见**模型包规范介绍**。
- 文件大小规范

  当使用公共资源池时，SWR的镜像大小（指下载后的镜像大小，非SWR界面显示的压缩后的镜像大小）和OBS模型包大小总和不大于30G。

https 示例

使用Flask启动https，Webserver代码示例如下：

```
from flask import Flask, request
import json

app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----------- in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
```

```
    print("----------- in goodbye func ----------")
    return '\nGoodbye!\n'


@app.route('/', methods=['POST'])
def default_func():
    print("----------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

@app.route('/health', methods=['GET'])
def healthy():
    return "{\"status\": \"OK\"}"

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080, ssl_context='adhoc')
```

在本地机器调试

自定义引擎的规范可以在安装有docker的本地机器上通过以下步骤提前验证：

1. 将自定义引擎镜像下载至本地机器，假设镜像名为custom_engine:v1。
2. 将模型包文件夹复制到本地机器，假设模型包文件夹名字为model。
3. 在模型包文件夹的同级目录下验证如下命令拉起服务：
   ```
   docker run --user 1000:100 -p 8080:8080 -v model:/home/mind/model    custom_engine:v1
   ```

   说明

   该指令无法完全模拟线上，主要是由于-v挂载进去的目录是root权限。在线上，模型文件从OBS下载到/home/mind/model目录之后，文件owner将统一修改为ma-user。
4. 在本地机器上启动另一个终端，执行以下验证指令，得到符合预期的推理结果。
   ```
   curl https://127.0.0.1:8080/${推理服务的请求路径}
   ```

推理部署示例

本节将详细说明以自定义引擎方式创建模型的步骤。

1. 创建模型并查看模型详情

   登录ModelArts管理控制台，进入“模型管理”页面中，单击“创建模型”，进入模型创建页面，设置相关参数如下：
   - 元模型来源：选择“从对象存储服务（OBS）中选择”。
   - 选择元模型：从OBS中选择一个模型包。
   - AI引擎：选择“Custom”。
   - 引擎包：从容器镜像中选择一个镜像。

     其他参数保持默认值。

   单击“立即创建”，跳转到模型列表页，查看模型状态，当状态变为“正常”，模型创建成功。

图 9-6 创建模型

单击模型名称，进入模型详情页面，查看模型详情信息。

2. 部署服务并查看详情

   在模型详情页面，单击右上角“部署>在线服务”，进入服务部署页面，模型和版本默认选中，选择合适的“实例规格”（例如CPU：2核 8GB），其他参数可保持默认值，单击“下一步”，跳转至服务列表页，当服务状态变为“运行中”，服务部署成功。

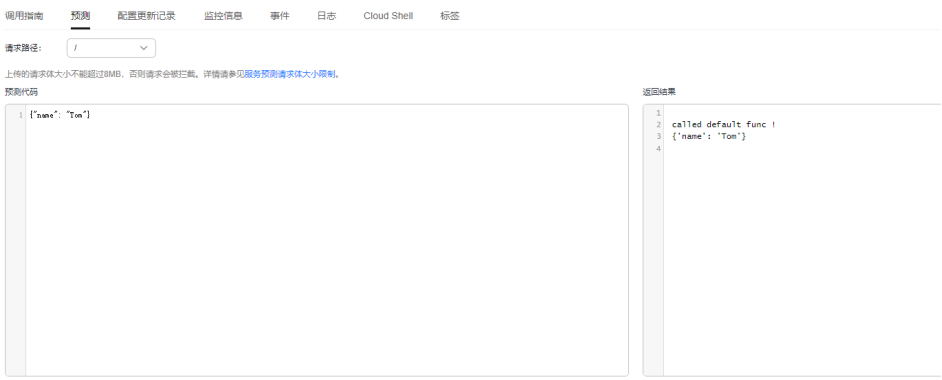
   单击服务名称，进入服务详情页面，查看服务详情信息，单击“日志”页签，查看服务日志信息。

图 9-7 查看服务日志信息

3. 服务预测

   在服务详情页面，单击“预测”页签，进行服务预测。

图 9-8 服务预测



9.3.5 自定义脚本代码示例

从OBS中导入模型文件创建模型时，模型文件包需符合ModelArts的模型包规范，推理代码和配置文件也需遵循ModelArts的要求。

本章节提供针对常用AI引擎的自定义脚本代码示例（包含推理代码示例）。模型推理代码编写的通用方法及说明请见模型推理代码编写说明。

Tensorflow

TensorFlow存在两种接口类型，keras接口和tf接口，其训练和保存模型的代码存在差异，但是推理代码编写方式一致。

训练模型（keras接口）

```
from keras.models import Sequential
model = Sequential()
from keras.layers import Dense
import tensorflow as tf

# 导入训练数据集
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

print(x_train.shape)

from keras.layers import Dense
from keras.models import Sequential
import keras
from keras.layers import Dense, Activation, Flatten, Dropout

# 定义模型网络
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(units=5120,activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(units=10, activation='softmax'))

# 定义优化器，损失函数等
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
# 训练
model.fit(x_train, y_train, epochs=2)
# 评估
model.evaluate(x_test, y_test)
```

保存模型（keras接口）

```
from keras import backend as K

# K.get_session().run(tf.global_variables_initializer())

# 定义预测接口的inputs和outputs
# inputs和outputs字典的key值会作为模型输入输出tensor的索引键
# 模型输入输出定义需要和推理自定义脚本相匹配
predict_signature = tf.saved_model.signature_def_utils.predict_signature_def(
    inputs={"images" : model.input},
    outputs={"scores" : model.output}
)

# 定义保存路径
```

```
builder = tf.saved_model.builder.SavedModelBuilder('./mnist_keras/')

builder.add_meta_graph_and_variables(

    sess = K.get_session(),
    # 推理部署需要定义tf.saved_model.tag_constants.SERVING标签
    tags=[tf.saved_model.tag_constants.SERVING],
    """
    signature_def_map：items只能有一个，或者需要定义相应的key为
    tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY
    """
    signature_def_map={
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
            predict_signature
    }

)
builder.save()
```

训练模型（tf接口）

```
from __future__ import print_function

import gzip
import os
import urllib

import numpy
import tensorflow as tf
from six.moves import urllib

# 训练数据来源于yann lecun官方网站http://yann.lecun.com/exdb/mnist/
SOURCE_URL = 'http://yann.lecun.com/exdb/mnist/'
TRAIN_IMAGES = 'train-images-idx3-ubyte.gz'
TRAIN_LABELS = 'train-labels-idx1-ubyte.gz'
TEST_IMAGES = 't10k-images-idx3-ubyte.gz'
TEST_LABELS = 't10k-labels-idx1-ubyte.gz'
VALIDATION_SIZE = 5000


def maybe_download(filename, work_directory):
    """Download the data from Yann's website, unless it's already here."""
    if not os.path.exists(work_directory):
        os.mkdir(work_directory)
    filepath = os.path.join(work_directory, filename)
    if not os.path.exists(filepath):
        filepath, _ = urllib.request.urlretrieve(SOURCE_URL + filename, filepath)
        statinfo = os.stat(filepath)
        print('Successfully downloaded %s %d bytes.' % (filename, statinfo.st_size))
    return filepath


def _read32(bytestream):
    dt = numpy.dtype(numpy.uint32).newbyteorder('>')
    return numpy.frombuffer(bytestream.read(4), dtype=dt)[0]


def extract_images(filename):
    """Extract the images into a 4D uint8 numpy array [index, y, x, depth]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2051:
            raise ValueError(
                'Invalid magic number %d in MNIST image file: %s' %
                (magic, filename))
        num_images = _read32(bytestream)
        rows = _read32(bytestream)
        cols = _read32(bytestream)
        buf = bytestream.read(rows * cols * num_images)
```



```
        data = numpy.frombuffer(buf, dtype=numpy.uint8)
        data = data.reshape(num_images, rows, cols, 1)
        return data


def dense_to_one_hot(labels_dense, num_classes=10):
    """Convert class labels from scalars to one-hot vectors."""
    num_labels = labels_dense.shape[0]
    index_offset = numpy.arange(num_labels) * num_classes
    labels_one_hot = numpy.zeros((num_labels, num_classes))
    labels_one_hot.flat[index_offset + labels_dense.ravel()] = 1
    return labels_one_hot


def extract_labels(filename, one_hot=False):
    """Extract the labels into a 1D uint8 numpy array [index]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2049:
            raise ValueError(
                'Invalid magic number %d in MNIST label file: %s' %
                (magic, filename))
        num_items = _read32(bytestream)
        buf = bytestream.read(num_items)
        labels = numpy.frombuffer(buf, dtype=numpy.uint8)
        if one_hot:
            return dense_to_one_hot(labels)
        return labels


class DataSet(object):
    """Class encompassing test, validation and training MNIST data set."""

    def __init__(self, images, labels, fake_data=False, one_hot=False):
        """Construct a DataSet. one_hot arg is used only if fake_data is true."""

        if fake_data:
            self._num_examples = 10000
            self.one_hot = one_hot
        else:
            assert images.shape[0] == labels.shape[0], (
                    'images.shape: %s labels.shape: %s' % (images.shape,
                                                           labels.shape))
            self._num_examples = images.shape[0]

            # Convert shape from [num examples, rows, columns, depth]
            # to [num examples, rows*columns] (assuming depth == 1)
            assert images.shape[3] == 1
            images = images.reshape(images.shape[0],
                                    images.shape[1] * images.shape[2])
            # Convert from [0, 255] -> [0.0, 1.0].
            images = images.astype(numpy.float32)
            images = numpy.multiply(images, 1.0 / 255.0)
        self._images = images
        self._labels = labels
        self._epochs_completed = 0
        self._index_in_epoch = 0

    @property
    def images(self):
        return self._images

    @property
    def labels(self):
        return self._labels

    @property
    def num_examples(self):
```



```
        return self._num_examples

    @property
    def epochs_completed(self):
        return self._epochs_completed

    def next_batch(self, batch_size, fake_data=False):
        """Return the next `batch_size` examples from this data set."""
        if fake_data:
            fake_image = [1] * 784
            if self.one_hot:
                fake_label = [1] + [0] * 9
            else:
                fake_label = 0
            return [fake_image for _ in range(batch_size)], [
                fake_label for _ in range(batch_size)
            ]
        start = self._index_in_epoch
        self._index_in_epoch += batch_size
        if self._index_in_epoch > self._num_examples:
            # Finished epoch
            self._epochs_completed += 1
            # Shuffle the data
            perm = numpy.arange(self._num_examples)
            numpy.random.shuffle(perm)
            self._images = self._images[perm]
            self._labels = self._labels[perm]
            # Start next epoch
            start = 0
            self._index_in_epoch = batch_size
            assert batch_size <= self._num_examples
        end = self._index_in_epoch
        return self._images[start:end], self._labels[start:end]


def read_data_sets(train_dir, fake_data=False, one_hot=False):
    """Return training, validation and testing data sets."""

    class DataSets(object):
        pass

    data_sets = DataSets()

    if fake_data:
        data_sets.train = DataSet([], [], fake_data=True, one_hot=one_hot)
        data_sets.validation = DataSet([], [], fake_data=True, one_hot=one_hot)
        data_sets.test = DataSet([], [], fake_data=True, one_hot=one_hot)
        return data_sets

    local_file = maybe_download(TRAIN_IMAGES, train_dir)
    train_images = extract_images(local_file)

    local_file = maybe_download(TRAIN_LABELS, train_dir)
    train_labels = extract_labels(local_file, one_hot=one_hot)

    local_file = maybe_download(TEST_IMAGES, train_dir)
    test_images = extract_images(local_file)

    local_file = maybe_download(TEST_LABELS, train_dir)
    test_labels = extract_labels(local_file, one_hot=one_hot)

    validation_images = train_images[:VALIDATION_SIZE]
    validation_labels = train_labels[:VALIDATION_SIZE]
    train_images = train_images[VALIDATION_SIZE:]
    train_labels = train_labels[VALIDATION_SIZE:]

    data_sets.train = DataSet(train_images, train_labels)
    data_sets.validation = DataSet(validation_images, validation_labels)
    data_sets.test = DataSet(test_images, test_labels)
```



```
    return data_sets

training_iteration = 1000

modelarts_example_path =  './modelarts-mnist-train-save-deploy-example'

export_path = modelarts_example_path + '/model/'
data_path = './'

print('Training model...')
mnist = read_data_sets(data_path, one_hot=True)
sess = tf.InteractiveSession()
serialized_tf_example = tf.placeholder(tf.string, name='tf_example')
feature_configs = {'x': tf.FixedLenFeature(shape=[784], dtype=tf.float32), }
tf_example = tf.parse_example(serialized_tf_example, feature_configs)
x = tf.identity(tf_example['x'], name='x')  # use tf.identity() to assign name
y_ = tf.placeholder('float', shape=[None, 10])
w = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
sess.run(tf.global_variables_initializer())
y = tf.nn.softmax(tf.matmul(x, w) + b, name='y')
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
values, indices = tf.nn.top_k(y, 10)
table = tf.contrib.lookup.index_to_string_table_from_tensor(
    tf.constant([str(i) for i in range(10)]))
prediction_classes = table.lookup(tf.to_int64(indices))
for _ in range(training_iteration):
    batch = mnist.train.next_batch(50)
    train_step.run(feed_dict={x: batch[0], y_: batch[1]})
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
print('training accuracy %g' % sess.run(
    accuracy, feed_dict={
        x: mnist.test.images,
        y_: mnist.test.labels
    }))
print('Done training!')
```

保存模型（tf接口）

```
# 导出模型
# 模型需要采用saved_model接口保存
print('Exporting trained model to', export_path)
builder = tf.saved_model.builder.SavedModelBuilder(export_path)

tensor_info_x = tf.saved_model.utils.build_tensor_info(x)
tensor_info_y = tf.saved_model.utils.build_tensor_info(y)

# 定义预测接口的inputs和outputs
# inputs和outputs字典的key值会作为模型输入输出tensor的索引键
# 模型输入输出定义需要和推理自定义脚本相匹配
prediction_signature = (
    tf.saved_model.signature_def_utils.build_signature_def(
        inputs={'images': tensor_info_x},
        outputs={'scores': tensor_info_y},
        method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME))

legacy_init_op = tf.group(tf.tables_initializer(), name='legacy_init_op')
builder.add_meta_graph_and_variables(
    # tag设为serve/tf.saved_model.tag_constants.SERVING
    sess, [tf.saved_model.tag_constants.SERVING],
    signature_def_map={
        'predict_images':
            prediction_signature,
    },
    legacy_init_op=legacy_init_op)

builder.save()
```

```
print('Done exporting!')
```

推理代码（keras接口和tf接口）

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表9-16**。本案例中调用父类“_inference(self, data)”推理请求方法，因此下文代码中不需要重写方法。

```
from PIL import Image
import numpy as np
from model_service.tfserving_model_service import TfServingBaseService


class MnistService(TfServingBaseService):

    # 预处理中处理用户HTTPS接口输入匹配模型输入
    # 对应上述训练部分的模型输入为{"images":<array>}
    def _preprocess(self, data):

        preprocessed_data = {}
        images = []
        # 对输入数据进行迭代
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1,784))
                images.append(image1)
        # 返回numpy array
        images = np.array(images,dtype=np.float32)
        # 对传入的多个样本做batch处理，shape保持和训练时输入一致
        images.resize((len(data), 784))
        preprocessed_data['images'] = images
        return preprocessed_data

    # 对应的上述训练部分保存模型的输出为{"scores":<array>}
    # 后处理中处理模型输出为HTTPS的接口输出
    def _postprocess(self, data):
        infer_output = {"mnist_result": []}
        # 迭代处理模型输出
        for output_name, results in data.items():
            for result in results:
                infer_output["mnist_result"].append(result.index(max(result)))
        return infer_output
```

Tensorflow2.1

训练并保存模型

```
from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    # 对输出层命名output，在模型推理时通过该命名取结果
    tf.keras.layers.Dense(10, activation='softmax', name="output")
])
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

tf.keras.models.save_model(model, "./mnist")
```

推理代码

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表9-16**。

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tfserving_model_service import TfServingBaseService

logger = logging.getLogger()
logger.setLevel(logging.INFO)


class MnistService(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

        # label文件可以在这里加载,在后处理函数里使用
        # label.txt放在obs和模型包的目录

        # with open(os.path.join(self.model_path, 'label.txt')) as f:
        #     self.label = json.load(f)
        # 非阻塞方式加载saved_model模型，防止阻塞超时
        thread = threading.Thread(target=self.load_model)
        thread.start()

    def load_model(self):
        # load saved_model 格式的模型
        self.model = tf.saved_model.load(self.model_path)

        signature_defs = self.model.signatures.keys()

        signature = []
        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)

        if len(signature) == 1:
            model_signature = signature[0]
        else:
            logging.warning("signatures more than one, use serving_default signature from %s", signature)
            model_signature = tf.saved_model.DEFAULT_SERVING_SIGNATURE_DEF_KEY

        self.predict = self.model.signatures[model_signature]

    def _preprocess(self, data):
        images = []
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((28, 28, 1))
                images.append(image1)
```



```
        images = tf.convert_to_tensor(images, dtype=tf.dtypes.float32)
        preprocessed_data = images

        return preprocessed_data

    def _inference(self, data):

        return self.predict(data)

    def _postprocess(self, data):

        return {
            "result": int(data["output"].numpy()[0].argmax())
        }
```

Pytorch

训练模型

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms

# 定义网络结构
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 输入第二维需要为784
        self.hidden1 = nn.Linear(784, 5120, bias=False)
        self.output = nn.Linear(5120, 10, bias=False)

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = F.relu((self.hidden1(x)))
        x = F.dropout(x, 0.2)
        x = self.output(x)
        return F.log_softmax(x)

def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                       100. * batch_idx / len(train_loader), loss.item()))

def test( model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()
```



```
    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

device = torch.device("cpu")

batch_size=64

kwargs={}

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor()
                   ])),
    batch_size=batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=False, transform=transforms.Compose([
        transforms.ToTensor()
    ])),
    batch_size=1000, shuffle=True, **kwargs)

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
optimizer = optim.Adam(model.parameters())

for epoch in range(1, 2 + 1):
    train(model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
```

保存模型

```
# 必须采用state_dict的保存方式，支持异地部署
torch.save(model.state_dict(), "pytorch_mnist/mnist_mlp.pt")
```

推理代码

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表9-16**。

```
from PIL import Image
import log
from model_service.pytorch_model_service import PTServingBaseService
import torch.nn.functional as F

import torch.nn as nn
import torch
import json

import numpy as np

logger = log.getLogger(__name__)

import torchvision.transforms as transforms

# 定义模型预处理
infer_transformation = transforms.Compose([
    transforms.Resize((28,28)),
    # 需要处理成pytorch tensor
    transforms.ToTensor()
])


import os


class PTVisionService(PTServingBaseService):
```



```
    def __init__(self, model_name, model_path):
        # 调用父类构造方法
        super(PTVisionService, self).__init__(model_name, model_path)
        # 调用自定义函数加载模型
        self.model = Mnist(model_path)
        # 加载标签
        self.label = [0,1,2,3,4,5,6,7,8,9]
        # 亦可通过文件标签文件加载
        # model目录下放置label.json文件，此处读取
        dir_path = os.path.dirname(os.path.realpath(self.model_path))
        with open(os.path.join(dir_path, 'label.json')) as f:
            self.label = json.load(f)


    def _preprocess(self, data):

        preprocessed_data = {}
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # 灰度处理
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
                        input_batch.append(infer_transformation(image1).cuda())
                    else:
                        input_batch.append(infer_transformation(image1))
            input_batch_var = torch.autograd.Variable(torch.stack(input_batch, dim=0), volatile=True)
            print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var

        return preprocessed_data

    def _postprocess(self, data):
        results = []
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

    def _inference(self, data):

        result = {}
        for k, v in data.items():
            result[k] = self.model(v)

        return result

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.hidden1 = nn.Linear(784, 5120, bias=False)
        self.output = nn.Linear(5120, 10, bias=False)

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = F.relu((self.hidden1(x)))
        x = F.dropout(x, 0.2)
        x = self.output(x)
        return F.log_softmax(x)



def Mnist(model_path, **kwargs):
    # 生成网络
    model = Net()
    # 加载模型
    if torch.cuda.is_available():
```



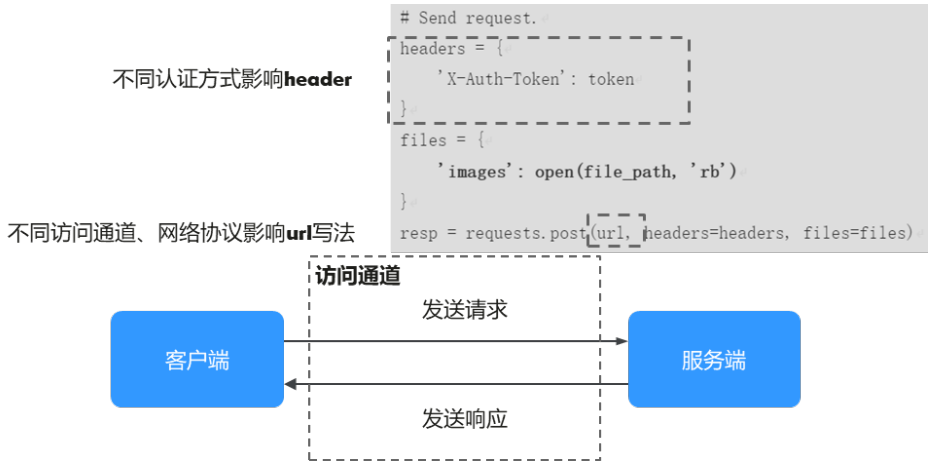
```
    device = torch.device('cuda')
    model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
else:
    device = torch.device('cpu')
    model.load_state_dict(torch.load(model_path, map_location=device))
# CPU或者GPU映射
model.to(device)
# 声明为推理模式
model.eval()

return model
```

9.4 将模型部署为实时推理作业

9.4.1 实时推理的部署及使用流程

在创建完模型后，可以将模型部署为一个在线服务。当在线服务的状态处于“运行中”，则表示在线服务已部署成功，部署成功的在线服务，将为用户提供一个可调用的API，此API为标准Restful API。访问在线服务时，您可以根据您的业务需求，分别确认使用何种认证方式、访问通道、传输协议，以上三个要素共同构成您的访问请求，三者可自由组合互不影响（例如不同的认证方式可以搭配不同的访问通道、不同的传输协议）。

图 9-9 认证方式、访问通道、传输协议



当前ModelArts支持访问在线服务的认证方式有以下方式（案例中均以HTTPS请求为例）：

- **Token认证**：Token具有时效性，有效期为24小时，需要使用同一个Token鉴权时，可以缓存起来，避免频繁调用。
- **AK/SK认证**：使用AK/SK对请求进行签名，在请求时将签名信息添加到消息头，从而通过身份认证。AK/SK签名认证方式仅支持消息体大小12M以内，12M以上的请求请使用Token认证。
- **APP认证**：在请求头部消息增加一个参数即可完成认证，认证方式简单，永久有效。

ModelArts支持通过以下几种方式调用API访问在线服务（案例中均以HTTPS请求为例）：

- **通过公网访问通道的方式访问在线服务**：ModelArts推理默认使用公网访问在线服务。在线服务部署成功后，将为用户提供一个可调用的API，此API为标准Restful API。
- **通过VPC高速访问通道的方式访问在线服务**：使用VPC直连的高速访问通道，用户的业务请求不需要经过推理平台，而是直接经VPC对等连接发送到实例处理，访问速度更快。

在线服务的API默认为HTTPS访问，同时还支持以下的传输协议：

- **使用WebSocket协议的方式访问在线服务**：WebSocket使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在WebSocket API中，浏览器和服务器只需要完成一次握手，两者之间就可以建立持久性的连接，并进行双向数据传输。
- **使用Server-Sent Events协议的方式访问在线服务**：Server-Sent Events访问主要解决了客户端与服务器之间的单向实时通信需求（例如ChatGPT回答的流式输出），相较于WebSocket（双向实时），它更加轻量级且易于实现。

9.4.2 部署模型为在线服务

模型准备完成后，您可以将模型部署为在线服务，对在线服务进行预测和调用。

约束与限制

单个用户最多可创建20个在线服务。

前提条件

- 数据已完成准备：已在ModelArts中创建状态“正常”可用的模型。
- 由于在线运行需消耗资源，确保账户未欠费。
- 部署服务操作需要镜像Owner拥有te_admin权限，否则部署服务过程中会报错：failed to set image shared, please check the agency permission。

操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型部署 > 在线服务”，默认进入“在线服务”列表。
2. 在“在线服务”列表中，单击左上角“部署”，进入“部署”页面。
3. 在“部署”页面，填写在线服务相关参数。
   a. 填写基本信息，详细参数说明请参见**表9-18**。

表 9-18 基本信息参数说明

参数名称	说明
“名称”	在线服务的名称，请按照界面提示规则填写。

参数名称	说明
“是否自动停止”	启用该参数并设置时间后，服务将在指定时间后自动停止。如果不启用此参数，在线服务将一直运行，同时一直收费，自动停止功能可以帮您避免产生不必要的费用。默认开启自动停止功能，且默认值为“1小时”。 目前支持设置为“1小时”、“2小时”、“4小时”、“6小时”、“自定义”。如果选择“自定义”的模式，可在右侧输入框中输入1~24范围内的任意整数。
“描述”	在线服务的简要说明。

b. 填写资源池和模型配置等关键信息，详情请参见**表9-19**。

表 9-19 参数说明

参数名称	子参数	说明
“资源池”	“公共资源池”	公共资源池有CPU或GPU两种规格，不同规格的资源池，其收费标准不同，详情请参见**产品价格详情**。当前仅支持按需付费模式。
	“专属资源池”	在专属资源池规格中选择对应的规格进行使用。暂不支持选择创建了逻辑子池的物理池。
“选择模型及配置”	“模型来源”	根据您的实际情况选择“自定义模型”或者“订阅模型”。
	“选择模型及版本”	选择状态“正常”的模型及版本。
	“分流”	设置当前实例节点的流量占比，服务调用请求根据该比例分配到当前版本上。 如您仅部署一个版本的模型，请设置为100%。如您添加多个版本进行灰度发布，多个版本分流之和设置为100%。
	“实例规格”	请根据界面显示的列表，选择可用的规格，置灰的规格表示当前环境无法使用。 如果公共资源池下规格为空数据，表示当前环境无公共资源。建议使用专属资源池。 **说明** 使用所选规格部署服务时，会产生必要的系统消耗，因此服务实际占用的资源会略大于该规格。
	“实例数”	设置当前版本模型的实例个数。如果实例数设置为1，表示后台的计算模式是单机模式；如果实例数设置大于1，表示后台的计算模式为分布式的。请根据实际编码情况选择计算模式。

参数名称	子参数	说明
	“环境变量”	设置环境变量，注入环境变量到容器实例。为确保您的数据安全，在环境变量中，请勿输入敏感信息，如明文密码。
	“部署超时时间”	用于设置单个模型实例的超时时间，包括部署和启动时间。默认值为20分钟，输入值必须在3到120之间。
	“添加模型版本进行灰度发布”	当选择的模型有多个版本时，您可以添加多个模型版本，并配置其分流占比，完成多版本和灵活流量策略的灰度发布，实现模型版本的平滑过渡升级。 **说明** 当前免费计算规格不支持多版本灰度发布。
	“存储挂载”	资源池为专属资源池时显示该参数。在服务运行时将存储卷以本地目录的方式挂载到计算节点（计算实例），模型或输入数据较大时建议使用。 SFS Turbo： ● 文件系统名称：选择对应的SFS Turbo极速文件。不支持选择跨区域（Region）的极速文件系统。 ● 挂载路径：指定容器内部的挂载路径，如“/sfs-turbo-mount/”。请选择全新目录，选择存量目录会覆盖存量文件。 **说明** ● 相同的文件系统只能挂载一次，且只能对应一个挂载路径，挂载路径均不可重复。最多可以挂载8个盘。 ● 使用专属资源池部署服务才允许使用存储挂载的能力，并且专属资源池需要打通VPC或关联SFS Turbo。 - 打通VPC为打通SFS Turbo所在VPC和专属资源池网络，打通步骤请见**打通VPC**章节。 - 关联SFS Turbo：如果SFS Turbo为HPC型的文件系统，可使用关联SFS Turbo功能。 ● 选择多挂载时请勿设置存在冲突的挂载路径如相同路径或相似路径如/obs-mount/与/obs-mount/tmp/等。 ● 选择SFS Turbo存储挂载后，请勿删除已经打通的VPC或解除SFS Turbo关联，否则会导致挂载功能无法使用。挂载时默认按客户端umask权限设置，为确保正常使用需在SFS Turbo界面绑定后端OBS存储后设置权限为777。
“服务流量限制”	-	服务流量限制是指每秒内一个服务能够被访问的次数上限。您可以根据实际需求设置每秒流量限制。

参数名称	子参数	说明
“升级为WebSocket”	-	设置在线服务是否部署为WebSocket服务。了解在线服务支持WebSocket，请参考**WebSocket在线服务全流程开发**。 **说明** • 要求模型的元模型来源为从容器镜像中选择，并且镜像支持WebSocket。 • 设置“升级为WebSocket”后，不支持设置“服务流量限制”。 • “升级为WebSocket”参数配置，不支持修改。
“支持APP认证”	“APP授权配置”	默认关闭。如需开启此功能，请参见**通过APP认证的方式访问在线服务**了解详情并根据实际情况进行设置。

c. 可选：配置高级选项。

表 9-20 高级选项参数说明

参数名称	说明
故障自动重启	开启该功能后，系统检测到在线服务异常，会自动重新部署在线服务。详细请参见**设置在线服务故障自动重启**。
“标签”	ModelArts支持对接标签管理服务TMS，在ModelArts中创建资源消耗性任务（例如：创建Notebook、训练作业、推理在线服务）时，可以为这些任务配置标签，通过标签实现资源的多维分组管理。 标签详细用法请参见**ModelArts如何通过标签实现资源分组管理**。 **说明** 可以在标签输入框下拉选择TMS预定义标签，也可以自己输入自定义标签。预定义标签对所有支持标签功能的服务资源可见。租户自定义标签只对自己服务可见。

4. 确认填写信息无误后，根据界面提示完成在线服务的部署。部署服务一般需要运行一段时间，根据您选择的数据量和资源不同，部署时间将耗时几分钟到几十分钟不等。

   说明

   在线服务部署完成后，将立即启动。

   您可以前往在线服务列表，查看在线服务的基本情况。在线服务列表中，刚部署的服务“状态”为“部署中”，当在线服务的“状态”变为“运行中”时，表示服务部署完成。

使用预测功能测试在线服务

模型部署为在线服务成功后，您可以在“预测”页签进行代码调试或添加文件测试。根据模型定义的输入请求不同（JSON文本或文件），测试服务包括如下两种方式：

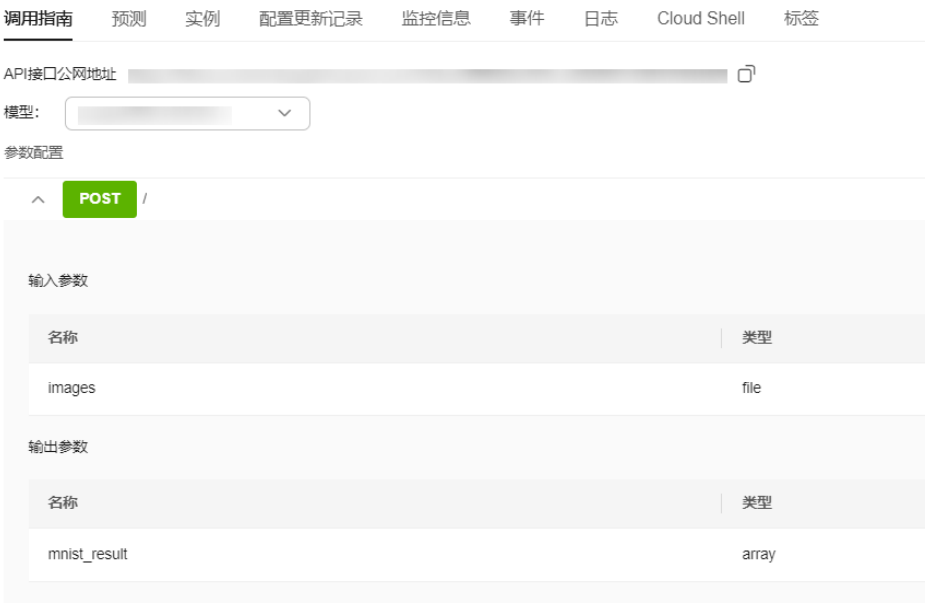
- **JSON文本预测**：如当前部署服务的模型，其输入类型指定的为JSON文本类，即不含有文件类型的输入，可以在“预测”页签输入JSON代码进行服务预测。
- **文件预测**：如当前部署服务的模型，其输入类型指定为文件类，可包含图片、音频或视频等场景，可以在“预测”页签添加图片进行服务预测。

说明

- 如果您的输入类型为图片，请注意测试服务单张图片输入应小于8MB。
- JSON文本预测，请求体的大小不超过8MB。
- 因APIG（API网关）的限制，单次预测的时间不能超过40S。
- 图片支持以下类型：“png”、“psd”、“jpg”、“jpeg”、“bmp”、“gif”、“webp”、“psd”、“svg”、“tiff”。
- 如果服务部署时使用的是“Ascend”规格，则无法预测含有透明度的PNG图片，因为Ascend仅支持RGB-3通道的图片。
- 该功能为调测使用，实际生产建议使用API调用。根据鉴权方式的不同，可以根据实际情况选择**通过Token认证的方式访问在线服务**、**通过AK/SK认证的方式访问在线服务**或者**通过APP认证的方式访问在线服务**。

针对您部署上线的服务，您可以在服务详情页面的“调用指南”中，了解本服务的输入参数，即上文提到的输入请求类型。

图 9-10 查看服务的调用指南



调用指南中的输入参数取决于您选择的模型来源：

- 如果您的元模型来源于自动学习或预置算法，其输入输出参数由ModelArts官方定义，请直接参考“调用指南”中的说明，并在预测页签中输入对应的JSON文本或文件进行服务测试。
- 如果您的元模型是自定义的，即推理代码和配置文件是自行编写的（**配置文件编写说明**），“调用指南”只是将您编写的配置文件进行了可视化展示。调用指南的输入参数与配置文件对应关系如下所示。

图 9-11 配置文件与调用指南的对应关系

不同输入请求的预测方式如下：

- **JSON文本预测**
  a. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署 > 在线服务”，进入“在线服务”管理页面。
  b. 单击目标服务名称，进入服务详情页面。在“预测”页签的预测代码下，输入预测代码，然后单击“预测”即可进行服务的预测。
- **文件预测**
  a. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署 > 在线服务”，进入“在线服务”管理页面。
  b. 单击目标服务名称，进入服务详情页面。在“预测”页签，单击“上传”，然后选择测试文件。文件上传成功后，单击“预测”即可进行服务的预测，如图9-12所示，输出标签名称，以及位置坐标和检测的评分。

图 9-12 图片预测

使用 CloudShell 调试在线服务实例容器

允许用户使用ModelArts控制台提供的CloudShell登录运行中在线服务实例容器。

约束限制：

- 只支持专属资源池部署的在线服务使用CloudShell访问容器。
- 在线服务必须处于“运行中”状态，才支持CloudShell访问容器。

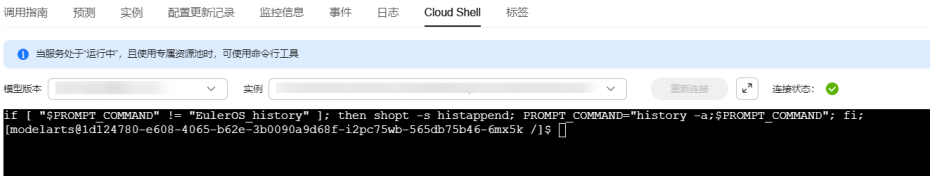
步骤1 登录ModelArts控制台，左侧菜单选择“模型部署 > 在线服务”。

步骤2 在线服务列表页面单击“名称/ID”，进入在线服务详情页面。

步骤3 单击CloudShell页签，选择模型版本和计算节点，当连接状态变为 时，即登录实例容器成功。

如果遇到异常情况服务器主动断开或超过10分钟未操作自动断开，此时可单击“重新连接”重新登录实例容器。

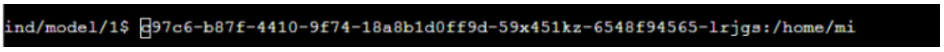
图 9-13 CloudShell 界面



说明

部分用户登录Cloud Shell界面时，可能会出现路径显示异常情况，此时在Cloud Shell中单击回车键即可恢复正常。

图 9-14 路径异常



----结束

9.4.3 访问在线服务支持的认证方式

9.4.3.1 通过 Token 认证的方式访问在线服务

如果在线服务的状态处于“运行中”，则表示在线服务已部署成功，部署成功的在线服务，将为用户提供一个可调用的API，此API为标准Restful API。在集成至生产环境之前，需要对此API进行调测，您可以使用以下方式向在线服务发起预测请求：

- **方式一：使用图形界面的软件进行预测（以Postman为例）**。Windows系统建议使用Postman。
- **方式二：使用curl命令发送预测请求**。Linux系统建议使用curl命令。
- **方式三：使用Python语言发送预测请求**。
- **方式四：使用Java语言发送预测请求**。

约束限制

调用API访问在线服务时，对预测请求体大小和预测时间有限制：

- 请求体的大小不超过12MB，超过后请求会被拦截。
- 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。

前提条件

已经获取**用户Token**、**预测文件的本地路径**、**在线服务的调用地址**和**在线服务的输入参数**信息。

- 用户Token的获取请参见**获取Token认证**。获取Token认证时，由于ModelArts生成的在线服务API不支持domain范围的token，因此需获取使用范围为project的Token信息，即scope参数的取值为project。
- 预测文件的本地路径既可使用绝对路径（如Windows格式"D:/test.png"，Linux格式"/opt/data/test.png"），也可以使用相对路径（如"./test.png"）。
- 在线服务的调用地址和输入参数信息，可以在控制台的“在线服务详情 > 调用指南”页面获取。

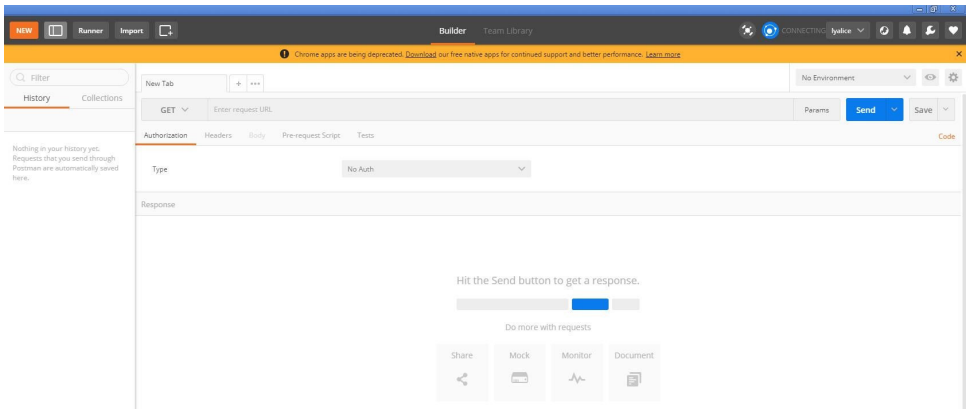
  “API接口公网地址”即在线服务的调用地址。当模型配置文件中apis定义了路径，调用地址后需拼接自定义路径。如：“{在线服务的调用地址}/predictions/poetry”。

图 9-15 获取在线服务 API 接口地址和文件预测输入参数信息

方式一：使用图形界面的软件进行预测（以 Postman 为例）

1. 下载Postman软件并安装，您也可以直接在Chrome浏览器添加Postman扩展程序（也可使用其他支持发送post请求的软件）。Postman推荐使用7.24.0版本。
2. 打开Postman，如**图9-16**所示。

图 9-16 Postman 界面

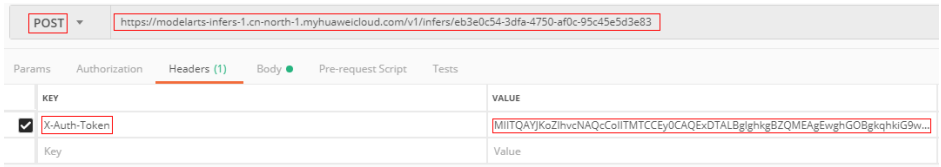


3. 在Postman界面填写参数，以图像分类举例说明。
   - 选择POST任务，将在线服务的调用地址复制到POST后面的方框。Headers页签的Key值填写为“X-Auth-Token”，Value值为用户Token。

说明

您也可以通过AK（Access Key ID）/SK（Secret Access Key)加密调用请求，具体可参见**用户AK-SK认证模式**。

图 9-17 参数填写

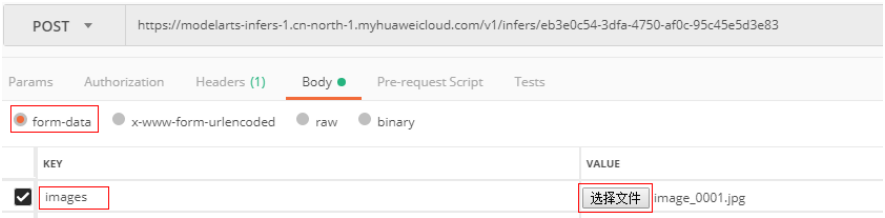


- 在Body页签，根据模型的输入参数不同，可分为2种类型：文件输入、文本输入。

  - **文件输入**

    选择“form-data”。在“KEY”值填写模型的入参，和在线服务的输入参数对应，比如本例中预测图片的参数为“images”。然后在“VALUE”值，选择文件，上传一张待预测图片（当前仅支持单张图片预测），如**图9-18**所示。

    图 9-18 填写 Body

  - **文本输入**

    选择“raw”，选择JSON(application/json)类型，在下方文本框中填写请求体，请求体样例如下：

    ```
    {
     "meta": {
       "uuid": "10eb0091-887f-4839-9929-cbc884f1e20e"
     },
     "data": {
       "req_data": [
         {
           "sepal_length": 3,
           "sepal_width": 1,
           "petal_length": 2.2,
           "petal_width": 4
         }
       ]
     }
    }
    ```

    其中，“meta”中可携带“uuid”，调用时传入一个“uuid”，返回预测结果时回传此“uuid”用于跟踪请求，如无此需要可不填写meta。“data”包含了一个“req_data”的数组，可传入单条或多条请求数据，其中每个数据的参数由模型决定，比如本例中的“sepal_length”、“sepal_width”等。

4. 参数填写完成，单击“send”发送请求，结果会在“Response”下的对话框里显示。

- 文件输入形式的预测结果样例如图9-19所示，返回结果的字段值根据不同模型可能有所不同。
- 文本输入形式的预测结果样例如图9-20所示，请求体包含“meta”及“data”。如输入请求中包含“uuid”，则输出结果中回传此“uuid”。如未输入，则为空。“data”包含了一个“resp_data”的数组，返回单条或多条输入数据的预测结果，其中每个结果的参数由模型决定，比如本例中的“sepal_length”、“predictresult”等。

图 9-19 文件输入预测结果

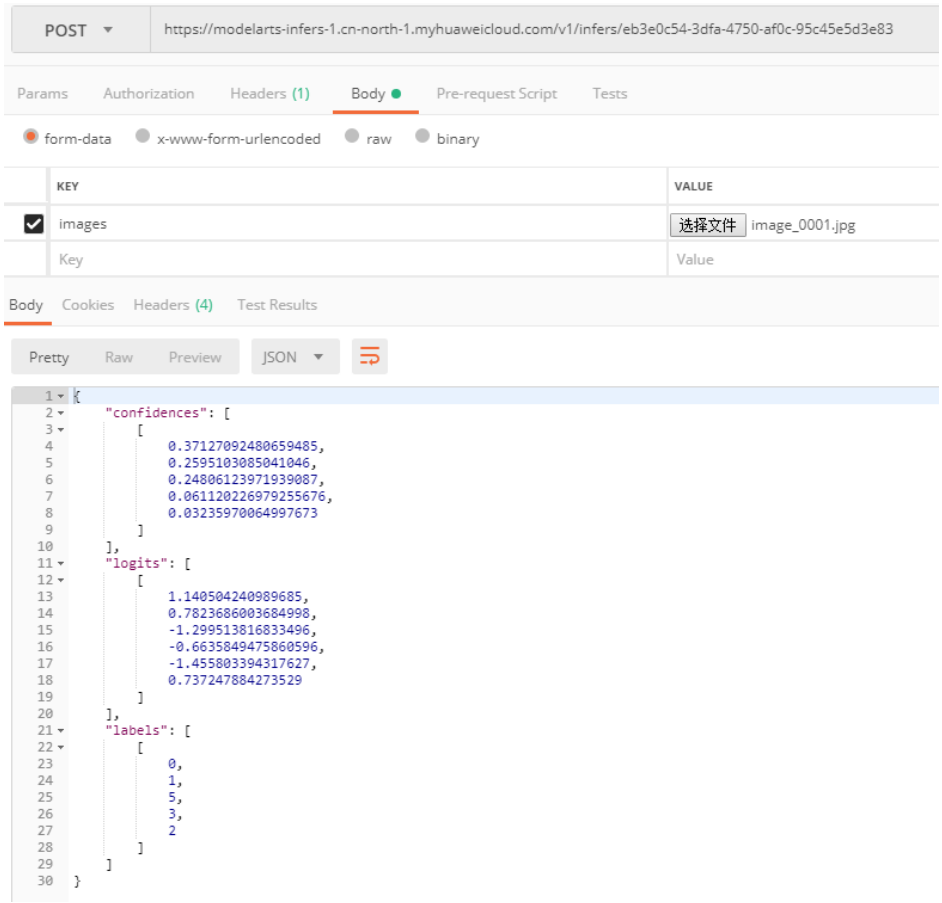
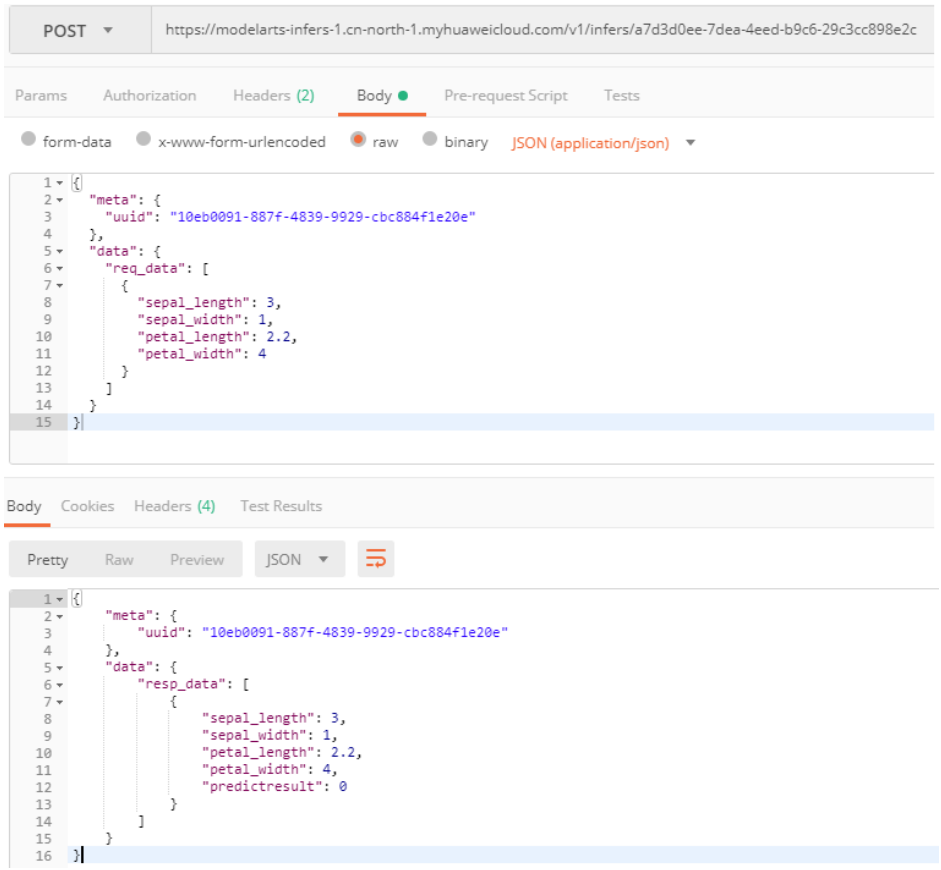


图 9-20 文本输入预测结果



方式二：使用 curl 命令发送预测请求

使用curl命令发送预测请求的命令格式也分为文件输入、文本输入两类。

- 文件输入
  curl -kv -F '**images**=@**图片路径**' -H '**X-Auth-Token**:**Token值**' -X POST **在线服务地址**
  - “-k”是指允许不使用证书到SSL站点。
  - “-F”是指上传数据的是文件，本例中参数名为“images”，这个名字可以根据具体情况变化，@后面是图片的存储路径。
  - “-H”是post命令的headers，Headers的Key值为“X-Auth-Token”，这个名字为固定的， Token值是获取的用户Token。
  - “POST”后面跟随的是在线服务的调用地址。

  curl命令文件输入样例：

  ```
  curl -kv -F 'images=@/home/data/test.png' -H 'X-Auth-Token:MIISkAY***80T9wHQ==' -X POST https://modelarts-infers-1.xxx/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
  ```

- 文本输入
  ```
  curl -kv -d '{"data":{"req_data":[{"sepal_length":3,"sepal_width":1,"petal_length":2.2,"petal_width":4}]}}' -H 'X-Auth-Token:MIISkAY***80T9wHQ==' -H 'Content-type: application/json' -X POST https://modelarts-infers-1.xxx/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
  ```

  “-d”是Body体的文本内容。

方式三：使用 Python 语言发送预测请求

1. 下载Python SDK并在开发工具中完成SDK配置。具体操作请参见**在Python环境中集成API请求签名的SDK**。
2. 创建请求体，进行预测请求。
   - **输入为文件格式**

     ```
     # coding=utf-8

     import requests

     if __name__ == '__main__':
         # Config url, token and file path.
         url = "在线服务的调用地址"
         token = "用户Token"
         file_path = "预测文件的本地路径"

         # Send request.
         headers = {
             'X-Auth-Token': token
         }
         files = {
             'images': open(file_path, 'rb')
         }
         resp = requests.post(url, headers=headers, files=files)

         # Print result.
         print(resp.status_code)
         print(resp.text)
     ```

     “files”中的参数名由在线服务的输入参数决定，需要和“类型”为“file”的输入参数“名称”保持一致。以**前提条件**里获取的文件预测输入参数“images”为例。
   - **输入为文本格式（json类型）**

     读取本地预测文件并进行base64编码的请求体示例如下：

     ```
     # coding=utf-8

     import base64
     import requests

     if __name__ == '__main__':
         # Config url, token and file path
         url = "在线服务的调用地址"
         token = "用户Token"
         file_path = "预测文件的本地路径"
         with open(file_path, "rb") as file:
             base64_data = base64.b64encode(file.read()).decode("utf-8")

         # Set body,then send request
         headers = {
             'Content-Type': 'application/json',
             'X-Auth-Token': token
         }
         body = {
             'image': base64_data
         }
         resp = requests.post(url, headers=headers, json=body)

         # Print result
         print(resp.status_code)
         print(resp.text)
     ```

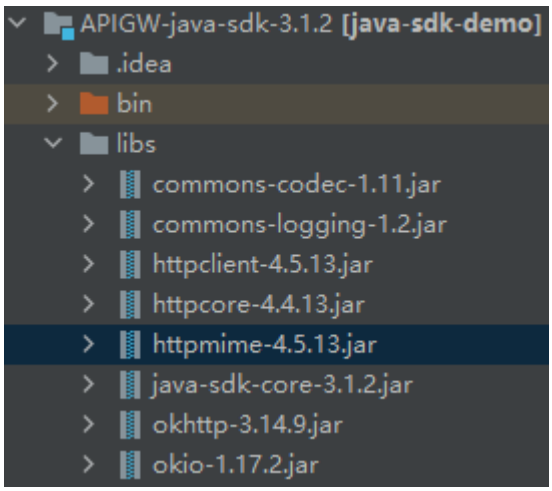
     “body”中的参数名由在线服务的输入参数决定，需要和“类型”为“string”的输入参数“名称”保持一致。以**前提条件**里获取的文本预测输入参数“image”为例。“body”中的base64_data值为string类型。

方式四：使用 Java 语言发送预测请求

1. 下载Java SDK并在开发工具中完成SDK配置。具体操作请参见**在Java环境中集成API请求签名的SDK**。
2. （可选）当预测请求的输入为文件格式时，Java工程依赖httpmime模块。
   a. 在工程“libs”中增加httpmime-x.x.x.jar。完整的Java依赖库如**图9-21**所示。

      httpmime-x.x.x.jar建议使用4.5及以上版本，下载地址：**https://mvnrepository.com/artifact/org.apache.httpcomponents/httpmime**。

      图 9-21 Java 依赖库

   b. httpmime-x.x.x.jar添加完成后，在Java工程的.classpath文件中，补充httpmime信息，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
<classpathentry kind="src" path="src"/>
<classpathentry kind="lib" path="libs/commons-codec-1.11.jar"/>
<classpathentry kind="lib" path="libs/commons-logging-1.2.jar"/>
<classpathentry kind="lib" path="libs/httpclient-4.5.13.jar"/>
<classpathentry kind="lib" path="libs/httpcore-4.4.13.jar"/>
<classpathentry kind="lib" path="libs/httpmime-x.x.x.jar"/>
<classpathentry kind="lib" path="libs/java-sdk-core-3.1.2.jar"/>
<classpathentry kind="lib" path="libs/okhttp-3.14.9.jar"/>
<classpathentry kind="lib" path="libs/okio-1.17.2.jar"/>
<classpathentry kind="output" path="bin"/>
</classpath>
```

3. 创建Java类，进行预测请求。
   - **输入为文件格式**

     Java的请求体示例如下：

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.Consts;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
```

```
import java.io.File;

public class MyTokenFile {

   public static void main(String[] args) {
      // Config url, token and filePath
      String url = "在线服务的调用地址";
      String token = "用户Token";
      String filePath = "预测文件的本地路径";

      try {
         // Create post
         HttpPost httpPost = new HttpPost(url);

         // Add header parameters
         httpPost.setHeader("X-Auth-Token", token);

         // Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
         File file = new File(filePath);
         HttpEntity entity = MultipartEntityBuilder.create().addBinaryBody("images", file).setContentType(ContentType.MULTIPART_FORM_DATA).setCharset(Consts.UTF_8).build();
         httpPost.setEntity(entity);

         // Send post
         CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

         // Print result
         System.out.println(response.getStatusLine().getStatusCode());
         System.out.println(EntityUtils.toString(response.getEntity()));
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

“addBinaryBody”中的参数名由在线服务的输入参数决定，需要和“类型”为“file”的输入参数“名称”保持一致。此处以**前提条件**里获取的“images”为例。

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyTokenTest {

   public static void main(String[] args) {
      // Config url, token and body
      String url = "在线服务的调用地址";
      String token = "用户Token";
      String body = "{}";

      try {
         // Create post
         HttpPost httpPost = new HttpPost(url);

         // Add header parameters
         httpPost.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
         httpPost.setHeader("X-Auth-Token", token);

         // Special characters, such as the double quotation mark ("), contained in the body
```

```
must be escaped.
            httpPost.setEntity(new StringEntity(body));

            // Send post.
            CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

“body”由具体文本格式决定，此处以json为例。

9.4.3.2 通过 AK/SK 认证的方式访问在线服务

如果在线服务的状态处于“运行中”，则表示在线服务已部署成功。部署成功的在线服务，将为用户提供一个可调用的API，此API为标准Restful API。用户可以通过AK/SK签名认证方式调用API。

使用AK/SK认证时，您可以通过APIG SDK访问，也可以通过ModelArts SDK访问。使用ModelArts SDK访问参见**用户AK-SK认证模式**。本文档详细介绍如何通过APIG SDK访问在线服务，具体操作流程如下：

1. **获取AK/SK**
2. **获取在线服务信息**
3. 发送预测请求
   - **方式一：使用Python语言发送预测请求**
   - **方式二：使用Java语言发送预测请求**

说明

1. AK/SK签名认证方式，仅支持Body体12M以内，12M以上的请求，需使用Token认证。
2. 客户端须注意本地时间与时钟服务器的同步，避免请求消息头X-Sdk-Date的值出现较大误差。因为API网关除了校验时间格式外，还会校验该时间值与网关收到请求的时间差，如果时间差超过15分钟，API网关将拒绝请求。

约束限制

调用API访问在线服务时，对预测请求体大小和预测时间有限制：

- 请求体的大小不超过12MB，超过后请求会被拦截。
- 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。

获取 AK/SK

如果已生成过AK/SK，则可跳过此步骤，找到原来已下载的AK/SK文件，文件名一般为：credentials.csv。

如下图所示，文件包含了租户名（User Name），AK（Access Key Id），SK（Secret Access Key）。

图 9-22 credential.csv 文件内容

	A	B	C
1	User Name	Access Key Id	Secret Access Key
2	hu[redacted]dg	QTWA[redacted]UT2QVKYUC	MFyfvK41ba2[redacted]npdUKGpownRZlmVmHc

AK/SK生成步骤：

1. 注册并登录管理控制台。
2. 单击右上角的用户名，在下拉列表中单击“我的凭证”。
3. 单击“访问密钥”。
4. 单击“新增访问密钥”，进入“身份验证”页面。
5. 根据提示完成身份验证，下载密钥，并妥善保管。

获取在线服务信息

在调用接口时，需获取在线服务的调用地址，以及在线服务的输入参数信息。步骤如下：

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型部署 > 在线服务”，默认进入“在线服务”列表。
2. 单击目标服务名称，进入服务详情页面。
3. 在“在线服务”的详情页面，可以获取该服务的调用地址和输入参数信息。
   “API接口公网地址”即在线服务的调用地址。当模型配置文件中apis定义了路径，调用地址后需拼接自定义路径。如：“{在线服务的调用地址}/predictions/poetry”。

图 9-23 获取在线服务 API 接口地址和文件预测输入参数信息

方式一：使用 Python 语言发送预测请求

1. 下载Python SDK并在开发工具中完成SDK配置。具体操作请参见**在Python环境中集成API请求签名的SDK**。
2. 创建请求体，进行预测请求。
   - **输入为文件格式**

```
# coding=utf-8

import requests
import os
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "在线服务的调用地址"
    # 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
```

```
    # 本示例以ak和sk保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    ak = os.environ["HUAWEICLOUD_SDK_AK"]
    sk = os.environ["HUAWEICLOUD_SDK_SK"]
    file_path = "预测文件的本地路径"

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {"x-sdk-content-sha256": "UNSIGNED-PAYLOAD"}
    request = signer.HttpRequest(method, url, headers)

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = ak
    sig.Secret = sk
    sig.Sign(request)

    # Send request
    files = {'images': open(file_path, 'rb')}
    resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri,
headers=request.headers, files=files)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

“file_path”为预测文件的本地路径，既可使用绝对路径（如Windows格式"D:/test.png"，Linux格式"/opt/data/test.png"），也可以使用相对路径（如"./test.png"）。

“files”参数的请求体样式为“files={"请求参数":("文件路径"，文件内容，“文件类型”)}”，参数填写可以参考**表9-21**。

表 9-21 files 参数说明

参数	是否必填	说明
请求参数	是	在线服务输入参数名称。
文件路径	否	上传文件的路径。
文件内容	是	上传文件的内容。
文件类型	否	上传文件类型。当前支持以下类型： • txt类型：text/plain • jpg/jpeg类型：image/jpeg • png类型：image/png

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
# coding=utf-8

import base64
import json
import os
import requests
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "在线服务的调用地址"
```

```
    # 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以ak和sk保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    ak = os.environ["HUAWEICLOUD_SDK_AK"]
    sk = os.environ["HUAWEICLOUD_SDK_SK"]
    file_path = "预测文件的本地路径"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {
        'Content-Type': 'application/json'
    }
    body = {
        'image': base64_data
    }
    request = signer.HttpRequest(method, url, headers, json.dumps(body))

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = ak
    sig.Secret = sk
    sig.Sign(request)

    # Send request
    resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri,
headers=request.headers, data=request.body)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

“body”中的参数名由在线服务的输入参数决定，需要和“类型”为“string”的输入参数“名称”保持一致。此处以“image”为例。“body”中的base64_data值为string类型。

方式二：使用 Java 语言发送预测请求

1. 下载Java SDK并在开发工具中完成SDK配置。
2. 创建Java类，进行预测请求。

   由于在APIG的Java SDK中，“request.setBody()”只支持String类型，所以只支持输入为文本格式的预测请求。如果输入的是文件格式，需要先进行base64编码转换成文本。

   - **输入为文件格式**

     此处以json格式为例介绍读取本地预测文件并进行base64编码的请求体，请求体示例如下：

```
package com.apig.sdk.demo;
import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.commons.codec.binary.Base64;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
public class MyAkSkTest2 {
    public static void main(String[] args) {
        String url = "在线服务的调用地址";
       // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环
```



```
境变量中密文存放，使用时解密，确保安全；
      // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");
        String filePath = "预测文件的本地路径";
        try {
            // Create request
            Request request = new Request();
            // Set the AK/SK to sign and authenticate the request.
            request.setKey(ak);
            request.setSecret(sk);
            // Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
            request.setMethod(HttpPost.METHOD_NAME);
            // Add header parameters
            request.addHeader(HttpHeaders.CONTENT_TYPE, "application/json");
            // Set a request URL in the format of https://{Endpoint}/{URI}.
            request.setUrl(url);
            // build your json body
            String body = "{\"image\":\"" + getBase64FromFile(filePath) + "\"}";
            // Special characters, such as the double quotation mark ("), contained in the body must be escaped.
            request.setBody(body);
            // Sign the request.
            HttpRequestBase signedRequest = Client.sign(request);
            // Send request.
            CloseableHttpResponse response = HttpClients.createDefault().execute(signedRequest);
            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * Convert the file into a byte array and Base64 encode it
     * @return
     */
    private static String getBase64FromFile(String filePath) {
        // Convert the file into a byte array
        InputStream in = null;
        byte[] data = null;
        try {
            in = new FileInputStream(filePath);
            data = new byte[in.available()];
            in.read(data);
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Base64 encode
        return new String(Base64.encodeBase64(data));
    }
}
```

注意

使用base64编码方式，需要在模型推理代码中增加对请求体解码的代码。

- **输入为文本格式（json类型）**

```
// Package name of the demo.
package com.apig.sdk.demo;

import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
```

```
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAkSkTest {

    public static void main(String[] args) {
        String url = "在线服务的调用地址";
       // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
       // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");

        try {
            // Create request
            Request request = new Request();

            // Set the AK/SK to sign and authenticate the request.
            request.setKey(ak);
            request.setSecret(sk);

            // Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
            request.setMethod(HttpPost.METHOD_NAME);

            // Add header parameters
            request.addHeader(HttpHeaders.CONTENT_TYPE, "application/json");

            // Set a request URL in the format of https://{Endpoint}/{URI}.
            request.setUrl(url);

            // Special characters, such as the double quotation mark ("), contained in the body must be escaped.
            String body = "{}";
            request.setBody(body);

            // Sign the request.
            HttpRequestBase signedRequest = Client.sign(request);

            // Send request.
            CloseableHttpResponse response = HttpClients.createDefault().execute(signedRequest);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

“body”由具体文本格式决定，此处以json为例。

9.4.3.3 通过 APP 认证的方式访问在线服务

部署在线服务支持开启APP认证，即ModelArts会为服务注册一个支持APP认证的接口，为此接口配置APP授权后，用户可以使用授权应用的AppKey+AppSecret或AppCode调用该接口。

针对在线服务的APP认证，具体操作流程如下。

1. **开启支持APP认证功能**：开启支持APP认证功能，选择已有APP应用或者创建新的APP应用。
2. **在线服务授权管理**：对创建的APP应用进行管理，包括查看、重置或删除应用，绑定或解绑应用对应的在线服务，获取“AppKey/AppSecret”或“AppCode”。

3. **APP认证鉴权**：调用支持APP认证的接口需要进行认证鉴权，支持两种鉴权方式（AppKey+AppSecret或AppCode），您可以选择其中一种进行认证鉴权。
4. 发送预测请求：
   - **方式一：使用Python语言通过AppKey+AppSecret认证鉴权方式发送预测请求**
   - **方式二：使用Java语言通过AppKey+AppSecret认证鉴权方式发送预测请求**
   - **方式三：使用Python语言通过AppCode认证鉴权方式发送预测请求**
   - **方式四：使用Java语言通过AppCode认证鉴权方式发送预测请求**

约束限制

调用API访问在线服务时，对预测请求体大小和预测时间有限制：

- 请求体的大小不超过12MB，超过后请求会被拦截。
- 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。

前提条件

- 数据已完成准备：已在ModelArts中创建状态“正常”可用的模型。
- 由于在线运行需消耗资源，确保账户未欠费。
- 已获取预测文件的本地路径，可使用绝对路径（如Windows格式"D:/test.png"，Linux格式"/opt/data/test.png"）或相对路径（如"./test.png"）。

开启支持 APP 认证功能

在部署为在线服务时，您可以开启支持APP认证功能。或者针对已部署完成的在线服务，您可以修改服务，开启支持APP认证功能。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署 > 在线服务”，进入在线服务管理页面。
2. 开启支持APP认证功能。
   - 在部署为在线服务时，即“部署”页面，填写部署服务相关参数时，开启支持APP认证功能。
   - 针对已部署完成的在线服务，进入在线服务管理页面，单击目标服务名称“操作”列的“修改”按钮，进入修改服务页面开启支持APP认证功能。

     图 9-24 部署页面开启支持 APP 认证功能

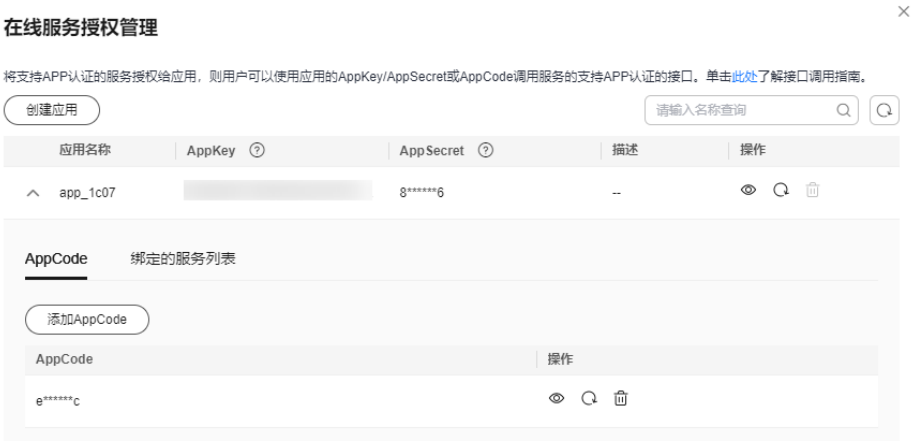
3. 选择APP授权配置。从下拉列表中选择您需要配置的APP应用，如果没有可选项，您可以通过如下方式创建应用。
   - 单击右侧“创建应用”，填写应用名称和描述之后单击“确定”完成创建。其中应用名称默认以“app_”开头，您也可以自行修改。
   - 进入“模型部署 > 在线服务”页面，单击“授权管理”，进入“在线服务授权管理”页面，选择“创建应用”，详情请参见**在线服务授权管理**。
4. 开启支持APP认证功能后，将支持APP认证的服务授权给应用，用户可以使用创建的“AppKey/AppSecret”或“AppCode”调用服务的支持APP认证的接口。

APP认证的服务授权给应用后，需要1-2分钟生效。

在线服务授权管理

如果您需要使用支持APP认证功能，建议您在部署在线服务之前进行授权管理操作完成应用创建。进入“模型部署 > 在线服务”页面，单击“授权管理”，进入“在线服务授权管理”对话框。在此页面您可以实现应用的创建和管理，包括查询明文、重置或删除应用，解绑应用对应的在线服务，获取“AppKey/AppSecret”或“AppCode”。

图 9-25 在线服务授权管理



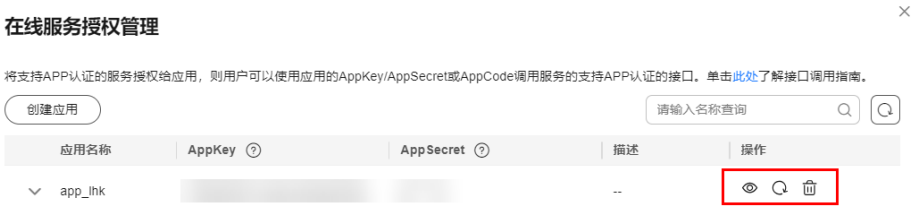
- **创建应用**

  选择“创建应用”，填写应用名称和描述之后单击“确定”完成创建。其中应用名称默认以“app_”开头，您也可以自行修改。

- **查看、重置或删除应用**

  您可以单击目标应用名称操作列的按钮完成应用的查询明文、重置或删除。创建完成后自动生成“AppKey/AppSecret”以供您后续调取接口进行APP鉴权使用。

  图 9-26 查询明文、重置或删除

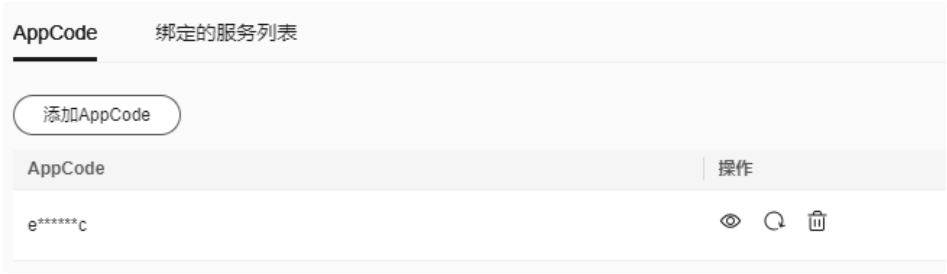
- **解绑服务**

  您可以单击目标应用名称前方的，在下拉列表中展示绑定的服务列表，即该应用对应的在线服务列表。单击操作列的“解绑”取消绑定，将不再支持调用该接口。

- **获取AppKey/AppSecret或AppCode**

  调用接口需要进行APP鉴权，在创建APP应用时自动生成“AppKey/AppSecret”，您可以在“在线服务授权管理”对话框中单击APP应用操作列的 查看完整的**AppSecret**。单击应用名称前方的展开下拉列表，通过单击“添加AppCode”自动生成“AppCode”，您可以单击操作列的 查看完整的**AppCode**。

图 9-27 添加 AppCode



APP 认证鉴权

当支持APP认证功能的在线服务运行成功处于“运行中”状态，就可以对服务进行调用 。在调用之前您需要进行APP认证鉴权。

当使用APP认证，且开启了简易认证模式，API请求既可以选择使用Appkey和AppSecret做签名和校验，也可以选择使用AppCode进行简易认证（ModelArts默认启用简易认证）。推荐使用AppKey/AppSecret认证，其安全性比AppCode认证要高。

- **AppKey/AppSecret认证**：通过AppKey与AppSecret对请求进行加密签名，可标识发送方并防止请求被修改。使用AppKey/AppSecret认证时，您需要使用专门的签名SDK对请求进行签名。
  - AppKey：APP访问密钥ID。与私有访问密钥关联的唯一标识符；访问密钥ID和私有访问密钥一起使用，对请求进行加密签名。
  - AppSecret：APP私有访问密钥，即与访问密钥ID结合使用的密钥，对请求进行加密签名，可标识发送方，并防止请求被修改。

  AppKey进行简易认证时，即在调用API的时候，在HTTP请求头部消息增加一个参数“apikey”（参数值为“AppKey”），实现快速认证。
- **AppCode认证**：通过AppCode认证通用请求。

  AppCode认证就是在调用API的时候，在HTTP请求头部消息增加一个参数“X-Apig-AppCode”（参数值为“AppCode”），而不需要对请求内容签名，API网关也仅校验AppCode，不校验请求签名，从而实现快速响应。

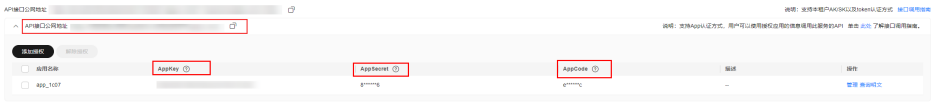
您可以在服务详情页的“调用指南”页签（如图9-28）获取API接口公网地址（对应下文示例中的在线服务的调用地址url）和AppKey/AppSecret（对应下文示例中的app_key、app_secret）和AppCode（对应下文示例中的app_code）。请注意使用图中第二行用于APP认证方式的API接口公网地址。

以下情况下需要对API接口公网地址进行拼接修改：

- 当模型配置文件中apis定义了路径，调用地址后需拼接自定义路径。如：“{在线服务的调用地址}/predictions/poetry”。

  如果是部署SD WebUI推理服务，调用地址后需添加"/"。如：“https://8e******5fe.apig.******.huaweicloudapis.com/v1/infers/f2682******f42/”。

图 9-28 获取 APP 认证鉴权相关信息



方式一：使用 Python 语言通过 AppKey+AppSecret 认证鉴权方式发送预测请求

1. 下载Python SDK并在开发工具中完成SDK配置。
2. 创建请求体，进行预测请求。
   - **输入为文件格式**

```
# coding=utf-8

import requests
import os
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    # API接口公网地址，例如"https://8e******5fe.apig.******.huaweicloudapis.com/v1/infers/f2682******f42",对应图9-28中的在线服务的调用地址url
    url = "在线服务的调用地址"
    # 认证用的app_key和app_secret硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以app_key和app_secret保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_KEY和HUAWEICLOUD_APP_SECRET。
    app_key = os.environ["HUAWEICLOUD_APP_KEY"]
    app_secret= os.environ["HUAWEICLOUD_APP_SECRET"]
    file_path = "预测文件的本地路径"

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {"x-sdk-content-sha256": "UNSIGNED-PAYLOAD"}
    request = signer.HttpRequest(method, url, headers)

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = app_key
    sig.Secret = app_secret
    sig.Sign(request)

    # Send request
    files = {'images': open(file_path, 'rb')}
    resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri, headers=request.headers, files=files)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

“files”参数的请求体样式为“files={"请求参数":("文件路径"，文件内容，“文件类型”)}”，参数填写可以参考表9-22。

表 9-22 files 参数说明

参数	是否必填	说明
请求参数	是	在线服务输入参数名称。
文件路径	否	上传文件的路径。
文件内容	是	上传文件的内容。
文件类型	否	上传文件类型。当前支持以下类型： • txt类型：text/plain • jpg/jpeg类型：image/jpeg • png类型：image/png

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
# coding=utf-8

import base64
import json
import os
import requests
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    # API接口公网地址，例如"https://8e******5fe.apig.******.huaweicloudapis.com/v1/infers/f2682******f42"
    url = "在线服务的调用地址"
    # 认证用的app_key和app_secret硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以app_key和app_secret保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_KEY和HUAWEICLOUD_APP_SECRET。
    app_key = os.environ["HUAWEICLOUD_APP_KEY"]
    app_secret= os.environ["HUAWEICLOUD_APP_SECRET"]
    file_path = "预测文件的本地路径"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {
        'Content-Type': 'application/json'
    }
    body = {
        'image': base64_data
    }
    request = signer.HttpRequest(method, url, headers, json.dumps(body))

    # Create sign, set the AppKey&AppSecret to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = app_key
    sig.Secret = app_secret
    sig.Sign(request)

    # Send request
    resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri, headers=request.headers, data=request.body)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

  “body”中的参数名由在线服务的输入参数决定，需要和“类型”为“string”的输入参数“名称”保持一致。此处以“image”为例。“body”中的base64_data值为string类型。

方式二：使用 Java 语言通过 AppKey+AppSecret 认证鉴权方式发送预测请求

1. 下载Java SDK并在开发工具中完成SDK配置。
2. 创建Java类，进行预测请求。

   由于在APIG的Java SDK中，“request.setBody()”只支持String类型，所以只支持输入为文本格式的预测请求。

   此处以json格式为例介绍读取本地预测文件并进行base64编码的请求体：

```
// Package name of the demo.
package com.apig.sdk.demo;
```



```
import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAkSkTest {

    public static void main(String[] args) {
        # API接口公网地址，例如"https://8e******5fe.apig.******.huaweicloudapis.com/v1/infers/f2682******f42"
        String url = "在线服务的调用地址";
        // 认证用的appKey和appSecret硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
        // 本示例以appKey和appSecret保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_KEY和HUAWEICLOUD_APP_SECRET。
        String appKey = System.getenv("HUAWEICLOUD_APP_KEY");
        String appSecret = System.getenv("HUAWEICLOUD_APP_SECRET");
        String body = "{}";

        try {
            // Create request
            Request request = new Request();

            // Set the AK/AppSecret to sign and authenticate the request.
            request.setKey(appKey);
            request.setSecret(appSecret);

            // Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
            request.setMethod(HttpPost.METHOD_NAME);

            // Add header parameters
            request.addHeader(HttpHeaders.CONTENT_TYPE, "application/json");

            // Set a request URL in the format of https://{Endpoint}/{URI}.
            request.setUrl(url);

            // Special characters, such as the double quotation mark ("), contained in the body must be escaped.
            request.setBody(body);

            // Sign the request.
            HttpRequestBase signedRequest = Client.sign(request);

            // Send request.
            CloseableHttpResponse response = HttpClients.createDefault().execute(signedRequest);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

“body”由具体文本格式决定，此处以json为例。

方式三：使用 Python 语言通过 AppCode 认证鉴权方式发送预测请求

1. 下载Python SDK并在开发工具中完成SDK配置。
2. 创建请求体，进行预测请求。
   - **输入为文件格式**


```
# coding=utf-8

import requests
import os

if __name__ == '__main__':
    # Config url, app code and file path.
    # API接口公网地址，例如"https://8e******5fe.apig.******.huaweicloudapis.com/v1/infers/f2682******f42"
    url = "在线服务的调用地址"
    # 认证用的app_code硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以app_code保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_CODE。
    app_code = os.environ["HUAWEICLOUD_APP_CODE"]
    file_path = "预测文件的本地路径"

    # Send request.
    headers = {
        'X-Apig-AppCode': app_code
    }
    files = {
        'images': open(file_path, 'rb')
    }
    resp = requests.post(url, headers=headers, files=files)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

“files”中的参数名由在线服务的输入参数决定，需要和“类型”为“file”的输入参数“名称”保持一致。此处以“images”为例。

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
# coding=utf-8

import base64
import requests
import os

if __name__ == '__main__':
    # Config url, app code and request body.
    # API接口公网地址，例如"https://8e******5fe.apig.******.huaweicloudapis.com/v1/infers/f2682******f42"
    url = "在线服务的调用地址"
    # 认证用的app_code硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以app_code保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_CODE。
    app_code = os.environ["HUAWEICLOUD_APP_CODE"]
    file_path = "预测文件的本地路径"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Send request
    headers = {
        'Content-Type': 'application/json',
        'X-Apig-AppCode': app_code
    }
    body = {
        'image': base64_data
    }
    resp = requests.post(url, headers=headers, json=body)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

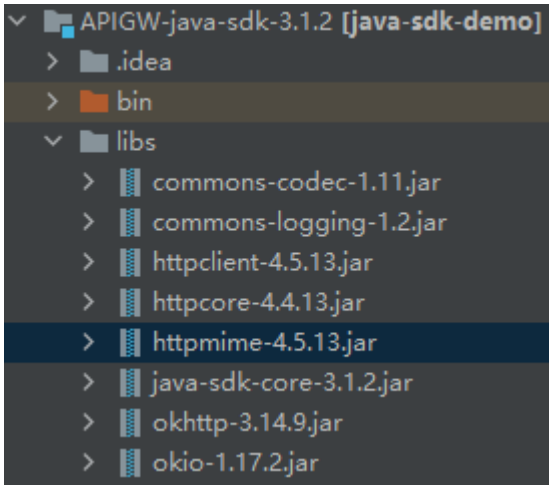
“body”中的参数名由在线服务的输入参数决定，需要和“类型”为“string”的输入参数“名称”保持一致。此处以“image”为例。“body”中的base64_data值为string类型。

方式四：使用 Java 语言通过 AppCode 认证鉴权方式发送预测请求

1. 下载Java SDK并在开发工具中完成SDK配置。
2. （可选）当预测请求的输入为文件格式时，Java工程依赖httpmime模块。
   a. 在工程“libs”中增加httpmime-x.x.x.jar。完整的Java依赖库如图9-29所示。

      httpmime-x.x.x.jar建议使用4.5及以上版本，下载地址：**https://mvnrepository.com/artifact/org.apache.httpcomponents/httpmime**。

      图 9-29 Java 依赖库

   b. httpmime-x.x.x.jar添加完成后，在Java工程的.classpath文件中，补充httpmime信息，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
<classpathentry kind="src" path="src"/>
<classpathentry kind="lib" path="libs/commons-codec-1.11.jar"/>
<classpathentry kind="lib" path="libs/commons-logging-1.2.jar"/>
<classpathentry kind="lib" path="libs/httpclient-4.5.13.jar"/>
<classpathentry kind="lib" path="libs/httpcore-4.4.13.jar"/>
<classpathentry kind="lib" path="libs/httpmime-x.x.x.jar"/>
<classpathentry kind="lib" path="libs/java-sdk-core-3.1.2.jar"/>
<classpathentry kind="lib" path="libs/okhttp-3.14.9.jar"/>
<classpathentry kind="lib" path="libs/okio-1.17.2.jar"/>
<classpathentry kind="output" path="bin"/>
</classpath>
```

3. 创建Java类，进行预测请求。
   - **输入为文件格式**

     Java的请求体示例如下：

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.Consts;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ContentType;
```

```
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import java.io.File;

public class MyAppCodeFile {

    public static void main(String[] args) {
        # API接口公网地址，例如"https://8e******5fe.apig.******.huaweicloudapis.com/v1/infers/f2682******f42"
        String url = "在线服务的调用地址";
        // 认证用的appCode硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
        // 本示例以appCode保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_CODE。
        String appCode = System.getenv("HUAWEICLOUD_APP_CODE");
        String filePath = "预测文件的本地路径";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader("X-Apig-AppCode", appCode);

            // Special characters, such as the double quotation mark ("), contained in the body must be escaped.
            File file = new File(filePath);
            HttpEntity entity = MultipartEntityBuilder.create().addBinaryBody("images", file).setContentType(ContentType.MULTIPART_FORM_DATA).setCharset(Consts.UTF_8).build();
            httpPost.setEntity(entity);

            // Send post
            CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

“addBinaryBody”中的参数名由在线服务的输入参数决定，需要和“类型”为“file”的输入参数“名称”保持一致。此处以“images”为例。

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAppCodeTest {

    public static void main(String[] args) {
        # API接口公网地址，例如"https://8e******5fe.apig.******.huaweicloudapis.com/v1/infers/f2682******f42"
        String url = "在线服务的调用地址";
        // 认证用的appCode硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
        // 本示例以appCode保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中
```



```
设置环境变量HUAWEICLOUD_APP_CODE。
        String appCode = System.getenv("HUAWEICLOUD_APP_CODE");
        String body = "{}";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
            httpPost.setHeader("X-Apig-AppCode", appCode);

            // Special characters, such as the double quotation mark ("), contained in the body
must be escaped.
            httpPost.setEntity(new StringEntity(body));

            // Send post
            CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

“body”由具体文本格式决定，此处以json为例。

9.4.4 访问在线服务支持的访问通道

9.4.4.1 通过公网访问通道的方式访问在线服务

背景描述

ModelArts推理默认使用公网访问在线服务。在线服务部署成功后，将为用户提供一个可调用的API，此API为标准Restful API。您可以在服务详情页面，调用指南页签中查看API接口公网地址。

图 9-30 API 接口公网地址

约束限制

调用API访问在线服务时，对预测请求体大小和预测时间有限制：

- 请求体的大小不超过12MB，超过后请求会被拦截。
- 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。

访问在线服务

公网访问在线服务有以下认证方式，API调用请参见认证详情：

- **通过Token认证的方式访问在线服务**
- **通过AK/SK认证的方式访问在线服务**
- **通过APP认证的方式访问在线服务**

9.4.4.2 通过 VPC 访问通道的方式访问在线服务

背景说明

如果您希望在自己账号的VPC内部节点访问ModelArts推理的在线服务，可以使用VPC访问通道的功能，用户通过在自己账号的指定VPC下创建终端节点，连接到ModelArts的终端节点服务，即可在自己的VPC节点中访问在线服务。

约束限制

调用API访问在线服务时，对预测请求体大小和预测时间有限制：

- 请求体的大小不超过12MB，超过后请求会被拦截。
- 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。

操作步骤

VPC访问通道访问在线服务操作步骤如下：

1. **获取ModelArts终端节点服务地址**
2. **购买连接ModelArts终端节点**
3. **创建DNS内网域名**
4. **VPC访问在线服务**

步骤1 提交工单，提供账号ID给华为云技术支持，用于获取ModelArts终端节点服务地址。

步骤2 购买连接ModelArts终端节点

1. 登录虚拟私有云（VPC）管理控制台，单击左侧导航栏中的“VPC 终端节点>终端节点”，进入“终端节点”页面。
2. 单击右上角的“购买终端节点”，进入购买页面。
   - 区域：终端节点所在区域。
     不同区域的资源之间内网不互通，请确保与ModelArts所在区域保持一致。
   - 服务类别：请选择“按名称查找服务”。
   - 服务名称：填入**步骤1**中获取的“终端节点服务地址”。单击右侧验证按钮，系统将为您自动填入虚拟私有云、子网和节点IP。
   - 创建内网域名：保持默认值。
3. 确认规格无误后，单击“立即购买”后提交任务，界面自动跳转至终端节点列表页面。

步骤3 创建DNS内网域名

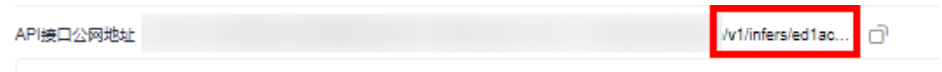
新创建的在线服务对接的是专享版APIG，需要使用ModelArts推理的独立公网域名，即infer-modelarts-<regionId>.modelarts-infer.com。内网VPC无法解析modelarts-infer.com域名，需要用户参考当前步骤和“步骤**步骤4** VPC访问在线服务”增加内网域名解析。

1. 登录云解析服务DNS管理控制台，左侧导航栏选择“内网域名”。
2. 单击“创建内网域名”，打开创建内网域名弹出框。填写以下参数配置：
   - 域名：遵循命名规范“infer-modelarts-<regionId>.modelarts-infer.com”，例如：infer-modelarts-cn-south-1.modelarts-infer.com
   - VPC：选择内网域名关联的VPC。
3. 单击“确定”，完成DNS内网域名的创建。

步骤4 VPC访问在线服务

1. 通过VPC访问通道访问在线服务，API如下：
   ```
   https://{DNS内网域名}/{URL}
   ```
   - DNS内网域名：设置的内网域名。您还可以通过在线服务列表页，单击“VPC访问通道”，打开弹出框，查看“访问域名”。
   - URL：在线服务的URL为服务详情页，调用指南页签中获取的“API接口公网地址”截取域名之后的地址部分。

   图 9-31 获取 URL

2. 使用图形界面的软件、curl命令、Python语言等多种方式访问在线服务。可参考**通过Token认证的方式访问在线服务**。

----结束

9.4.4.3 通过 VPC 高速访问通道的方式访问在线服务

背景说明

访问在线服务的实际业务中，用户可能会存在如下需求：

- 高吞吐量、低时延
- TCP或者RPC请求

因此，ModelArts提供了VPC直连的高速访问通道功能以满足用户的需求。

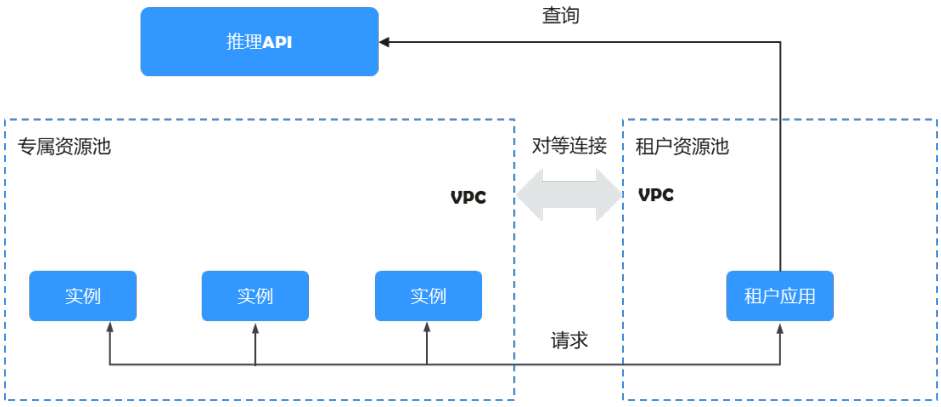
使用VPC直连的高速访问通道，用户的业务请求不需要经过推理平台，而是直接经VPC对等连接发送到实例处理，访问速度更快。

说明

由于请求不经过推理平台，所以会丢失以下功能：

- 认证鉴权
- 流量按配置分发
- 负载均衡
- 告警、监控和统计

图 9-32 VPC 直连的高速访问通道示意图



约束限制

调用API访问在线服务时，对预测请求体大小和预测时间有限制：

- 请求体的大小不超过12MB，超过后请求会被拦截。
- 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。

准备工作

使用专属资源池部署在线服务，服务状态为“运行中”。

须知

- 只有专属资源池部署的服务才支持VPC直连的高速访问通道。
- VPC直连的高速访问通道，目前只支持访问在线服务。
- 因流量限控，获取在线服务的IP和端口号次数有限制，每个主账号租户调用次数不超过2000次/分钟，每个子账号租户不超过20次/分钟。
- 目前仅支持自定义镜像导入模型，部署的服务支持高速访问通道。

操作步骤

使用VPC直连的高速访问通道访问在线服务，基本操作步骤如下：

1. **将专属资源池的网络打通VPC**
2. **VPC下创建弹性云服务器**
3. **获取在线服务的IP和端口号**
4. **通过IP和端口号直连应用**

步骤1 将专属资源池的网络打通VPC

登录ModelArts控制台，进入“AI专属资源池 > 弹性集群Cluster”找到服务部署使用的专属资源池，单击“名称/ID”，进入资源池详情页面，查看网络配置信息。返回专属资源池列表，选择“网络”页签，找到专属资源池关联的网络，打通VPC。打通VPC网络后，网络列表和资源池详情页面将显示VPC名称，单击后可以跳转至VPC详情页面。

图 9-33 查看网络配置

图 9-34 打通 VPC

步骤2 VPC下创建弹性云服务器

登录弹性云服务器ECS控制台，单击右上角“购买弹性云服务器”，进入购买弹性云服务器页面，完成基本配置后单击“下一步：网络配置”，进入网络配置页面，选择**步骤1**中打通的VPC，完成其他参数配置，完成高级配置并确认配置，下发购买弹性云服务器的任务。等待服务器的状态变为“运行中”时，弹性云服务器创建成功。单击“名称/ID”，进入服务器详情页面，查看虚拟私有云配置信息。

图 9-35 购买弹性云服务器时选择 VPC

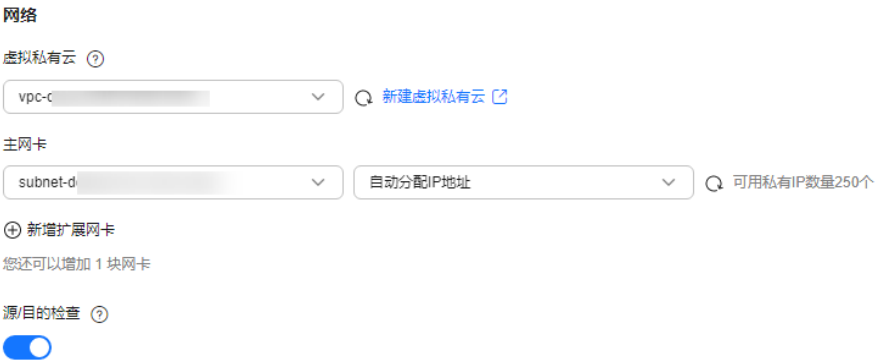


图 9-36 查看虚拟私有云配置信息

步骤3 获取在线服务的IP和端口号

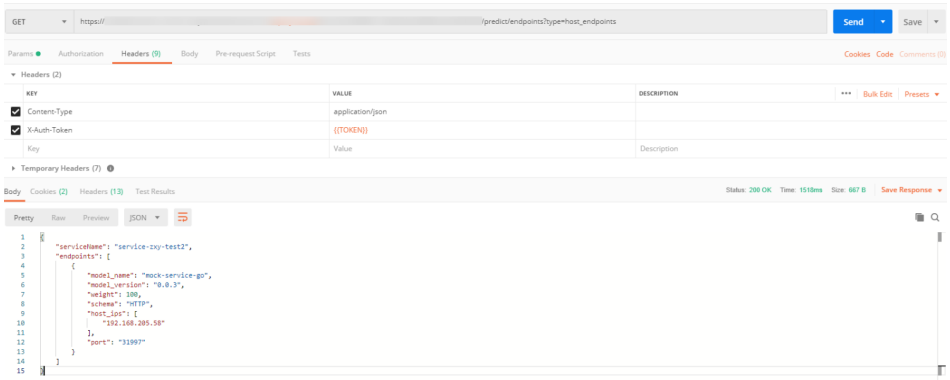
可以通过使用图形界面的软件（以Postman为例）获取服务的IP和端口号，也可以登录弹性云服务器（ECS），创建Python环境运行代码，获取服务IP和端口号。

API接口：

```
GET /v1/{project_id}/services/{service_id}/predict/endpoints?type=host_endpoints
```

- 方式一：图形界面的软件获取服务的IP和端口号

  图 9-37 接口返回示例

- 方式二：Python语言获取IP和端口号

  Python代码如下，下述代码中以下参数需要手动修改：

  - project_id：用户项目ID，获取方法请参见**获取项目ID和名称**。
  - service_id：服务ID，在服务详情页可查看。
  - REGION_ENDPOINT：服务的终端节点，查询请参见**终端节点**。

  ```
  def get_app_info(project_id, service_id):
      list_host_endpoints_url = "{}/v1/{}/services/{}/predict/endpoints?type=host_endpoints"
  ```

```
    url = list_host_endpoints_url.format(REGION_ENDPOINT, project_id, service_id)
    headers = {'X-Auth-Token': X_Auth_Token}
    response = requests.get(url, headers=headers)
    print(response.content)
```

步骤4 通过IP和端口号直连应用

登录弹性云服务器（ECS），可以通过Linux命令行访问在线服务，也可以创建Python环境运行Python代码访问在线服务。schema、 ip、port参数值从**步骤3**获取。

- 执行命令示例如下，直接访问在线服务。

```
curl --location --request POST 'http://192.168.205.58:31997' \
--header 'Content-Type:  application/json' \
--data-raw '{"a":"a"}'
```

图 9-38 访问在线服务

- 创建Python环境，运行Python代码访问在线服务。

```
def vpc_infer(schema, ip, port, body):
    infer_url = "{}://{}:{}"
    url = infer_url.format(schema, ip, port)
    response = requests.post(url, data=body)
    print(response.content)
```

说明

由于高速通道特性会缺失负载均衡的能力，因此在多实例时需要自主制定负载均衡策略。

----结束

9.4.5 访问在线服务支持的传输协议

9.4.5.1 使用 WebSocket 协议的方式访问在线服务

背景说明

WebSocket是一种网络传输协议，可在单个TCP连接上进行全双工通信，位于OSI模型的应用层。WebSocket协议在2011年由IETF标准化为RFC 6455，后由RFC 7936补充规范。Web IDL中的WebSocket API由W3C标准化。

WebSocket使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在WebSocket API中，浏览器和服务器只需要完成一次握手，两者之间就可以建立持久性的连接，并进行双向数据传输。

前提条件

- 在线服务部署时需选择“升级为WebSocket”。
- 在线服务中的模型导入选择的镜像需支持WebSocket协议。

约束与限制

- WebSocket协议只支持部署在线服务。
- 只支持自定义镜像导入模型部署的在线服务。

- 调用API访问在线服务时，对预测请求体大小和预测时间有限制：
  - 请求体的大小不超过12MB，超过后请求会被拦截。
  - 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。

WebSocket 在线服务调用

WebSocket协议本身不提供额外的认证方式。不管自定义镜像里面是ws还是wss，经过ModelArts平台出去的WebSocket协议都是wss的。同时wss只支持客户端对服务端的单向认证，不支持服务端对客户端的双向认证。

可以使用ModelArts提供的以下认证方式：

- **token认证**
- **AK/SK**
- **APP认证**

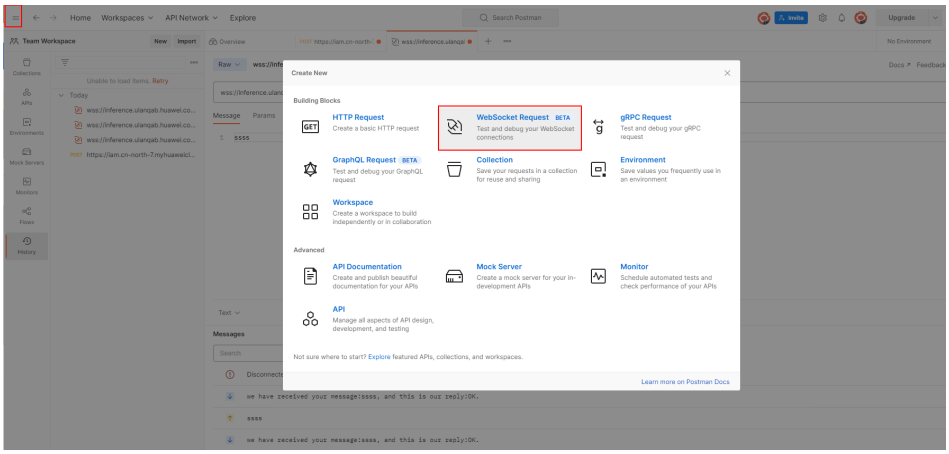
WebSocket服务调用步骤如下（以图形界面的软件Postman进行预测，token认证为例）：

1. **WebSocket连接的建立**
2. **WebSocket客户端和服务端双向传输数据**

步骤1 WebSocket连接的建立

1. 打开Postman（需选择8.5 以上版本，以10.12.0为例）工具，单击左上角，选择“File>New”，弹出新建对话框，选择“WebSocket Request”（当前为beta版本）功能：

   图 9-39 选择 WebSocket Request 功能

2. 在新建的窗口中填入WebSocket连接信息：

   左上角选择Raw，不要选择Socket.IO（一种WebSocket实现，要求客户端跟服务端都要基于Socket.IO），地址栏中填入从服务详情页“调用指南”页签中获取“API接口调用公网地址”后面的地址。如果自定义镜像中有更细粒度的地址，则在地址后面追加该URL。如果有queryString，那么在params栏中添加参数。在header中添加认证信息（不同认证方式有不同header，跟https的推理服务相同）。选择单击右上的connect按钮，建立WebSocket连接。

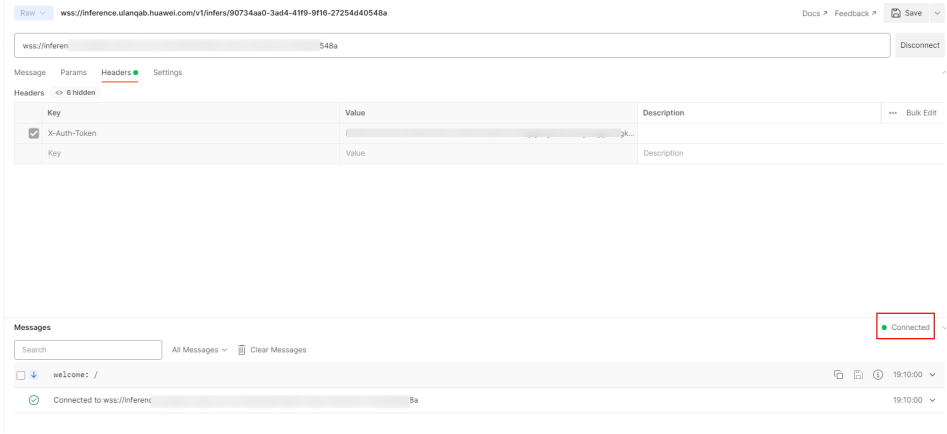
图 9-40 获取 API 接口调用公网地址

说明

- 如果信息正确，右下角连接状态处会显示：CONNECTED；
- 如果无法建立连接，如果是401状态码，检查认证信息；
- 如果显示WRONG_VERSION_NUMBER等关键字，检查自定义镜像的端口和ws跟wss的配置是否正确。

连接成功后结果如下：

图 9-41 连接成功



须知

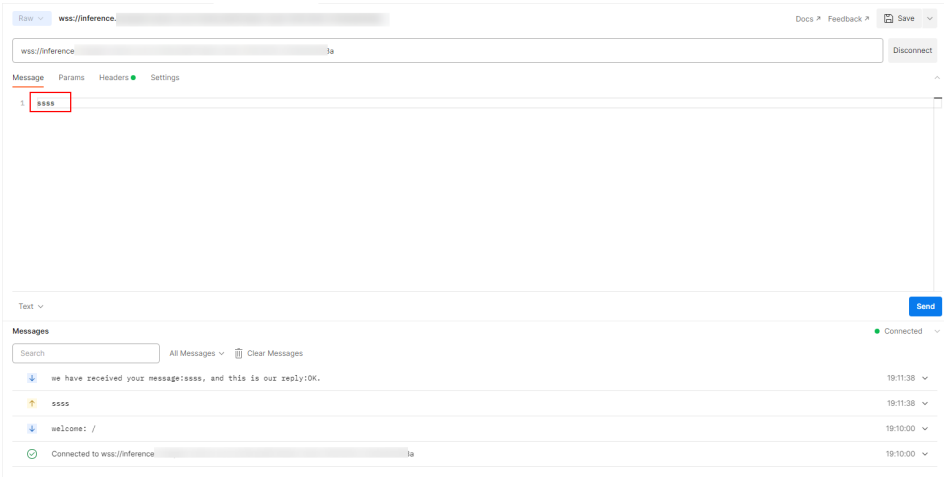
优先验证自定义镜像提供的websocket服务的情况，不同的工具实现的websocket服务会有不同，可能出现连接建立后维持不住，可能出现请求一次后连接就中断需要重新连接的情况，ModelArts平台只保证，未上ModelArts前自定义镜像的websocket的形态跟上了ModelArts平台后的websocket形态相同（除了地址跟认证方式不同）。

步骤2 WebSocket客户端和服务端双向传输数据

连接建立后，WebSocket使用TCP完成全双工通信。WebSocket的客户端可以往服务端发送数据，客户端有不同的实现，同一种语言也存在不同的lib包的实现，这里不考虑实现的不同种类。

客户端发送的内容在协议的角度不限定格式，Postman支持Text/Json/XML/HTML/Binary，以text为例，在输入框中输入要发送的文本，单击右侧中部的Send按钮即可将请求发往服务端，当文本内容过长，可能会导致postman工具卡住。

图 9-42 发送数据



----结束

9.4.5.2 使用 Server-Sent Events 协议的方式访问在线服务

背景说明

Server-Sent Events（SSE）是一种服务器向客户端推送数据的技术，它是一种基于HTTP的推送技术，服务器可以向客户端推送事件。这种技术通常用于实现服务器向客户端推送实时数据，例如聊天应用、实时新闻更新等。

SSE主要解决了客户端与服务器之间的单向实时通信需求（例如ChatGPT回答的流式输出），相较于WebSocket（双向实时），它更加轻量级且易于实现。

前提条件

在线服务中的模型导入选择的镜像需支持SSE协议。

约束与限制

- SSE协议只支持部署在线服务。
- 只支持自定义镜像导入模型部署的在线服务。
- 调用API访问在线服务时，对预测请求体大小和预测时间有限制：
  - 请求体的大小不超过12MB，超过后请求会被拦截。
  - 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。

SSE 在线服务调用

SSE协议本身不提供额外的认证方式，和HTTP请求方式一致。

可以使用ModelArts提供的以下认证方式：

- **token认证**
- **AK/SK**
- **APP认证**

SSE服务调用如下（以图形界面的软件Postman进行预测，token认证为例）：

图 9-43 SSE 服务调用

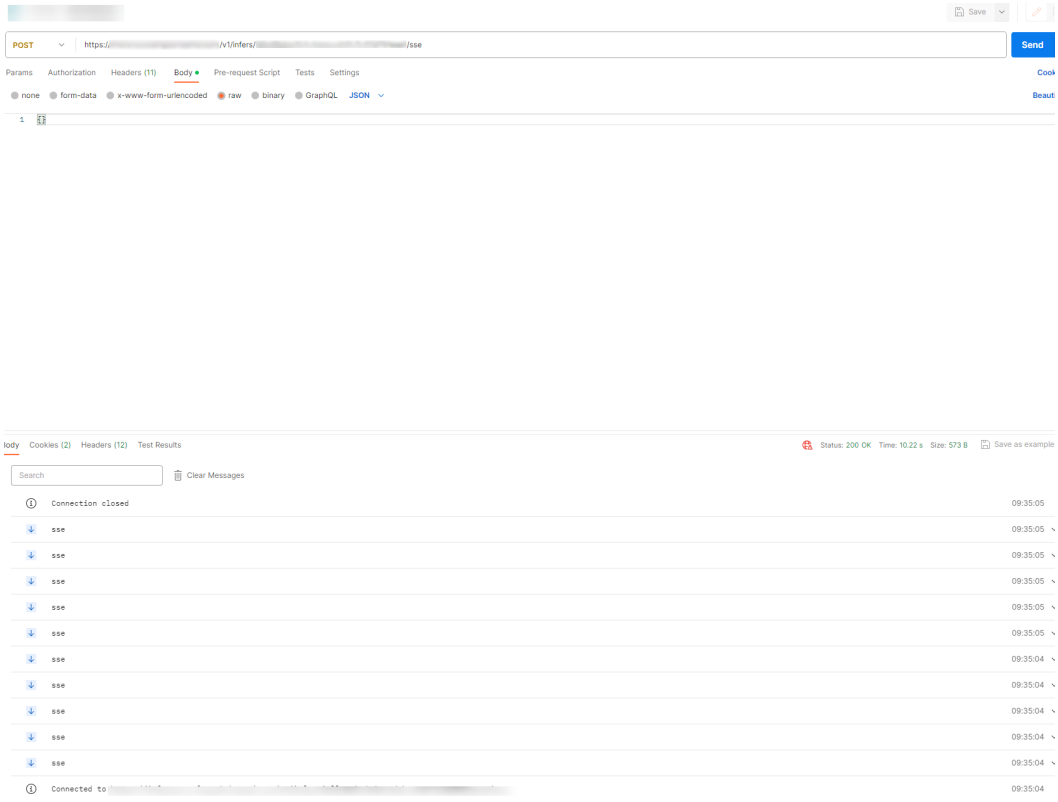


图 9-44 响应头 Content-Type

说明

正常情况下，可以观察到响应头Content-Type为text/event-stream;charset=UTF-8。

9.5 将模型部署为批量推理服务

模型准备完成后，您可以将模型部署为批量服务。在“模型部署>批量服务”界面，列举了用户所创建的批量服务。

前提条件

- 数据已完成准备：已在ModelArts中创建状态“正常”可用的模型。
- 准备好需要批量处理的数据，并上传至OBS目录。
- 已在OBS创建至少1个空的文件夹，用于存储输出的内容。

背景信息

- 用户最多可创建1000个批量服务。
- 根据模型定义的输入请求不同（JSON文本或文件），不同的模型输入，需要填写的参数不同。当模型输入为JSON文件时，则需要根据配置文件生成映射文件；如果模型输入为文件时，则不需要。

操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型部署 > 批量服务”，默认进入“批量服务”列表。
2. 在批量服务列表中，单击左上角“部署”，进入“部署”页面。
3. 在部署页面，填写批量服务相关参数。
   a. 填写基本信息。基本信息包含“名称”、“描述”。其中“名称”默认生成。例如：service-bc0d，您也可以根据实际情况填写“名称”和“描述”信息等。
   b. 填写服务参数。包含资源池、模型配置等关键信息。

表 9-23 参数说明

参数名称	说明
“资源池”	“公共资源池” 公共资源池有CPU或GPU两种规格。如需使用，需联系管理员创建公共资源池。
	“专属资源池” 您可以在资源池规格中选择对应的规格进行使用。
“模型来源”	根据您的实际情况选择“自定义模型”或者“订阅模型”。
“选择模型及版本”	选择状态“正常”的模型及版本。
“输入数据目录位置”	选择输入数据的OBS路径，即您上传数据的OBS目录。只能选择文件夹或“.manifest”文件。“.manifest”文件规范请参见**Manifest文件规范**。 **说明** • 输入数据为图片时，建议单张图片小于12MB。 • 输入数据格式为csv时，建议不要包含中文。 • 输入数据格式为csv时，建议文件大小不超过12MB。 • 如果单张图片/csv文件超过文件12MB，会提示报错，建议调整文件大小使其符合要求，或联系技术支持人员调整文件大小限制。
“请求路径”	批量服务中调用模型的接口URL，表示服务的请求路径，此值来自模型配置文件中apis的url字段。

参数名称	说明
“映射关系”	如果模型输入是json格式时，系统将根据此模型对应的配置文件自动生成映射关系。如果模型的输入是文件，则不需要映射关系。 自动生成的映射关系文件，填写每个参数对应到csv单行数据的字段索引，索引index从0开始计数。 映射关系生成规则：映射规则来源于模型配置文件“config.json”中输入参数（request）。当“type”定义为“string/number/integer/boolean”基本类型时，需要配置映射规则参数，即index参数。请参见**映射关系示例**了解其规则。 index必须是从0开始的正整数，当index设置不规则不符时，最终的请求将忽略此参数。配置映射规则后，其对应的csv数据必须以英文半角逗号分隔。
“输出数据目录位置”	选择批量预测结果的保存位置，可以选择您创建的空文件夹。
“实例规格”	系统将根据您的模型匹配提供可用的计算资源。请在下拉框中选择可用资源，如果资源标识为售罄，表示暂无此资源。 例如，模型来源于自动学习项目，则计算资源将自动关联自动学习规格供使用。
“实例数”	设置当前版本模型的实例个数。如果节点个数设置为1，表示后台的计算模式是单机模式；如果节点个数设置大于1，表示后台的计算模式为分布式的。请根据实际编码情况选择计算模式。
“环境变量”	设置环境变量，注入环境变量到容器实例。为确保您的数据安全，在环境变量中，请勿输入敏感信息，如明文密码。
“部署超时时间”	用于设置单个模型实例的超时时间，包括部署和启动时间。默认值为20分钟，输入值必须在3到120之间。
“运行日志输出”	默认关闭，批量服务的运行日志仅存放在ModelArts日志系统，在服务详情页的“日志”支持简单查询。 如果开启此功能，批量服务的运行日志会输出存放到云日志服务LTS。LTS自动创建日志组和日志流，默认缓存7天内的运行日志。如需了解LTS专业日志管理功能，请参见**云日志服务**。 **说明** • “运行日志输出”开启后，不支持关闭。 • LTS服务提供的日志查询和日志存储功能涉及计费，详细请参见**了解LTS的计费规则**。 • 请勿打印无用的audio日志文件，这会导致系统日志卡死，无法正常显示日志，可能会出现“Failed to load audio”的报错。

4. 完成参数填写后，根据界面提示完成批量服务的部署。部署服务一般需要运行一段时间，根据您选择的数据量和资源不同，部署时间将耗时几分钟到几十分钟不等。

   说明

   批量服务部署完成后，将立即启动，运行过程中将按照您选择的资源按需计费。

   您可以前往批量服务列表，查看批量服务的基本情况。在批量服务列表中，刚部署的服务“状态”为“部署中”，当批量服务的“状态”变为“运行完成”时，表示服务部署完成。

Manifest 文件规范

推理平台批量服务支持使用manifest文件，manifest文件可用于描述数据的输入输出。

输入manifest文件样例

- 文件名：“test.manifest”
- 文件内容：

```
{"source": "obs://test/data/1.jpg"}
{"source": "s3://test/data/2.jpg"}
{"source": "https://infers-data.obs.cn-north-1.myhuaweicloud.com:443/xgboosterdata/data.csv?AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz1uDzwve8GpY%3D&x-obs-security-token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-ZaMSxHvl68kKLAy5feYvLDM..."}
```

- 文件要求：
  a. 文件名后缀需为“.manifest”；
  b. 文件内容是多行JSON，每行JSON描述一个输入数据，需精确到文件，不能是文件夹；
  c. JSON内容需定义一个source字段，字段值是OBS的文件地址，有2种表达形式：
     i. 桶路径“<obs path>{{桶名}}/{{对象名}}/文件名”，适用于访问自己名下的OBS数据；您可以访问OBS服务的对象获取路径。<obs path>可以为“obs://”或“s3://”。
     ii. OBS生成的分享链接，包含签名信息。适用于访问其他人的OBS数据。分享链接有效时间限制，请在有效时间内操作。

输出manifest文件样例

批量服务的输出结果目录会有一个manifest文件。

- 假设用户输出结果路径为/test-bucket/test/，则结果存放位置如下：

```
OBS桶/目录名
├── test-bucket
│   ├── test
│   │   ├── infer-result-{{task_id}}.manifest
│   │   ├── infer-result
│   │   │   ├── 1.jpg_result.txt
│   │   │   ├── 2.jpg_result.txt
```

- infer-result-0.manifest文件内容：

```
{"source": "obs://obs-data-bucket/test/data/1.jpg","result":"SUCCESSFUL","inference-loc": "obs://test-bucket/test/infer-result/1.jpg_result.txt"}
{"source": "s3://obs-data-bucket/test/data/2.jpg","result":"FAILED","error_message": "Download file failed."}
{"source ": "https://infers-data.obs.example.com:443/xgboosterdata/2.jpg?AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz1uDzwve8GpY%3D&x-obs-security-token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-
```

ZaMSxHvl68kKLAy5feYvLDMNZWxzhBZ6Q-3HcoZMh9gISwQOVBwm4ZytB_m8sg1fL6isU7T3CnoL9jmv
DGgT9VBC7dC1EyfSJrUcqfB_N0ykCsfrA1Tt_IQYZFDu_HyqVk-
GunUcTVdDfWlCV3TrYcpmznZjliAnYUO89kAwCYGeRZsCsC0ePu4PHMsBvYV9gWmN9AUZIDn1sfRL4vo
BpwQnp6tnAgHW49y5a6hP2hCAoQ-95SpUriJ434QlymoeKfTHVMKOeZxZea-
JxOvevOCGI5CcGehEJaz48sgH81UiHzl21zocNB_hpPfus2jY6KPglEJxMv6Kwmro-
ZBXWuSJUDOnSYXI-3ciYjg9-
h10b8W3sW1mOTFCWNGoWsd74it7l_5-7UUhoIeyPByO_REwkur2FOJsuMpGlRaPyglZxXm_jfdLFXobYtz
Zhbul4yWXga6oxTOkfcwykTOYH0NPoPRt5MYGYweOXXxFs3d5w2rd0y7p0QYhyTzIkk5CIz7FlWNapFISL
7zdhsl8RfchTqESq94KgkeqatSF_iIvnYMW2r8P8x2k_eb6NJ7U_q5ztMbO9oWEcfr0D2f7n7Bl_nb2HIB_H9tj
zKvqwngaimYhBbMRPfibvttW86GiwVP8vrC27FOn39Be9z2hSfJ_8pHej0yMlyNqZ481FQ5vWT_vFV3JHM-
7I1ZB0_hIdaHfItm-J69cTfHSEOzt7DGaMIES1o7U3w%3D%3D","result":"SUCCESSFUL","inference-loc":
"obs://test-bucket/test/infer-result/2.jpg_result.txt"}

- 文件格式：
  a. 文件名为“infer-result-{{task_id}}.manifest”，task_id为批量任务id，批量服务对应唯一的批量任务id。
  b. 当处理文件数目较多时，可能会有多个manifest文件，后缀相同，均为“.manifest”，文件名以后缀区分，例如“infer-result-{{task_id}}_1.manifest”等。
  c. manifest同一目录下会创建infer-result-{{task_id}}目录存放文件处理结果。
  d. 文件内容是多行JSON，每行JSON描述一个输入数据的对应输出结果。
  e. JSON内容包含多个字段。
     i. source：输入数据描述，与输入的manifest一致。
     ii. result：文件处理结果，值为SUCCESSFUL或FAILED，分别代表成功与失败。
     iii. inference-loc：输出结果路径，result为SUCCESSFUL时有此字段，格式为“obs://{{桶名}}/{{对象名}}”。
     iv. error_message：错误信息，result为FAILED时有此字段。

映射关系示例

如下示例展示了配置文件、映射规则、csv数据以及最终推理请求的关系。

假设，您的模型所用配置文件，其apis参数如下所示：

```
[
  {
    "method": "post",
    "url": "/",
    "request": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "data": {
            "type": "object",
            "properties": {
              "req_data": {
                "type": "array",
                "items": [
                  {
                    "type": "object",
                    "properties": {
                      "input_1": {
                        "type": "number"
                      },
                      "input_2": {
                        "type": "number"
                      },
                      "input_3": {
```



```
                                    "type": "number"
                                },
                                "input_4": {
                                    "type": "number"
                                }
                            }
                        }
                    ]
                }
            }
        }
    }
]
```

此时，其对应的映射关系如下所示。ModelArts管理控制台将从配置文件中自动解析映射关系，如果您调用ModelArts API时，需要自行根据规则编写映射关系。

```
{
    "type": "object",
    "properties": {
        "data": {
            "type": "object",
            "properties": {
                "req_data": {
                    "type": "array",
                    "items": [
                        {
                            "type": "object",
                            "properties": {
                                "input_1": {
                                    "type": "number",
                                    "index": 0
                                },
                                "input_2": {
                                    "type": "number",
                                    "index": 1
                                },
                                "input_3": {
                                    "type": "number",
                                    "index": 2
                                },
                                "input_4": {
                                    "type": "number",
                                    "index": 3
                                }
                            }
                        }
                    ]
                }
            }
        }
    }
}
```

用户需要进行推理的数据，即CSV数据，格式如下所示。数据必须以英文逗号隔开。

```
5.1,3.5,1.4,0.2
4.9,3.0,1.4,0.2
4.7,3.2,1.3,0.2
```

根据定义好的映射关系，最终推理请求样例如下所示，与在线服务使用的格式类似：

```
{
    "data": {
        "req_data": [{
            "input_1": 5.1,
            "input_2": 3.5,
```



```
            "input_3": 1.4,
            "input_4": 0.2
        }]
    }
}
```

查看批量服务预测结果

当您在部署批量服务时，会选择输出数据目录位置，您可以查看“运行完成”状态的批量服务运行结果。

步骤1 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>批量服务”，进入“批量服务”管理页面。

步骤2 单击状态为“运行完成”的目标服务名称，进入服务详情页面。

- 您可以查看服务的“名称”、“状态”、“服务ID”、“输入数据目录位置”、“输出数据目录位置”和“描述”。
- 您也可以通过单击描述右侧的，对描述信息进行编辑。

步骤3 从“输出数据目录位置”参数右侧获取详细OBS地址，前往此OBS目录，可以获取批量服务预测结果，包括预测结果文件和模型预测结果。

如果预测成功，目录下有预测结果文件和模型预测结果；如果预测失败，目录下只有预测结果文件。

- 预测结果文件：文件格式为“xxx.manifest”，里面包含文件路径、预测结果等信息。
- 模型预测结果输出：
  - 当输入为图片时，每张图片输出一个结果，输出结果格式为“图片名_result.txt”。例如：IMG_20180919_115016.jpg_result.txt。
  - 当输入为音频时，每个音频输出一个结果，输出结果格式为“音频名_result.txt”。例如：1-36929-A-47.wav_result.txt。
  - 当输入为表格数据时，输出结果格式为“表格名_result.txt”。例如：train.csv_result.txt。

----结束

9.6 管理 ModelArts 模型

9.6.1 查看 ModelArts 模型详情

查看模型列表

当模型创建成功后，您可在模型列表页查看所有创建的模型。模型列表页包含以下信息。

表 9-24 模型列表

参数	说明
模型名称	模型的名称。

参数	说明
最新版本	模型的当前最新版本。
状态	模型当前状态。
部署类型	模型支持部署的服务类型。
版本数量	模型的版本数量。
请求模式	在线服务的请求模式。 • 同步请求：单次推理，可同步返回结果（约<60s）。例如: 图片、较小视频文件。 • 异步请求：单次推理，需要异步处理返回结果（约>60s）。例如: 实时视频推理、大视频文件。
创建时间	模型的创建时间。
描述	模型的描述。
操作	• 部署：将模型发布为在线服务、批量服务或边缘服务。 • 创建新版本：创建新的模型版本。参数配置除版本外，将默认选择上一个版本的配置信息，您可以对参数配置进行修改。 • 删除：删除对应的模型。 **说明** 如果模型的版本已经部署服务，需先删除关联的服务后再执行删除操作。模型删除后不可恢复，请谨慎操作。

单击模型的“版本数量”，可查看版本列表信息。

图 9-45 版本列表

版本列表中包含以下信息。

表 9-25 版本列表

参数	说明
版本	模型当前版本。
状态	模型当前状态。
部署类型	模型支持部署的服务类型。

参数	说明
模型大小	模型的大小。
模型来源	显示模型的来源。
创建时间	模型的创建时间。
描述	模型的描述。
操作	• 部署：将模型发布为在线服务、批量服务或边缘服务。 • 删除：针对模型的某一版本进行删除。

查看模型详情

当模型创建成功后，您可以进入模型详情页查看模型的信息。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型管理”，进入“自定义模型”列表页面。
2. 单击目标模型名称，进入模型详情页面。
   您可以查看模型的基本信息、模型精度，以及切换页签查看更多信息。

表 9-26 模型基本信息

参数	说明
名称	模型的名称。
状态	模型当前状态。
版本	模型当前版本。
ID	模型的ID。
描述	单击编辑按钮，可以添加模型的描述。
部署类型	模型支持部署的服务类型。
元模型来源	显示元模型的来源，主要有从训练中选择、从对象存储服务（OBS）中选择、从容器镜像中选择。不同来源的元模型，模型显示的参数会不同。
训练作业名称	如果元模型来源于训练作业，则显示关联的训练作业，单击训练作业名称可以直接跳转到训练作业详情页面。
训练作业版本	如果元模型来源于训练作业且为旧版训练作业，显示训练作业版本。
元模型存储路径	如果元模型来源于对象存储服务，显示元模型的存放路径。
容器镜像存储路径	如果元模型来源于容器镜像，显示容器镜像存储路径。
AI引擎	如果元模型来源于训练作业/对象存储服务，显示模型使用的AI引擎。

参数	说明
引擎包地址	如果元模型来源于对象存储服务（AI引擎为Custom），显示引擎包地址。
运行环境	如果元模型来源于训练作业/对象存储服务（AI引擎为预置引擎），显示元模型依赖的运行环境。
容器调用接口	如果元模型来源于对象存储服务（AI引擎为Custom）/容器镜像，显示模型启动的协议和端口号。
推理代码	如果元模型来源于训练作业且为旧版训练作业，则显示推理代码的存放路径。
镜像复制	如果元模型来源于容器镜像，显示镜像复制功能状态。
动态加载	如果元模型来源于训练作业/对象存储服务，显示模型是否支持动态加载。
大小	模型的大小。
健康检查	如果元模型来源于对象存储服务/容器镜像，显示健康检查状态。当健康检查为开启时，会根据您启用的探针显示对应探针的参数设置情况。 • **启动探针：**用于检测应用实例是否已经启动。如果提供了启动探针(startup probe)，则禁用所有其他探针，直到它成功为止。如果启动探针失败，将会重启实例。如果没有提供启动探针，则默认状态为成功Success。 • **就绪探针：**用于检测应用实例是否已经准备好接收流量。如果就绪探针失败，即实例未准备好，会从服务负载均衡的池中剔除该实例，不会将流量路由到该实例，直到探测成功。 • **存活探针：**用于检测应用实例内应用程序的健康状态。如果存活探针失败，即应用程序不健康，将会自动重启实例。 每种探针下会显示以下字段：检查方式、健康检查URL（检查方式为“HTTP请求检查”时显示）、健康检查命令（检查方式为“执行命令检查”时显示）、健康检查周期、延迟时间、超时时间、最大失败次数。
模型说明	显示创建模型时添加的模型说明文档信息。
系统运行架构	显示系统运行架构。
推理加速卡类型	显示推理加速卡类型。

表 9-27 模型页签详情

参数	说明
模型精度	显示该模型的模型召回率、精准率、准确率和F1值。
参数配置	可以查看模型的apis定义详情，以及模型的入参和出参。

参数	说明
运行时依赖	查看模型对环境的依赖。当构建任务失败后可以编辑运行时依赖，保存修改后将触发镜像重新构建。
事件	展示模型创建过程中的关键操作进展。 事件保存周期为3个月，3个月后自动清理数据。 查看模型的事件类型和事件信息，请参见**查看ModelArts模型事件**
使用约束	根据创建模型时的设置，显示部署服务的使用约束，如请求模式、启动命令、模型加密等。对于异步请求模式的模型，可显示输入模式、输出模式、服务启动参数和作业配置参数等参数。
关联服务	展示使用该模型部署的服务列表，单击服务名称可以直接跳转到服务详情页面。

9.6.2 查看 ModelArts 模型事件

创建模型的（从用户可看见创建模型任务开始）过程中，每一个关键事件点在系统后台均有记录，用户可随时在对应模型的详情页面进行查看。

方便用户更清楚的了解创建模型过程，遇到任务异常时，更加准确的排查定位问题。可查看的事件点包括：

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	开始导入模型。	-
异常	构建镜像失败。	构建镜像失败原因较多，需根据具体的报错定位和处理问题。**FAQ**
异常	自定义镜像不支持指定依赖。	自定义镜像导入不支持配置运行时依赖，在构建镜像的dockerfile文件中安装pip依赖包。**FAQ**
异常	非自定义镜像不支持指定swr_location字段。	请删除模型配置文件config.json中的swr_location字段后重试。
异常	自定义镜像健康检查接口必须是xxx。	请修改自定义镜像健康检查接口后重试。
正常	当前镜像构建任务状态为xxx。	-
异常	镜像xxx不存在名为xxx的标签。	请联系技术支持。

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
异常	模型配置文件包含非法参数值：xxx。	请删除模型配置文件中的非法参数后重试。
异常	获取镜像xxx的标签列表失败。	请联系技术支持。
异常	xxx大于xxxG，无法导入。	模型或镜像大小超过限制，请精简模型或镜像后，重新导入。FAQ
异常	用户xxx没有OBS的obs:object:PutObjectAcl权限。	子用户没有OBS的obs:object:PutObjectAcl权限，为子用户添加委托权限。FAQ
异常	镜像构建任务超时。限制超时时间为xxx分钟。	imagePacker构建镜像有超时时间限制，请精简代码，提高编译效率。FAQ
正常	模型描述已更新。	-
正常	模型运行时依赖未更新。	-
正常	模型运行时依赖已更新。正在重新构建镜像	-
异常	触发SWR限流，请稍后重试。	触发SWR限流，请稍后重试。
正常	系统升级中，请稍后重试。	-
异常	获取源镜像失败。认证错误，token已失效。	请联系技术支持。
异常	获取源镜像失败。检查该镜像是否存在。	请联系技术支持。
正常	源镜像大小计算完成。	-
正常	源镜像共享成功。	-
异常	构建镜像失败，因为触发了限流。请稍后重试。	触发了限流，请稍后重试。
异常	发送构建镜像请求失败。	请联系技术支持。
异常	共享源镜像失败。请检查镜像是否存在或者是否有权限共享该镜像。	请检查镜像是否存在或者是否有权限共享该镜像。
正常	模型导入成功。	-
正常	模型文件导入成功。	-

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	模型大小计算完成。	-
异常	模型导入失败。	模型导入失败情况较多，请参考FAQ定位和处理。
异常	复制模型文件失败，请检查OBS权限是否正常。	请检查OBS权限是否正常。FAQ
异常	镜像构建任务调度失败。	请联系技术支持。
异常	启动镜像构建任务失败。	请联系技术支持。
异常	罗马镜像构建完成，无法分享给资源租户。	请联系技术支持。
正常	镜像构建完成。	-
正常	启动镜像构建任务。	-
正常	启动环境镜像构建任务。	-
正常	收到构建模型环境镜像请求。	-
正常	收到构建模型镜像请求。	-
正常	使用现有环境镜像。	-
异常	构建镜像失败。详细信息请查看构建日志。	查看构建日志定位和处理问题。FAQ
异常	因系统内部原因构建镜像失败。请联系技术支持。	请联系技术支持。
异常	模型文件xxx大于5G，无法导入。	模型文件xxx大于5G，请精简模型文件后重试，或者使用动态加载功能进行导入。FAQ
异常	因系统内部原因创建OBS桶失败，请联系技术支持。	请联系技术支持。
异常	模型大小计算失败。子路径xxx在路径xxx下不存在。	修改子路径为正确的路径后重试，或者联系技术支持。
异常	模型大小计算失败。xxx类型模型不存在路径xxx下。	检查xxx类型模型的存储位置，修改为正确的路径后重试，或者联系技术支持。
提示	模型大小计算失败。多于一个xxx模型文件在路径xxx下。	-

创建模型的过程中，关键事件支持手动/自动刷新。

查看操作

1. 在ModelArts管理控制台的左侧导航栏中选择“模型管理”，在模型列表中，您可以单击模型名称，进入模型详情页面。
2. 在模型详情页面，切换到“事件”页签，查看事件信息。

9.6.3 管理 ModelArts 模型版本

为方便溯源和模型反复调优，在ModelArts中提供了模型版本管理的功能，您可以基于版本对模型进行管理。

前提条件

已在ModelArts中创建模型。

创建新版本

在“模型”页面，单击操作列的“创建新版本”进入“创建新版本”页面，参数配置除版本外，将默认选择上一个版本的配置信息，您可以对参数配置进行修改，参数说明请参见**创建模型**。单击“立即创建”，完成新版本的创建操作。

删除版本

在“模型管理”页面，单击模型的“版本数量”，在展开的版本列表中，单击“操作”列的“删除”，即可删除对应的版本。

说明

如果模型的版本已经部署服务，需先删除关联的服务后再执行删除操作。版本删除后不可恢复，请谨慎操作。

删除模型

在“模型管理”页面，单击模型“操作”列的“删除”，即可删除对应的模型。

说明

如果模型的版本已经部署服务，需先删除关联的服务后再执行删除操作。模型删除后不可恢复，请谨慎操作。

9.7 管理同步在线服务

9.7.1 查看在线服务详情

当模型部署为在线服务成功后，您可以进入“在线服务”页面，来查看服务详情。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署 > 在线服务”，进入“在线服务”管理页面。
2. 单击目标服务名称，进入服务详情页面。
   您可以查看服务的“名称”、“状态”等信息，详情说明请参见**表9-28**。

表 9-28 在线服务配置

参数	说明
名称	在线服务名称。
状态	在线服务当前状态。
来源	在线服务的来源。
服务ID	在线服务的ID。
描述	您可以单击编辑按钮，添加服务描述。
资源池	当前服务使用的资源池规格。如果使用公共资源池部署，则不显示该参数。
个性化配置	您可以为在线服务的不同版本设定不同配置条件，并支持携带自定义运行参数，丰富版本分流策略或同一版本内的不同运行配置。您可以打开个性化配置按钮，单击“查看配置”**修改服务个性化配置**。
服务流量限制	服务流量限制是指每秒内一个服务能够被访问的次数上限。
运行日志输出	默认关闭，在线服务的运行日志仅存放在ModelArts日志系统。 启用运行日志输出后，在线服务的运行日志会输出存放到云日志服务LTS。LTS自动创建日志组和日志流，默认缓存7天内的运行日志。如需了解LTS专业日志管理功能，请参见**云日志服务**。 **说明** • “运行日志输出”开启后，不支持关闭。 • LTS服务提供的日志查询和日志存储功能涉及计费，详细请参见**了解LTS的计费规则**。 • 请勿打印无用的audio日志文件，这会导致系统日志卡死，无法正常显示日志，可能会出现“Failed to load audio”的报错。
升级为WebSocket	是否升级为WebSocket服务。

3. 您可以进入在线服务的详情页面，通过切换页签查看更多详细信息，详情说明请参见**表9-29**。

表 9-29 在线服务详情

参数	说明
调用指南	展示API接口公网地址、模型信息、输入参数、输出参数。 您可以通过复制API接口公网地址，调用服务。如果您支持APP认证方式，可以在调用指南查看API接口公网地址和授权管理详情，包括“应用名称”、“AppKey”、“AppSecret”等信息。您也可以在此处对APP应用进行“添加授权”或“解除授权”的操作。

参数	说明
预测	对在线服务进行预测。具体操作请参见**使用预测功能测试在线服务**。
实例	查看异步在线服务的实例信息。这里的实例个数和部署服务时设置的“实例数”相对应，如果修改服务或服务异常，实例数会有变化。如果存在某个实例异常希望重建实例，您可单击“删除”按钮，该实例被删除后会自动新建一个相同计算规格的实例。
配置更新记录	展示“当前配置”详情和“历史更新记录”。 ● “当前配置”：展示模型名称、版本、状态、实例规格、分流、实例数、部署超时时间、环境变量、存储挂载等信息。专属资源池部署的服务，同时展示资源池信息。 ● “历史更新记录”：展示历史模型相关信息。
监控信息	展示当前服务的“资源统计信息”和“模型调用次数统计”。 ● “资源统计信息”：包括CPU、内存、GPU、NPU的可用和已用信息。 ● “模型调用次数统计”：当前模型的调用次数，从模型状态为“已就绪”后开始统计。（websocket服务不显示）
事件	展示当前服务使用过程中的关键操作，比如服务部署进度、部署异常的详细原因、服务被启动、停止、更新的时间点等。 事件保存周期为1个月，1个月后自动清理数据。 查看服务的事件类型和事件信息，请参见**查看在线服务的事件**

参数	说明
日志	展示当前服务下每个模型的日志信息。包含最近5分钟、最近30分钟、最近1小时和自定义时间段。 自定义时间段您可以选择开始时间和结束时间。 当服务启用运行日志输出后，页面展示存放到云日志服务LTS中的日志信息。您可以单击“到LTS查看完整日志”查看全量的日志。 日志搜索规则说明： • 不支持带有分词符的字符串搜索（当前默认分词符有 ,'";=()[]{}@&<>/:\n\t\r）。 • 支持关键词精确搜索。关键词指相邻两个分词符之间的单词。 • 支持关键词模糊匹配搜索，例如输入“error”或“er?or”或“rro*”或“er*r”。 • 支持短语精确搜索。例如输入“Start to refresh”。 • 启用运行日志输出前，支持关键词的“与”、“或”组合搜索。格式为“query logs&&erro*”或“query logs||erro*”。启用运行日志输出后，支持关键词的“与”、“或”组合搜索。格式为“query logs AND erro*”或“query logs OR erro*”。
标签	展示服务已添加的标签。支持添加、修改、删除标签。 标签详细用法请参见**ModelArts如何通过标签实现资源分组管理**。
Cloud Shell	允许用户使用ModelArts控制台提供的CloudShell登录运行中在线服务实例容器，详情请见**使用CloudShell调试在线服务实例容器**。

修改服务个性化配置

服务个性化配置规则由配置条件、访问版本、自定义运行参数（包括配置项名称和配置项值）组成。

您可以为在线服务的不同版本设定不同配置条件，并支持携带自定义运行参数。

个性化配置规则的优先级与顺序相对应，从高到低设置。您可以通过拖动个性化配置规则的顺序更换优先级。

当匹配了某一规则后就不再继续下一规则的判断，最多允许配置10个条件。

表 9-30 个性化配置参数

参数	是否必选	说明
配置条件	必选	SPEL（Spring Expression Language）规则的表达式，当前仅支持字符型的“相等”、“matches”和hashCode函数计算。

参数	是否必选	说明
访问版本	必选	服务个性化配置规则对应的访问版本。当匹配到规则时，请求该版本的在线服务。
配置项名称	可选	自定义运行参数的Key值，不超过128个字符。 当需要通过Header（http消息头）携带自定义运行参数至在线服务时，可以配置。
配置项值	可选	自定义运行参数的Value值，不超过256个字符。 当需要通过Header（http消息头）携带自定义运行参数至在线服务时，可以配置。

可以设置以下三种场景：

- 如果在线服务部署多个版本用于灰度发布，可以使用个性化配置实现按用户分流。

  表 9-31 按内置变量配置条件

内置变量	说明
DOMAIN_NAME	调用预测请求的账号名。
DOMAIN_ID	调用预测请求的账号ID。
PROJECT_NAME	调用预测请求的项目名。
PROJECT_ID	调用预测请求的项目ID。
USER_NAME	调用预测请求的用户名。
USER_ID	调用预测请求的用户ID。

  “#”表示引用变量，匹配的字符串需要用单引号。

  ```
  #{内置变量} == '字符串'
  #{内置变量} matches '正则表达式'
  ```

  - 示例一：

    当调用预测请求的账号名为“zhangsan”时，匹配至指定版本。

    ```
    #DOMAIN_NAME == 'zhangsan'
    ```

  - 示例二：

    当调用预测请求的账号名以“op”开头时，匹配至指定版本。

    ```
    #DOMAIN_NAME matches 'op.*'
    ```

表 9-32 常用的正则匹配表达式

字符	描述
“.”	匹配除“\n”之外的任何单个字符串。需匹配包括“\n”在内的任何字符，请使用“(.|\n)”的模式。
“*”	匹配前面的子表达式零次或多次。例如，“zo*”能匹配“z”以及“zoo”。
“+”	匹配前面的子表达式一次或多次。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。
“?”	匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“does”或“does”中的“do”。
“^”	匹配输入字符串的开始位置。
“$”	匹配输入字符串的结束位置。
“{n}”	*n*是一个非负整数。匹配确定的*n*次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个“o”。
“x|y”	匹配x或y。例如，“z|food”能匹配“z”或“food”。“(z|f)ood”则匹配“zood”或“food”。
“[xyz]”	字符集合。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。

图 9-46 按用户分流

- 如果在线服务部署多个版本用于灰度发布，可以使用个性化配置实现通过Header来访问不同版本。

  您需要通过"#HEADER_"开头说明引用header作为条件

  ```
  #HEADER_{key} == '{value}'
  #HEADER_{key} matches '{value}'
  ```

  - 示例一：

    当预测的http请求的header中存在version，且值为0.0.1则符合条件。不存在此header或者值不为0.0.1都不符合条件。

    ```
    #HEADER_version == '0.0.1'
    ```

  - 示例二：

    当预测的http请求的header中存在testheader且值符合正以mock开头时，可匹配到这条规则。

```
#HEADER_testheader matches 'mock.*'
```

- 示例三：

  当预测的http请求的header中存在uid且其哈希值符合指定的分桶算法时，可匹配到这条规则。

```
#HEADER_uid.hashCode() % 100 < 10
```

图 9-47 通过 Header 访问不同版本

- 如果在线服务部署的版本支持使用不同的运行配置，您可以通过“配置项名称”和“配置项值”携带自定义运行参数至在线服务，实现不同用户使用不同运行配置。

  示例：

  用户zhangsan访问时，模型使用配置A；用户lisi访问时，模型使用配置B。当匹配到运行配置条件时，ModelArts会在请求里增加一个Header，传入自定义运行参数，其中Key是“配置项名称”，Value是“配置项值”。

图 9-48 个性化配置规则支持传入自定义运行参数。

9.7.2 查看在线服务的事件

服务的（从用户可看见部署服务任务开始）整个生命周期中，每一个关键事件点在系统后台均有记录，用户可随时在对应服务的详情页面进行查看。

方便用户更清楚的了解服务部署和运行过程，遇到任务异常时，更加准确的排查定位问题。可查看的事件点包括：

表 9-33 事件

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	开始部署服务。	-
异常	资源不足，等待资源释放。	等待资源释放后重试。
异常	xxx资源不足，服务调度失败。补充信息：xxx	根据补充信息，了解资源不足详情，参考FAQ处理。
正常	开始构建镜像。	-
异常	构建模型(xxx) 镜像失败，构建日志:\nxxx。	根据构建日志定位和处理问题。
异常	构建镜像失败。	请联系技术支持。
正常	构建镜像完成。	-
异常	xxx服务失败。错误信息：xxx	请根据错误信息定位和处理问题。
异常	更新服务失败，执行回滚操作。	请联系技术支持。
正常	服务更新中。	-
正常	服务启动中。	-
正常	服务停止中。	-
正常	服务已停止。	-
正常	自动停止开关已关闭。	-
正常	自动关闭功能开启，服务将在xs后停止。	-
正常	到达自动停止时间，服务停止。	-
异常	配额超限，服务停止。	请联系技术支持。
异常	自动停止服务失败，错误信息: xxx	请根据错误信息定位和处理问题。
正常	删除资源池(xxx)上服务实例。	-
正常	停止资源池(xxx)上服务实例。	-

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
异常	批量服务失败，请稍后重试。错误信息：xxx	请根据错误信息定位和处理问题。
正常	服务运行完成。	-
异常	停止服务失败，错误信息：xxx	请根据错误信息定位和处理问题。
正常	订阅许可即将超期：xxx	-
正常	服务xxx启动成功。	-
异常	启动服务xxx失败。	启动服务失败情况较多，请参考FAQ定位和处理。
异常	部署服务超时，错误信息：xxx	请根据错误信息定位和处理问题。
正常	更新服务失败，执行回滚操作成功。	-
异常	更新服务失败，执行回滚操作失败。	请联系技术支持。
正常	[model 0.0.1] OBS桶，OBS并行文件系统，SFS Turbo挂载成功。 [%s] %s volume successfully.	-

服务部署和运行过程中，关键事件支持手动/自动刷新。

查看操作

1. 在ModelArts管理控制台的左侧导航栏中选择“模型部署 > 在线服务”，在服务列表中，您可以单击名称/ID，进入服务详情页面。
2. 在服务详情页面，切换到“事件”页签，查看事件信息。

9.7.3 管理在线服务生命周期

启动服务

您可以对处于“运行完成”、“异常”和“停止”状态的服务进行启动操作，“部署中”状态的服务无法启动。启动服务，当服务处于“运行中”状态后，ModelArts将开始计费。您可以通过如下方式启动服务：

- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。您可以单击“操作”列的“启动”，启动服务。

- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。单击目标服务名称，进入服务详情页面。您可以单击页面右上角“启动”，启动服务。

说明

部署方式为ModelArts边缘节点和ModelArts边缘资源池的服务不支持启动。

停止服务

停止服务，ModelArts将停止计费。您可以通过如下方式停止服务：

- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>在线服务”，进入在线服务管理页面。您可以单击“操作”列的“更多 > 停止”，停止服务。
- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>在线服务”，进入在线服务管理页面。单击目标服务名称，进入服务详情页面。您可以单击页面右上角“停止”，停止正在运行中服务。

说明

部署方式为ModelArts边缘节点和ModelArts边缘资源池的服务不支持停止。

删除服务

如果服务不再使用，您可以删除服务释放资源。

登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>在线服务”，进入在线服务管理页面。

- 单击在线服务列表“操作”列的“更多>删除”删除服务。
- 勾选在线服务列表中的服务，然后单击列表左上角“删除”按钮，批量删除服务。
- 单击目标服务名称，进入服务详情页面，单击右上角“删除”删除服务。

说明

- 删除操作无法恢复，请谨慎操作。
- 没有委托授权时，无法删除服务。
- 如果在线服务开启了“运行日志输出”，删除服务时，推荐同时删除LTS中的日志以及日志流，避免LTS日志流超过限额产生额外费用，如后续不再使用，建议删除。

重启服务

只有当在线服务处于“运行中”或“告警”状态时，才可进行重启操作。批量服务、边缘服务不支持重启。您可以通过如下方式重启在线服务：

- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>在线服务”，进入在线服务列表页面。您可以单击“操作”列的“更多>重启”，重启服务。
- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>在线服务”，进入在线服务列表页面。单击目标服务名称，进入服务详情页面。您可以单击页面右上角“重启”，重启在线服务。

说明

部署方式为ModelArts边缘节点和ModelArts边缘资源池的服务不支持重启。

9.7.4 修改在线服务配置

对于已部署的服务，您可以修改服务的基本信息以匹配业务变化，更换模型的版本号，实现服务升级。

您可以通过如下两种方式修改服务的基本信息：

方式一：通过服务管理页面修改服务信息

方式二：通过服务详情页面修改服务信息

前提条件

服务已部署成功，“部署中”的服务不支持修改服务信息进行升级。

约束限制

- 服务升级关系着业务实现，不当的升级操作会导致升级期间业务中断的情况，请谨慎操作。
- ModelArts支持部分场景下在线服务进行无损滚动升级。按要求进行升级前准备，做好验证，即可实现业务不中断的无损升级。

表 9-34 支持无损滚动升级的场景

创建模型的元模型来源	服务使用的是公共资源池	服务使用的是专属资源池
从训练中选择元模型	不支持	不支持
从容器镜像中选择元模型	不支持	支持，创建模型的自定义镜像需要满足**创建模型的自定义镜像规范**。
从OBS中选择元模型	不支持	不支持

方式一：通过服务管理页面修改服务信息

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。
2. 在服务列表中，单击目标服务操作列的“修改”，修改服务基本信息，然后根据提示提交修改任务。

   当修改了服务的某些参数配置时，系统会自动重启服务使修改生效。在提交修改服务任务时，如果涉及重启，会有弹窗提醒。

   在线服务参数说明请参见**部署模型为在线服务**。修改在线服务还需要配置“最大无效实例数”设置并行升级的最大节点数，升级阶段节点无效。

方式二：通过服务详情页面修改服务信息

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。
2. 单击目标服务名称，进入服务详情页面。

3. 您可以通过单击页面右上角“修改”，修改服务基本信息，然后根据提示提交修改任务。

   当修改了服务的某些参数配置时，系统会自动重启服务使修改生效。在提交修改服务任务时，如果涉及重启，会有弹窗提醒。

   在线服务参数说明请参见**部署模型为在线服务**。修改在线服务还需要配置“最大无效实例数”设置并行升级的最大节点数，升级阶段节点无效。

9.7.5 在云监控平台查看在线服务性能指标

ModelArts 支持的监控指标

为使用户更好地掌握自己的ModelArts在线服务和对应模型负载的运行状态，云服务平台提供了云监控。您可以使用该服务监控您的ModelArts在线服务和对应模型负载，执行自动实时监控、告警和通知操作，帮助您更好地了解服务和模型的各项性能指标。

表 9-35 ModelArts 支持的监控指标

指标ID	指标名称	指标含义	取值范围	测量对象	监控周期
cpu_usage	CPU使用率	该指标用于统计ModelArts用户服务的CPU使用率。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
mem_usage	内存使用率	该指标用于统计ModelArts用户服务的内存使用率。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
gpu_util	GPU使用率	该指标用于统计ModelArts用户服务的GPU使用情况。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
gpu_mem_usage	GPU显存使用率	该指标用于统计ModelArts用户服务的GPU显存使用情况。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
npu_util	NPU使用率	该指标用于统计ModelArts用户服务的NPU使用情况。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟

指标ID	指标名称	指标含义	取值范围	测量对象	监控周期
npu_mem_usage	NPU显存使用率	该指标用于统计ModelArts用户服务的NPU显存使用情况。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
successfully_called_times	调用成功次数	统计ModelArts用户调用服务的成功次数。 单位：次/分钟。	≥Count/min	ModelArts模型负载 ModelArts在线服务	1分钟
failed_called_times	调用失败次数	统计ModelArts用户调用服务的失败次数。 单位：次/分钟。	≥Count/min	ModelArts模型负载 ModelArts在线服务	1分钟
total_called_times	调用总次数	统计ModelArts用户调用服务的次数。 单位：次/分钟。	≥Count/min	ModelArts模型负载 ModelArts在线服务	1分钟
disk_read_rate	磁盘读取速率	统计ModelArts用户服务的磁盘读取速率 单位：bit/min	≥bit/min	ModelArts模型负载	1分钟
disk_write_rate	磁盘写入速率	统计ModelArts用户服务的磁盘写入速率 单位：bit/min	≥bit/min	ModelArts模型负载	1分钟
send_bytes_rate	上行速率	统计ModelArts用户服务的 出方向网络流速。 单位：bit/min	≥bit/min	ModelArts模型负载	1分钟
recv_bytes_rate	下行速率	统计ModelArts用户服务的 入方向网络流速。	≥bit/min	ModelArts模型负载	1分钟
req_count_2xx	2xx响应次数	统计api接口2xx响应的次数	≥Count/min	ModelArts在线服务	1分钟
req_count_4xx	4xx异常次数	统计api接口返回4xx错误的次数	≥Count/min	ModelArts在线服务	1分钟
req_count_5xx	5xx异常次数	统计api接口返回5xx错误的次数	≥Count/min	ModelArts在线服务	1分钟

指标ID	指标名称	指标含义	取值范围	测量对象	监控周期
avg_latency	平均延迟毫秒数	统计api接口平均响应延时时间	≥ms	ModelArts在线服务	1分钟
tp_99	TP99	上1分钟内，统计该请求每次响应所消耗的时间，并将这些时间按从小到大的顺序进行排序，取第99%的值作为TP99的值。	≥ms	ModelArts在线服务	1分钟
tp_999	TP99.9	上1分钟内，统计该请求每次响应所消耗的时间，并将这些时间按从小到大的顺序进行排序，取第99.9%的值作为TP99.9的值。	≥ms	ModelArts在线服务	1分钟

对于有多个测量维度的测量对象，使用接口查询监控指标时，所有测量维度均为必选。

- 查询单个监控指标时，多维度dim使用样例：
  dim.0=service_id,530cd6b0-86d7-4818-837f-935f6a27414d&dim.1="model_id,3773b058-5b4f-4366-9035-9bbd9964714a。
- 批量查询监控指标时，多维度dim使用样例：
  "dimensions": [
  {
  "name": "service_id",
  "value": "530cd6b0-86d7-4818-837f-935f6a27414d"
  }
  {
  "name": "model_id",
  "value": "3773b058-5b4f-4366-9035-9bbd9964714a"
  }
  ]

表 9-36 维度说明

Key	Value
service_id	在线服务ID。
model_id	模型负载ID。

设置告警规则

通过设置ModelArts在线服务和模型负载告警规则，用户可自定义监控目标与通知策略，及时了解ModelArts在线服务和模型负载状况，从而起到预警作用。

设置ModelArts服务和模型的告警规则包括设置告警规则名称、监控对象、监控指标、告警阈值、监控周期和是否发送通知等参数。本节介绍了设置ModelArts服务和模型告警规则的具体方法。

说明

只有“运行中”的在线服务，支持对接CES监控。

前提条件：

- 已创建ModelArts在线服务。
- 已在云监控服务创建ModelArts监控服务。登录“云监控服务”控制台，在“自定义监控”页面，根据界面提示创建ModelArts监控服务。

设置告警规则有多种方式。您可以根据实际应用场景，选择设置告警规则的方式。

- 对ModelArts服务设置告警规则
- 对单个服务设置告警规则
- 对模型版本设置告警规则
- 对服务或模型版本的单个指标设置告警规则

方式一：对整个ModelArts服务设置告警规则

步骤1 登录管理控制台。

步骤2 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。

步骤3 在左侧导航栏，选择“告警 > 告警规则”页面，单击“创建告警规则”。

步骤4 在“创建告警规则”页面，“资源类型”选择“ModelArts”，“维度”选择“服务”，“触发规则”选择“自定义创建”，设置告警策略，完成其他信息填写后，单击“立即创建”。

----结束

方式二：对单个服务设置告警规则

步骤1 登录管理控制台。

步骤2 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。

步骤3 在左侧导航栏，选择“云服务监控 > ModelArts”。

步骤4 选择需要添加告警规则的在线服务名称，单击操作列的“创建告警规则”。

步骤5 在“创建告警规则”界面，根据界面提示设置ModelArts在线服务和模型负载的告警规则。

----结束

方式三：对单个版本设置告警规则

步骤1 登录管理控制台。

步骤2 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。

步骤3 在左侧导航栏，选择“云服务监控 > ModelArts”。

步骤4 单击在线服务名称前面的小三角，展示模型版本列表，选择需要设置告警规则的模型版本，单击操作列的“创建告警规则”。

步骤5 在“创建告警规则”界面，根据界面提示设置模型负载的告警规则。

----结束

方式四：对服务或模型版本的单个指标设置告警规则

步骤1 登录管理控制台。

步骤2 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。

步骤3 在左侧导航栏，选择“云服务监控 > ModelArts”。

步骤4 单击在线服务名称或单击在线服务名称前面的小三角，展示模型版本列表，单击模型版本名称，查看告警规则详情。

步骤5 在告警规则详情页，单击单个指标右上角的加号按钮，对服务或模型版本的单个指标设置告警规则。

----结束

查看监控指标

云服务平台提供的云监控，可以对ModelArts在线服务和模型负载运行状态进行日常监控。您可以通过管理控制台，直观地查看ModelArts在线服务和模型负载的各项监控指标。由于监控数据的获取与传输会花费一定时间，因此，云监控显示的是当前时间5～10分钟前的状态。如果您的在线服务刚创建完成，请等待5～10分钟后查看监控数据。

前提条件：

- ModelArts在线服务正常运行。
- 已在云监控页面设置告警规则，具体操作请参见**设置告警规则**。
- 在线服务已正常运行一段时间（约10分钟）。
- 对于新创建的在线服务，需要等待一段时间，才能查看上报的监控数据和监控视图。
- 故障、删除状态的在线服务，无法在云监控中查看其监控指标。当在线服务再次启动或恢复后，即可正常查看。

对接云监控之前，用户无法查看到未对接资源的监控数据。具体操作，请参见**设置告警规则**。

步骤1 登录管理控制台。

步骤2 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。

步骤3 在左侧导航栏，选择“云服务监控 > ModelArts”。

步骤4 查看监控图表。

- 查看在线服务监控图表：单击目标在线服务“操作”列的“查看监控指标”。
- 查看模型负载监控图标：单击目标在线服务左侧的，在下拉列表中选择模型负载“操作”列的“查看监控指标”。

步骤5 在监控区域，您可以通过选择时长，查看对应时间的监控数据。

当前支持查看近1小时、近3小时和近12小时的监控数据，查看更长时间范围监控曲线，请在监控视图中单击进入大图模式查看。

----结束

9.7.6 集成在线服务 API 至生产环境中应用

针对已完成调测的API，可以将在线服务API集成至生产环境中应用。

前提条件

确保在线服务一直处于“运行中”状态，否则会导致生产环境应用不可用。

集成方式

ModelArts在线服务提供的API是一个标准的Restful API，可使用HTTPS协议访问。ModelArts提供了SDK用于调用在线服务API，SDK调用方式请参见《**SDK参考**》>“场景1：部署在线服务Predictor的推理预测”。

除此之外，您还可以使用常见的开发工具及开发语言调用此接口，建议通过互联网搜索并获取调用标准Restful API的指导。

9.7.7 设置在线服务故障自动重启

场景描述

当系统检测到Snt9b硬件故障时，自动复位Snt9B芯片并重启推理在线服务，提升了推理在线服务的恢复速度。

约束限制

仅支持使用Snt9b资源的同步在线服务。

只支持针对整节点资源复位，请确保部署的在线服务为8*N卡规格，请谨慎评估对部署在该节点的其他服务的影响。

开启故障自动重启

用户可以在部署在线服务任务时，勾选“高级选项”的“现在配置”，可以看到“故障自动重启”参数，打开开关即可。

9.8 管理批量推理作业

9.8.1 查看批量服务详情

当模型部署为批量服务成功后，您可以进入“批量服务”页面，来查看服务详情。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>批量服务”，进入“批量服务”管理页面。
2. 单击目标服务名称，进入服务详情页面。
   您可以查看服务的“名称”、“状态”等信息，详情说明请参见**表9-37**。

   表 9-37 批量服务参数

参数	说明
名称	批量服务名称。
服务ID	批量服务的ID。
状态	批量服务当前状态。
任务ID	批量服务的任务ID。
实例规格	批量服务的节点规格。
实例数	批量服务的节点个数。
任务开始时间	本次批量服务的任务开始时间。
环境变量	批量服务创建时填写的环境变量。
任务结束时间	本次批量服务的任务结束时间。
描述	您可以单击编辑按钮，添加服务描述。
输入数据目录位置	本次批量服务中，输入数据的OBS路径。
输出数据目录位置	本次批量服务中，输出数据的OBS路径。
模型名称&版本	本次批量服务所使用的模型名称及版本。
运行日志输出	默认关闭，批量服务的运行日志仅存放在ModelArts日志系统。 启用运行日志输出后，批量服务的运行日志会输出存放到云日志服务LTS。LTS自动创建日志组和日志流，默认缓存7天内的运行日志。如需了解LTS专业日志管理功能，请参见**云日志服务**。 **说明** • “运行日志输出”开启后，不支持关闭。 • LTS服务提供的日志查询和日志存储功能涉及计费，详细请参见**了解LTS的计费规则**。 • 请勿打印无用的audio日志文件，这会导致系统日志卡死，无法正常显示日志，可能会出现“Failed to load audio”的报错。

3. 您可以进入批量服务的详情页面，通过切换页签查看更多详细信息，详情说明请参见**表9-38**。

表 9-38 批量服务页签

参数	说明
事件	展示当前服务使用过程中的关键操作，比如服务部署进度、部署异常的详细原因、服务被启动、停止、更新的时间点等。 事件保存周期为1个月，1个月后自动清理数据。 查看服务的事件类型和事件信息，请参见**查看在线服务的事件**
日志	展示当前服务下每个模型的日志信息。包含最近5分钟、最近30分钟、最近1小时和自定义时间段。 自定义时间段您可以选择开始时间和结束时间。 当服务启用运行日志输出后，页面展示存放到云日志服务LTS中的日志信息。您可以单击“到LTS查看完整日志”查看全量的日志。 日志搜索规则说明： • 不支持带有分词符的字符串搜索（当前默认分词符有 ,'";=()[]{}@&<>/:\n\t\r）。 • 支持关键词精确搜索。关键词指相邻两个分词符之间的单词。 • 支持关键词模糊匹配搜索，例如输入“error”或“er?or”或“rro*”或“er*r”。 • 支持短语精确搜索。例如输入“Start to refresh”。 • 启用运行日志输出前，支持关键词的“与”、“或”组合搜索。格式为“query logs&&erro*”或“query logs||erro*”。启用运行日志输出后，支持关键词的“与”、“或”组合搜索。格式为“query logs AND erro*”或“query logs OR erro*”。

9.8.2 查看批量服务的事件

服务的（从用户可看见部署服务任务开始）整个生命周期中，每一个关键事件点在系统后台均有记录，用户可随时在对应服务的详情页面进行查看。

方便用户更清楚的了解服务部署和运行过程，遇到任务异常时，更加准确的排查定位问题。可查看的事件点包括：

表 9-39 事件

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	开始部署服务。	-
异常	资源不足，等待资源释放。	等待资源释放后重试。

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
异常	xxx资源不足，服务调度失败。补充信息：xxx	根据补充信息，了解资源不足详情，参考FAQ处理。
正常	开始构建镜像。	-
异常	构建模型(xxx) 镜像失败，构建日志:\nxxx。	根据构建日志定位和处理问题。
异常	构建镜像失败。	请联系技术支持。
正常	构建镜像完成。	-
异常	xxx服务失败。错误信息：xxx	请根据错误信息定位和处理问题。
异常	更新服务失败，执行回滚操作。	请联系技术支持。
正常	服务更新中。	-
正常	服务启动中。	-
正常	服务停止中。	-
正常	服务已停止。	-
正常	自动停止开关已关闭。	-
正常	自动关闭功能开启，服务将在xs后停止。	-
正常	到达自动停止时间，服务停止。	-
异常	配额超限，服务停止。	请联系技术支持。
异常	自动停止服务失败，错误信息: xxx	请根据错误信息定位和处理问题。
正常	删除资源池(xxx)上服务实例。	-
正常	停止资源池(xxx)上服务实例。	-
异常	批量服务失败，请稍后重试。错误信息：xxx	请根据错误信息定位和处理问题。
正常	服务运行完成。	-
异常	停止服务失败，错误信息：xxx	请根据错误信息定位和处理问题。

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	订阅许可即将超期：xxx	-
正常	服务xxx启动成功。	-
异常	启动服务xxx失败。	启动服务失败情况较多，请参考FAQ定位和处理。
异常	部署服务超时，错误信息：xxx	请根据错误信息定位和处理问题。
正常	更新服务失败，执行回滚操作成功。	-
异常	更新服务失败，执行回滚操作失败。	请联系技术支持。
正常	[model 0.0.1] OBS桶，OBS并行文件系统，SFS Turbo挂载成功。 [%s] %s volume successfully.	-

服务部署和运行过程中，关键事件支持手动/自动刷新。

查看操作

1. 在ModelArts管理控制台的左侧导航栏中选择“模型部署 > 批量服务”，在服务列表中，您可以单击名称/ID，进入服务详情页面。
2. 在服务详情页面，切换到“事件”页签，查看事件信息。

9.8.3 管理批量服务生命周期

启动服务

您可以对处于“运行完成”、“异常”和“停止”状态的服务进行启动操作，“部署中”状态的服务无法启动。启动服务，当服务处于“运行中”状态后，ModelArts将开始计费。您可以通过如下方式启动服务：

- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。您可以单击“操作”列的“启动”，启动服务。
- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。单击目标服务名称，进入服务详情页面。您可以单击页面右上角“启动”，启动服务。

说明

部署方式为ModelArts边缘节点和ModelArts边缘资源池的服务不支持启动。

停止服务

停止服务，ModelArts将停止计费。您可以通过如下方式停止服务：

- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>批量服务”，进入批量服务管理页面。您可以单击“操作”列的“停止”，停止服务。
- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>批量服务”，进入批量服务管理页面。单击目标服务名称，进入服务详情页面。您可以单击页面右上角“停止”，停止正在运行中服务。

  说明

  部署方式为ModelArts边缘节点和ModelArts边缘资源池的服务不支持停止。

删除服务

如果服务不再使用，您可以删除服务释放资源。

登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>批量服务”，进入批量服务管理页面。

- 单击批量服务列表“操作”列的“删除”，删除服务。
- 勾选批量服务列表中的服务，然后单击列表左上角“删除”按钮，批量删除服务。
- 单击目标服务名称，进入服务详情页面，单击右上角“删除”按钮进行删除。

说明

- 删除操作无法恢复，请谨慎操作。
- 没有委托授权时，无法删除服务。

重启服务

批量服务不支持重启。

9.8.4 修改批量服务配置

对于已部署的服务，您可以修改服务的基本信息以匹配业务变化，更换模型的版本号，实现服务升级。

您可以通过如下两种方式修改服务的基本信息：

方式一：通过服务管理页面修改服务信息

方式二：通过服务详情页面修改服务信息

前提条件

服务已部署成功，“部署中”的服务不支持修改服务信息进行升级。

约束限制

- 服务升级关系着业务实现，不当的升级操作会导致升级期间业务中断的情况，请谨慎操作。
- ModelArts支持部分场景下在线服务进行无损滚动升级。按要求进行升级前准备，做好验证，即可实现业务不中断的无损升级。

表 9-40 支持无损滚动升级的场景

创建模型的元模型来源	服务使用的是公共资源池	服务使用的是专属资源池
从训练中选择元模型	不支持	不支持
从容器镜像中选择元模型	不支持	支持，创建模型的自定义镜像需要满足**创建模型的自定义镜像规范**。
从OBS中选择元模型	不支持	不支持

方式一：通过服务管理页面修改服务信息

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。
2. 在服务列表中，单击目标服务操作列的“修改”，修改服务基本信息，然后根据提示提交修改任务。

   当修改了服务的某些参数配置时，系统会自动重启服务使修改生效。在提交修改服务任务时，如果涉及重启，会有弹窗提醒。批量服务参数说明请参见**将模型部署为批量推理服务**。

方式二：通过服务详情页面修改服务信息

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。
2. 单击目标服务名称，进入服务详情页面。
3. 您可以通过单击页面右上角“修改”，修改服务基本信息，然后根据提示提交修改任务。

   当修改了服务的某些参数配置时，系统会自动重启服务使修改生效。在提交修改服务任务时，如果涉及重启，会有弹窗提醒。批量服务参数说明请参见**将模型部署为批量推理服务**。

10 制作自定义镜像用于 ModelArts Standard

10.1 自定义镜像使用场景

在AI业务开发以及运行的过程中，一般都会有复杂的环境依赖需要进行调测并固化。面对开发中的开发环境的脆弱和多轨切换问题，在ModelArts的AI开发最佳实践中，通过容器镜像的方式将运行环境进行固化，以这种方式不仅能够进行依赖管理，而且可以方便的完成工作环境切换。配合ModelArts提供的云化容器资源使用，可以更加快速、高效地进行AI开发与模型实验的迭代等。

本章节会先介绍镜像相关概念，然后介绍预置镜像和自定义镜像使用场景，并且提供自定义镜像制作的相关指导。

概念解释

预置镜像：ModelArts提供的镜像，可以在创建Notebook/训练作业/模型时，直接选择ModelArts提供的镜像。

ModelArts提供的预置镜像分为两种：

- 统一镜像：适用创建Notebook/训练作业/模型，后续新上线的镜像都为统一镜像。对应章节**ModelArts统一镜像列表**。
- 各模块独有的镜像：仅适用单个模块（例如训练的预置镜像只能用于训练），此类镜像为ModelArts早期的镜像，后续会陆续下线。对应章节**Notebook专属预置镜像列表**、**训练专属预置镜像列表**、**推理专属预置镜像列表**。

自定义镜像：用户参照ModelArts镜像规范制作的镜像。

基础镜像：镜像制作的一个基本概念，先有基础镜像然后在此基础上做镜像。基础镜像可以是ModelArts预置镜像、第三方镜像。

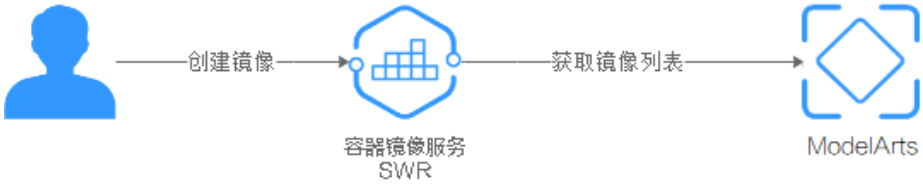
自定义镜像功能关联服务介绍

- 容器镜像服务

  容器镜像服务（Software Repository for Container，SWR）是一种支持镜像全生命周期管理的服务， 提供简单易用、安全可靠的镜像管理功能，帮助您快速部署容器化服务。您可以通过界面、社区CLI和原生API上传、下载和管理容器镜像。

  您制作的自定义镜像需要上传至SWR服务。ModelArts开发环境、训练和创建模型使用的自定义镜像需要从SWR服务管理列表获取。

  图 10-1 获取镜像列表

- 对象存储服务

  对象存储服务（Object Storage Service，OBS）是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力。

  在使用ModelArts时存在与OBS的数据交互，您需要使用的数据可以存储至OBS。

- 弹性云服务器

  弹性云服务器（Elastic Cloud Server，ECS）是由CPU、内存、操作系统、云硬盘组成的基础的计算组件。弹性云服务器创建成功后，您就可以像使用自己的本地PC或物理服务器一样，使用弹性云服务器。

  在制作自定义镜像时，您可以在本地环境或者ECS上完成自定义镜像制作。

说明

在您使用自定义镜像功能时，ModelArts可能需要访问您的容器镜像服务SWR、对象存储服务OBS等依赖服务，如果没有授权，这些功能将不能正常使用。建议您使用委托授权功能，将依赖服务操作权限委托给ModelArts服务，让ModelArts以您的身份使用依赖服务，代替您进行一些资源操作。详细操作参见使用**委托授权**。

ModelArts 的预置镜像使用场景

ModelArts给用户提供了一组预置镜像，用户可以直接使用预置镜像创建Notebook实例，在实例中进行依赖安装与配置后，保存为自定义镜像，可直接用于ModelArts训练，而不需要做适配。同时也可以使用预置镜像直接提交训练作业、创建模型等。

ModelArts提供的预置镜像版本是依据用户反馈和版本稳定性决定的。当用户的功能开发基于ModelArts提供的版本能够满足的时候，比如用户开发基于MindSpore1.X，建议用户使用预置镜像，这些镜像经过充分的功能验证，并且已经预置了很多常用的安装包，用户无需花费过多的时间来配置环境即可使用。

ModelArts默认提供了一组预置镜像供开发使用，这些镜像有以下特点：

- 零配置，即开即用，面向特定的场景，将AI开发过程中常用的依赖环境进行固化，提供合适的软件、操作系统、网络等配置策略，通过在硬件上的充分测试，确保其兼容性和性能最合适。

- 方便自定义，预置镜像已经在SWR仓库中，通过对预置镜像的扩展完成自定义镜像注册。
- 安全可信，基于安全加固最佳实践，访问策略、用户权限划分、开发软件漏洞扫描、操作系统安全加固等方式，确保镜像使用的安全性。

ModelArts 的自定义镜像使用场景

当用户对深度学习引擎、开发库有特殊需求场景的时候，预置镜像已经不能满足用户需求。ModelArts提供自定义镜像功能支持用户自定义运行引擎。

ModelArts底层采用容器技术，自定义镜像指的是用户自行制作容器镜像并在ModelArts上运行。自定义镜像功能支持自由文本形式的命令行参数和环境变量，灵活性比较高，便于支持任意计算引擎的作业启动需求。

在制作自定义镜像的时候，可以把ModelArts提供的预置镜像作为基础镜像，通过在Dockerfile中使用预置镜像的SWR地址来拉取预置镜像后进行改造。可在ModelArts预置镜像列表里获取镜像的SWR地址，参考**ModelArts支持的预置镜像列表**章节。

- **制作自定义镜像用于创建Notebook**

  当Notebook预置镜像不能满足需求时，用户可以制作自定义镜像。在镜像中自行安装与配置环境依赖软件及信息，并制作为自定义镜像，用于创建新的Notebook实例。同时也支持用户在Notebook中，基于已有镜像制作新的自定义镜像。
- **制作自定义镜像用于训练模型**

  如果您已经在本地完成模型开发或训练脚本的开发，且您使用的AI引擎是ModelArts不支持的框架。您可以制作自定义镜像，并上传至SWR服务。您可以在ModelArts使用此自定义镜像创建训练作业，使用ModelArts提供的资源训练模型。
- **制作自定义镜像用于推理**

  如果您使用了ModelArts不支持的AI引擎开发模型，可以通过制作自定义镜像，导入ModelArts创建为模型，并支持进行统一管理和部署为服务。

10.2 ModelArts 支持的预置镜像列表

10.2.1 ModelArts 预置镜像更新说明

本章节提供了ModelArts预置镜像的变更说明 ，比如依赖包的变化，方便用户感知镜像能力的差异，减少镜像使用问题。

统一镜像更新说明

表 10-1 统一镜像更新说明

镜像名称	更新时间	更新说明
mindspore_2.3.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	2024-05-21	基于昇腾415商发版本，mindspore更新至2.3.0-rc4，cann更新至8.0.rc1 下线ma-cau 1.1.6、ma-cau-adapter 1.1.3

镜像名称	更新时间	更新说明
pytorch_2.1.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	2024-05-21	基于昇腾415商发版本，cann更新至8.0.rc1
pytorch_1.11.0-cann_8.0.rc1-py_3.9-euler_2.10.7-aarch64-snt9b	2024-05-21	基于昇腾415商发版本，cann更新至8.0.rc1
mindspore_2.3.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b	2024-07-27	基于昇腾715商发版本，mindspore更新至2.3.0，cann更新至8.0.rc2，配套驱动Ascend HDK 24.1.RC2
pytorch_2.1.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b	2024-07-27	基于昇腾715商发版本，cann更新至8.0.rc2，配套驱动Ascend HDK 24.1.RC2
pytorch_1.11.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b	2024-07-27	基于昇腾715商发版本，cann更新至8.0.rc2，配套驱动Ascend HDK 24.1.RC2

10.2.2 ModelArts 统一镜像列表

统一镜像列表

ModelArts提供了ARM+Ascend规格的统一镜像，包括MindSpore、PyTorch。适用于开发环境，模型训练，服务部署，请参考**统一镜像列表**。

表 10-2 MindSpore

预置镜像	适配芯片	适用范围
mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook、训练、推理部署
mindspore_2.3.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook、训练、推理部署

表 10-3 PyTorch

预置镜像	适配芯片	适用范围
pytorch_2.1.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook、训练、推理部署
pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook、训练、推理部署
pytorch_2.1.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook、训练、推理部署

预置镜像	适配芯片	适用范围
pytorch_1.11.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b	Ascend snt9b	Notebook、训练、推理部署

mindspore_2.3.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b

表 10-4 mindspore_2.3.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b

AI引擎框架	URL	包含的依赖项	
mindspore 2.3.0 + mindspore-lite 2.3.0 + Ascend CANN Toolkit 8.0.rc2	swr.<region>.myhuaweicloud.com/atelier/mindspore_2_3_ascend:mindspore_2.3.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b-20240727152329-0f2c29a 例如： 中国-香港 swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_2_3_ascend:mindspore_2.3.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b-20240727152329-0f2c29a	PyPI 程序包	Yum 软件包
		mindspore 2.3.0 mindspore-lite 2.3.0 mindinsight 2.3.0 mindarmour 2.0.0 mindformers 1.2.0 seccomponent 1.1.8 moxing-framework 2.2.8.0aa484aa ipykernel 6.7.0 ipython 8.18.1 jupyter-client 7.4.9 matplotlib 3.5.1 numpy 1.22.0 pandas 1.3.5 Pillow 10.0.1 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.4	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

pytorch_2.1.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b

表 10-5 pytorch_2.1.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b

AI引擎框架	URL	包含的依赖项	
pytorch 2.1.0 + mindspore-lite 2.3.0 + Ascend CANN Toolkit 8.0.rc2	swr.<region>.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	PyPI 程序包	Yum 软件包
		torch 2.1.0 torch-npu 2.1.0.post6.dev20240716 mindspore-lite 2.3.0 seccomponent 1.1.8 moxing-framework 2.2.8.0aa484aa ipykernel 6.7.0 ipython 8.18.1 jupyter-client 7.4.9 ma-cau 1.1.7 ma-cau-adapter 1.1.3 ma-cli 1.2.3 matplotlib 3.5.1 numpy 1.22.0 pandas 1.3.5 Pillow 10.0.1 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.4	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

pytorch_1.11.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b

表 10-6 pytorch_1.11.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b

AI引擎框架	URL	包含的依赖项	
pytorch 1.11 + mindspore-lite 2.3.0 + Ascend CANN Toolkit 8.0.rc2	swr.<region>.myhuaweicloud.com/atelier/pytorch_1_11_ascend:pytorch_1.11.0-cann_8.0.rc2-py_3.9-euler_2.10.7-aarch64-snt9b-20240727152329-0f2c29a	PyPI 程序包	Yum 软件包
		torch 1.11.0 torch-npu 1.11.0.post14.dev20240716 apex 0.1.dev20240716+ascend mindspore-lite 2.3.0 seccomponent 1.1.8 moxing-framework 2.2.8.0aa484aa ipykernel 6.7.0 ipython 8.18.1 jupyter-client 7.4.9 ma-cau 1.1.7 ma-cau-adapter 1.1.3 ma-cli 1.2.3 matplotlib 3.5.1 numpy 1.22.0 pandas 1.3.5 Pillow 10.3.0 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.4	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b

表 10-7 mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b

AI引擎框架	URL	包含的依赖项	
mindspore 2.2.0 + mindspore-lite 2.2.0 + Ascend CANN Toolkit 7.0.RC1	swr.<region>.myhuaweicloud.com/atelier/mindspore_2_2_ascend:mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83 例如： 中国-香港 swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/mindspore_2_2_ascend:mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83	PyPI 程序包	Yum 软件包
		mindspore 2.2.0 ipykernel 6.7.0 ipython 8.18.1 jupyter-client 7.4.9 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.2.3.2c7f2141 numpy 1.22.0 pandas 1.3.5 Pillow 10.0.1 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.4 mindinsight 2.2.0	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

pytorch_2.1.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b

表 10-8 pytorch_2.1.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b 镜像介绍

AI引擎框架	URL	包含的依赖项	
pytorch 2.1.0 + mindspore-lite 2.2.0 + Ascend CANN Toolkit 7.0.RC1	swr.<region>.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83 例如： 中国-香港 swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83	PyPI 程序包	Yum 软件包
		torch 2.1.0 apex 0.1-ascend-20231013 torch-npu 2.1.0rc1.post20231013 ipykernel 5.3.4 ipython 7.34.0 jupyter-client 7.3.4 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.26 moxing-framework 2.2.3.2c7f2141 numpy 1.26.1 pandas 1.3.5 Pillow 10.0.1 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.3.3	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b

表 10-9 pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b 镜像介绍

AI引擎框架	URL	包含的依赖项	
pytorch 1.11 + mindspore-lite 2.2.0 + Ascend CANN Toolkit 7.0.RC1	swr.<region>.myhuaweicloud.com/atelier/pytorch_1_11_ascend:pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83 例如： 中国-香港 swr.cn-ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_11_ascend:pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83	PyPI 程序包	Yum 软件包
		torch 1.11.0 torch-npu 1.11.0.post4-20231013 apex 0.1-ascend-20231013 ipykernel 6.7.0 ipython 8.17.2 jupyter-client 7.4.9 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.20 moxing-framework 2.2.3.2c7f2141 numpy 1.26.1 pandas 1.3.5 Pillow 10.0.1 pip 21.0.1 psutil 5.9.5 PyYAML 6.0.1 scipy 1.10.1 scikit-learn 1.0.2 tornado 6.3.3	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

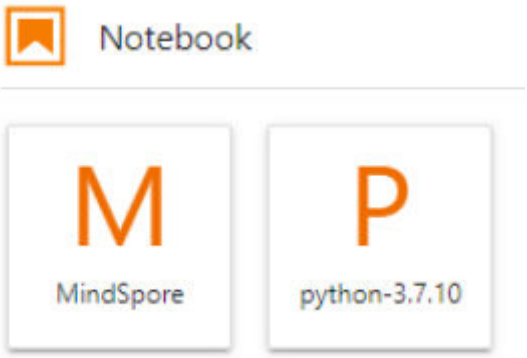
10.2.3 Notebook 专属预置镜像列表

ModelArts开发环境提供Docker容器镜像，可作为预构建容器运行。预置镜像里面包含PyTorch，Tensorflow，MindSpore等常用AI引擎框架，镜像命名以AI引擎为主，并且每个镜像里面都预置了很多常用包，用户可以直接使用而无需重新安装。

ModelArts开发环境提供的预置镜像主要包含：

- 常用预置包：基于标准的Conda环境，预置了常用的AI引擎，常用的数据分析软件包，例如Pandas，Numpy等，常用的工具软件，例如cuda，cudnn等，满足AI开发常用需求。

- 预置Conda环境：每个预置镜像都会创建一个相对应的Conda环境和一个基础Conda环境python（不包含任何AI引擎），如预置MindSpore所对应的Conda环境如下。



  用户可以根据是否使用AI引擎Mindspore参与功能调试，选择不同的Conda环境。
- Notebook：是一款Web应用，用户能够在界面编写代码，并且将代码、数学方程和可视化内容组合到一个文档中。
- JupyterLab插件：插件包括规格切换，分享案例到AI Gallery进行交流，停止实例（实例停止后CPU、Memory不再计费）等，提升用户体验。
- 支持SSH远程连接功能：通过SSH连接启动实例，在本地调试就可以操作实例，方便调试。
- 预置镜像支持功能开发：基于ModelArts预置镜像进行依赖安装配置后，保存为自定义镜像，能直接在ModelArts用于训练作业。

表 10-10 X86 预置镜像列表

引擎类型	镜像名称
PyTorch	pytorch1.8-cuda10.2-cudnn7-ubuntu18.04
	pytorch1.10-cuda10.2-cudnn7-ubuntu18.04
	pytorch1.4-cuda10.1-cudnn7-ubuntu18.04
Tensorflow	tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04
	tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04
MindSpore	mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04
	mindspore1.7.0-py3.7-ubuntu18.04
	mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04
	mindspore1.2.0-openmpi2.1.1-ubuntu18.04
无AI引擎（专用于自定义镜像的基础镜像）	conda3-cuda10.2-cudnn7-ubuntu18.04
	conda3-ubuntu18.04

Notebook 基础镜像 x86 PyTorch

PyTorch包含三种镜像：

- **镜像一：pytorch1.8-cuda10.2-cudnn7-ubuntu18.04**
- **镜像二：pytorch1.10-cuda10.2-cudnn7-ubuntu18.04**
- **镜像三：pytorch1.4-cuda10.1-cudnn7-ubuntu18.04**

镜像一：pytorch1.8-cuda10.2-cudnn7-ubuntu18.04

表 10-11 pytorch1.8-cuda10.2-cudnn7-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
PyTorch 1.8	是（cuda 10.2）	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e	PyPI 程序包	Ubuntu 软件包

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
			torch 1.8.0 torchvision 0.9.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.4 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 Pillow 9.3.0 pip 21.0.1 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 tensorboard 2.1.1	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx pandoc python3 rpm screen tar tmux unzip vim wget zip

镜像二：pytorch1.10-cuda10.2-cudnn7-ubuntu18.04

表 10-12 pytorch1.10-cuda10.2-cudnn7-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
Pytorch 1.10	是（cuda 10.2）	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_10:pytorch_1.10.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221008154718-2b3e39c	PyPI 程序包	Ubuntu 软件包

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
			torch 1.10.2 torchvision 0.11.3 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.4 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 Pillow 9.3.0 pip 21.0.1 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx pandoc python3 rpm screen tar tmux unzip vim wget zip

镜像三：pytorch1.4-cuda10.1-cudnn7-ubuntu18.04

表 10-13 pytorch1.4-cuda10.1-cudnn7-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
Pytorch 1.4	是（cuda 10.1）	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_4:pytorch_1.4-cuda_10.1-py37-ubuntu_18.04-x86_64-20220926104017-041ba2e	PyPI 程序包	Ubuntu 软件包

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
			torch 1.4.0 torchvision 0.5.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.7 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 Pillow 6.2.0 pip 21.0.1 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 tensorboard 2.1.1	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx pandoc python3 rpm screen tar tmux unzip vim wget zip

Notebook 基础镜像 x86 Tensorflow

Tensorflow包含两种镜像：

- **镜像一：tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04**
- **镜像二：tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04**

镜像一：tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04

表 10-14 tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
Tensorflow 2.1	是（cuda 10.1）	swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926144607-041ba2e	PyPI 程序包	Ubuntu 软件包

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
			tensorflow 2.1.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.4 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 Pillow 9.3.0 pip 21.0.1 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 tensorboard 2.1.1	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

镜像二：tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

表 10-15 tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
Tensorflow 1.13-gpu	是（cuda 10.0）	swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_1_13:tensorflow_1.13-cuda_10.0-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e	PyPI 程序包	Ubuntu 软件包

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
			tensorflow-gpu 1.13.1 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.6 ma-cli 1.2.3 matplotlib 3.5.2 modelarts 1.4.25 moxing-framework 2.0.1.rc0.ffd1c0c8 numpy 1.17.0 opencv-python 4.1.2.30 pandas 1.1.5 Pillow 6.2.0 pip 21.0.1 psutil 5.8.0 PyYAML 5.1 scipy 1.2.2 scikit-learn 0.22.1 tornado 6.2	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

Notebook 基础镜像 x86 MindSpore

MindSpore包含四种镜像：

- **镜像一：mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04**
- **镜像二：mindspore1.7.0-py3.7-ubuntu18.04**
- **镜像三：mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04**
- **镜像四：mindspore1.2.0-openmpi2.1.1-ubuntu18.04**

镜像一：mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04

表 10-16 mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
Mindspore-gpu 1.7.0	是（cuda 10.1）	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e	PyPI 程序包	Ubuntu 软件包

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
			mindspore-gpu 1.7.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.4 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 pandas 1.1.5 Pillow 9.3.0 pip 21.0.1 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 mindinsight 1.7.0 mindvision 0.1.0	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

镜像二：mindspore1.7.0-py3.7-ubuntu18.04

表 10-17 mindspore1.7.0-py3.7-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
Mindspore 1.7.0	无	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e	PyPI 程序包	Ubuntu 软件包

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
			mindspore 1.7.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.6 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 pandas 1.1.5 Pillow 9.3.0 pip 21.0.1 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 mindinsight 1.7.0 mindvision 0.1.0	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

镜像三：mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04

表 10-18 mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
Mindspore-gpu 1.2.0	是（cuda 10.1）	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_2_0:mindspore_1.2.0-py_3.7-cuda_10.1-ubuntu_18.04-x86_64-20220926104106-041ba2e	PyPI 程序包	Ubuntu 软件包

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
			mindspore-gpu 1.2.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.6 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 pandas 1.1.5 Pillow 6.2.0 pip 21.0.1 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 mindinsight 1.2.0	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

镜像四：mindspore1.2.0-openmpi2.1.1-ubuntu18.04

表 10-19 mindspore1.2.0-openmpi2.1.1-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
Mindspore 1.2.0	无	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_2_0:mindspore_1.2.0-py_3.7-ubuntu_18.04-x86_64-20220926104106-041ba2e	PyPI 程序包	Ubuntu 软件包

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
			mindspore 1.2.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.6 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 pandas 1.1.5 Pillow 6.2.0 pip 21.0.1 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 mindinsight 1.2.0	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

Notebook 基础镜像 x86 自定义专用镜像

自定义镜像包含两种镜像：conda3-cuda10.2-cudnn7-ubuntu18.04，conda3-ubuntu18.04，该类镜像是无AI引擎以及相关的软件包，镜像较小，只有2~5G。用户

使用此类镜像做基础镜像，安装自己需要的引擎版本和依赖包，可扩展性更高。并且这些镜像预置了一些开发环境启动所必要的配置，用户无需对此做任何适配，安装所需的软件包即可使用。

此类镜像为最基础的镜像，主要应对用户做自定义镜像时基础镜像太大的问题，所以镜像中未安装任何组件；如果需使用OBS SDK相关功能，推荐使用ModelArts SDK进行文件复制等操作，详细操作请参考**文件传输**。

镜像一：conda3-cuda10.2-cudnn7-ubuntu18.04

表 10-20 conda3-cuda10.2-cudnn7-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
无	是（cuda 10.2）	swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:cuda_10.2-ubuntu_18.04-x86_64-20221008154718-2b3e39c	PyPI 程序包	Ubuntu 软件包
			ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.9 ma-cli 1.2.3 matplotlib 3.5.2 modelarts 1.4.25 moxing-framework 2.1.6.879ab2f4 numpy 1.21.6 pandas 1.3.5 Pillow 9.5.0 pip 20.3.3 psutil 5.9.4 PyYAML 6.0 scipy 1.7.3 tornado 6.2	automake build-essential ca-certificates cmake cpp curl g++ gcc gfortran grep libcudnn7 libcudnn7-dev nginx python3 rpm tar unzip vim wget zip

镜像二：conda3-ubuntu18.04

表 10-21 conda3-ubuntu18.04 镜像介绍

AI引擎框架	是否使用GPU（CUDA版本）	URL	包含的依赖项	
无	否	swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c 华东-上海一 swr.cn-east-3.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c 华南-广州 swr.cn-south-1.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c	PyPI 程序包	Ubuntu 软件包
			ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.9 ma-cli 1.2.3 matplotlib 3.5.2 modelarts 1.4.25 moxing-framework 2.1.6.879ab2f4 numpy 1.21.6 pandas 1.3.5 Pillow 9.5.0 pip 20.3.3 psutil 5.9.4 PyYAML 6.0 scipy 1.7.3 tornado 6.2	automake build-essential ca-certificates cmake cpp curl g++ gcc gfortran grep nginx python3 rpm tar unzip vim wget zip

Notebook 基础镜像 ARM MindSpore

ARM MindSpore包含三种镜像：

- **镜像一：mindspore_1.10.0-cann_6.0.1-py_3.7-euler_2.8.3**
- **镜像二：mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3**

- 镜像三：mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3

镜像一：mindspore_1.10.0-cann_6.0.1-py_3.7-euler_2.8.3

表 10-22 mindspore_1.10.0-cann_6.0.1-py_3.7-euler_2.8.3 镜像介绍

AI引擎框架	URL	包含的依赖项	
Mindspore-Ascend 1.10.0	{region_id}.myhuaweicloud.com/atelier/mindspore_1_10_ascend:mindspore_1.10.0-cann_6.0.1-py_3.7-euler_2.8.3-aarch64-d910-20230303173945-815d627 例如： 华北-北京四 swr.cn-north-4.myhuaweicloud.com/atelier/mindspore_1_10_ascend:mindspore_1.10.0-cann_6.0.1-py_3.7-euler_2.8.3-aarch64-d910-20230303173945-815d627	PyPI 程序包	Yum 软件包
		mindspore-ascend 1.10.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.5 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.0.1.rc0.ffd1c0c8 numpy 1.21.2 pandas 1.1.3 Pillow 9.4.0 pip 21.0.1 psutil 5.7.0 PyYAML 5.3.1 scipy 1.5.4 scikit-learn 0.24.0 tornado 6.2 mindinsight 1.9.0	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

镜像二：mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3

表 10-23 mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3 镜像介绍

AI引擎框架	URL	包含的依赖项	
MindSpore 1.9.0	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_9_ascend:mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3-aarch64-d910-20221116111529	PyPI 程序包	Yum 软件包
		mindspore-ascend 1.9.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.4.5 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.0.1.rc0.ffd1c0c8 numpy 1.21.2 pandas 1.1.3 Pillow 9.3.0 pip 22.3.1 psutil 5.7.0 PyYAML 5.3.1 scipy 1.5.4 scikit-learn 0.24.0 tornado 6.2 mindinsight 1.9.0	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

镜像三：mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3

表 10-24 mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3 镜像介绍

AI引擎框架	URL	包含的依赖项	
Mindspore-Ascend 1.7.0	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220906	PyPI 程序包	Yum 软件包

AI引擎框架	URL	包含的依赖项	
		mindspore-ascend 1.7.0 ipykernel 5.3.4 ipython 7.34.0 jupyter-client 7.3.4 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.0.1.rc0.ffd1c0c8 numpy 1.21.2 pandas 1.1.3 Pillow 9.2.0 pip 22.1.2 psutil 5.7.0 PyYAML 5.3.1 scipy 1.5.4 scikit-learn 0.24.0 tornado 6.2 mindinsight 1.7.0	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

Notebook 基础镜像 ARM TenSorFlow

ARM TenSorFlow镜像包含两种：

- **镜像一：tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64**
- **镜像二：tensorflow1.15-cann5.1.0-py3.7-euler2.8.3**

镜像一：tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64

表 10-25 tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64 镜像介绍

AI引擎框架	URL	包含的依赖项	
Mindspore-Ascend 1.7.0	swr.{region_id}.myhuaweicloud.com/atelier/notebook2.0-mul-kernel-arm-ascend-cp37:5.0.1-c81-20220726	PyPI 程序包	Yum 软件包

AI引擎框架	URL	包含的依赖项	
		mindspore-ascend 1.7.0 ipykernel 6.7.0 ipython 7.29.0 jupyter-client 7.0.6 ma-cli 1.2.3 matplotlib 3.1.2 modelarts 1.4.25 moxing-framework 2.0.0.rc2.4b57a67b numpy 1.17.5 pandas 1.1.3 Pillow 7.0.0 pip 21.2.4 psutil 5.7.0 PyYAML 5.3.1 scipy 1.5.4 scikit-learn 0.24.0 tornado 6.1 mindinsight 1.7.0	cmake cpp curl ffmpeg g++ gcc git grep python3 rpm tar unzip wget zip

镜像二：tensorflow1.15-cann5.1.0-py3.7-euler2.8.3

表 10-26 tensorflow1.15-cann5.1.0-py3.7-euler2.8.3 镜像介绍

AI引擎框架	是否使用昇腾（CANN版本）	URL	包含的依赖项	
Tensorflow 1.15	是（CANN 5.1）	swr.{region-id}.{局点域名}/atelier/tensorflow_1_15_ascend:tensorflow_1.15-cann_5.1.0-py_3.7-euler_2.8.3-aarch64-d910-20220906	PyPI 程序包	Yum 软件包

AI引擎框架	是否使用昇腾（CANN版本）	URL	包含的依赖项	
			tensorflow 1.15.0 tensorboard 1.15.0 ipykernel 5.3.4 ipython 7.34.0 jupyter-client 7.3.4 ma-cli 1.2.3 matplotlib 3.5.1 modelarts 1.4.25 moxing-framework 2.0.1.rc0.ffd1c0c8 numpy 1.17.5 pandas 0.24.2 Pillow 9.2.0 pip 22.1.2 psutil 5.7.0 PyYAML 5.3.1 scipy 1.3.3 scikit-learn 0.20.0 tornado 6.2	ca-certificates.noarch cmake cpp curl gcc-c++ gcc gdb grep nginx python3 rpm tar unzip vim wget zip

10.2.4 训练专属预置镜像列表

ModelArts平台提供了Tensorflow，PyTorch，MindSpore等常用深度学习任务的基础镜像，镜像里已经安装好运行任务所需软件。当基础镜像里的软件无法满足您的程序运行需求时，您可以基于这些基础镜像制作一个新的镜像并进行训练。

训练基础镜像列表

ModelArts中预置的训练基础镜像如下表所示。

表 10-27 ModelArts 训练基础镜像列表

引擎类型	版本名称
PyTorch	pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
TensorFlow	tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64
Horovod	horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64
	horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
MPI	mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

说明

不同区域支持的AI引擎有差异，请以实际环境为准。

训练基础镜像详情（PyTorch）

介绍预置的PyTorch镜像详情。

引擎版本：pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

引擎版本：pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region}.myhuaweicloud.com/aip/pytorch_1_8:train-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-roma-20220309171256-40adcc1
- 镜像构建时间：20220309171256 (yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.2.89
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python, python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages

训练基础镜像详情（TensorFlow）

介绍预置的TensorFlow镜像详情。

- **引擎版本：tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**

引擎版本：tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region}.myhuaweicloud.com/aip/tensorflow_2_1:train-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d

- 镜像构建时间：20210912152543(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.1.243
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages

训练基础镜像详情（Horovod）

介绍预置的Horovod镜像详情。

- **引擎版本一：horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **引擎版本二： horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64**

引擎版本一：horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region}.myhuaweicloud.com/aip/horovod_tensorflow:train-horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- 镜像构建时间：20210912152543(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.1.243
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/lib/python3.7/site-packages

引擎版本二： horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region}.myhuaweicloud.com/aip/horovod_pytorch:train-horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- 镜像构建时间：20210912152543(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：11.1.1
- cudnn：8.0.5.39
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/lib/python3.7/site-packages

训练基础镜像详情（MPI）

介绍预置的mindspore_1.3.0镜像详情。

引擎版本：mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

引擎版本：mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

- 镜像地址：swr.{region}.myhuaweicloud.com/aip/mindspore_1_3_0:train-mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-roma-20211104202338-f258e59
- 镜像构建时间：20211104202338(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.1.243
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/lib/python3.7/site-packages

10.2.5 推理专属预置镜像列表

ModelArts的推理平台提供了一系列的基础镜像，用户可以基于这些基础镜像构建自定义镜像，用于部署推理服务。

X86 架构（CPU/GPU）的推理基础镜像

表 10-28 TensorFlow

AI引擎版本	支持的运行环境	镜像名称	URI
2.1.0	CPU GPU(cuda10.1)	tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64	swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817
1.15.5	CPU GPU(cuda11.4)	tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64	swr.{region_id}.myhuaweicloud.com/aip/tensorflow_1_15:tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64-20220524162601-50d6a18

AI引擎版本	支持的运行环境	镜像名称	URI
2.6.0	CPU GPU(cuda11.2)	tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64	swr.{region_id}.myhuaweicloud.com/aip/tensorflow_2_6:tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18

表 10-29 PyTorch

AI引擎版本	支持的运行环境	镜像名称	URI
1.8.0	CPU GPU(cuda10.2)	pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221118143845-d65d817
1.8.2	CPU GPU(cuda11.1)	pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64	swr.{region_id}.myhuaweicloud.com/aip/pytorch_1_8:pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18

表 10-30 MindSpore

AI引擎版本	支持的运行环境	镜像名称	URI
1.7.0	CPU	mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76

AI引擎版本	支持的运行环境	镜像名称	URI
1.7.0	GPU(cuda 10.1)	mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
1.7.0	GPU(cuda 11.1)	mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76

推理基础镜像详情 TensorFlow（CPU/GPU）

ModelArts提供了以下TensorFlow（CPU/GPU）推理基础镜像：

- **引擎版本一：tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **引擎版本二： tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64**
- **引擎版本三：tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64**

引擎版本一：tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817
- 镜像构建时间：20220713110657(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.1.243
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages
- 部分pip安装包列表：

```
Cython                  0.29.21
easydict                1.9
Flask                   2.0.1
grpcio                  1.47.0
gunicorn                20.1.0
h5py                    3.7.0
ipykernel               6.7.0
Jinja2                  3.0.1
```

```
lxml                    4.9.1
matplotlib              3.5.1
moxing-framework        2.1.0.5d9c87c8
numpy                   1.19.5
opencv-python           4.1.2.30
pandas                  1.1.5
Pillow                  9.2.0
pip                     22.1.2
protobuf                3.20.1
psutil                  5.8.0
PyYAML                  5.1
requests                2.27.1
scikit-learn            0.22.1
scipy                   1.5.2
sklearn                 0.0
tensorboard             2.1.1
tensorboardX            2.0
tensorflow              2.1.0
tensorflow-estimator    2.1.0
wheel                   0.37.1
zipp                    3.8.0
...
```

- 部分apt安装包列表：

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
```

```
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

引擎版本二： tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/aip/tensorflow_1_15:tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64-20220524162601-50d6a18
- 镜像构建时间：20220524162601(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 20.04.4 LTS
- cuda：11.4.3
- cudnn：8.2.4.15
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/TensorFlow-1.15.5/bin/python， python 3.8.13
- 三方包安装路径：/home/ma-user/anaconda3/envs/TensorFlow-1.15.5/lib/python3.8/site-packages
- 部分pip安装包列表：

  ```
  Cython 0.29.21
  psutil 5.9.0
  matplotlib 3.5.1
  protobuf 3.20.1
  tensorflow 1.15.5+nv
  Flask 2.0.1
  grpcio 1.46.1
  gunicorn 20.1.0
  Pillow 9.0.1
  tensorboard 1.15.0
  PyYAML 6.0
  pip 22.0.4
  lxml 4.7.1
  numpy 1.18.5
  tensorflow-estimator 1.15.1
  ...
  ```

- 部分apt安装包列表：

  ```
  apt
  ca-certificates
  cmake
  cuda
  curl
  ethtool
  fdisk
  ffmpeg
  g++
  gcc
  git
  gpg
  graphviz
  ```



```
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

引擎版本三：tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/aip/tensorflow_2_6:tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18
- 镜像构建时间：20220524162601(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：11.2.0
- cudnn：8.1.1.33
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/TensorFlow-2.6.0/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/TensorFlow-2.6.0/lib/python3.7/site-packages

- 部分pip安装包列表：

```
Cython 0.29.21
requests 2.27.1
easydict 1.9
tensorboardX 2.0
tensorflow 2.6.0
Flask 2.0.1
grpcio 1.46.1
gunicorn 20.1.0
idna 3.3
tensorflow-estimator 2.9.0
pandas 1.1.5
Pillow 9.0.1
lxml 4.8.0
matplotlib 3.5.1
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 5.1
numpy 1.17.0
opencv-python 4.1.2.30
protobuf 3.20.1
pip 21.2.2
...
```

- 部分apt安装包列表：

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
```



```
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

推理基础镜像详情 PyTorch（CPU/GPU）

ModelArts提供了以下PyTorch（CPU/GPU）推理基础镜像：

- **引擎版本一：pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64**
- **引擎版本二：pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64**

引擎版本一：pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221118143845-d65d817
- 镜像构建时间：20220713110657 (yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.2.89
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python, python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages
- 部分pip安装包列表：

```
Cython                  0.27.3
easydict                1.9
Flask                   2.0.1
fonttools               4.34.4
gunicorn                20.1.0
ipykernel               6.7.0
Jinja2                  3.0.1
lxml                    4.9.1
matplotlib              3.5.1
mmcv                    1.2.7
moxing-framework        2.1.0.5d9c87c8
numpy                   1.19.5
opencv-python           4.1.2.30
pandas                  1.1.5
Pillow                  9.2.0
pip                     22.1.2
protobuf                3.20.1
psutil                  5.8.0
PyYAML                  5.1
requests                2.27.1
```

```
scikit-learn                0.22.1
scipy                       1.5.2
sklearn                     0.0
tensorboard                 2.1.1
tensorboardX                2.0
torch                       1.8.0
torchtext                   0.5.0
torchvision                 0.9.0
tornado                     6.2
tqdm                        4.64.0
traitlets                   5.3.0
typing_extensions           4.3.0
urllib3                     1.26.10
watchdog                    2.0.0
wcwidth                     0.2.5
Werkzeug                    2.1.2
wheel                       0.37.1
yapf                        0.32.0
zipp                        3.8.0
...
```

- 部分apt安装包列表：

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
```



```
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

引擎版本二：pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/aip/pytorch_1_8:pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18
- 镜像构建时间：20220524162601(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：11.1.1
- cudnn：8.0.5.39
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/PyTorch-1.8.2/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/PyTorch-1.8.2/lib/python3.7/site-packages
- 部分pip安装包列表：

  ```
  Cython 0.27.3
  mmcv 1.2.7
  easydict 1.9
  tensorboardX 2.0
  torch 1.8.2+cu111
  Flask 2.0.1
  pandas 1.1.5
  gunicorn 20.1.0
  PyYAML 5.1
  torchaudio 0.8.2
  Pillow 9.0.1
  psutil 5.8.0
  lxml 4.8.0
  matplotlib 3.5.1
  torchvision 0.9.2+cu111
  pip 21.2.2
  protobuf 3.20.1
  numpy 1.17.0
  opencv-python 4.1.2.30
  scikit-learn 0.22.1
  ...
  ```

- 部分apt安装包列表：

  ```
  apt
  ca-certificates
  cmake
  cuda
  curl
  ethtool
  fdisk
  ffmpeg
  g++
  gcc
  git
  gpg
  ```



```
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

推理基础镜像详情 MindSpore（CPU/GPU）

ModelArts提供了以下MindSpore（CPU/GPU）推理基础镜像：

- **引擎版本一：mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64**
- **引擎版本二：mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **引擎版本三：mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64**

引擎版本一：mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76

- 镜像构建时间：20220702120711(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/MindSpore/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- 部分pip安装包列表：

```
cycler                          0.11.0
easydict                         1.9
Flask                           2.0.1
grpcio                          1.47.0
gunicorn                         20.1.0
ipykernel                       6.7.0
Jinja2                          3.0.1
lxml                            4.9.0
matplotlib                       3.5.1
mindinsight                      1.7.0
mindspore                        1.7.0
mindvision                       0.1.0
moxing-framework                    2.1.0.5d9c87c8
numpy                           1.17.0
opencv-contrib-python-headless    4.6.0.66
opencv-python-headless            4.6.0.66
pandas                          1.1.5
Pillow                          9.1.1
pip                             22.1.2
protobuf                         3.20.1
psutil                          5.8.0
PyYAML                           5.1
requests                         2.27.1
scikit-learn                     0.22.1
scipy                           1.5.2
setuptools                       62.6.0
sklearn                          0.0
tensorboardX                        2.0
threadpoolctl                    3.1.0
tomli                           2.0.1
tornado                          6.1
tqdm                             4.64.0
traitlets                       5.3.0
treelib                         1.6.1
urllib3                         1.26.9
wheel                            0.37.1
zipp                            3.8.0
...
```

- 部分apt安装包列表：

```
apt
ca-certificates
cmake
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
```



```
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

引擎版本二：mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- 镜像构建时间：20220702120711(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：10.1.243
- cudnn：7.6.5.32
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/MindSpore/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- 部分pip安装包列表：

```
cycler                  0.11.0
easydict                 1.9
Flask                   2.0.1
grpcio                  1.47.0
gunicorn                 20.1.0
```



```
ipykernel                       6.7.0
Jinja2                          3.0.1
lxml                            4.9.0
matplotlib                      3.5.1
mindinsight                     1.7.0
mindspore                       1.7.0
mindvision                      0.1.0
moxing-framework                2.1.0.5d9c87c8
numpy                           1.17.0
opencv-contrib-python-headless  4.6.0.66
opencv-python-headless          4.6.0.66
pandas                          1.1.5
Pillow                          9.1.1
pip                             22.1.2
protobuf                        3.20.1
psutil                          5.8.0
PyYAML                          5.1
requests                        2.27.1
scikit-learn                    0.22.1
scipy                           1.5.2
setuptools                      62.6.0
sklearn                         0.0
tensorboardX                    2.0
threadpoolctl                   3.1.0
tomli                           2.0.1
tornado                         6.1
tqdm                            4.64.0
traitlets                       5.3.0
treelib                         1.6.1
urllib3                         1.26.9
wheel                           0.37.1
zipp                            3.8.0
...
```

- 部分apt安装包列表：

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
```

```
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

引擎版本三：mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- 镜像地址：swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- 镜像构建时间：20220702120711(yyyy-mm-dd-hh-mm-ss)
- 镜像系统版本：Ubuntu 18.04.4 LTS
- cuda：11.1.1
- cudnn：8.0.5.39
- Python解释器路径及版本：/home/ma-user/anaconda3/envs/MindSpore/bin/python， python 3.7.10
- 三方包安装路径：/home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- 部分pip安装包列表：

```
cycler                          0.11.0
easydict                         1.9
Flask                           2.0.1
grpcio                          1.47.0
gunicorn                         20.1.0
ipykernel                       6.7.0
Jinja2                          3.0.1
lxml                            4.9.0
matplotlib                       3.5.1
mindinsight                      1.7.0
mindspore                        1.7.0
mindvision                       0.1.0
moxing-framework                    2.1.0.5d9c87c8
numpy                           1.17.0
opencv-contrib-python-headless   4.6.0.66
opencv-python-headless              4.6.0.66
pandas                          1.1.5
Pillow                          9.1.1
pip                             22.1.2
protobuf                         3.20.1
```

```
psutil                  5.8.0
PyYAML                    5.1
requests                2.27.1
scikit-learn            0.22.1
scipy                   1.5.2
setuptools               62.6.0
sklearn                 0.0
tensorboardX              2.0
threadpoolctl             3.1.0
tomli                   2.0.1
tornado                   6.1
tqdm                    4.64.0
traitlets               5.3.0
treelib                 1.6.1
urllib3                 1.26.9
wheel                    0.37.1
zipp                    3.8.0
...
```

- 部分apt安装包列表：

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
```



```
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

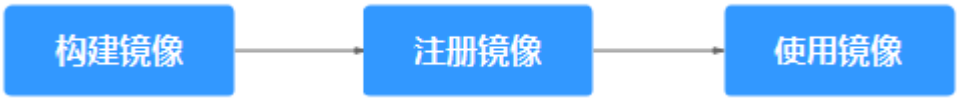
10.3 制作自定义镜像用于创建 Notebook

10.3.1 Notebook 的自定义镜像制作方法

用户在使用ModelArts开发环境时，经常需要对开发环境进行一些改造，如安装、升级或卸载一些包。但是某些包的安装升级需要root权限，运行中的Notebook实例中无root权限，所以在Notebook实例中安装需要root权限的软件，目前在预置的开发环境镜像中是无法实现的。用户可以使用ModelArts提供的基础镜像或用户第三方镜像来编写Dockerfile，构建出完全适合自己的镜像。

Notebook 自定义镜像制作流程

图 10-2 Notebook 自定义镜像制作流程图（适用于场景一和场景二）



场景一：基于Notebook预置镜像或第三方镜像，在服务器上配置docker环境，编写Dockerfile后构建镜像并注册，具体案例参考**在ECS上构建自定义镜像并在Notebook中使用**

场景二：基于Notebook提供的预置镜像或第三方镜像，借助ModelArts命令行工具(**ma-cli镜像构建命令介绍**)制作和注册镜像，构建一个面向AI开发的自定义镜像。此场景Notebook作为制作镜像的平台。具体案例参考**在Notebook中通过Dockerfile从0制作自定义镜像**。

场景三：通过预置的镜像创建Notebook实例，在预置镜像上安装对应的自定义软件和依赖，进而将运行的实例环境以容器镜像的方式保存下来。具体案例参考**在Notebook中通过镜像保存功能制作自定义镜像**。

Notebook 自定义镜像规范

制作自定义镜像时，Base镜像需满足如下规范：

- 基于昇腾、Dockerhub官网等官方开源的镜像制作，开源镜像需要满足如下操作系统约束：

  x86：Ubuntu18.04、Ubuntu20.04

  ARM：Euler2.8.3、Euler2.10.7

> **说明**
>
> Ubuntu20.04.6可能有兼容性问题，请优先使用低于该版本的操作系统。

- 不满足以上镜像规范，所制作的镜像使用可能会出现故障，请用户检查镜像规范，并参考**Notebook自定义镜像故障基础排查**自行排查，如未解决请联系华为技术工程师协助解决。

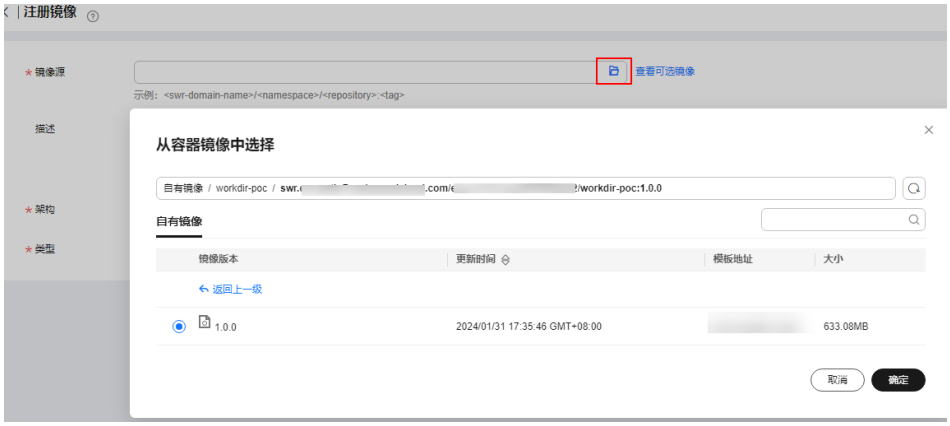
构建后需要注册镜像

用户的自定义镜像构建完成后，需要在ModelArts“镜像管理”页面注册后，方可在Notebook中使用。

> **说明**
>
> SWR镜像类型设置为“私有”时，同一账号下的子用户（IAM用户）可以注册使用。
> SWR镜像类型设置为“公开”时，其他用户才可以注册使用。

1. 进入ModelArts控制台，单击“镜像管理 > 注册镜像”，进入“注册镜像”页面。
2. 根据界面提示填写相关信息，然后单击“立即注册”。
   - “镜像源”选择构建好的镜像。可直接复制完整的SWR地址，或单击 选择SWR构建好的镜像进行注册。

     图 10-3 选择镜像源

   - “架构”和“类型”：根据自定义镜像的实际框架选择。
3. 注册后的镜像会显示在ModelArts“镜像管理”页面。

10.3.2 在 ECS 上构建自定义镜像并在 Notebook 中使用

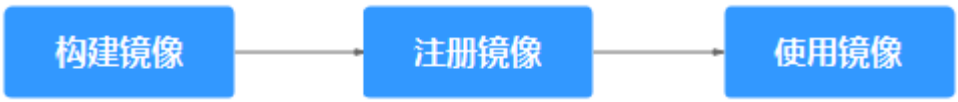
使用场景和构建流程说明

用户可以使用ModelArts提供的基础镜像或第三方的镜像来编写Dockerfile，在ECS服务器上构建出完全适合自己的镜像。然后将镜像进行注册，用以创建新的开发环境，满足自己的业务需求。

本案例将基于ubuntu镜像，安装pytorch 1.8、ffmpeg 3和gcc 8，构建一个面向AI开发的新环境。

主要流程如下图所示：

图 10-4 构建与调测镜像流程



Notebook 自定义镜像规范

制作自定义镜像时，Base镜像需满足如下规范：

- 基于昇腾、Dockerhub官网等官方开源的镜像制作，开源镜像需要满足如下操作系统约束：

  x86：Ubuntu18.04、Ubuntu20.04

  ARM：Euler2.8.3、Euler2.10.7

  说明

  Ubuntu20.04.6可能有兼容性问题，请优先使用低于该版本的操作系统。

- 不满足以上镜像规范，所制作的镜像使用可能会出现故障，请用户检查镜像规范，并参考**Notebook自定义镜像故障基础排查**自行排查，如未解决请联系华为技术工程师协助解决。

操作流程

1. 准备一台Linux环境，这里以ECS为例。
2. 在ECS中构建镜像（本文档提供了Dockflie样例文件）。
3. 将构建的镜像推到SWR。
4. 注册SWR镜像到ModelArts。
5. 创建Notebook并验证新镜像。

准备 Docker 机器并配置环境信息

准备一台具有Docker功能的机器，如果没有，建议申请一台弹性云服务器并购买弹性公网IP，并在准备好的机器上安装必要的软件。

ModelArts提供了ubuntu系统的脚本，方便安装docker。

说明

本地Linux机器的操作等同ECS服务器上的操作，请参考本案例。

1. 登录ECS控制台，购买弹性云服务器，镜像选择“公共镜像”，推荐使用ubuntu18.04的镜像；系统盘设置为100GiB。具体操作请参考**购买并登录弹性云服务器**。

图 10-5 选择镜像和磁盘

2. 购买弹性公网IP并绑定到弹性云服务器。具体操作请参考**配置网络**。
3. 配置VM环境。
   a. 在docker机器中，使用如下命令下载安装脚本。
   ```
   wget https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/modelarts/custom-image-build/install_on_ubuntu1804.sh
   ```

   说明

   当前仅支持ubuntu系统的脚本。

   b. 在docker机器中并执行如下命令，即可完成环境配置。
   ```
   bash install_on_ubuntu1804.sh
   ```

   图 10-6 配置成功


   ```
   source /etc/profile
   ```

   安装脚本依次执行了如下任务：

   i. 安装docker。
   ii. 如果挂载了GPU，则会安装nvidia-docker2，用以将GPU挂载到docker容器中。

制作自定义镜像

这一节描述如何编写一个Dockerfile，并据此构建出一个新镜像在Notebook创建实例并使用。关于Dockerfile的具体编写方法，请参考**官网**。

1. 查询基础镜像（第三方镜像可跳过此步骤）

   ModelArts提供的公共镜像，请参考**Notebook专属预置镜像列表**，根据预置镜像的引擎类型在对应的章节查看镜像URL。
2. 连接容器镜像服务。
   a. 登录容器镜像服务控制台。选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

图 10-7 获取登录指令

说明

- 此处生成的登录指令有效期为24小时，如果需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
- 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。

b. 在安装容器引擎的机器中执行上一步复制的登录指令。登录成功会显示“Login Succeeded”。

3. 拉取基础镜像或第三方镜像（此处以第三方镜像举例）。

```
docker pull swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/ubuntu:18.04 #组织名和镜像替换成自己的
```

4. 编写Dockerfile。

   vim一个Dockerfile，如果使用的基础镜像是ModelArts提供的公共镜像，Dockerfile的具体内容可参考**Dockerfile文件（基础镜像为ModelArts提供）**。

   如果使用的基础镜像是第三方镜像（非ModelArts提供的公共镜像），Dockerfile文件中需要添加uid为1000的用户ma-user和gid为100的用户组ma-group，具体可参考**Dockerfile文件（基础镜像为非ModelArts提供）**。

   本例的Dockerfile将基于ubuntu镜像安装pytorch 1.8、ffmpeg 3和gcc 8，构建一个面向AI任务的镜像。

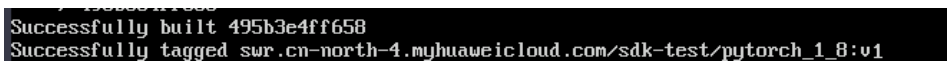
5. 构建镜像

   使用docker build命令从Dockerfile构建出一个新镜像。命令参数解释如下：

   - “-t”指定了新的镜像地址，包括{局点信息}/{组织名称}/{镜像名称}:{版本名称}，请根据实际填写。建议使用完整的swr地址，因为后续的调试和注册需要使用。
   - “-f ”指定了Dockerfile的文件名，根据实际填写。
   - 最后的“ . ”指定了构建的上下文是当前目录，根据实际填写。

```
docker build -t swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/pytorch_1_8:v1 -f Dockerfile .
```

图 10-8 构建成功



注册新镜像

调试完成后，将新镜像注册到ModelArts镜像管理服务中，进而能够在ModelArts中使用该镜像。

1. 将镜像推到SWR

   推送前需要登录SWR，请参考**登录SWR**。登录后使用docker push命令进行推送，如下：

   ```
   docker push swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/pytorch_1_8:v1
   ```

   完成后即可在SWR上看到该镜像。

   图 10-9 将镜像推到 SWR

2. 注册镜像

   在ModelArts Console上注册镜像

   登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。

   a. 单击“注册镜像”，镜像源即为步骤1中推送到SWR中的镜像。请将完整的SWR地址复制到这里即可，或单击 可直接从SWR选择自有镜像进行注册。

   b. “架构”和“类型”根据实际情况选择，与镜像源保持一致。

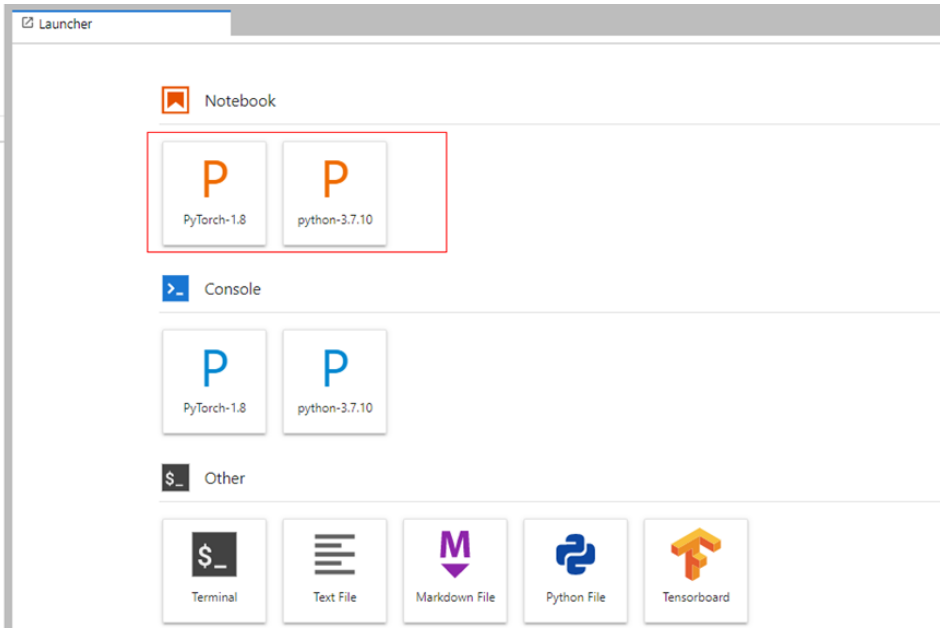
   说明

   注册镜像时，“架构”和“类型”需要和镜像源保持一致，否则在使用此自定义镜像创建Notebook时会创建失败。

创建开发环境并使用

1. 镜像注册成功后，即可在ModelArts控制台的Notebook页面，创建开发环境时选择自定义镜像，选中2中注册的镜像。
2. Notebook创建成功后，在ModelArts Notebook列表页，单击“打开”，启动该开发环境，启动之后Notebook Launcher界面展示如下：

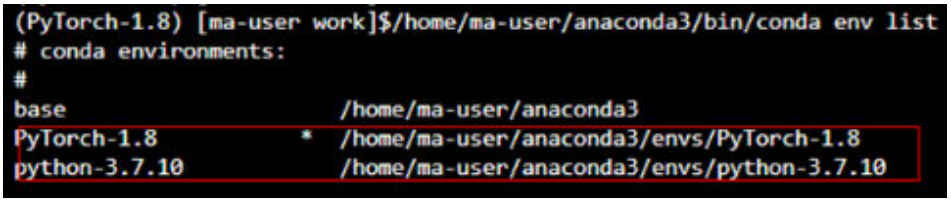
   图 10-10 打开开发环境

3. 打开一个Terminal，查看conda env环境。conda更多知识可以通过**conda官网**了解。

   开发环境中展示的每个kenrel本质是安装在/home/ma-user/anaconda3/下面的conda env环境。conda env环境可通过命令/home/ma-user/anaconda3/bin/conda env list查看。

   图 10-11 查看 conda env 环境



Dockerfile 文件（基础镜像为 ModelArts 提供）

vim一个Dockerfile文件。基础镜像为ModelArts提供的镜像时，Dockerfile文件的具体内容如下：

```
FROM swr.ap-southeast-1.myhuaweicloud.com/atelier/notebook2.0-pytorch-1.4-kernel-cp37:3.3.3-release-v1-20220114

USER root
# section1: config apt source
RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak && \
    echo -e "deb http://repo.huaweicloud.com/ubuntu/ bionic main restricted\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates main restricted\ndeb http://repo.huaweicloud.com/ubuntu/ bionic universe\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates universe\ndeb http://repo.huaweicloud.com/ubuntu/ bionic multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-backports main restricted universe multiverse\ndeb http://repo.huaweicloud.com/ubuntu bionic-security main restricted\ndeb http://repo.huaweicloud.com/ubuntu bionic-security universe\ndeb http://repo.huaweicloud.com/ubuntu bionic-security multiverse" > /etc/apt/sources.list && \
    apt-get update
# section2: install ffmpeg and gcc
RUN apt-get -y install ffmpeg && \
    apt -y install gcc-8 g++-8 && \
    update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 80 --slave /usr/bin/g++ g++ /usr/bin/g++-8 && \
    rm $HOME/.pip/pip.conf
USER ma-user
# section3: configure conda source and pip source
RUN echo -e "channels:\n  - defaults\nshow_channel_urls: true\ndefault_channels:\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2\ncustom_channels:\n  conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  msys2: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  bioconda: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  menpo: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  pytorch-lts: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  simpleitk: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud" > $HOME/.condarc && \
    echo -e "[global]\nindex-url = https://pypi.tuna.tsinghua.edu.cn/simple\n[install]\ntrusted-host = https://pypi.tuna.tsinghua.edu.cn" > $HOME/.pip/pip.conf
# section4: create a conda environment(only support python=3.7) and install pytorch1.8
RUN source /home/ma-user/anaconda3/bin/activate && \
    conda create -y --name pytorch_1_8 python=3.7 && \
    conda activate pytorch_1_8 && \
    pip install torch==1.8.0 torchvision==0.9.0 torchaudio==0.8.0 && \
    conda deactivate
```

Dockerfile 文件（基础镜像为非 ModelArts 提供）

如果使用的镜像是第三方镜像，Dockerfile文件中需要添加uid为1000的用户ma-user和gid为100的用户组ma-group。如果基础镜像中uid 1000或者gid 100已经被其他用

户和用户组占用，需要将其对应的用户和用户组删除。**如下Dockerfile文件已添加指定的用户和用户组，您直接使用即可。**

说明

用户只需要设置uid为1000的用户ma-user和gid为100的用户组ma-group，并使ma-user有对应目录的读写执行权限，其他如启动cmd不需要关心，无需设置或更改。

vim一个Dockerfile文件，添加第三方镜像（即非ModelArts提供的官方镜像）为基础镜像，如以ubuntu18.04为例。Dockerfile文件的具体内容如下：

```
# Replace it with the actual image version.
FROM ubuntu:18.04
# Set the user ma-user whose UID is 1000 and the user group ma-group whose GID is 100
USER root
RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
    fi && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
    fi && \
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user && \
# Grant the read, write, and execute permissions on the target directory to the user ma-user.
chmod -R 750 /home/ma-user

#Configure the APT source and install the ZIP and Wget tools (required for installing conda).
RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak && \
    echo "deb http://repo.huaweicloud.com/ubuntu/ bionic main restricted\ndeb http://
repo.huaweicloud.com/ubuntu/ bionic-updates main restricted\ndeb http://repo.huaweicloud.com/ubuntu/
bionic universe\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates universe\ndeb http://
repo.huaweicloud.com/ubuntu/ bionic multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates
multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-backports main restricted universe multiverse
\ndeb http://repo.huaweicloud.com/ubuntu bionic-security main restricted\ndeb http://
repo.huaweicloud.com/ubuntu bionic-security universe\ndeb http://repo.huaweicloud.com/ubuntu bionic-
security multivers e" > /etc/apt/sources.list && \
apt-get update && \
apt-get install -y zip wget

#Modifying the system Configuration of the image (required for creating the Conda environment)
RUN rm /bin/sh && ln -s /bin/bash /bin/sh

#Switch to user ma-user , download miniconda from the Tsinghua repository, and install miniconda in /
home/ma-user.
USER ma-user
RUN cd /home/ma-user/ && \
    wget --no-check-certificate https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/Miniconda3-4.6.14-
Linux-x86_64.sh && \
    bash Miniconda3-4.6.14-Linux-x86_64.sh -b -p /home/ma-user/anaconda3 && \
    rm -rf Miniconda3-4.6.14-Linux-x86_64.sh

#Configure the conda and pip sources
RUN mkdir -p /home/ma-user/.pip && \
    echo -e "channels:\n  - defaults\nshow_channel_urls: true\ndefault_channels:\n  - https://
mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2" > /home/ma-user/.condarc && \
    echo -e "[global]\nindex-url = https://pypi.tuna.tsinghua.edu.cn/simple\n[install]\ntrusted-host = https://
pypi.tuna.tsinghua.edu.cn" > /home/ma-user/.pip/pip.conf

#Create the conda environment and install the Python third-party package. The ipykernel package is
mandatory for starting a kernel.
RUN source /home/ma-user/anaconda3/bin/activate && \
    conda create -y --name pytorch_1_8 python=3.7 && \
    conda activate pytorch_1_8 && \
    pip install torch==1.8.1 torchvision==0.9.1 && \
    pip install ipykernel==6.7.0 && \
    conda init bash && \
```



```
    conda deactivate

#Install FFmpeg and GCC
USER root
RUN apt-get -y install ffmpeg && \
    apt -y install gcc-8 g++-8
```

10.3.3 在 Notebook 中通过 Dockerfile 从 0 制作自定义镜像

场景说明

本案例将基于ModelArts提供的PyTorch预置镜像，并借助ModelArts命令行工具(请参考**ma-cli镜像构建命令介绍**)，通过加载镜像构建模板并修改Dockerfile，构建出一个新镜像，最后注册后在Notebook使用。

操作流程

1. 创建Notebook。
2. 在Notebook中制作自定义镜像。
3. 注册镜像到ModelArts。
4. 创建Notebook并验证新镜像。

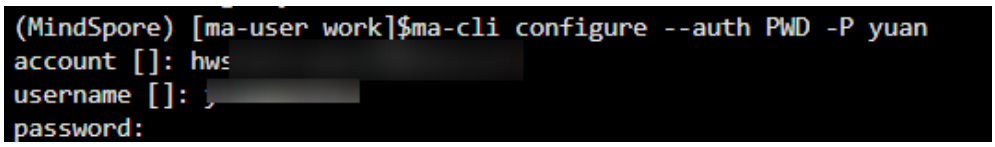
创建 Notebook

1. 登录ModelArts控制台，进入“开发空间>Notebook”，单击“创建”，进入创建Notebook页面。“公共镜像”选择“PyTorch”的，其他参数默认。具体操作请参考**创建Notebook实例**。
2. 创建完成后Notebook的状态为“运行中”，单击“操作列”的“打开”，自动进入JupyterLab界面，打开Terminal。

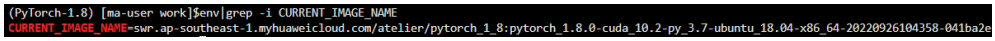
在 Notebook 中制作自定义镜像

步骤1 首先配置鉴权信息，指定profile，根据提示输入账号、用户名及密码。鉴权更多信息请查看**配置登录信息**。

```
ma-cli configure --auth PWD -P xxx
```



步骤2 执行**env|grep -i CURRENT_IMAGE_NAME**命令查询当前实例所使用的镜像。



步骤3 制作新镜像。

1. 获取上步查询的基础镜像的SWR地址。

   CURRENT_IMAGE_NAME=swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e

2. 加载镜像构建模板。

执行**ma-cli image get-template**命令查询镜像模板。

```
(MindSpore) [ma-user work]$ma-cli image get-template
Template Name                                Description
-------------------------------------------  ------------------------------------------------------------------------------------------------------------------------
customize_from_ubuntu_18.04_to_modelarts     Add ma-user, apt install packages and create a new conda environment with pip based on scratch ubuntu 18.04
upgrade_current_notebook_apt_packages        Install apt packages like ffmpeg, gcc-8, g++-8 based on current Notebook image
migrate_3rd_party_image_to_modelarts         General template for migrating your own or open source image to ModelArts
build_handwritten_number_inference_application  Create a new AI application, used to generate an image to deploy and infer in ModelArts
update_dli_image_pip_package                 Install pip packages based on DLI image
forward_compat_cuda_11_image_to_modelarts    Migrate and forward compat cuda-11.x to ModelArts by upgrading only user-mode CUDA components
```

然后执行**ma-cli image add-template**命令将镜像模板加载到指定文件夹下，默认路径为当前命令所在的路径。例如：加载upgrade_current_notebook_apt_packages镜像构建模板。

```
ma-cli image add-template upgrade_current_notebook_apt_packages
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image add-template upgrade_current_notebook_apt_packages
[ OK ] Successfully add configuration template [ upgrade_current_notebook_apt_packages ] under folder [ /home/ma-user/work/.ma/upgrade_current_notebook_apt_packages ]
```

3. 修改Dockerfile。

   本例的Dockerfile将基于PyTorch基础镜像pytorch1.8-cuda10.2-cudnn7-ubuntu18.04进行改造，加载镜像模板upgrade_current_notebook_apt_packages，升级gcc和g++。

   加载镜像模板后，Dockerfile文件自动加载，在“.ma/upgrade_current_notebook_apt_packages”路径下，双击Dockerfile文件打开，内容参考如下，根据实际需求修改：

   ```
   FROM swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e

   # Set proxy to download internet resources
   ENV HTTP_PROXY=http://proxy.modelarts.com:80 \
       http_proxy=http://proxy.modelarts.com:80 \
       HTTPS_PROXY=http://proxy.modelarts.com:80 \
       https_proxy=http://proxy.modelarts.com:80

   USER root

   # Config apt source which can accelerate the apt package download speed. Also install ffmpeg and gcc-8 in root mode
   RUN cp -f /etc/apt/sources.list /etc/apt/sources.list.bak && \
       sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
       sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
       apt update && \
       apt -y install ffmpeg && \
       apt install -y --no-install-recommends gcc-8 g++-8 && apt-get autoremove -y && \
       update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 80 --slave /usr/bin/g++ g++ /usr/bin/g++-8

   # ModelArts requires ma-user as the default user to start image
   USER ma-user
   ```

4. 构建镜像

   使用**ma-cli image build**命令从Dockerfile构建出一个新镜像。命令更多信息请参考**镜像构建命令**。

   ```
   ma-cli image build .ma/upgrade_current_notebook_apt_packages/Dockerfile -swr notebook-test/my_image:0.0.1 -P XXX
   ```

   其中“.ma/upgrade_current_notebook_apt_package/Dockerfile”为Dockerfile文件所在路径，“notebook-test/my_image:0.0.1”为构建的新镜像的SWR路径。“XXX”为鉴权时指定的profile。

----结束

注册新镜像

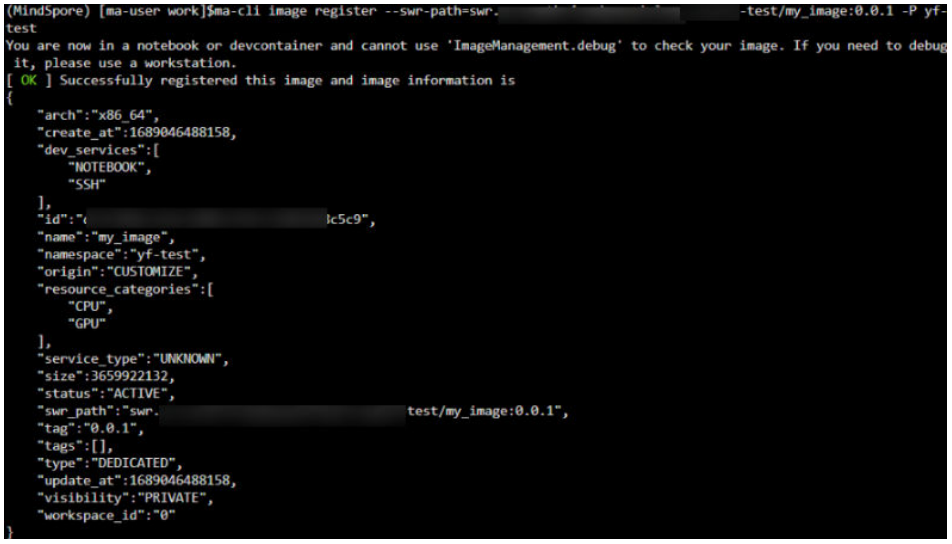
构建完成后，将新镜像注册到ModelArts镜像管理服务中，进而能够在ModelArts中使用该镜像。

有两种方式来注册镜像。

- **方式一：使用ma-cli image register命令来注册镜像。**注册命令会返回注册好的镜像信息，包括镜像id，name等，如下图所示。该命令的更多信息可参考**镜像构建命令**。

  ```
  ma-cli image register --swr-path=swr.ap-southeast-1.myhuaweicloud.com/notebook-test/
  my_image:0.0.1 -P XXX
  ```

  图 10-12 注册镜像

- **方式二：在ModelArts Console上注册镜像**

  登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。

  a. 单击“注册镜像”。请将完整的SWR地址复制到这里即可，或单击 可直接从SWR选择自有镜像进行注册。

  b. “架构”和“类型”根据实际情况选择，与镜像源保持一致。

创建 Notebook 并使用

镜像注册成功后，即可在ModelArts控制台的“开发环境 > Notebook”页面，创建开发环境时选择该自定义镜像。

10.3.4 在 Notebook 中通过镜像保存功能制作自定义镜像

通过预置的镜像创建Notebook实例，在基础镜像上安装对应的自定义软件和依赖，在管理页面上进行操作，进而完成将运行的实例环境以容器镜像的方式保存下来。镜像保存后，默认工作目录是根目录“/”路径。

保存的镜像中，安装的依赖包不丢失，持久化存储的部分（home/ma-user/work目录的内容）不会保存在最终产生的容器镜像中。VS Code远程开发场景下，在Server端安装的插件不丢失。

说明

当镜像保存失败时，请在Notebook实例详情页查看事件，事件描述请参考**查看Notebook实例事件**。

建议保存的镜像大小不要超过35G，镜像层数不要超过125层，因为节点容器存储Rootfs差异（详细请参考**容器引擎空间分配**），可能会导致镜像保存失败。

- 如使用的是专属资源池，可尝试在“专属资源池>弹性集群”页面按需调整容器引擎空间大小，具体步骤请参考**扩缩容专属资源池的“修改容器引擎空间大小”**。
- 如果问题仍未解决，请联系技术支持。

前提条件

Notebook实例状态为“运行中”。

保存镜像

1. 在Notebook列表中，对于要保存的Notebook实例，单击右侧“操作”列中的“更多 > 保存镜像”，进入“保存镜像”对话框。

图 10-13 保存镜像

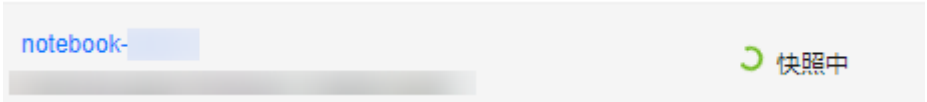
2. 在保存镜像对话框中，设置组织、镜像名称、镜像版本和描述信息。单击“确定”保存镜像。

   在“组织”下拉框中选择一个组织。如果没有组织，可以单击右侧的“立即创建”，创建一个组织。

   同一个组织内的用户可以共享使用该组织内的所有镜像。

3. 镜像会以快照的形式保存，保存过程约5分钟，请耐心等待。此时不可再操作实例。

图 10-14 保存镜像



须知

快照中耗费的时间仍占用实例的总运行时长，如果在快照中时，实例因运行时间到期停止，将导致镜像保存失败。

4. 镜像保存成功后，实例状态变为“运行中”，用户可在“镜像管理”页面查看到该镜像详情。
5. 单击镜像的名称，进入镜像详情页，可以查看镜像版本/ID，状态，资源类型，镜像大小，SWR地址等。

基于自定义镜像创建 Notebook 实例

从Notebook中保存的镜像可以在镜像管理中查询到，可以用于创建新的Notebook实例，完全继承保存状态下的实例软件环境配置。

方式一：在Notebook实例创建页面，镜像类型选择“自定义镜像”，名称选择上述保存的镜像。

图 10-15 创建基于自定义镜像的 Notebook 实例

方式二：在“镜像管理”页面，单击某个镜像的镜像详情，在镜像详情页，单击“创建Notebook”，也会跳转到基于该自定义镜像创建Notebook的页面。

镜像保存时，哪些目录的数据可以被保存

- 可以保存的目录：包括容器构建时静态添加到镜像中的文件和目录，可以保存在镜像环境里。

  例如：安装的依赖包、“/home/ma-user”目录
- 不会被保存的目录：容器启动时动态连接到宿主机的挂载目录或数据卷，这些内容不会被保存在镜像中。可以通过**df -h**命令查看挂载的动态目录，非“/”路径下的不会保存。

  例如：持久化存储的部分“home/ma-user/work”目录的内容不会保存在最终产生的容器镜像中、动态挂载在“/data”下的目录不会被保存。

10.4 制作自定义镜像用于训练模型

10.4.1 训练作业的自定义镜像制作流程

如果您已经在本地完成模型开发或训练脚本的开发，且您使用的AI引擎是ModelArts不支持的框架。您可以制作自定义镜像，并上传至SWR服务。您可以在ModelArts使用此自定义镜像创建训练作业，使用ModelArts提供的资源训练模型。

制作流程

图 10-16 训练作业的自定义镜像制作流程

场景一：预置镜像满足ModelArts训练平台约束，但不满足代码依赖的要求，需要额外安装软件包。

具体案例参考**使用预置镜像制作自定义镜像用于训练模型**。

场景二：已有本地镜像满足代码依赖的要求，但是不满足ModelArts训练平台约束，需要适配。

具体案例参考**已有镜像迁移至ModelArts用于训练模型**。

场景三： 当前无可使用的镜像，需要从0制作镜像（既需要安装代码依赖，又需要制作出的镜像满足MAModelArts平台约束）。具体案例参考：

从0制作自定义镜像用于创建训练作业（PyTorch+CPU/GPU）

从0制作自定义镜像用于创建训练作业（MPI+CPU/GPU）

从0制作自定义镜像用于创建训练作业（Tensorflow+GPU）

训练框架的自定义镜像约束

- 推荐自定义镜像使用ubuntu-18.04的操作系统，避免出现版本不兼容的问题。
- 自定义镜像的大小推荐15GB以内，最大不要超过资源池的容器引擎空间大小的一半。镜像过大会直接影响训练作业的启动时间。

  ModelArts公共资源池的容器引擎空间为50G，专属资源池的容器引擎空间的默认为50G，支持在创建专属资源池时自定义容器引擎空间。
- 自定义镜像的默认用户必须为“uid”为“1000”的用户。
- 自定义镜像中不能安装GPU或Ascend驱动程序。当用户选择GPU资源运行训练作业时，ModelArts后台自动将GPU驱动程序放置在训练环境中的 /usr/local/nvidia 目录；当用户选择Ascend资源运行训练作业时，ModelArts后台自动将Ascend驱动程序放置在/usr/local/Ascend/driver目录。
- X86 CPU架构和ARM CPU架构的自定义镜像分别只能运行于对应CPU架构的规格中。

  执行如下命令，查看自定义镜像的CPU架构。
  ```
  docker inspect {自定义镜像地址} | grep Architecture
  ```
  ARM CPU架构的自定义镜像，上述命令回显如下。
  ```
  "Architecture": "arm64"
  ```
  - 规格中带有ARM字样的显示，为ARM CPU架构。
  - 规格中未带有ARM字样的显示，为X86 CPU架构。
- ModelArts后台暂不支持下载开源安装包，建议用户在自定义镜像中安装训练所需的依赖包。
- 自定义镜像需上传至容器镜像服务（SWR）才能在ModelArts上用于训练。

10.4.2 使用预置镜像制作自定义镜像用于训练模型

使用预置框架构建自定义镜像原理介绍

如果先前基于预置框架且通过指定代码目录和启动文件的方式来创建的训练作业；但是随着业务逻辑的逐渐复杂，您期望可以基于预置框架修改或增加一些软件依赖的时候，可以使用预置框架构建自定义镜像，即在创建训练作业页面选择预置框架名称后，在预置框架版本下拉列表中选择“自定义”。

该方式的训练流程与直接基于预置框架创建的训练作业相同，例如：

- 系统会自动注入一系列环境变量，如下所示。
  - PATH=${MA_HOME}/anaconda/bin:${PATH}
  - LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
  - PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
- 选择的启动文件将会被系统自动以python命令直接启动，因此请确保镜像中的Python命令为您预期的Python环境。通过系统自动注入的PATH环境变量，可以参考下述命令确认训练作业最终使用的Python版本。
  - export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
  - docker run --rm {image} $(which python) -V
- 系统会自动添加预置框架关联的超参。

使用预置框架构建训练镜像

ModelArts平台提供了Tensorflow，PyTorch，MindSpore等常用深度学习任务的基础镜像，镜像里已经安装好运行任务所需软件。当基础镜像里的软件无法满足您的程序运行需求时，您可以基于这些基础镜像制作一个新的镜像并进行训练。

您可以参考如下步骤基于训练基础镜像来构建新镜像。

1. 安装Docker。如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

   以linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

2. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

3. 准备可用的pip源文件pip.conf 。

   ```
   [global]
   index-url = https://repo.huaweicloud.com/repository/pypi/simple
   trusted-host = repo.huaweicloud.com
   timeout = 120
   ```

4. 参考如下Dockerfile文件内容来基于ModelArts提供的训练基础镜像来构建一个新镜像。将编写好的Dockerfile文件放置在context文件夹内。训练基础镜像地址请参见**训练专属预置镜像列表**。

   ```
   FROM {ModelArts提供的训练基础镜像地址}

   # 配置pip
   RUN mkdir -p /home/ma-user/.pip/
   COPY --chown=ma-user:ma-group pip.conf /home/ma-user/.pip/pip.conf
   ```



```
# 设置容器镜像预置环境变量
# 将python解释器路径加入到PATH环境变量中
# 请务必设置PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=${ANACONDA_DIR}/envs/${ENV_NAME}/bin:$PATH \
    PYTHONUNBUFFERED=1

RUN /home/ma-user/anaconda/bin/pip install --no-cache-dir numpy
```

5. 构建新镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像training:v1。

```
docker build . -t training:v1
```

6. 将构建好的新镜像上传至SWR。
   a. 登录容器镜像服务控制台，选择区域。
   b. 单击右上角“创建组织”，输入组织名称完成组织创建。您可以自定义组织名称，本示例使用“deep-learning”，实际操作时请重新命名一个组织名称。后续所有命令中使用到组织名称deep-learning时，均需要替换为此处实际创建的组织名称。
   c. 单击右上角“登录指令”，获取登录访问指令。以root用户登录ECS环境，输入登录指令。

      图 10-17 在 ECS 中执行登录指令

   d. 登录SWR后，使用**docker tag**命令给上传镜像打标签。下面命令中的组织名称deep-learning，请替换为**a.登录容器镜像服务控制台，选择区域。**中实际创建的组织名称，以下所有命令中的deep-learning都需要替换。

```
sudo docker tag tf-1.13.2:latest swr.实际域名.com/deep-learning/tf-1.13.2:latest
```

   e. 使用**docker push**命令上传镜像。

```
sudo docker push swr.实际域名.com/deep-learning/tf-1.13.2:latest
```

   f. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。

      “swr.<region>.myhuaweicloud.com/deep-learning/tf-1.13.2:latest”即为此自定义镜像的“SWR_URL”。

7. 在ModelArts上创建训练作业。
   a. 登录ModelArts管理控制台。
   b. 在左侧导航栏中，选择“模型训练 > 训练作业”进入训练作业列表。
   c. 单击“创建训练作业”，进入创建训练作业页面，填写作业信息，创建方式参考**表10-31**，其他参数填写请参考**创建训练作业**。

      表 10-31 创建训练作业的创建方式

参数名称	说明
创建方式	必选，选择“自定义算法”。
启动方式	必选，选择“预置框架”，并选择训练作业要使用的预置框架引擎和引擎版本，引擎版本选择“自定义”。 

参数名称	说明
镜像	容器镜像选择**上一步上传到SWR的镜像**。
代码目录	必填，选择训练代码文件所在的OBS目录。 • 需要提前将代码上传至OBS桶中，目录内文件总大小要小于或等于5GB，文件数要小于或等于1000个，文件深度要小于或等于32。 • 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“${MA_JOB_DIR}/code”目录中。
启动文件	必填，选择代码目录中训练作业的Python启动脚本。 ModelArts只支持使用Python语言编写的启动文件，因此启动文件必须以“.py”结尾。

10.4.3 已有镜像迁移至 ModelArts 用于训练模型

场景描述

本地已有镜像，需要做云上适配，用于ModelArts模型训练。

操作步骤

1. 参考如下Dockerfile，修改已有镜像，使其符合模型训练的自定义镜像规范。

```
FROM {已有镜像}

USER root

# 如果已存在 gid = 100 用户组，则删除 groupadd 命令。
RUN groupadd ma-group -g 100
# 如果已存在 uid = 1000 用户，则删除 useradd 命令。
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 修改镜像中相关文件权限，使得 ma-user, uid = 1000 用户可读写。
RUN chown -R ma-user:100 {Python软件包路径}

# 设置容器镜像预置环境变量。
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失。
ENV PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录。
USER ma-user
WORKDIR /home/ma-user
```

Dockerfile需要重点关注以下几点：

a. 为镜像增加模型训练的默认用户组ma-group，“gid = 100”。

说明

如果已存在“gid = 100”用户组，可能会报错“groupadd: GID '100' already exists”。可通过命令“cat /etc/group | grep 100”查询是否已存在gid = 100用户组。

如果已存在“gid = 100”用户组，则该步骤跳过，下文Dockerfile中删除“RUN groupadd ma-group -g 100”命令。

b. 为镜像增加模型训练的默认用户ma-user，“uid = 1000”。

说明

如果已存在“uid = 1000”用户，可能会报错“useradd: UID 1000 is not unique”。可通过命令“cat /etc/passwd | grep 1000”查询是否已存在uid = 1000用户。

如果已存在“uid = 1000”用户，则该步骤跳过，下文Dockerfile中删除“RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user”命令。

c. 修改镜像中相关文件权限，使得ma-user，“uid = 1000”用户可读写。

2. 编写好Dockerfile后，通过执行如下所示命令进行新镜像构建。

```
docker build -f Dockerfile . -t {新镜像}
```

3. 构建成功后将新镜像上传至SWR（参考**6**）。
4. 在ModelArts上创建训练作业。
   a. 登录ModelArts管理控制台。
   b. 在左侧导航栏中，选择“模型训练 > 训练作业”进入训练作业列表。
   c. 单击“创建训练作业”，进入创建训练作业页面，填写作业信息，创建方式参考**表10-31**，其他参数填写请参考**创建训练作业**。

表 10-32 创建训练作业的创建方式（使用自定义镜像）

参数名称	说明
创建方式	必选，选择“自定义算法”。
启动方式	必选，选择“自定义”。 
镜像	必填，单击右边的“选择”，从容器镜像中选择**上一步上传到SWR的镜像**。

参数名称	说明
代码目录	选择训练代码文件所在的OBS目录。如果自定义镜像中不含训练代码则需要配置该参数，如果自定义镜像中已包含训练代码则不需要配置。 ● 需要提前将代码上传至OBS桶中，目录内文件总大小要小于或等于5GB，文件数要小于或等于1000个，文件深度要小于或等于32。 ● 训练代码文件会在训练作业启动的时候被系统自动下载到训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为存放代码目录的最后一级OBS目录。例如，“代码目录”选择的是“/test/code”，则训练代码文件会被下载到训练容器的“${MA_JOB_DIR}/code”目录中。
运行用户ID	容器运行时的用户ID，该参数为选填参数，建议使用默认值1000。 如果需要指定uid，则uid数值需要在规定范围内，不同资源池的uid范围如下： ● 公共资源池：1000-65535 ● 专属资源池：0-65535
启动命令	必填，镜像的启动命令。 运行训练作业时，当“代码目录”下载完成后，“启动命令”会被自动执行。 ● 如果训练启动脚本用的是py文件，例如“train.py”，则启动命令如下所示。 `python ${MA_JOB_DIR}/demo-code/train.py` ● 如果训练启动脚本用的是sh文件，例如“main.sh”，则启动命令如下所示。 `bash ${MA_JOB_DIR}/demo-code/main.sh` 启动命令支持使用“;”和“&&”拼接多条命令，命令中的“demo-code”为存放代码目录的最后一级OBS目录，以实际情况为准。
本地代码目录	指定训练容器的本地目录，启动训练时系统会将代码目录下载至此目录。 此参数可选，默认本地代码目录为“/home/ma-user/modelarts/user-job-dir”。
工作目录	训练时，系统会自动cd到此目录下执行启动文件。

10.4.4 从 0 制作自定义镜像用于创建训练作业（Pytorch+Ascend）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是PyTorch，训练使用的资源是专属资源池的Ascend芯片。

准备工作

准备一套可以连接外部网络，装有Linux系统并安装18.09.7及以上版本docker的虚拟机或物理机用作镜像构建节点，以下称“构建节点”。

可以通过执行docker pull、apt-get update/upgrade和pip install命令判断是否可正常访问外部可用的开源软件仓库，若可以正常访问表示环境已连接外部网络。

须知

- 上述的虚拟机或物理机需要为arm64架构。
- 建议构建节点安装的Linux系统版本为Ubuntu 18.04。
- 本指导使用/opt目录作为构建任务承载目录，请确保该目录下可用存储空间大于30GB。
- Docker的安装可以参考官方文档：**Install Docker Engine on Ubuntu**。MiniConda与tflite安装包为第三方安装包，ModelArts不对其安全相关问题进行负责，如用户有安全方面的需求，可以对该安装包进行加固后发布成同样名称的文件上传到构建节点。

制作自定义镜像

步骤1 确认Docker Engine版本。执行如下命令。

```
docker version | grep -A 1 Engine
```

命令回显如下。

```
Engine:
 Version: 18.09.0
```

说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

步骤2 准备名为context的文件夹。

```
mkdir -p context
```

步骤3 准备可用的pip源文件pip.conf 。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

```
[global]
index-url =
https://repo.huaweicloud.com/repository/pypi/simple
trusted-host =
repo.huaweicloud.com
timeout = 120
```

步骤4 准备可用的apt源文件Ubuntu-Ports-bionic.list。本示例使用华为开源镜像站提供的apt源，执行如下命令获取apt源文件。

```
wget -O Ubuntu-Ports-bionic.list --no-check-certificate
https://repo.huaweicloud.com/repository/conf/Ubuntu-Ports-bionic.list
```

步骤5 下载Ascend-cann-nnae_7.0.0_linux-aarch64.run与torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl以及torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl安装文件。

- 下载Ascend-cann-nnae_7.0.0_linux-aarch64.run文件：请根据您的用户类型打开下方对应的链接。版本过滤选择CANN7，筛选后版本单击CANN7.0.0链接，然后页面中找到Ascend-cann-nnae_7.0.0_linux-aarch64.run并下载。

  - 企业用户：**下载地址**。
  - 运营商用户：**下载地址**。
- 下载torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl文件：请单击**下载地址**下载。
- 下载torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl文件：请单击**下载地址**下载。

说明

ModelArts当前仅支持CANN商用版本，不支持社区版。

步骤6 下载Miniconda3安装文件。

使用地址**下载地址**，下载Miniconda3-py39_24.5.0-0安装文件（对应python 3.9）。

说明

如果需要其他版本的Python，可以从**Miniconda3文件列表**下载，需注意MindSpore要下载对应其Python版本的包，上下文版本替换要保持一致。

步骤7 将上述pip源文件、*.list文件、*.run文件、 *.whl文件、Miniconda3安装文件放置在context文件夹内，context文件夹内容如下。

```
context
├── Ascend-cann-nnae_7.0.0_linux-aarch64.run
├── torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
├── Miniconda3-py39_24.5.0-0-Linux-aarch64.sh
├── torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

步骤8 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网
FROM ubuntu:18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 安装 OS 依赖
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y \
    # utils
    ca-certificates vim curl \
    # CANN 7.0.0
    gcc g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils net-tools
libblas-dev gfortran libblas3 libopenblas-dev \
    # MindSpore 2.2.0
    libgmp-dev && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    # 修改 CANN 7.0.0 安装目录的父目录权限，使得 ma-user 可以写入
    chmod o+w /usr/local

RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 设置容器镜像默认用户与工作目录
USER ma-user
```



```
WORKDIR /home/ma-user

# 使用开源镜像站提供的 pypi 配置
RUN mkdir -p /home/ma-user/.pip/
COPY --chown=ma-user:100 pip.conf /home/ma-user/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY --chown=ma-user:100 Miniconda3-py39_24.5.0-0-Linux-aarch64.sh /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py39_24.5.0-0-Linux-aarch64.sh -b -p /home/ma-user/miniconda3



ENV PATH=$PATH:/home/ma-user/miniconda3/bin

# 安装 CANN 7.0.0 Python Package 依赖
RUN pip install numpy~=1.19.2 decorator~=4.4.0 sympy~=1.5.1 cffi~=1.12.3 protobuf~=3.13.0 \
    attrs pyyaml pathlib2 scipy requests psutil absl-py

# 安装 CANN 7.0.0 至 /usr/local/Ascend 目录
COPY --chown=ma-user:100 Ascend-cann-nnae_7.0.0_linux-aarch64.run /tmp
RUN chmod +x /tmp/Ascend-cann-nnae_7.0.0_linux-aarch64.run && \
    echo Y|/tmp/Ascend-cann-nnae_7.0.0_linux-aarch64.run --install --install-path=/usr/local/Ascend

# 安装 Pytorch 2.1.0
COPY --chown=ma-user:100 torch-2.1.0-cp39-cp39-
manylinux_2_17_aarch64.manylinux2014_aarch64.whl /tmp
RUN chmod +x /tmp/torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl && \
    pip install /tmp/torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl

# 安装 touch-npu
COPY --chown=ma-user:100 torch_npu-2.1.0.post7-cp39-cp39-
manylinux_2_17_aarch64.manylinux2014_aarch64.whl /tmp
RUN chmod +x /tmp/torch_npu-2.1.0.post7-cp39-cp39-
manylinux_2_17_aarch64.manylinux2014_aarch64.whl && \
    pip install /tmp/torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl

# 构建最终容器镜像
FROM ubuntu:18.04

# 安装 OS 依赖
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y \
    # utils
    ca-certificates vim curl \
    # CANN 7.0.RC1
    gcc g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils net-tools
libblas-dev gfortran libblas3 libopenblas-dev \
    # MindSpore 2.2.0
    libgmp-dev && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3
COPY --chown=ma-user:100 --from=builder /home/ma-user/Ascend /home/ma-user/Ascend
COPY --chown=ma-user:100 --from=builder /home/ma-user/var /home/ma-user/var
COPY --chown=ma-user:100 --from=builder /usr/local/Ascend /usr/local/Ascend

# 设置容器镜像预置环境变量
# 请务必设置 CANN 相关环境变量
```

```
# 请务必设置 Ascend Driver 相关环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/usr/local/Ascend/nnae/latest/bin:/usr/local/Ascend/nnae/latest/compiler/ccec_compiler/bin:/home/ma-user/miniconda3/bin \
    LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/Ascend/driver/lib64:/usr/local/Ascend/driver/lib64/common:/usr/local/Ascend/driver/lib64/driver:/usr/local/Ascend/nnae/latest/lib64:/usr/local/Ascend/nnae/latest/lib64/plugin/opskernel:/usr/local/Ascend/nnae/latest/lib64/plugin/nnengine \
    PYTHONPATH=$PYTHONPATH:/usr/local/Ascend/nnae/latest/python/site-packages:/usr/local/Ascend/nnae/latest/opp/op_impl/built-in/ai_core/tbe \
    ASCEND_AICPU_PATH=$ASCEND_AICPU_PATH:/usr/local/Ascend/nnae/latest \
    ASCEND_OPP_PATH=$ASCEND_OPP_PATH:/usr/local/Ascend/nnae/latest/opp \
    ASCEND_HOME_PATH=$ASCEND_HOME_PATH:/usr/local/Ascend/nnae/latest \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user
```

关于Dockerfile文件编写的更多指导内容参见**Docker官方文档**。

步骤9 确认已创建完成Dockerfile文件。此时context文件夹内容如下。

```
context
├── Ascend-cann-nnae_7.0.0_linux-aarch64.run
├── Dockerfile
├── torch-2.1.0-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
├── torch_npu-2.1.0.post7-cp39-cp39-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
├── Miniconda3-py39_24.5.0-0-Linux-aarch64.sh
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

步骤10 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像。

```
docker build . -t pytorch:2.1.0-cann7.0.0
```

说明

如果构建中访问https://registry-1.docker.io/v2/ 出现connection refused或者Client.Timeout exceeded问题，需要配置下docker代理。

vi /etc/docker/daemon.json

文件中写入以下内容，并保存文件

```
{
"registry-mirrors":[
"https://docker.m.daocloud.io",
"https://docker.jianmuhub.com",
"https://huecker.io",
"https://dockerhub.timeweb.cloud",
"https://dockerhub1.beget.com",
"https://noohub.ru"]
}
```

依次执行systemctl daemon-reload和systemctl restart docker

重新构建

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged pytorch:2.1.0-cann7.0.0
```

----结束

上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。

2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker tag pytorch:2.1.0-cann7.0.0 swr.{region-id}.{domain}/deep-learning/pytorch:2.1.0-cann7.0.0
   ```
   b. 使用docker push命令上传镜像。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:2.1.0-cann7.0.0
   ```
6. 完成镜像上传后，在容器镜像服务控制台的“我的镜像”页面可查看已上传的自定义镜像。

在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**快速配置ModelArts委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像地址：swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:2.1.0-cann7.0.0
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/pytorch/demo-code/”，训练代码会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 启动命令：“/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py”，此处的“demo-code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 资源池：选择专属资源池。
   - 类型：选择驱动/固件版本匹配的专属资源池Ascend规格。
   - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/pytorch/log/”
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。您可以在作业详情页面，查看日志信息。

10.4.5 从 0 制作自定义镜像用于创建训练作业（PyTorch+CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是PyTorch，训练使用的资源是CPU或GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练作业。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **前提条件**
2. **Step1 创建OBS桶和文件夹**
3. **Step2 准备训练脚本并上传至OBS**
4. **Step3 准备镜像主机**
5. **Step4 制作自定义镜像**
6. **Step5 上传镜像至SWR服务**
7. **Step6 在ModelArts上创建训练作业**

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表10-33**所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 10-33 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/pytorch/demo-code/”	用于存储训练脚本文件。
“obs://test-modelarts/pytorch/log/”	用于存储训练日志文件。

Step2 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本“pytorch-verification.py”文件，并上传至OBS桶的“obs://test-modelarts/pytorch/demo-code/”文件夹下。

“pytorch-verification.py”文件内容如下：

```
import torch
import torch.nn as nn

x = torch.randn(5, 3)
print(x)

available_dev = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
y = torch.randn(5, 3).to(available_dev)
print(y)
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用Ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤 。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以执行以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 执行如下命令确认Docker Engine版本。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。


```
...
Engine:
 Version:      18.09.0
```

说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

```
mkdir -p context
```

4. 准备可用的pip源文件pip.conf 。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```

说明

在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索pypi ，也可以查看“pip.conf”文件内容。

5. 下载“torch*.whl ”文件。

在网站“https://download.pytorch.org/whl/torch_stable.html”搜索并下载如下whl文件。

- torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
- torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
- torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

说明

“+”符号的URL编码为“%2B”，在上述网站中搜索目标文件名时，需要将原文件名中的“+”符号替换为“%2B”。

例如“torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl”。

6. 下载Miniconda3安装文件。

使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 将上述pip源文件、torch*.whl文件、Miniconda3安装文件放置在context文件夹内，context文件夹内容如下。

```
context
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

8. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root
```



```
# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 torch*.whl
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim和curl 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y vim curl && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user
```

关于Dockerfile文件编写的更多指导内容参见**Docker官方文档**。

9. 确认已创建完成Dockerfile文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

10. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像pytorch:1.8.1-cuda11.1。

```
docker build . -t pytorch:1.8.1-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged pytorch:1.8.1-cuda11.1
```


Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker tag pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
   ```
   b. 使用docker push命令上传镜像。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
   ```
6. 完成镜像上传后，在容器镜像服务控制台的“我的镜像”页面可查看已上传的自定义镜像。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**快速配置ModelArts委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”
   - 启动方式：选择“自定义”
   - 镜像地址：**Step5 上传镜像至SWR服务**中创建的镜像。
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/pytorch/demo-code/”，训练代码会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 启动命令：“/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py”，此处的“demo-code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 资源池：选择公共资源池
   - 类型：选择GPU或者CPU规格。
   - 永久保存日志：打开
   - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/pytorch/log/”
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

图 10-18 GPU 规格运行日志信息

```
tensor([[-0.4181,  0.8150, -0.2581],
        [-0.6062,  0.5347,  0.1890],
        [ 0.5751,  1.2730, -0.3907],
        [ 0.4812, -0.4064, -0.2753],
        [ 1.0377, -1.1248,  1.2977]])
tensor([[-0.7440, -0.8577, -0.2340],
        [ 0.9569,  0.5516, -1.3350],
        [-1.2878, -0.2791,  0.3486],
        [-1.0997,  0.7627, -0.3188],
        [-1.0865, -1.2626, -0.5900]], device='cuda:0')

```

10.4.6 从 0 制作自定义镜像用于创建训练作业（MPI+CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MPI，训练使用的资源是CPU或GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练作业。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **前提条件**
2. **Step1 创建OBS桶和文件夹**
3. **Step2 准备脚本文件并上传至OBS中**
4. **Step3 准备镜像主机**
5. **Step4 制作自定义镜像**
6. **Step5 上传镜像至SWR服务**
7. **Step6 在ModelArts上创建训练作业**

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表10-34**所示，示例中的桶名称“test-modelarts” 和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 10-34 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/mpi/demo-code/”	用于存储MPI启动脚本与训练脚本文件。
“obs://test-modelarts/mpi/log/”	用于存储训练日志文件。

Step2 准备脚本文件并上传至 OBS 中

准备本案例所需的MPI启动脚本run_mpi.sh文件和训练脚本mpi-verification.py文件，并上传至OBS桶的“obs://test-modelarts/mpi/demo-code/”文件夹下。

- MPI启动脚本run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
```

```
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-
p ${MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG -x NCCL_SOCKET_IFNAME -x NCCL_IB_HCA -x NCCL_IB_TIMEOUT -x
NCCL_IB_GID_INDEX -x NCCL_IB_TC \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x PATH -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?

    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    else
        echo "[run_mpi] exec command successfully, exited with $RET_CODE"
    fi

    # stop 1...N worker by killing the sleep proc
    sed -i '1d' ${MY_HOME}/hostfile
```

```
    if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
        echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

        sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
        printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

        mpirun \
        --hostfile ${MY_HOME}/hostfile \
        --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -x PATH -x LD_LIBRARY_PATH \
        pkill sleep \
        > /dev/null 2>&1
    fi

    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

说明

“run_mpi.sh”脚本需要以LF作为换行符。使用CRLF作为换行符会导致训练作业运行失败，日志中会打印“$'\r': command not found”的错误信息。

- 训练脚本mpi-verification.py文件内容如下：

```
import os
import socket

if __name__ == '__main__':
    print(socket.gethostname())

    # https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
    print('OMPI_COMM_WORLD_SIZE: ' + os.environ['OMPI_COMM_WORLD_SIZE'])
    print('OMPI_COMM_WORLD_RANK: ' + os.environ['OMPI_COMM_WORLD_RANK'])
    print('OMPI_COMM_WORLD_LOCAL_RANK: ' + os.environ['OMPI_COMM_WORLD_LOCAL_RANK'])
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker官方文档**。


```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```

如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

```
docker version | grep -A 1 Engine
```

命令回显如下。

```
 Engine:
  Version:          18.09.0
```

说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

```
mkdir -p context
```

4. 下载Miniconda3安装文件。

使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

5. 下载openmpi 3.0.0安装文件。

使用地址https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz，下载horovod v0.22.1已经编译好的openmpi 3.0.0文件。

6. 将上述Miniconda3安装文件、openmpi 3.0.0文件放置在context文件夹内，context文件夹内容如下。

```
context
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```

7. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 复制 Miniconda3 (python 3.7.13) 安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp

# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim / curl / net-tools / ssh 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping \
    openssh-client openssh-server && \
    ssh -V && \
```



```
    mkdir -p /run/sshd && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 配置 sshd，使得 ssh 可以免密登录
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
  StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config
```

关于Dockerfile文件编写的更多指导内容参见**Docker官方文档**。

8. 确认已创建完成Dockerfile文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```

9. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像mpi:3.0.0-cuda11.1。

```
docker build . -t mpi:3.0.0-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
naming to docker.io/library/mpi:3.0.0-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。

2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker tag mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
   b. 使用docker push命令上传镜像。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。

   “swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**快速配置ModelArts委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在ModelArts管理控制台，左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”
   - 启动方式：选择“自定义”
   - 镜像地址：“swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1”
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/mpi/demo-code/”
   - 启动命令：bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/mpi-verification.py
   - 环境变量：添加“MY_SSHD_PORT = 38888”
   - 资源池：选择公共资源池
   - 类型：选择GPU规格
   - 计算节点个数：选择“1”或“2”
   - 永久保存日志：打开
   - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/mpi/log/”
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如**图10-19**所示。

图 10-19 1 个计算节点 GPU 规格 worker-0 运行日志信息

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 0
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888           0.0.0.0:*               LISTEN      60/sshd
tcp6       0      0 :::38888                ::::*                   LISTEN      60/sshd
172.16.0.122 slots=1
modelarts-job-8cf8a682-21cb-4d73-9bb3-789cecdc458b-worker-0
OMPI_COMM_WORLD_SIZE: 1
OMPI_COMM_WORLD_RANK: 0
OMPI_COMM_WORLD_LOCAL_RANK: 0
```

计算节点个数选择为2，训练作业也可以运行。日志信息如图10-20和图10-21所示。

图 10-20 2 个计算节点 worker-0 运行日志信息

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 0
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888           0.0.0.0:*               LISTEN      61/sshd
tcp6       0      0 :::38888                ::::*                   LISTEN      61/sshd
172.16.0.39 slots=1
172.16.0.123 slots=1
Warning: Permanently added '[172.16.0.123]:38888' (RSA) to the list of known hosts.
modelarts-job-31732752-6857-4e33-96ff-7a28afae26fb-worker-0
OMPI_COMM_WORLD_SIZE: 2
OMPI_COMM_WORLD_RANK: 0
OMPI_COMM_WORLD_LOCAL_RANK: 0
modelarts-job-31732752-6857-4e33-96ff-7a28afae26fb-worker-1
OMPI_COMM_WORLD_SIZE: 2
OMPI_COMM_WORLD_RANK: 1
OMPI_COMM_WORLD_LOCAL_RANK: 0
```

图 10-21 2 个计算节点 worker-1 运行日志信息

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 1
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888           0.0.0.0:*               LISTEN      62/sshd
tcp6       0      0 :::38888                ::::*                   LISTEN      62/sshd
/home/ma-user/modelarts/user-job-dir/[illegible]e/run_mpi.sh: line 109:    66 Terminated              sleep 365d
```

10.4.7 从 0 制作自定义镜像用于创建训练作业（Tensorflow +GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是Tensorflow，训练使用的资源是GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用GPU规格资源运行训练作业。

- ubuntu-18.04
- cuda-11.2
- python-3.7.13
- mlnx ofed-5.4
- tensorflow gpu-2.10.0

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **前提条件**
2. **Step1 创建OBS桶和文件夹**
3. **Step2 创建数据集并上传至OBS**
4. **Step3 准备训练脚本并上传至OBS**
5. **Step4 准备镜像主机**
6. **Step5 制作自定义镜像**
7. **Step6 上传镜像至SWR服务**
8. **Step7 在ModelArts上创建训练作业**

前提条件

已注册华为云账号，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表10-35**所示，示例中的桶名称“test-modelarts” 和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 10-35 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/tensorflow/code/”	用于存储训练脚本文件。

文件夹名称	用途
“obs://test-modelarts/tensorflow/data/”	用于存储数据集文件。
“obs://test-modelarts/tensorflow/log/”	用于存储训练日志文件。

Step2 创建数据集并上传至 OBS

使用网站**https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz**，下载“mnist.npz”文件并上传至OBS桶的“obs://test-modelarts/tensorflow/data/”文件夹下。

Step3 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本mnist.py，并上传至OBS桶的“obs://test-modelarts/tensorflow/code/”文件夹下。

mnist.py文件内容如下：

```
import argparse
import tensorflow as tf

parser = argparse.ArgumentParser(description='TensorFlow quick start')
parser.add_argument('--data_url', type=str, default="./Data", help='path where the dataset is saved')
args = parser.parse_args()

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data(args.data_url)
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
```

Step4 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step5 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
   Engine:
    Version:          18.09.0
   ```

   说明

   推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

4. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

   ```
   [global]
   index-url = https://repo.huaweicloud.com/repository/pypi/simple
   trusted-host = repo.huaweicloud.com
   timeout = 120
   ```

   说明

   在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索pypi，也可以查看pip.conf文件内容。

5. 下载tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl文件。

   使用网站https://pypi.org/project/tensorflow-gpu/2.10.0/#files，下载tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl文件。

6. 下载Miniconda3安装文件。

   使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 编写容器镜像Dockerfile文件。

   在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

   ```
   # 容器镜像构建主机需要连通公网

   # 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
   #
   # https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
   # require Docker Engine >= 17.05
   #
   ```

```
# builder stage
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 tensorflow whl
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl

RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir keras==2.10.0

# 构建最终容器镜像
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp

# 安装 vim / curl / net-tools / mlnx ofed（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping && \
    # mlnx ofed
    apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1
libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-
dev flex chrpath libltdl-dev && \
    cd /tmp && \
    tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
    MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --
without-fw-update -q && \
    cd - && \
    rm -rf /tmp/* && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
    LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH \
    PYTHONUNBUFFERED=1
```

关于Dockerfile文件编写的更多指导内容参见**Docker官方文档**。

8. 下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。
   进入**地址**，单击“Download”，“Version”选择“5.4-3.5.8.0-LTS”，“OSDistributionVersion”选择“Ubuntu 18.04”，“Architecture”选择“x86_64”，下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。
9. 将上述Dockerfile文件、 Miniconda3 安装文件等放置在context文件夹内，context文件夹内容如下。
   ```
   context
   ├── Dockerfile
   ├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
   ├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
   ├── pip.conf
   └── tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
   ```
10. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像tensorflow:2.10.0-ofed-cuda11.2。
    ```
    docker build . -t tensorflow:2.10.0-ofed-cuda11.2
    ```
    构建过程结束时出现如下构建日志说明镜像构建成功。
    ```
    Successfully tagged tensorflow:2.10.0-ofed-cuda11.2
    ```

Step6 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。
      ```
      #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
      sudo docker tag tensorflow:2.10.0-ofed-cuda11.2 swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
      ```
   b. 使用docker push命令上传镜像。
      ```
      #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
      sudo docker push swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
      ```
6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。

Step7 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**快速配置ModelArts委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“下一步”。
   - 创建方式：选择“自定义算法”。
   - 镜像来源：选择“自定义”。
   - 镜像地址：**Step5 制作自定义镜像**中创建的镜像。
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/tensorflow/code/”，训练代码会被自动下载至训练容器的“${MA_JOB_DIR}/code”目录中，“code”为OBS存放代码路径的最后一级目录，可以根据实际修改。

- 启动命令：“python ${MA_JOB_DIR}/code/mnist.py”，此处的“code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
- 训练输入：单击“增加训练输入”，参数名称设置为“data_path”，选择OBS中存放“mnist.npz”的目录，例如“obs://test-modelarts/tensorflow/data/mnist.npz”，获取方式设置为“超参”。
- 资源池：选择公共资源池。
- 资源类型：选择GPU规格。
- 计算节点个数：1个。
- 永久保存日志：打开。
- 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/mindspore-gpu/log/”。

4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

图 10-22 GPU 规格运行日志信息

```
    0.9767
323 1503/1875 [=====================>......] - ETA: 0s - loss: 0.0741 - accuracy:
    0.9769
324 1533/1875 [======================>......] - ETA: 0s - loss: 0.0743 - accuracy:
    0.9769
325 1564/1875 [======================>.....] - ETA: 0s - loss: 0.0746 - accuracy:
    0.9768
326 1595/1875 [======================>.....] - ETA: 0s - loss: 0.0741 - accuracy:
    0.9770
327 1624/1875 [======================>.....] - ETA: 0s - loss: 0.0742 - accuracy:
    0.9770
328 1654/1875 [=======================>....] - ETA: 0s - loss: 0.0745 - accuracy:
    0.9770
329 1685/1875 [=======================>....] - ETA: 0s - loss: 0.0747 - accuracy:
    0.9768
330 1716/1875 [=======================>...] - ETA: 0s - loss: 0.0752 - accuracy:
    0.9767
331 1747/1875 [========================>...] - ETA: 0s - loss: 0.0755 - accuracy:
    0.9767
332 1778/1875 [========================>..] - ETA: 0s - loss: 0.0753 - accuracy:
    0.9767
333 1809/1875 [========================>..] - ETA: 0s - loss: 0.0751 - accuracy:
    0.9768
334 1841/1875 [=========================>.] - ETA: 0s - loss: 0.0753 - accuracy:
    0.9767
335 1872/1875 [=========================>.] - ETA: 0s - loss: 0.0753 - accuracy:
    0.9767
336 1875/1875 [==============================] - 3s 2ms/step - loss: 0.0752 - accuracy: 0.9767
```

10.4.8 从 0 制作自定义镜像用于创建训练作业（MindSpore +Ascend）

本案例介绍如何从0到1制作Ascend容器镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MindSpore，训练使用的资源是专属资源池的Ascend芯片。

场景描述

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用Ascend规格资源运行训练作业。

- ubuntu-18.04
- cann-6.3.RC2 (商用版本)
- python-3.7.13
- mindspore-2.1.1

说明

- 本教程以cann-6.3.RC2、mindspore-2.1.1为例介绍。
- 本示例仅用于示意Ascend容器镜像制作流程，且在匹配正确的Ascend驱动/固件版本的专属资源池上运行通过。

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **Step1 创建OBS桶和文件夹**
2. **Step2 准备脚本文件并上传至OBS中**
3. **Step3 制作自定义镜像**
4. **Step4 上传镜像至SWR**
5. **Step5 在ModelArts上创建Notebook并调试**
6. **Step6 在ModelArts上创建训练作业**

约束限制

- 由于案例中需要下载商用版CANN，因此本案例仅面向有下载权限的渠道用户，非渠道用户建议参考其他自定义镜像制作教程。
- Mindspore版本与CANN版本，CANN版本与Ascend驱动/固件版本均有严格的匹配关系，版本不匹配会导致训练失败。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。如下示例中，请创建命名为“test-modelarts”的桶，并创建如**表10-36**所示的文件夹。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 10-36 OBS 桶文件夹列表

文件夹名称	用途
obs://test-modelarts/ascend/demo-code/	用于存储Ascend训练脚本文件。

文件夹名称	用途
obs://test-modelarts/ascend/demo-code/run_ascend/	用于存储Ascend训练脚本的启动脚本。
obs://test-modelarts/ascend/log/	用于存储训练日志文件。

Step2 准备脚本文件并上传至 OBS 中

1. 准备本案例所需训练脚本mindspore-verification.py文件和Ascend的启动脚本文件（共5个）。
   - 训练脚本文件具体内容请参见**训练mindspore-verification.py文件**。
   - Ascend的启动脚本文件包括以下5个，具体脚本内容请参见**Ascend的启动脚本文件**。
     i. run_ascend.py
     ii. common.py
     iii. rank_table.py
     iv. manager.py
     v. fmk.py

   说明

   mindspore-verification.py和run_ascend.py脚本文件在创建训练作业时的“启动命令”参数中调用，具体请参见**启动命令**。

   run_ascend.py脚本运行时会调用common.py、rank_table.py、manager.py、fmk.py脚本。
2. 上传训练脚本mindspore-verification.py文件至OBS桶的“obs://test-modelarts/ascend/demo-code/”文件夹下。
3. 上传Ascend的启动脚本文件（共5个）至OBS桶的“obs://test-modelarts/ascend/demo-code/run_ascend/”文件夹下。

Step3 制作自定义镜像

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cann-6.3.RC2(商用版本)
- python-3.7.13
- mindspore-2.1.1

说明

Mindspore版本与CANN版本，CANN版本和Ascend驱动/固件版本均有严格的匹配关系，版本不匹配会导致训练失败。

本示例仅用于示意Ascend容器镜像制作流程，且在匹配正确的Ascend驱动/固件版本的专属资源池上运行通过。

1. 准备一台Linux **aarch64**架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

2. 安装Docker。

   以Linux **aarch64**架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

   启动docker。

   ```
   systemctl start docker
   ```

3. 确认Docker Engine版本。执行如下命令。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
   Engine:
    Version:          18.09.0
   ```

   说明

   推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

4. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

5. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

   ```
   [global]
   index-url = https://repo.huaweicloud.com/repository/pypi/simple
   trusted-host = repo.huaweicloud.com
   timeout = 120
   ```

   说明

   在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索pypi，可以查看pip.conf文件内容。

6. 准备可用的apt源文件Ubuntu-Ports-bionic.list。本示例使用华为开源镜像站提供的apt源，执行如下命令获取apt源文件。

   ```
   wget -O Ubuntu-Ports-bionic.list https://repo.huaweicloud.com/repository/conf/Ubuntu-Ports-bionic.list
   ```

   说明

   在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索Ubuntu-Ports，可以查看获取apt源文件的命令。

7. 下载CANN 6.3.RC2-linux aarch64与mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl安装文件。

   - 下载run文件“Ascend-cann-nnae_6.3.RC2_linux-aarch64.run”（**下载链接**）。
   - 下载whl文件“mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl”（**下载链接**）。

   说明

   ModelArts当前仅支持CANN商用版本，不支持社区版。

8. 下载Miniconda3安装文件。

   使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.10.3-Linux-aarch64.sh，下载Miniconda3-py37-4.10.3安装文件（对应python 3.7.10）。

9. 将上述pip源文件、*.run文件、*.whl文件、Miniconda3安装文件放置在context文件夹内，context文件夹内容如下。

```
context
├── Ascend-cann-nnae_6.3.RC2_linux-aarch64.run
├── mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl
├── Miniconda3-py37_4.10.3-Linux-aarch64.sh
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

10. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网
FROM arm64v8/ubuntu:18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 安装 OS 依赖（使用华为开源镜像站）
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y \
    # utils
    ca-certificates vim curl \
    # CANN 6.3.RC2
    gcc-7 g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils
net-tools libblas-dev gfortran libblas3 && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    # 修改 CANN 6.3.RC2 安装目录的父目录权限，使得 ma-user 可以写入
    chmod o+w /usr/local

RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /home/ma-user/.pip/
COPY --chown=ma-user:100 pip.conf /home/ma-user/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY --chown=ma-user:100 Miniconda3-py37_4.10.3-Linux-aarch64.sh /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.10.3-Linux-aarch64.sh -b -p /home/ma-user/miniconda3

ENV PATH=$PATH:/home/ma-user/miniconda3/bin

# 安装 CANN 6.3.RC2 Python Package 依赖
RUN pip install numpy~=1.14.3 decorator~=4.4.0 sympy~=1.4 cffi~=1.12.3 protobuf~=3.11.3 \
    attrs pyyaml pathlib2 scipy requests psutil absl-py

# 安装 CANN 6.3.RC2 至 /usr/local/Ascend 目录
COPY --chown=ma-user:100 Ascend-cann-nnae_6.3.RC2_linux-aarch64.run /tmp
RUN chmod +x /tmp/Ascend-cann-nnae_6.3.RC2_linux-aarch64.run && \
    /tmp/Ascend-cann-nnae_6.3.RC2_linux-aarch64.run --install --install-path=/usr/local/Ascend

# 安装 MindSpore 2.1.1
COPY --chown=ma-user:100 mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl /tmp
RUN chmod +x /tmp/mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl && \
    pip install /tmp/mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl

# 构建最终容器镜像
FROM arm64v8/ubuntu:18.04
```



```
# 安装 OS 依赖（使用华为开源镜像站）
COPY Ubuntu-Ports-bionic.list /tmp
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    mv /tmp/Ubuntu-Ports-bionic.list /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y \
    # utils
    ca-certificates vim curl \
    # CANN 6.3.RC2
    gcc-7 g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev libffi-dev unzip pciutils
net-tools libblas-dev gfortran libblas3 && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3
COPY --chown=ma-user:100 --from=builder /home/ma-user/Ascend /home/ma-user/Ascend
COPY --chown=ma-user:100 --from=builder /home/ma-user/var /home/ma-user/var
COPY --chown=ma-user:100 --from=builder /usr/local/Ascend /usr/local/Ascend

# 设置容器镜像预置环境变量
# 请务必设置 CANN 相关环境变量
# 请务必设置 Ascend Driver 相关环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/usr/local/Ascend/nnae/latest/bin:/usr/local/Ascend/nnae/latest/compiler/
ccec_compiler/bin:/home/ma-user/miniconda3/bin \
    LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/Ascend/driver/lib64:/usr/local/Ascend/driver/
lib64/common:/usr/local/Ascend/driver/lib64/driver:/usr/local/Ascend/nnae/latest/lib64:/usr/local/
Ascend/nnae/latest/lib64/plugin/opskernel:/usr/local/Ascend/nnae/latest/lib64/plugin/nnengine \
    PYTHONPATH=$PYTHONPATH:/usr/local/Ascend/nnae/latest/python/site-packages:/usr/local/
Ascend/nnae/latest/opp/built-in/op_impl/ai_core/tbe \
    ASCEND_AICPU_PATH=/usr/local/Ascend/nnae/latest \
    ASCEND_OPP_PATH=/usr/local/Ascend/nnae/latest/opp \
    ASCEND_HOME_PATH=/usr/local/Ascend/nnae/latest \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user
```

关于Dockerfile文件编写的更多指导内容参见**Docker官方文档**。

11. 确认已创建完成Dockerfile文件。此时context文件夹内容如下。

```
context
├── Ascend-cann-nnae_6.3.RC2_linux-aarch64.run
├── Dockerfile
├── mindspore-2.1.1-cp37-cp37m-linux_aarch64.whl
├── Miniconda3-py37_4.10.3-Linux-aarch64.sh
├── pip.conf
└── Ubuntu-Ports-bionic.list
```

12. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像。

```
docker build . -t mindspore:2.1.1-cann6.3.RC2
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged mindspore:2.1.1-cann6.3.RC2
```

13. 将制作完成的镜像上传至SWR服务，具体参见**Step4 上传镜像至SWR**。

Step4 上传镜像至 SWR

本章节介绍如何将制作好的镜像上传至SWR服务，方便后续在ModelArts上创建训练作业时调用。

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。

2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker tag mindspore:2.1.1-cann6.3.RC2 swr.{region}.{domain}/deep-learning/mindspore:2.1.1-cann6.3.RC2
   ```
   b. 使用docker push命令上传镜像。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker push swr.{region}.{domain}/deep-learning/mindspore:2.1.1-cann6.3.RC2
   ```
6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。

Step5 在 ModelArts 上创建 Notebook 并调试

1. 将上传到SWR上的镜像注册到ModelArts的镜像管理中。

   登录ModelArts管理控制台，在左侧导航栏中选择“镜像管理 ”，单击“注册镜像”，根据界面提示注册镜像。注册后的镜像可以用于创建Notebook。
2. 在Notebook中使用自定义镜像创建Notebook并调试，调试成功后，保存镜像。
   a. 在Notebook中使用自定义镜像创建Notebook操作请参见**基于自定义镜像创建Notebook实例**。
   b. 保存Notebook镜像操作请参见**保存Notebook镜像环境**。
3. 已有的镜像调试成功后，再**使用ModelArts训练模块训练作业**。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。
2. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”
   - 启动方式：选择“自定义”
   - 镜像地址：
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/ascend/demo-code/”
   - 启动命令：“python ${MA_JOB_DIR}/demo-code/run_ascend/run_ascend.py python ${MA_JOB_DIR}/demo-code/mindspore-verification.py”
   - 资源池：选择专属资源池
   - 类型：选择驱动/固件版本匹配的专属资源池Ascend规格。
   - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/ascend/log/”
3. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
4. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如图10-23所示。

图 10-23 专属资源池 Ascend 规格运行日志信息

```
Ascend Envs
------
JOB_ID:  modelarts-job-2436291a-8543-4ab8-84ad-2dda8f1e4f5c
RANK_TABLE_FILE:  /home/ma-user/modelarts/rank_table/jobstart_hccl.json
RANK_SIZE:  1
ASCEND_DEVICE_ID:  0
DEVICE_ID:  0
RANK_ID:  0
------
   [2. 2. 2. 2.]
   [2. 2. 2. 2.]]

  [[2. 2. 2. 2.]
   [2. 2. 2. 2.]
   [2. 2. 2. 2.]]

  [[2. 2. 2. 2.]
   [2. 2. 2. 2.]
   [2. 2. 2. 2.]]]
```

训练 mindspore-verification.py 文件

mindspore-verification.py文件内容如下：

```
import os
import numpy as np
from mindspore import Tensor
import mindspore.ops as ops
import mindspore.context as context

print('Ascend Envs')
print('------')
print('JOB_ID: ', os.environ['JOB_ID'])
print('RANK_TABLE_FILE: ', os.environ['RANK_TABLE_FILE'])
print('RANK_SIZE: ', os.environ['RANK_SIZE'])
print('ASCEND_DEVICE_ID: ', os.environ['ASCEND_DEVICE_ID'])
print('DEVICE_ID: ', os.environ['DEVICE_ID'])
print('RANK_ID: ', os.environ['RANK_ID'])
print('------')

context.set_context(device_target="Ascend")
x = Tensor(np.ones([1,3,3,4]).astype(np.float32))
y = Tensor(np.ones([1,3,3,4]).astype(np.float32))

print(ops.add(x, y))
```

Ascend 的启动脚本文件

- run_ascend.py

  ```
  import sys
  import os

  from common import RunAscendLog
  from common import RankTableEnv

  from rank_table import RankTable, RankTableTemplate1, RankTableTemplate2
  ```

```
from manager import FMKManager

if __name__ == '__main__':
    log = RunAscendLog.setup_run_ascend_logger()

    if len(sys.argv) <= 1:
        log.error('there are not enough args')
        sys.exit(1)

    train_command = sys.argv[1:]
    log.info('training command')
    log.info(train_command)

    if os.environ.get(RankTableEnv.RANK_TABLE_FILE_V1) is not None:
        # new format rank table file
        rank_table_path = os.environ.get(RankTableEnv.RANK_TABLE_FILE_V1)
        RankTable.wait_for_available(rank_table_path)
        rank_table = RankTableTemplate1(rank_table_path)
    else:
        # old format rank table file
        rank_table_path_origin = RankTableEnv.get_rank_table_template2_file_path()
        RankTable.wait_for_available(rank_table_path_origin)
        rank_table = RankTableTemplate2(rank_table_path_origin)

    if rank_table.get_device_num() >= 1:
        log.info('set rank table %s env to %s' % (RankTableEnv.RANK_TABLE_FILE,
rank_table.get_rank_table_path()))
        RankTableEnv.set_rank_table_env(rank_table.get_rank_table_path())
    else:
        log.info('device num < 1, unset rank table %s env' % RankTableEnv.RANK_TABLE_FILE)
        RankTableEnv.unset_rank_table_env()

    instance = rank_table.get_current_instance()
    server = rank_table.get_server(instance.server_id)
    current_instance = RankTable.convert_server_to_instance(server)

    fmk_manager = FMKManager(current_instance)
    fmk_manager.run(rank_table.get_device_num(), train_command)
    return_code = fmk_manager.monitor()

    fmk_manager.destroy()

    sys.exit(return_code)
```

- common.py

```
import logging
import os

logo = 'Training'


# Rank Table Constants
class RankTableEnv:
    RANK_TABLE_FILE = 'RANK_TABLE_FILE'

    RANK_TABLE_FILE_V1 = 'RANK_TABLE_FILE_V_1_0'

    HCCL_CONNECT_TIMEOUT = 'HCCL_CONNECT_TIMEOUT'

    # jobstart_hccl.json is provided by the volcano controller of Cloud-Container-Engine(CCE)
    HCCL_JSON_FILE_NAME = 'jobstart_hccl.json'

    RANK_TABLE_FILE_DEFAULT_VALUE = '/user/config/%s' % HCCL_JSON_FILE_NAME

    @staticmethod
    def get_rank_table_template1_file_dir():
        parent_dir = os.environ[ModelArts.MA_MOUNT_PATH_ENV]
        return os.path.join(parent_dir, 'rank_table')
```

```
    @staticmethod
    def get_rank_table_template2_file_path():
        rank_table_file_path = os.environ.get(RankTableEnv.RANK_TABLE_FILE)
        if rank_table_file_path is None:
            return RankTableEnv.RANK_TABLE_FILE_DEFAULT_VALUE

        return os.path.join(os.path.normpath(rank_table_file_path),
RankTableEnv.HCCL_JSON_FILE_NAME)

    @staticmethod
    def set_rank_table_env(path):
        os.environ[RankTableEnv.RANK_TABLE_FILE] = path

    @staticmethod
    def unset_rank_table_env():
        del os.environ[RankTableEnv.RANK_TABLE_FILE]


class ModelArts:
    MA_MOUNT_PATH_ENV = 'MA_MOUNT_PATH'
    MA_CURRENT_INSTANCE_NAME_ENV = 'MA_CURRENT_INSTANCE_NAME'
    MA_VJ_NAME = 'MA_VJ_NAME'

    MA_CURRENT_HOST_IP = 'MA_CURRENT_HOST_IP'

    CACHE_DIR = '/cache'

    TMP_LOG_DIR = '/tmp/log/'

    FMK_WORKSPACE = 'workspace'

    @staticmethod
    def get_current_instance_name():
        return os.environ[ModelArts.MA_CURRENT_INSTANCE_NAME_ENV]

    @staticmethod
    def get_current_host_ip():
        return os.environ.get(ModelArts.MA_CURRENT_HOST_IP)

    @staticmethod
    def get_job_id():
        ma_vj_name = os.environ[ModelArts.MA_VJ_NAME]
        return ma_vj_name.replace('ma-job', 'modelarts-job', 1)

    @staticmethod
    def get_parent_working_dir():
        if ModelArts.MA_MOUNT_PATH_ENV in os.environ:
            return os.path.join(os.environ.get(ModelArts.MA_MOUNT_PATH_ENV),
ModelArts.FMK_WORKSPACE)

        return ModelArts.CACHE_DIR


class RunAscendLog:

    @staticmethod
    def setup_run_ascend_logger():
        name = logo
        formatter = logging.Formatter(fmt='[run ascend] %(asctime)s - %(levelname)s - %(message)s')

        handler = logging.StreamHandler()
        handler.setFormatter(formatter)

        logger = logging.getLogger(name)
        logger.setLevel(logging.INFO)
        logger.addHandler(handler)
        logger.propagate = False
        return logger
```

```
    @staticmethod
    def get_run_ascend_logger():
        return logging.getLogger(logo)
```

- rank_table.py

```
import json
import time
import os

from common import ModelArts
from common import RunAscendLog
from common import RankTableEnv

log = RunAscendLog.get_run_ascend_logger()


class Device:
    def __init__(self, device_id, device_ip, rank_id):
        self.device_id = device_id
        self.device_ip = device_ip
        self.rank_id = rank_id


class Instance:
    def __init__(self, pod_name, server_id, devices):
        self.pod_name = pod_name
        self.server_id = server_id
        self.devices = self.parse_devices(devices)

    @staticmethod
    def parse_devices(devices):
        if devices is None:
            return []
        device_object_list = []
        for device in devices:
            device_object_list.append(Device(device['device_id'], device['device_ip'], ''))

        return device_object_list

    def set_devices(self, devices):
        self.devices = devices


class Group:
    def __init__(self, group_name, device_count, instance_count, instance_list):
        self.group_name = group_name
        self.device_count = int(device_count)
        self.instance_count = int(instance_count)
        self.instance_list = self.parse_instance_list(instance_list)

    @staticmethod
    def parse_instance_list(instance_list):
        instance_object_list = []
        for instance in instance_list:
            instance_object_list.append(
                Instance(instance['pod_name'], instance['server_id'], instance['devices']))

        return instance_object_list


class RankTable:
    STATUS_FIELD = 'status'
    COMPLETED_STATUS = 'completed'

    def __init__(self):
        self.rank_table_path = ""
        self.rank_table = {}

    @staticmethod
    def read_from_file(file_path):
```

```
        with open(file_path) as json_file:
            return json.load(json_file)

    @staticmethod
    def wait_for_available(rank_table_file, period=1):
        log.info('Wait for Rank table file at %s ready' % rank_table_file)
        complete_flag = False
        while not complete_flag:
            with open(rank_table_file) as json_file:
                data = json.load(json_file)
                if data[RankTable.STATUS_FIELD] == RankTable.COMPLETED_STATUS:
                    log.info('Rank table file is ready for read')
                    log.info('\n' + json.dumps(data, indent=4))
                    return True

            time.sleep(period)

        return False

    @staticmethod
    def convert_server_to_instance(server):
        device_list = []
        for device in server['device']:
            device_list.append(
                Device(device_id=device['device_id'], device_ip=device['device_ip'],
rank_id=device['rank_id']))

        ins = Instance(pod_name='', server_id=server['server_id'], devices=[])
        ins.set_devices(device_list)
        return ins

    def get_rank_table_path(self):
        return self.rank_table_path

    def get_server(self, server_id):
        for server in self.rank_table['server_list']:
            if server['server_id'] == server_id:
                log.info('Current server')
                log.info('\n' + json.dumps(server, indent=4))
                return server

        log.error('server [%s] is not found' % server_id)
        return None


class RankTableTemplate2(RankTable):

    def __init__(self, rank_table_template2_path):
        super().__init__()

        json_data = self.read_from_file(file_path=rank_table_template2_path)

        self.status = json_data[RankTableTemplate2.STATUS_FIELD]
        if self.status != RankTableTemplate2.COMPLETED_STATUS:
            return

        # sorted instance list by the index of instance
        # assert there is only one group
        json_data["group_list"][0]["instance_list"] = sorted(json_data["group_list"][0]["instance_list"],
                                                             key=RankTableTemplate2.get_index)

        self.group_count = int(json_data['group_count'])
        self.group_list = self.parse_group_list(json_data['group_list'])

        self.rank_table_path, self.rank_table = self.convert_template2_to_template1_format_file()

    @staticmethod
    def parse_group_list(group_list):
        group_object_list = []
```

```
        for group in group_list:
            group_object_list.append(
                Group(group['group_name'], group['device_count'], group['instance_count'],
group['instance_list']))

        return group_object_list

    @staticmethod
    def get_index(instance):
        # pod_name example: job94dc1dbf-job-bj4-yolov4-15
        pod_name = instance["pod_name"]
        return int(pod_name[pod_name.rfind("-") + 1:])

    def get_current_instance(self):
        """
        get instance by pod name
        specially, return the first instance when the pod name is None
        :return:
        """
        pod_name = ModelArts.get_current_instance_name()
        if pod_name is None:
            if len(self.group_list) > 0:
                if len(self.group_list[0].instance_list) > 0:
                    return self.group_list[0].instance_list[0]

            return None

        for group in self.group_list:
            for instance in group.instance_list:
                if instance.pod_name == pod_name:
                    return instance
        return None

    def convert_template2_to_template1_format_file(self):
        rank_table_template1_file = {
            'status': 'completed',
            'version': '1.0',
            'server_count': '0',
            'server_list': []
        }

        logic_index = 0
        server_map = {}
        # collect all devices in all groups
        for group in self.group_list:
            if group.device_count == 0:
                continue
            for instance in group.instance_list:
                if instance.server_id not in server_map:
                    server_map[instance.server_id] = []

                for device in instance.devices:
                    template1_device = {
                        'device_id': device.device_id,
                        'device_ip': device.device_ip,
                        'rank_id': str(logic_index)
                    }
                    logic_index += 1
                    server_map[instance.server_id].append(template1_device)

        server_count = 0
        for server_id in server_map:
            rank_table_template1_file['server_list'].append({
                'server_id': server_id,
                'device': server_map[server_id]
            })
            server_count += 1

        rank_table_template1_file['server_count'] = str(server_count)
```

```
        log.info('Rank table file (Template1)')
        log.info('\n' + json.dumps(rank_table_template1_file, indent=4))

        if not os.path.exists(RankTableEnv.get_rank_table_template1_file_dir()):
            os.makedirs(RankTableEnv.get_rank_table_template1_file_dir())

        path = os.path.join(RankTableEnv.get_rank_table_template1_file_dir(),
RankTableEnv.HCCL_JSON_FILE_NAME)
        with open(path, 'w') as f:
            f.write(json.dumps(rank_table_template1_file))
            log.info('Rank table file (Template1) is generated at %s', path)

        return path, rank_table_template1_file

    def get_device_num(self):
        total_device_num = 0
        for group in self.group_list:
            total_device_num += group.device_count
        return total_device_num


class RankTableTemplate1(RankTable):
    def __init__(self, rank_table_template1_path):
        super().__init__()
        self.rank_table_path = rank_table_template1_path
        self.rank_table = self.read_from_file(file_path=rank_table_template1_path)

    def get_current_instance(self):
        current_server = None
        server_list = self.rank_table['server_list']
        if len(server_list) == 1:
            current_server = server_list[0]
        elif len(server_list) > 1:
            host_ip = ModelArts.get_current_host_ip()
            if host_ip is not None:
                for server in server_list:
                    if server['server_id'] == host_ip:
                        current_server = server
                        break
            else:
                current_server = server_list[0]

        if current_server is None:
            log.error('server is not found')
            return None
        return self.convert_server_to_instance(current_server)

    def get_device_num(self):
        server_list = self.rank_table['server_list']
        device_num = 0
        for server in server_list:
            device_num += len(server['device'])
        return device_num
```

- manager.py

```
import time
import os
import os.path
import signal

from common import RunAscendLog
from fmk import FMK


log = RunAscendLog.get_run_ascend_logger()


class FMKManager:
    # max destroy time: ~20 (15 + 5)
```

```
# ~ 15 (1 + 2 + 4 + 8)
MAX_TEST_PROC_CNT = 4

def __init__(self, instance):
    self.instance = instance
    self.fmk = []
    self.fmk_processes = []
    self.get_sigterm = False
    self.max_test_proc_cnt = FMKManager.MAX_TEST_PROC_CNT

# break the monitor and destroy processes when get terminate signal
def term_handle(func):
    def receive_term(signum, stack):
        log.info('Received terminate signal %d, try to destroyed all processes' % signum)
        stack.f_locals['self'].get_sigterm = True

    def handle_func(self, *args, **kwargs):
        origin_handle = signal.getsignal(signal.SIGTERM)
        signal.signal(signal.SIGTERM, receive_term)
        res = func(self, *args, **kwargs)
        signal.signal(signal.SIGTERM, origin_handle)
        return res

    return handle_func

def run(self, rank_size, command):
    for index, device in enumerate(self.instance.devices):
        fmk_instance = FMK(index, device)
        self.fmk.append(fmk_instance)

        self.fmk_processes.append(fmk_instance.run(rank_size, command))

@term_handle
def monitor(self, period=1):
    # busy waiting for all fmk processes exit by zero
    # or there is one process exit by non-zero

    fmk_cnt = len(self.fmk_processes)
    zero_ret_cnt = 0
    while zero_ret_cnt != fmk_cnt:
        zero_ret_cnt = 0
        for index in range(fmk_cnt):
            fmk = self.fmk[index]
            fmk_process = self.fmk_processes[index]
            if fmk_process.poll() is not None:
                if fmk_process.returncode != 0:
                    log.error('proc-rank-%s-device-%s (pid: %d) has exited with non-zero code: %d'
                              % (fmk.rank_id, fmk.device_id, fmk_process.pid, fmk_process.returncode))
                    return fmk_process.returncode

                zero_ret_cnt += 1
        if self.get_sigterm:
            break
        time.sleep(period)

    return 0

def destroy(self, base_period=1):
    log.info('Begin destroy training processes')
    self.send_sigterm_to_fmk_process()
    self.wait_fmk_process_end(base_period)
    log.info('End destroy training processes')

def send_sigterm_to_fmk_process(self):
    # send SIGTERM to fmk processes (and process group)
    for r_index in range(len(self.fmk_processes) - 1, -1, -1):
        fmk = self.fmk[r_index]
        fmk_process = self.fmk_processes[r_index]
        if fmk_process.poll() is not None:
```

```
                log.info('proc-rank-%s-device-%s (pid: %d) has exited before receiving the term signal',
                        fmk.rank_id, fmk.device_id, fmk_process.pid)
                del self.fmk_processes[r_index]
                del self.fmk[r_index]

            try:
                os.killpg(fmk_process.pid, signal.SIGTERM)
            except ProcessLookupError:
                pass

    def wait_fmk_process_end(self, base_period):
        test_cnt = 0
        period = base_period
        while len(self.fmk_processes) > 0 and test_cnt < self.max_test_proc_cnt:
            for r_index in range(len(self.fmk_processes) - 1, -1, -1):
                fmk = self.fmk[r_index]
                fmk_process = self.fmk_processes[r_index]
                if fmk_process.poll() is not None:
                    log.info('proc-rank-%s-device-%s (pid: %d) has exited',
                            fmk.rank_id, fmk.device_id, fmk_process.pid)
                    del self.fmk_processes[r_index]
                    del self.fmk[r_index]
            if not self.fmk_processes:
                break

            time.sleep(period)
            period *= 2
            test_cnt += 1

        if len(self.fmk_processes) > 0:
            for r_index in range(len(self.fmk_processes) - 1, -1, -1):
                fmk = self.fmk[r_index]
                fmk_process = self.fmk_processes[r_index]
                if fmk_process.poll() is None:
                    log.warn('proc-rank-%s-device-%s (pid: %d) has not exited within the max waiting time,
'
                            'send kill signal',
                            fmk.rank_id, fmk.device_id, fmk_process.pid)
                    os.killpg(fmk_process.pid, signal.SIGKILL)
```

- fmk.py

```
import os
import subprocess
import pathlib
from contextlib import contextmanager

from common import RunAscendLog
from common import RankTableEnv
from common import ModelArts

log = RunAscendLog.get_run_ascend_logger()


class FMK:

    def __init__(self, index, device):
        self.job_id = ModelArts.get_job_id()
        self.rank_id = device.rank_id
        self.device_id = str(index)

    def gen_env_for_fmk(self, rank_size):
        current_envs = os.environ.copy()

        current_envs['JOB_ID'] = self.job_id

        current_envs['ASCEND_DEVICE_ID'] = self.device_id
        current_envs['DEVICE_ID'] = self.device_id

        current_envs['RANK_ID'] = self.rank_id
        current_envs['RANK_SIZE'] = str(rank_size)
```

```
    FMK.set_env_if_not_exist(current_envs, RankTableEnv.HCCL_CONNECT_TIMEOUT, str(1800))

    log_dir = FMK.get_log_dir()
    process_log_path = os.path.join(log_dir, self.job_id, 'ascend', 'process_log', 'rank_' + self.rank_id)
    FMK.set_env_if_not_exist(current_envs, 'ASCEND_PROCESS_LOG_PATH', process_log_path)
    pathlib.Path(current_envs['ASCEND_PROCESS_LOG_PATH']).mkdir(parents=True, exist_ok=True)

    return current_envs

@contextmanager
def switch_directory(self, directory):
    owd = os.getcwd()
    try:
        os.chdir(directory)
        yield directory
    finally:
        os.chdir(owd)

def get_working_dir(self):
    fmk_workspace_prefix = ModelArts.get_parent_working_dir()
    return os.path.join(os.path.normpath(fmk_workspace_prefix), 'device%s' % self.device_id)

@staticmethod
def get_log_dir():
    parent_path = os.getenv(ModelArts.MA_MOUNT_PATH_ENV)
    if parent_path:
        log_path = os.path.join(parent_path, 'log')
        if os.path.exists(log_path):
            return log_path

    return ModelArts.TMP_LOG_DIR

@staticmethod
def set_env_if_not_exist(envs, env_name, env_value):
    if env_name in os.environ:
        log.info('env already exists. env_name: %s, env_value: %s ' % (env_name, env_value))
        return

    envs[env_name] = env_value

def run(self, rank_size, command):
    envs = self.gen_env_for_fmk(rank_size)
    log.info('bootstrap proc-rank-%s-device-%s' % (self.rank_id, self.device_id))

    log_dir = FMK.get_log_dir()
    if not os.path.exists(log_dir):
        os.makedirs(log_dir)

    log_file = '%s-proc-rank-%s-device-%s.txt' % (self.job_id, self.rank_id, self.device_id)
    log_file_path = os.path.join(log_dir, log_file)

    working_dir = self.get_working_dir()
    if not os.path.exists(working_dir):
        os.makedirs(working_dir)

    with self.switch_directory(working_dir):
        # os.setsid: change the process(forked) group id to itself
        training_proc = subprocess.Popen(command, env=envs, preexec_fn=os.setsid,
                                         stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

        log.info('proc-rank-%s-device-%s (pid: %d)', self.rank_id, self.device_id, training_proc.pid)

        # https://docs.python.org/3/library/subprocess.html#subprocess.Popen.wait
        subprocess.Popen(['tee', log_file_path], stdin=training_proc.stdout)

        return training_proc
```

10.5 制作自定义镜像用于推理

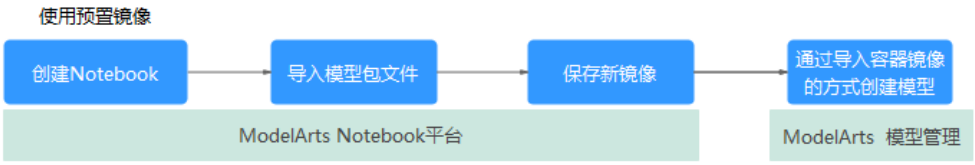
10.5.1 模型的自定义镜像制作流程

如果您使用了ModelArts不支持的AI引擎开发模型，也可通过制作自定义镜像，导入ModelArts创建为模型，并支持进行统一管理和部署为服务。

制作流程

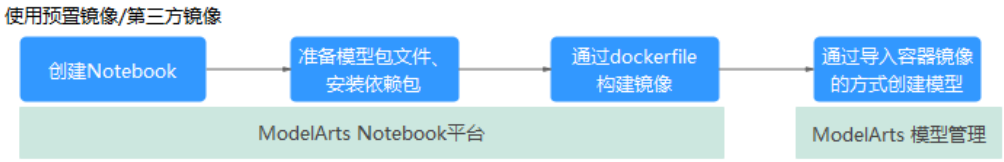
场景一： 预置镜像的环境软件满足要求，只需要导入模型包，就能用于创建模型，通过镜像保存功能制作。具体案例参考**在Notebook中通过镜像保存功能制作自定义镜像用于推理**。

图 10-24 模型的自定义镜像制作场景一



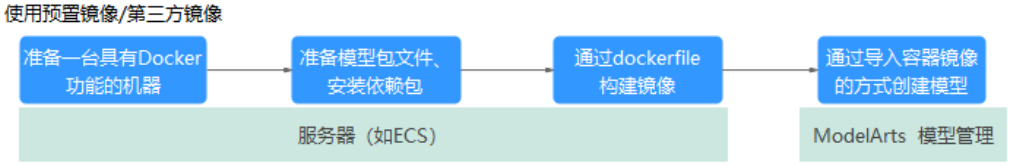
场景二： 预置镜像既不满足软件环境要求，同时需要放入模型包，在Notebook中通过Dockerfile制作。具体案例参考**在Notebook中通过Dockerfile从0制作自定义镜像用于推理**。

图 10-25 模型的自定义镜像制作场景二



场景三： 预置镜像既不满足软件环境要求，同时需要放入模型包，新的镜像超过35G，在服务器（如ECS）上制作。具体案例参考**在ECS中通过Dockerfile从0制作自定义镜像用于推理**。

图 10-26 模型的自定义镜像制作场景三



约束限制

- 自定义镜像中不能包含恶意代码。

- 创建模型的自定义镜像大小不超过50GB。
- 对于同步请求模式的模型，如果预测请求时延超过60s，会造成请求失败，甚至会有服务业务中断的风险，预测请求时延超过60s时，建议制作异步请求模式的模型。

自定义镜像的配置规范

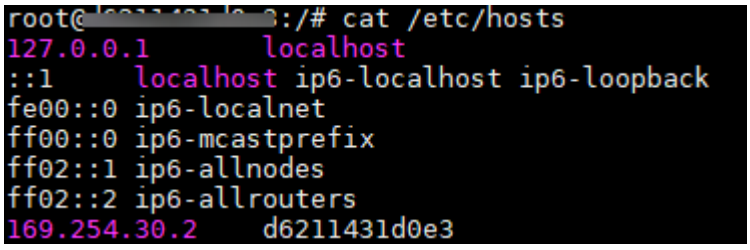
- **镜像对外接口**

  设置镜像的对外服务接口，推理接口需与config.json文件中apis定义的url一致，当镜像启动时可以直接访问。下面是mnist镜像的访问示例，该镜像内含mnist数据集训练的模型，可以识别手写数字。其中listen_ip为容器IP，您可以通过启动自定义镜像，在容器中获取容器IP。

  - 请求示例

    ```
    curl -X POST \ http://{listen_ip}:8080/ \ -F images=@seven.jpg
    ```

    图 10-27 listen_ip 获取示例

  - 返回示例

    ```
    {"mnist_result": 7}
    ```

- **（可选）健康检查接口**

  如果在滚动升级时要求不中断业务，那么必须在config.json文件中配置健康检查的接口，供ModelArts调用，在config.json文件中配置。当业务可提供正常服务时，健康检查接口返回健康状态，否则返回异常状态。

  须知

  - 如果要实现无损滚动升级，必须配置健康检查接口。
  - 自定义镜像如果需要在“在线服务”模块使用OBS外部存储挂载功能，需要新建一个OBS挂载专属目录如“/obs-mount/”，避免选择存量目录覆盖已有文件。OBS挂载仅开放对挂载目录文件新增、查看、修改功能，如果需要删除文件请到OBS并行文件系统中手动删除。

  健康检查接口示例如下。

  - URI

    ```
    GET /health
    ```

  - 请求示例**curl -X GET \ http://{listen_ip}:8080/health**
  - 响应示例

    ```
    {"health": "true"}
    ```

  - 状态码

表 10-37 状态码

状态码	编码	状态码说明
200	OK	请求成功

- **日志文件输出**

  为保证日志内容可以正常显示，日志信息需要打印到标准输出。

- **镜像启动入口**

  如果需要部署批量服务，镜像的启动入口文件需要为“/home/run.sh”，采用CMD设置默认启动路径，例如Dockerfile配置如下：

  CMD ["sh", "/home/run.sh"]

- **镜像依赖组件**

  如果需要部署批量服务，镜像内需要集成python、jre/jdk、zip等组件包。

- **（可选）保持Http长链接，无损滚动升级**

  如果需要支持滚动升级的过程中不中断业务，那么需要将服务的Http的“keep-alive”参数设置为200s。以gunicorn服务框架为例，gunicorn缺省情形下不支持keep-alive，需要同时安装gevent并配置启动参数“--keep-alive 200 -k gevent”。不同服务框架参数设置有区别，请以实际情况为准。

- **（可选）处理SIGTERM信号，容器优雅退出**

  如果需要支持滚动升级的过程中不中断业务，那么需要在容器中捕获SIGTERM信号，并且在收到SIGTERM信号之后等待60秒再优雅退出容器。提前优雅退出容器可能会导致在滚动升级的过程中业务概率中断。要保证容器优雅退出，从收到SIGTERM信号开始，业务需要将收到的请求全部处理完毕再结束，这个处理时长最多不超过90秒。例如run.sh如下所示：

```
#!/bin/bash
gunicorn_pid=""

handle_sigterm() {
  echo "Received SIGTERM, send SIGTERM to $gunicorn_pid"
  if [ $gunicorn_pid != "" ]; then
      sleep 60
      kill -15 $gunicorn_pid  # 传递 SIGTERM 给gunicorn进程
      wait $gunicorn_pid          # 等待gunicorn进程完全终止
  fi
}

trap handle_sigterm TERM
```

10.5.2 在 Notebook 中通过镜像保存功能制作自定义镜像用于推理

场景说明

本文详细介绍如何将本地已经制作好的模型包导入ModelArts的开发环境Notebook中进行调试和保存，然后将保存后的镜像部署到推理。本案例仅适用于华为云北京四和上海一站点。

操作流程如下：

1. **Step1 在Notebook中复制模型包**
2. **Step2 在Notebook中调试模型**

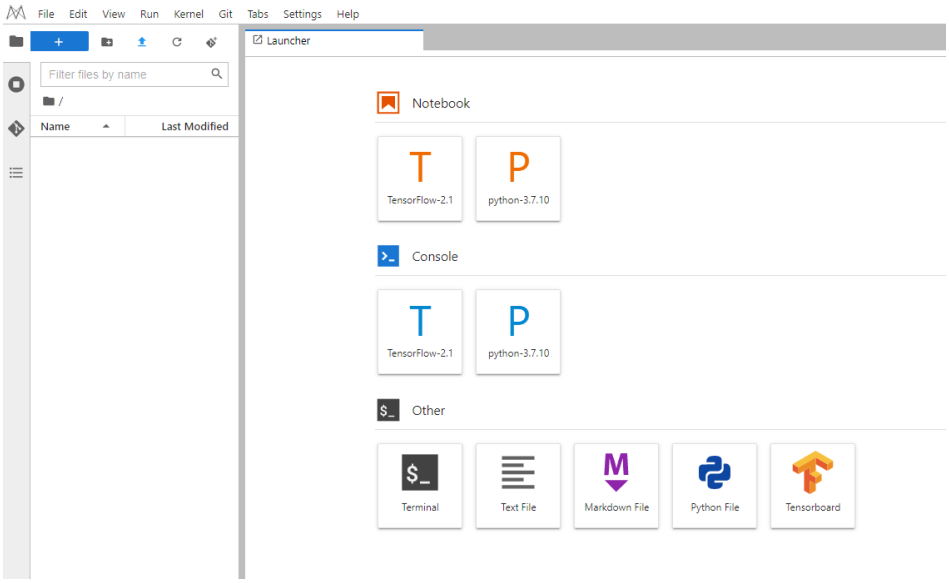
3. **Step3 Notebook中保存镜像**
4. **Step4 使用保存成功的镜像用于推理部署**

Step1 在 Notebook 中复制模型包

1. 登录ModelArts控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”管理页面。
2. 单击右上角“创建”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止。
   b. 填写Notebook详细参数，如选择镜像、资源规格等。
      - “镜像”：选择统一镜像**tensorflow_2.1-cuda_10.1-cudnn7-ubuntu_18.04**（详见**引擎版本一：tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**）或者**pytorch1.8-cuda10.2-cudnn7-ubuntu18.04**（详见**引擎版本一：pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64**）。
      - “资源池”：选择公共资源池或专属资源池，此处以公共资源池为例。
      - “类型”：推荐选择GPU。
      - “规格”：推荐选择GP Tnt004规格，如果没有再选择其他规格。
3. 参数填写完成后，单击“立即创建”进行规格确认。参数确认无误后，单击“提交”，完成Notebook的创建操作。

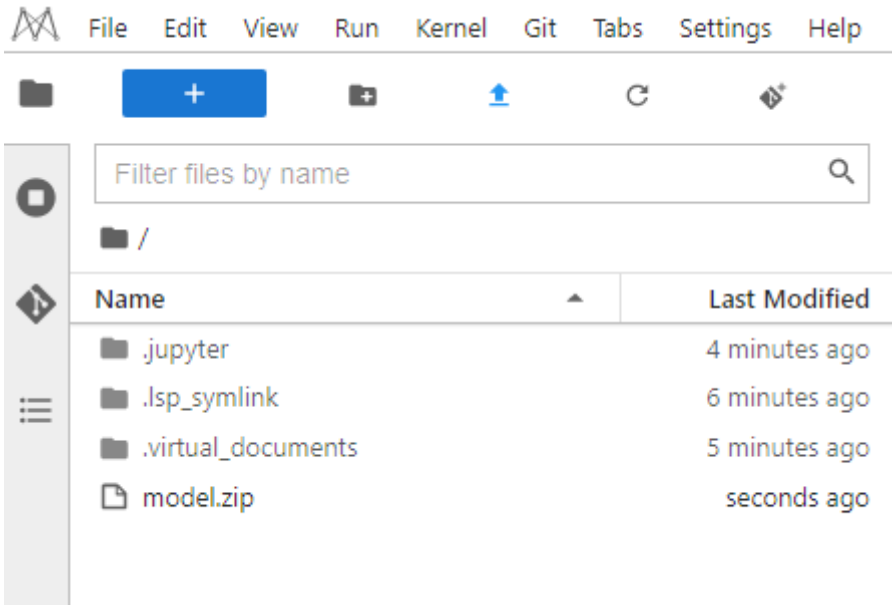
   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。
4. 当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。单击“操作列”的“打开”，进入JupyterLab的Launcher界面。

图 10-28 打开后进入 JupyterLab 的 Launcher 界面



5. 通过 功能，上传模型包文件到Notebook中，默认工作目录/home/ma-user/work/。模型包文件需要用户自己准备，样例内容参见**模型包文件样例**。

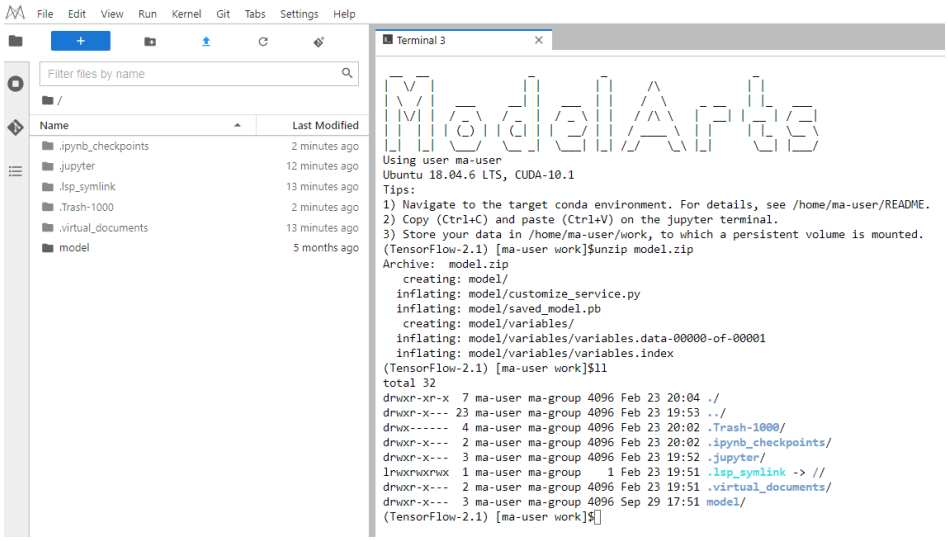
图 10-29 上传模型包



6. 打开Terminal终端，解压model.zip，解压后删除zip文件。

```
#解压命令
unzip model.zip
```

图 10-30 在 Terminal 终端中解压 model.zip



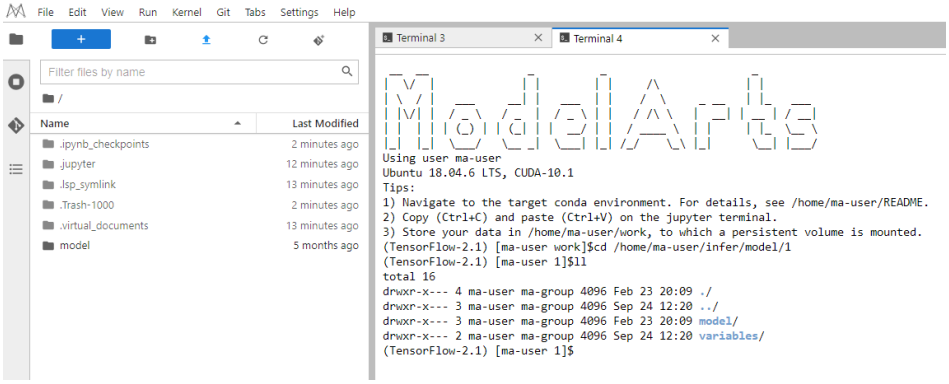
7. 在Terminal运行界面，执行复制命令。

```
cp -rf model/*  /home/ma-user/infer/model/1
```

然后执行如下命令查看镜像文件复制成功。

```
cd /home/ma-user/infer/model/1
ll
```

图 10-31 查看镜像文件复制成功

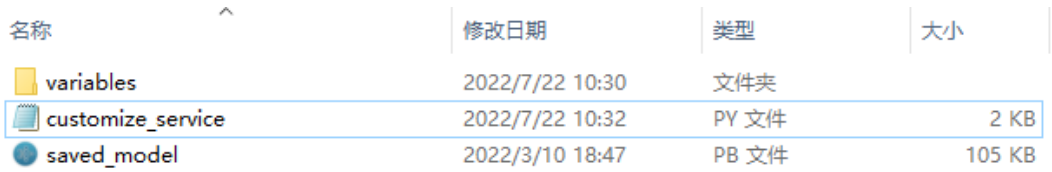


模型包文件样例

模型包文件model.zip中需要用户自己准备模型文件，此处仅是举例示意说明，以一个手写数字识别模型为例。

Model目录下必须要包含推理脚本文件customize_service.py，目的是为开发者提供模型预处理和后处理的逻辑。

图 10-32 推理模型 model 目录示意图（需要用户自己准备模型文件）



推理脚本customize_service.py的具体写法要求可以参考**模型推理代码编写说明**。

本案例中提供的customize_service.py文件具体内容如下：

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tfserving_model_service import TfServingBaseService


class mnist_service(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

        # 非阻塞方式加载saved_model模型，防止阻塞超时
        thread = threading.Thread(target=self.load_model)
        thread.start()

    def load_model(self):
        # load saved_model 格式的模型
        self.model = tf.saved_model.load(self.model_path)

        signature_defs = self.model.signatures.keys()
```

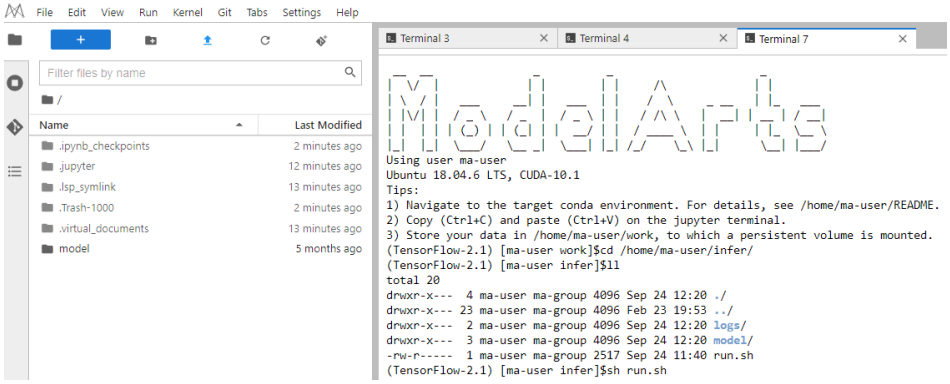
```
    signature = []
    # only one signature allowed
    for signature_def in signature_defs:
        signature.append(signature_def)

    if len(signature) == 1:
        model_signature = signature[0]
    else:
        logging.warning("signatures more than one, use serving_default signature from %s", signature)
        model_signature = tf.saved_model.DEFAULT_SERVING_SIGNATURE_DEF_KEY

    self.predict = self.model.signatures[model_signature]

def _preprocess(self, data):
    images = []
    for k, v in data.items():
        for file_name, file_content in v.items():
            image1 = Image.open(file_content)
            image1 = np.array(image1, dtype=np.float32)
            image1.resize((28, 28, 1))
            images.append(image1)

    images = tf.convert_to_tensor(images, dtype=tf.dtypes.float32)
    preprocessed_data = images

    return preprocessed_data

def _inference(self, data):

    return self.predict(data)

def _postprocess(self, data):

    return {
        "result": int(data["output"].numpy()[0].argmax())
    }
```

Step2 在 Notebook 中调试模型

1. 打开一个新的Terminal终端，进入“/home/ma-user/infer/”目录，运行启动脚本run.sh，并预测模型。基础镜像中默认提供了run.sh作为启动脚本。启动命令如下：

   ```
   sh  run.sh
   ```

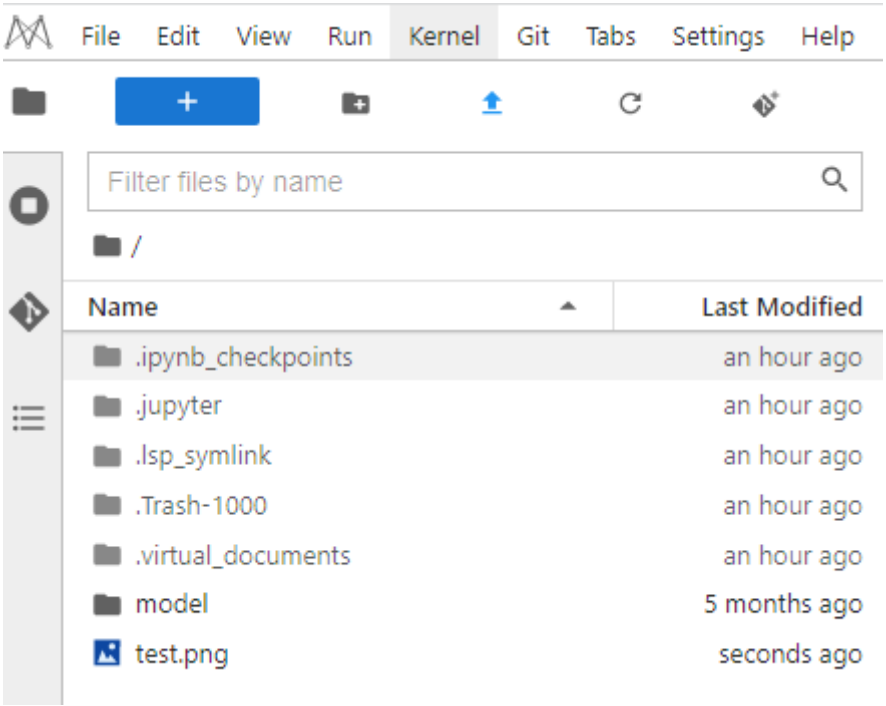
   图 10-33 运行启动脚本

2. 上传一张预测图片（手写数字图片）到Notebook中。

图 10-34 手写数字图片

图 10-35 上传预测图片



3. 重新打开一个新的Terminal终端，执行如下命令进行预测。

```
curl -kv -F 'images=@/home/ma-user/work/test.png' -X POST http://127.0.0.1:8080/
```

图 10-36 预测

在调试过程中，如果有修改模型文件或者推理脚本文件，需要重启run.sh脚本。执行如下命令先停止nginx服务，再运行run.sh脚本。

```
#查询nginx进程
ps -ef |grep nginx
#关闭所有nginx相关进程
kill -9 {进程ID}
#运行run.sh脚本
sh run.sh
```

也可以执行pkill nginx命令直接关闭所有nginx进程。

```
#关闭所有nginx进程
pkill nginx
#运行run.sh脚本
sh run.sh
```

图 10-37 重启 run.sh 脚本

```
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$ps -ef |grep nginx
ma-user   6474     1  0 10:57 ?        00:00:00 nginx: master process /usr/sbin/nginx
ma-user   6476  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user   6477  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user   7925  1752  0 10:59 pts/0    00:00:00 grep --color=auto nginx
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$kill -9 6474
(TensorFlow-2.1) [ma-user infer]$kill -9 6476
(TensorFlow-2.1) [ma-user infer]$kill -9 6477
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$sh run.sh
```

Step3 Notebook 中保存镜像

说明

Notebook实例状态必须为“运行中”才可以一键进行镜像保存。

1. 在Notebook列表中，对于要保存的Notebook实例，单击右侧“操作”列中的“更多 > 保存镜像”，进入“保存镜像”对话框。
2. 在保存镜像对话框中，设置组织、镜像名称、镜像版本和描述信息。单击“确认”保存镜像。

   在“组织”下拉框中选择一个组织。如果没有组织，可以单击右侧的“立即创建”，创建一个组织。

   同一个组织内的用户可以共享使用该组织内的所有镜像。
3. 镜像会以快照的形式保存，保存过程约5分钟，请耐心等待。此时不可再操作实例（对于打开的JupyterLab界面和本地IDE仍可操作）。

须知

快照中耗费的时间仍占用实例的总运行时长，如果在快照中时，实例因运行时间到期停止，将导致镜像保存失败。

4. 镜像保存成功后，实例状态变为“运行中”，用户可在“镜像管理”页面查看到该镜像详情。
5. 单击镜像的名称，进入镜像详情页，可以查看镜像版本/ID，状态，资源类型，镜像大小，SWR地址等。

Step4 使用保存成功的镜像用于推理部署

将**Step2 在Notebook中调试模型**的自定义镜像导入到模型中，并部署为在线服务。

1. 登录ModelArts控制台，在左侧导航栏中选择“模型管理”，单击“创建模型”，进入创建模型。
2. 设置模型的参数，如**图10-38**所示。
   - 元模型来源：从容器镜像中选择。
   - 容器镜像所在的路径：单击选择镜像文件。具体路径查看**5**SWR地址。
   - 容器调用接口：选择HTTPS。
   - host：设置为8443。
   - 部署类型：选择在线服务。

图 10-38 设置模型参数

3. 填写启动命令，启动命令内容如下：

```
sh /home/ma-user/infer/run.sh
```

4. 填写apis定义，单击“保存”生效。apis定义中指定输入为文件，具体内容参见下面代码样例。

图 10-39 填写 apis 定义

apis定义具体内容如下：

```
[{
    "url": "/",
    "method": "post",
    "request": {
        "Content-type": "multipart/form-data",
        "data": {
            "type": "object",
            "properties": {
                "images": {
                    "type": "file"
                }
            }
        }
    },
    "response": {
        "Content-type": "applicaton/json",
        "data": {
            "type": "object",
            "properties": {
                "result": {
                    "type": "integer"
                }
            }
        }
    }
}]
```

须知

apis定义提供模型对外Restfull api数据定义，用于定义模型的输入、输出格式。

- 创建模型填写apis。在创建的模型部署服务成功后，进行预测时，会自动识别预测类型。
- 创建模型时不填写apis。在创建的模型部署服务成功后，进行预测，需选择“请求类型”。“请求类型”可选择“application/json”或“multipart/form-data”。请根据元模型，选择合适的类型。
  - 选择“application/json”时，直接填写“预测代码”进行文本预测。
  - 选择“multipart/form-data”时，需填写“请求参数”，请求参数取值等同于**使用图形界面的软件进行预测（以Postman为例）**Body页签中填写的“KEY”的取值，也等同于**使用curl命令发送预测请求**上传数据的参数名。

5. 设置完成后，单击“立即创建”，等待模型状态变为“正常”。
6. 单击新建的模型名称左侧的小三角形，展开模型的版本列表。在操作列单击“部署 > 在线服务”，跳转至在线服务的部署页面。
7. 在部署页面，参考如下说明填写关键参数。

   “名称”：自定义一个在线服务的名称，也可以使用默认值。

   “资源池”：选择“公共资源池”。

   “模型来源”和“选择模型及版本”：会自动选择模型和版本号。

   “实例规格”：在下拉框中选择“限时免费”资源，勾选并阅读免费规格说明。

   其他参数可使用默认值。

   说明

   如果限时免费资源售罄，建议选择收费CPU资源进行部署。当选择收费CPU资源部署在线服务时会收取少量资源费用，具体费用以界面信息为准。
8. 参数配置完成后，单击“下一步”，确认规格参数后，单击“提交”启动在线服务的部署。
9. 进入“部署上线 > 在线服务”页面，等待服务状态变为“运行中”时，表示服务部署成功。单击操作列的“预测”，进入服务详情页的“预测”页面。上传图片，预测结果。

10.5.3 在 Notebook 中通过 Dockerfile 从 0 制作自定义镜像用于推理

场景说明

针对ModelArts目前不支持的AI引擎，您可以通过自定义镜像的方式将编写的模型导入ModelArts，创建为模型。

本文详细介绍如何在ModelArts的开发环境Notebook中使用基础镜像构建一个新的推理镜像，并完成模型的创建，部署为在线服务。本案例仅适用于华为云北京四和上海一站点。

操作流程如下：

1. **Step1 在Notebook中构建一个新镜像**：在ModelArts的开发环境Notebook中制作自定义镜像，镜像规范可参考**创建模型的自定义镜像规范**。
2. **Step2 构建成功的镜像注册到镜像管理模块**：将构建成功的自定义镜像注册到ModelArts的镜像管理模块中，方便下一步调试。
3. **Step3 在Notebook中变更镜像并调试**：在Notebook中调试镜像。
4. **Step4 使用调试成功的镜像用于推理部署**：将调试完成的镜像导入ModelArts的模型管理中，并部署上线。

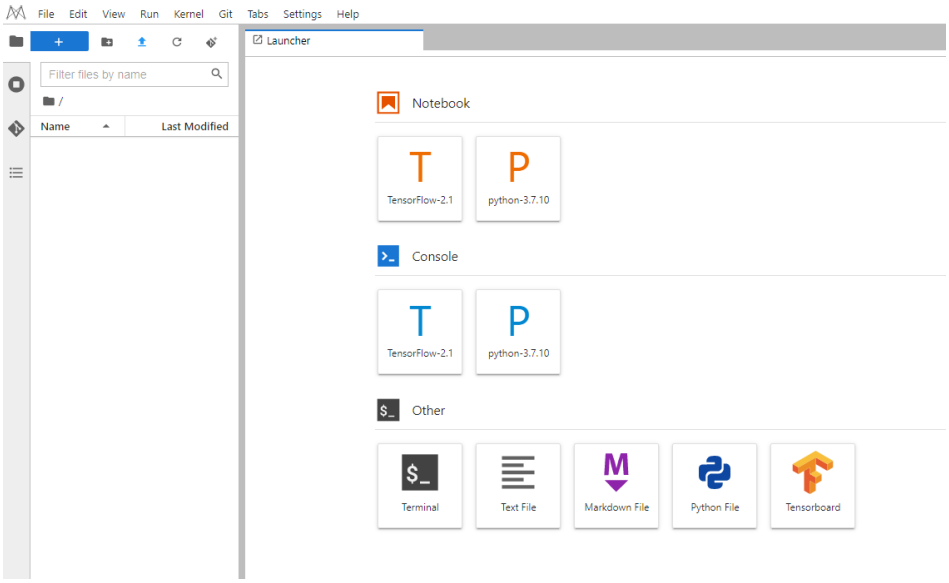
Step1 在 Notebook 中构建一个新镜像

本章节以ModelArts提供的基础镜像tensorflow为例介绍如何在ModelArts的Notebook中构建一个新镜像并用于模型部署。

1. 登录ModelArts控制台，在左侧导航栏中选择“全局配置”，检查是否配置了访问授权。如果未配置，请先配置访问授权。参考**使用委托授权**完成操作。
2. 登录ModelArts控制台，在左侧导航栏中选择“开发环境 > Notebook”，进入“Notebook”管理页面。
3. 单击右上角“创建”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止。
   b. 填写Notebook详细参数，如选择镜像、资源规格等。
      - “镜像”：选择公共镜像下任意一个支持CPU类型的镜像，例如：**tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04**
      - “资源池”：选择公共资源池或专属资源池，此处以公共资源池为例。
      - “类型”：推荐选择GPU。
      - “规格”：推荐选择GP Tnt004规格，如果没有再选择其他规格。
4. 参数填写完成后，单击“立即创建”进行规格确认。参数确认无误后，单击“提交”，完成Notebook的创建操作。

   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。
5. 打开运行中的Notebook实例。

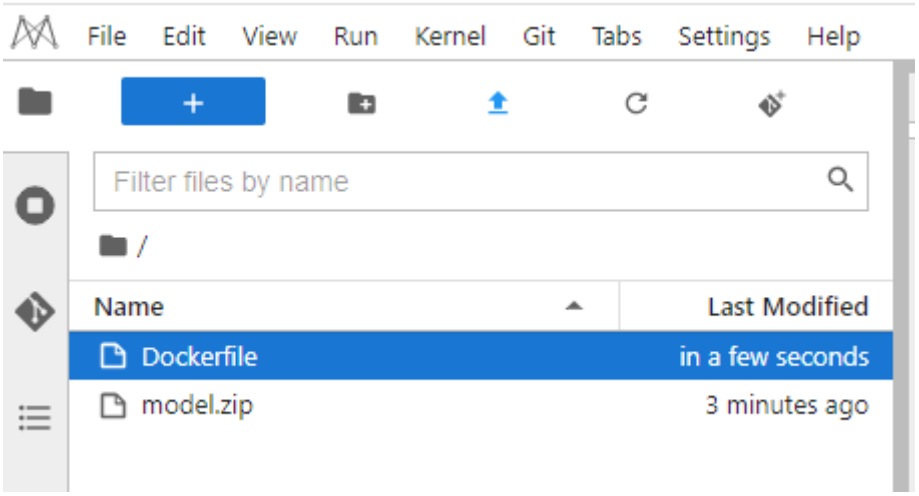
图 10-40 打开 Notebook 实例



6. 通过 功能，上传dockerfile文件和模型包文件到Notebook中，默认工作目录/home/ma-user/work/。

   dockerfile文件的具体内容可以参见**Dockerfile模板**。模型包文件需要用户自己准备，样例内容参见**模型包文件样例**。

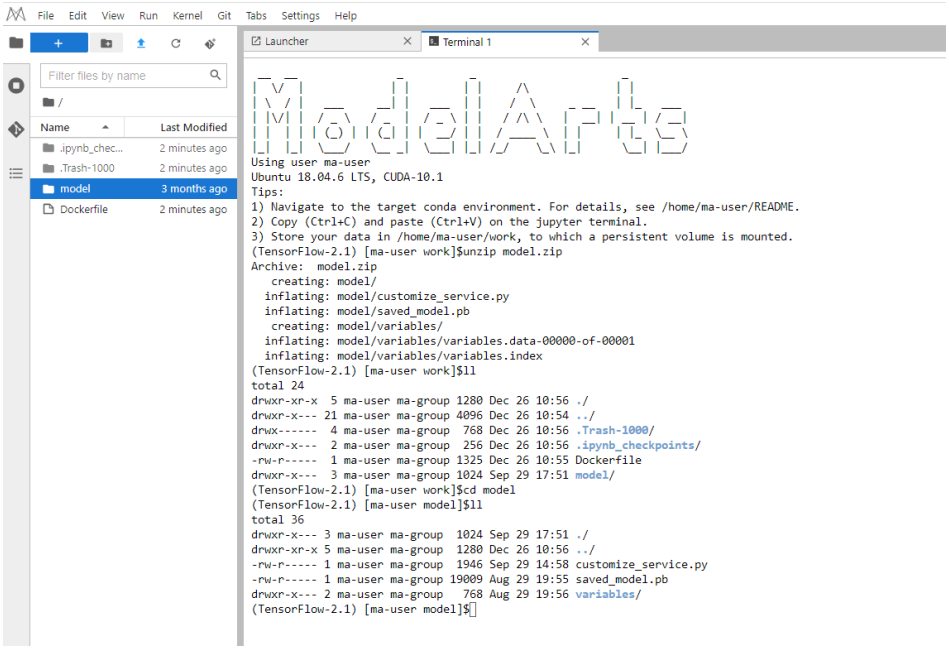
图 10-41 上传 dockerfile 文件和模型包文件



7. 打开Terminal终端，解压model.zip，解压后删除zip文件。

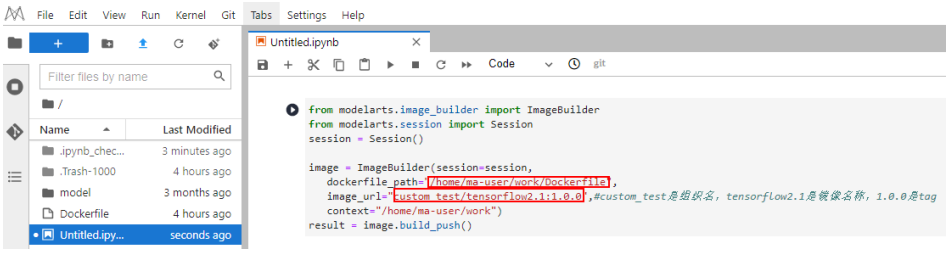
```
#解压命令
unzip model.zip
```

图 10-42 在 Terminal 终端中解压 model.zip



8. 打开一个新的.ipynb文件，启动构建脚本，在构建脚本中指定dockerfile文件和镜像的推送地址。构建脚本当前仅支持华为云北京四和上海一站点。

图 10-43 启动构建脚本

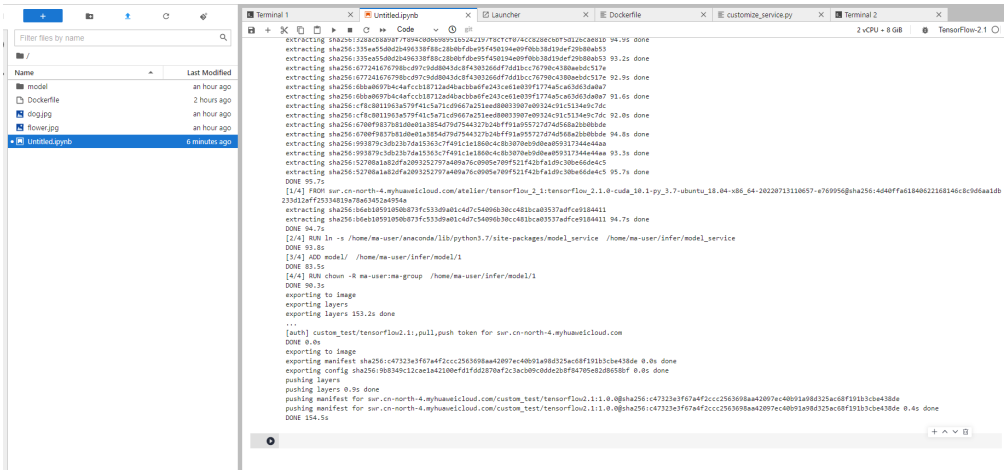


构建脚本内容如下：

```
from modelarts.image_builder import ImageBuilder
from modelarts.session import Session
session = Session()

image = ImageBuilder(session=session,
  dockerfile_path="/home/ma-user/work/Dockerfile",
  image_url="custom_test/tensorflow2.1:1.0.0",#custom_test是组织名，tensorflow2.1是镜像名称，1.0.0是tag
  context="/home/ma-user/work")
result = image.build_push()
```

等待镜像构建完成。镜像构建完成后会自动推送到SWR中。

图 10-44 等待镜像构建完成



Dockerfile 模板

Dockerfile样例，此样例可以直接另存为一个Dockerfile文件使用。此处可以使用的基础镜像列表请参见**推理专属预置镜像列表**。

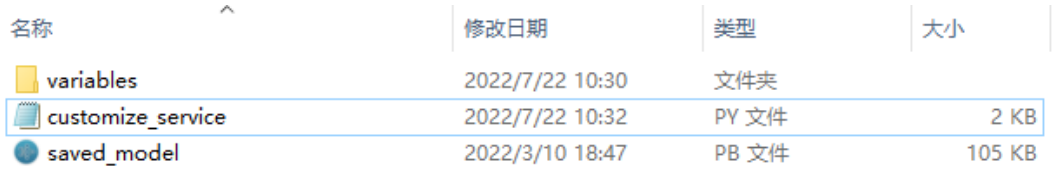
```
FROM swr.cn-north-4.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817

# here create a soft link from '/home/ma-user/anaconda/lib/python3.7/site-packages/model_service' to '/home/ma-user/infer/model_service'. It’s the build-in inference framework code dir
# if the installed python version of this base image is python3.8, you should create a soft link from '/home/ma-user/anaconda/lib/python3.8/site-packages/model_service' to '/home/ma-user/infer/model_service'.
USER root
RUN ln -s /home/ma-user/anaconda/lib/python3.7/site-packages/model_service  /home/ma-user/infer/model_service
USER ma-user

# here we supply a demo, you can change it to your own model files
ADD model/  /home/ma-user/infer/model/1
USER root
RUN chown -R ma-user:ma-group  /home/ma-user/infer/model/1
USER ma-user

# default MODELARTS_SSL_CLIENT_VERIFY switch is "true". In order to debug, we set it to be "false"
ENV MODELARTS_SSL_CLIENT_VERIFY="false"

# change your port and protocol here, default is 8443 and https
# ENV MODELARTS_SERVICE_PORT=8080
# ENV MODELARTS_SSL_ENABLED="false"

# add pip install here
# RUN pip install numpy==1.16.4
# RUN pip install -r requirements.txt

# default cmd, you can chage it here
# CMD sh /home/ma-user/infer/run.sh
```

模型包文件样例

模型包文件model.zip中需要用户自己准备模型文件，此处仅是举例示意说明，以一个手写数字识别模型为例。

Model目录下必须要包含推理脚本文件customize_service.py，目的是为开发者提供模型预处理和后处理的逻辑。

图 10-45 推理模型 model 目录示意图（需要用户自己准备模型文件）



推理脚本customize_service.py的具体写法要求可以参考**模型推理代码编写说明**。

本案例中提供的customize_service.py文件具体内容如下：

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tfserving_model_service import TfServingBaseService


class mnist_service(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

        # 非阻塞方式加载saved_model模型，防止阻塞超时
        thread = threading.Thread(target=self.load_model)
        thread.start()

    def load_model(self):
        # load saved_model 格式的模型
        self.model = tf.saved_model.load(self.model_path)

        signature_defs = self.model.signatures.keys()

        signature = []
        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)

        if len(signature) == 1:
            model_signature = signature[0]
        else:
            logging.warning("signatures more than one, use serving_default signature from %s", signature)
            model_signature = tf.saved_model.DEFAULT_SERVING_SIGNATURE_DEF_KEY

        self.predict = self.model.signatures[model_signature]

    def _preprocess(self, data):
        images = []
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((28, 28, 1))
                images.append(image1)

        images = tf.convert_to_tensor(images, dtype=tf.dtypes.float32)
        preprocessed_data = images

        return preprocessed_data
```

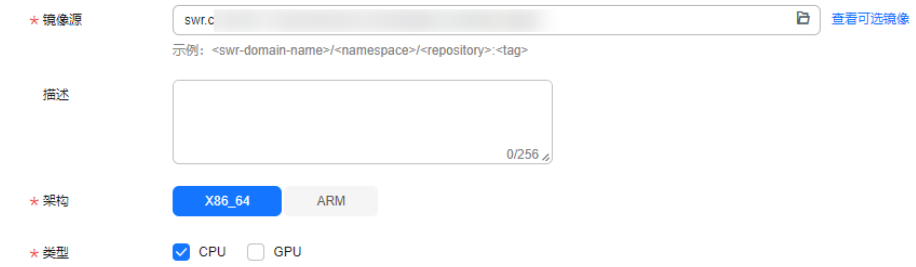
```
def _inference(self, data):

    return self.predict(data)

def _postprocess(self, data):

    return {
        "result": int(data["output"].numpy()[0].argmax())
    }
```

Step2 构建成功的镜像注册到镜像管理模块

将**Step1 在Notebook中构建一个新镜像**中构建成功的自定义镜像注册到镜像管理中，方便后续使用。

1. 登录ModelArts控制台，在左侧导航栏中选择“镜像管理”，单击“注册镜像”，进入注册镜像页面。
2. 输入镜像源地址，选择架构和类型后，单击“立即注册”。
   - “镜像源”：地址为swr.cn-north-4-myhuaweicloud.com/custom_test/tensorflow2.1:1.0.0。其中custom_test/tensorflow2.1:1.0.0为**8**镜像构建脚本中设置的镜像地址。
   - “架构”：选择X86_64
   - “类型”：选择CPU

   图 10-46 注册镜像

3. 注册完成后，可以在镜像管理页面查看到注册成功的镜像。

Step3 在 Notebook 中变更镜像并调试

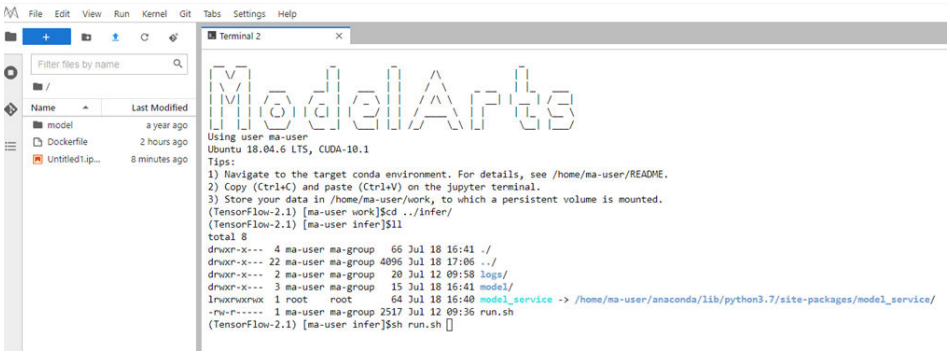
使用制作完成的自定义镜像进行推理服务调试，调试成功后再导入到ModelArts的模型中并部署为在线服务。

1. 登录ModelArts控制台，在左侧导航栏中选择“开发环境 > Notebook”，进入“Notebook”管理页面。停止**Step1 在Notebook中构建一个新镜像**中创建的Notebook。
2. 在Notebook对应操作列，单击“更多 > 变更镜像”，打开“变更镜像”弹出框，变更镜像选择“自定义镜像”，将当前镜像变更为**Step2 构建成功的镜像注册到镜像管理模块**注册的镜像，如**图10-47**所示。

图 10-47 变更镜像

3. 启动变更后的Notebook，并打开。进入Terminal运行界面，在工作目录，运行启动脚本run.sh，并预测模型。基础镜像中默认提供了run.sh作为启动脚本。

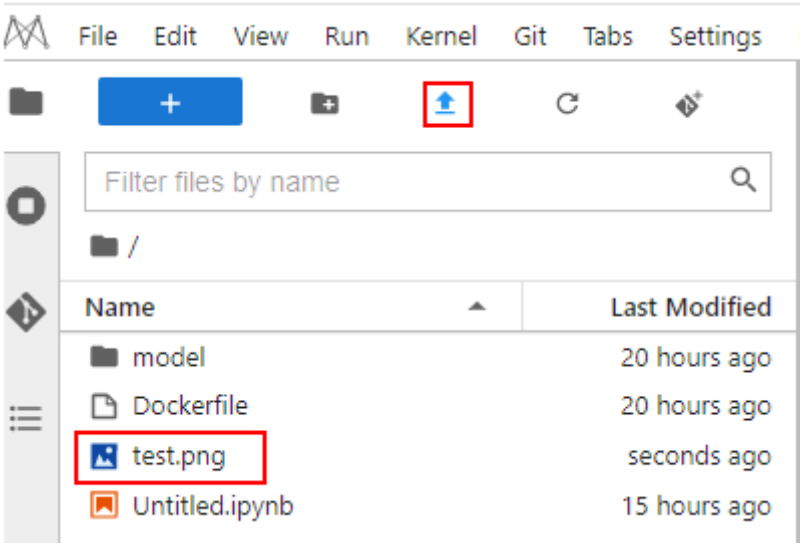
图 10-48 运行启动脚本



4. 上传一张预测图片（手写数字图片）到Notebook中。

图 10-49 手写数字图片

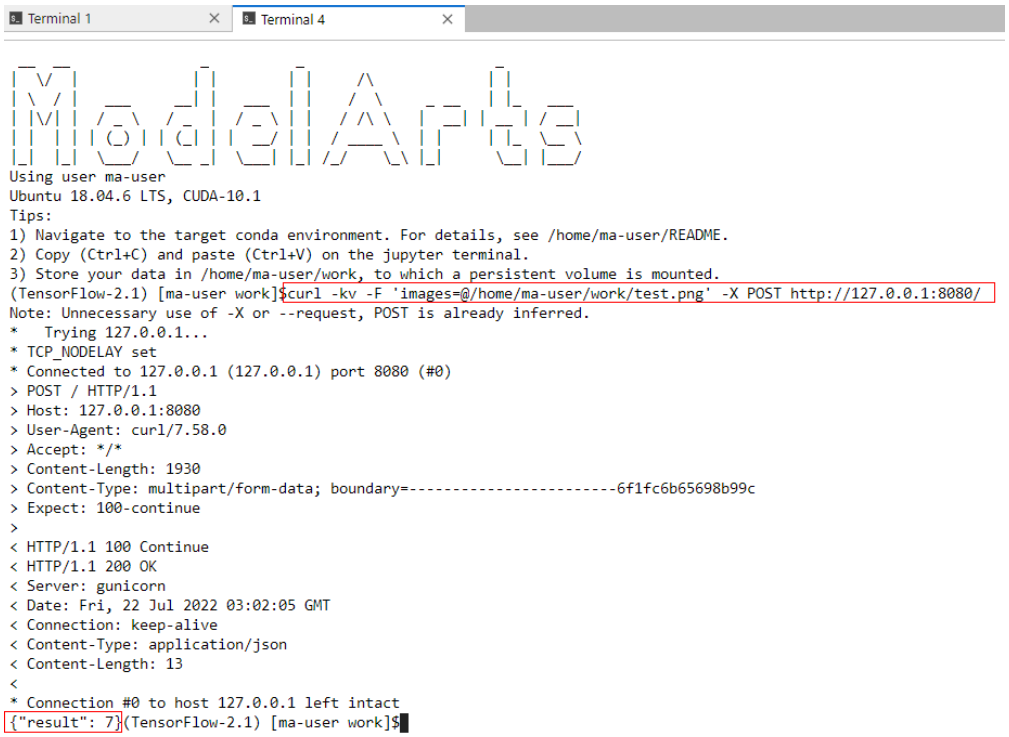
图 10-50 上传预测图片



5. 重新打开一个新的Terminal终端，执行如下命令进行预测。

```
curl -kv -F 'images=@/home/ma-user/work/test.png' -X POST http://127.0.0.1:8080/
```

图 10-51 预测



在调试过程中，如果有修改模型文件或者推理脚本文件，需要重启run.sh脚本。执行如下命令先停止nginx服务，再运行run.sh脚本。

```
#查询nginx进程
ps -ef |grep nginx
#关闭所有nginx相关进程
kill -9 {进程ID}
#运行run.sh脚本
sh run.sh
```

也可以执行pkill nginx命令直接关闭所有nginx进程。

```
#关闭所有nginx进程
pkill nginx
#运行run.sh脚本
sh run.sh
```

图 10-52 重启 run.sh 脚本

```
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$ps -ef |grep nginx
ma-user    6474     1  0 10:57 ?        00:00:00 nginx: master process /usr/sbin/nginx
ma-user    6476  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user    6477  6474  0 10:57 ?        00:00:00 nginx: worker process
ma-user    7925  1752  0 10:59 pts/0    00:00:00 grep --color=auto nginx
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$kill -9 6474
(TensorFlow-2.1) [ma-user infer]$kill -9 6476
(TensorFlow-2.1) [ma-user infer]$kill -9 6477
(TensorFlow-2.1) [ma-user infer]$
(TensorFlow-2.1) [ma-user infer]$sh run.sh
```

Step4 使用调试成功的镜像用于推理部署

将**Step3 在Notebook中变更镜像并调试**中调试成功的自定义镜像导入到模型中，并部署为在线服务。

1. 登录ModelArts控制台，在左侧导航栏中选择“模型管理”，单击“创建”，进入模型管理。
2. 设置模型的参数，如**图10-53**所示。
   - 元模型来源：从容器镜像中选择。
   - 容器镜像所在的路径：单击选择前面创建的镜像。
   - 容器调用接口：选择HTTPS。
   - host：设置为8443。
   - 部署类型：选择在线部署。

图 10-53 设置模型参数

3. 填写apis定义，单击“保存”生效。apis定义中指定输入为文件，具体内容参见下面代码样例。

图 10-54 填写 apis 定义

apis定义具体内容如下：

```
[{
    "url": "/",
    "method": "post",
    "request": {
        "Content-type": "multipart/form-data",
        "data": {
            "type": "object",
            "properties": {
                "images": {
                    "type": "file"
                }
            }
        }
    },
    "response": {
        "Content-type": "applicaton/json",
        "data": {
            "type": "object",
            "properties": {
                "result": {
                    "type": "integer"
                }
            }
        }
    }
}]
```

须知

apis定义提供模型对外Restfull api数据定义，用于定义模型的输入、输出格式。

- 创建模型填写apis。在创建的模型部署服务成功后，进行预测时，会自动识别预测类型。
- 创建模型时不填写apis。在创建的模型部署服务成功后，进行预测，需选择“请求类型”。“请求类型”可选择“application/json”或“multipart/form-data”。请根据元模型，选择合适的类型。
  - 选择“application/json”时，直接填写“预测代码”进行文本预测。
  - 选择“multipart/form-data”时，需填写“请求参数”，请求参数取值等同于**使用图形界面的软件进行预测（以Postman为例）**Body页签中填写的“KEY”的取值，也等同于**使用curl命令发送预测请求**上传数据的参数名。

4. 设置完成后，单击“立即创建”，等待模型状态变为“正常”。
5. 单击新建的模型名称左侧的小三角形，展开模型的版本列表。在操作列单击“部署 > 在线服务”，跳转至在线服务的部署页面。
6. 在部署页面，参考如下说明填写关键参数。

   “名称”：按照界面提示规则自定义一个在线服务的名称，也可以使用默认值。

   “资源池”：选择“公共资源池”。

   “模型来源”和“选择模型及版本”：会自动选择模型和版本号。

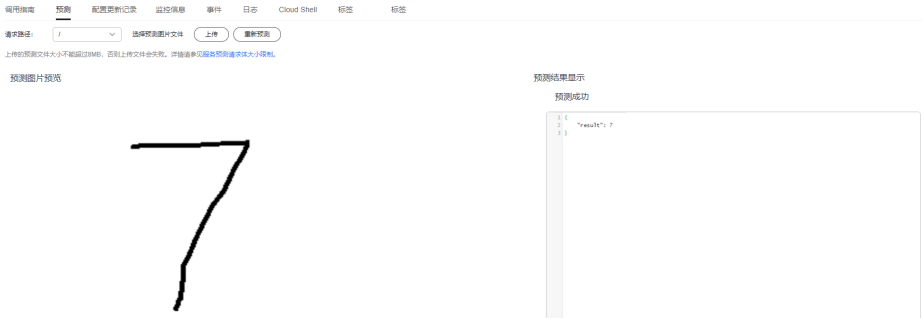
   “计算节点规格”：在下拉框中选择“限时免费”资源，勾选并阅读免费规格说明。

   其他参数可使用默认值。

   说明

   如果限时免费资源售罄，建议选择收费CPU资源进行部署。当选择收费CPU资源部署在线服务时会收取少量资源费用，具体费用以界面信息为准。
7. 参数配置完成后，单击“下一步”，确认规格参数后，单击“提交”启动在线服务的部署。
8. 进入“部署上线 > 在线服务”页面，等待服务状态变为“运行中”时，表示服务部署成功。单击操作列的“预测”，进入服务详情页的“预测”页面。上传图片，预测结果。

图 10-55 预测



10.5.4 在 ECS 中通过 Dockerfile 从 0 制作自定义镜像用于推理

针对ModelArts目前不支持的AI引擎，您可以针对该引擎构建自定义镜像，并将镜像导入ModelArts，创建为模型。本文详细介绍如何使用自定义镜像完成模型的创建，并部署成在线服务。

操作流程如下：

1. **本地构建镜像**：在本地制作自定义镜像包，镜像包规范可参考**创建模型的自定义镜像规范**。
2. **本地验证镜像并上传镜像至SWR服务**：验证自定义镜像的API接口功能，无误后将自定义镜像上传至SWR服务。
3. **将自定义镜像创建为模型**：将上传至SWR服务的镜像导入ModelArts的模型管理。
4. **将模型部署为在线服务**：将导入的模型部署上线。

本地构建镜像

以linux x86_x64架构的主机为例，您可以购买相同规格的ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录弹性云服务器**。镜像选择公共镜像，推荐使用ubuntu18.04的镜像。

图 10-56 创建 ECS 服务器-选择 X86 架构的公共镜像

1. 登录主机后，安装Docker，可参考**Docker官方文档**。也可执行以下命令安装docker。
   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```
2. 获取基础镜像。本示例以Ubuntu18.04为例。
   ```
   docker pull ubuntu:18.04
   ```
3. 新建文件夹“self-define-images”，在该文件夹下编写自定义镜像的“Dockerfile”文件和应用服务代码“test_app.py”。本样例代码中，应用服务代码采用了flask框架。

   文件结构如下所示
   ```
   self-define-images/
     --Dockerfile
     --test_app.py
   ```
   - “Dockerfile”
     ```
     From ubuntu:18.04
     # 配置华为云的源，安装 python、python3-pip 和 Flask
     RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
       sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
     ```

```
&& \
  sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
&& \
  apt-get update && \
  apt-get install -y python3 python3-pip && \
  pip3 install  --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/
repository/pypi/simple  Flask

# 复制应用服务代码进镜像里面
COPY test_app.py /opt/test_app.py

# 指定镜像的启动命令
CMD python3  /opt/test_app.py
```

- “test_app.py”

```
from flask import Flask, request
import json
app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----------- in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----------- in goodbye func ----------")
    return '\nGoodbye!\n'


@app.route('/', methods=['POST'])
def default_func():
    print("----------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```

4. 进入“self-define-images”文件夹，执行以下命令构建自定义镜像“test:v1”。

```
docker build -t test:v1 .
```

5. 您可以使用“docker images”查看您构建的自定义镜像。

本地验证镜像并上传镜像至 SWR 服务

1. 在本地环境执行以下命令启动自定义镜像

```
docker run -it -p 8080:8080 test:v1
```

图 10-57 启动自定义镜像

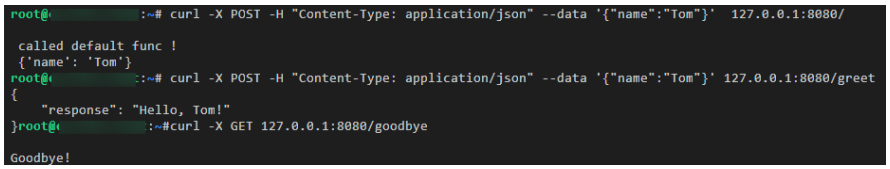
```
                :/opt/file# docker run -it -p 8080:8080 test:v1
 * Serving Flask app "test_app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

2. 另开一个终端，执行以下命令验证自定义镜像的三个API接口功能。

```
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}'  127.0.0.1:8080/
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
curl -X GET 127.0.0.1:8080/goodbye
```

如果验证自定义镜像功能成功，结果如下图所示。

图 10-58 校验接口



3. 上传自定义镜像至SWR服务。
4. 完成自定义镜像上传后，您可以在“容器镜像服务>我的镜像>自有镜像”列表中看到已上传镜像。

将自定义镜像创建为模型

参考**从容器镜像中选择元模型**导入元模型，您需要特别关注以下参数：

- 元模型来源：选择“从容器镜像中选择”
  - 容器镜像所在的路径：选择已制作好的自有镜像

    图 10-59 选择已制作好的自有镜像

  - 容器调用接口：指定模型启动的协议和端口号。请确保协议和端口号与自定义镜像中提供的协议和端口号保持一致。
  - 镜像复制：选填，选择是否将容器镜像中的模型镜像复制到ModelArts中。
  - 健康检查：选填，用于指定模型的健康检查。仅当自定义镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致模型创建失败。
- apis定义：选填，用于编辑自定义镜像的apis定义。模型apis定义需要遵循ModelArts的填写规范，参见**模型配置文件说明**。

  本样例的配置文件如下所示：

```
[{
      "url": "/",
      "method": "post",
      "request": {
          "Content-type": "application/json"
      },
      "response": {
          "Content-type": "application/json"
      }
   },
{
      "url": "/greet",
      "method": "post",
      "request": {
          "Content-type": "application/json"
      },
      "response": {
          "Content-type": "application/json"
      }
   },
```

```
{
      "url": "/goodbye",
      "method": "get",
      "request": {
          "Content-type": "application/json"
      },
      "response": {
          "Content-type": "application/json"
      }
   }
]
```

将模型部署为在线服务

1. 参考**部署为在线服务**将模型部署为在线服务。
2. 在线服务创建成功后，您可以在服务详情页查看服务详情。
3. 您可以通过“预测”页签访问在线服务。

11 ModelArts Standard 资源监控

11.1 ModelArts Standard 资源监控概述

为了满足用户对资源使用的监控诉求，ModelArts Standard提供了多种监控查看方式。

- 方式一：通过ModelArts Standard控制台查看

  您在可通过ModelArts控制台的总览页或各模块资源监控页签查看监控指标。具体涉及以下几个方面：

  - 通过ModelArts控制台的总览页查看，具体请参见**通过ModelArts控制台查看监控指标**。
  - Standard训练作业：用户在运行训练作业时，可以查看训练作业占用的CPU、GPU或NPU资源使用情况。具体请参见**训练资源监控**章节。
  - Standard在线服务：用户将模型部署为在线服务后，可以通过监控功能查看该推理服务的CPU、内存或GPU等资源使用统计信息和模型调用次数统计，具体参见**查看推理服务详情**章节。

- 方式二：通过AOM查看所有监控指标

  ModelArts Standard上报的所有监控指标都保存在AOM中，当ModelArts控制台可以查看的指标不满足诉求时，用户可以通过AOM服务提供的指标消费和使用的能力来查看指标。设置指标阈值告警、告警上报等，都可以直接在AOM控制台操作。具体参见**通过AOM控制台查看ModelArts所有监控指标**。

- 方式三：通过Grafana查看所有监控指标

  当AOM的监控模板不能满足用户诉求时，用户可以使用Grafana可视化工具来查看与分析监控指标。Grafana支持灵活而又复杂多样的监控视图和模板，为用户提供基于网页仪表面板的可视化监控效果，使用户更加直观地查看到实时资源使用情况。

将Grafana的数据源配置完成后，就可以通过Grafana查看AOM保存的所有ModelArts Standard的所有指标。具体参见**使用Grafana查看AOM中的监控指标**。

通过Grafana插件查看AOM中的监控指标的操作流程如下：

a. 安装配置Grafana

安装配置Grafana有**在Windows上安装配置Grafana**、**在Linux上安装配置Grafana**和**在Notebook上安装配置Grafana**三种方式，请您根据实际情况选择。

b. **配置Grafana数据源**

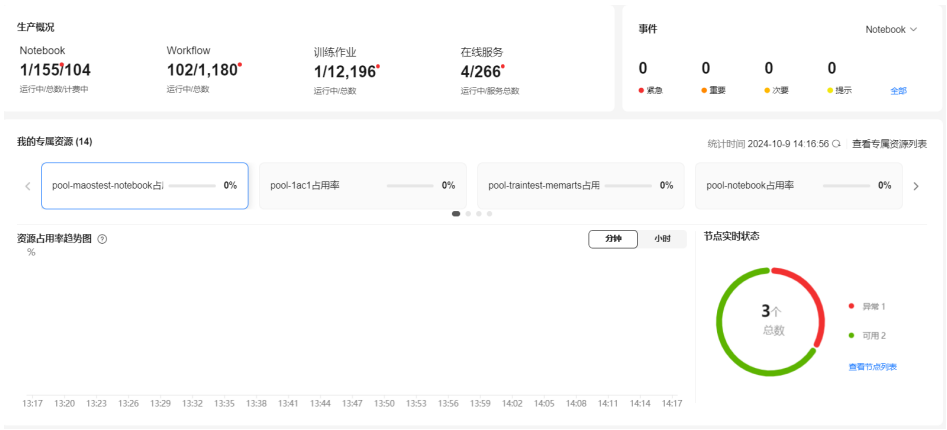
c. **配置仪表盘查看指标数据**

11.2 在 ModelArts 控制台查看监控指标

在总览页查看 ModelArts 监控指标

在ModelArts控制台的总览页，支持查看生产概况（即总体作业运行数量）、资源占用情况、训练作业资源利用情况。您可以单击生产概况的链接、资源池名称、训练作业，跳转到对应界面查看更多详情。

图 11-1 总览页查看监控信息



说明

在总览页查看全部事件时，如果顶部事件总数和底部的“总条数”数量不一致，请刷新重试。

在各模块资源监控页签查看 ModelArts 监控指标

- 训练作业：用户在运行训练作业时，可以查看多个计算节点的CPU、GPU、NPU资源使用情况。具体请参见**训练资源监控**章节。
- 在线服务：用户将模型部署为在线服务后，可以通过监控功能查看CPU、内存、GPU等资源使用统计信息和模型调用次数统计，具体参见**查看服务详情**章节。

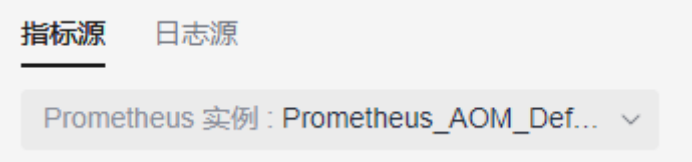
11.3 在 AOM 控制台查看 ModelArts 所有监控指标

ModelArts会定期收集资源池中各节点的关键资源（GPU、NPU、CPU、Memory等）的使用情况以及开发环境、训练作业、推理服务的关键资源的使用情况，并上报到AOM，用户可直接在AOM上查看。

登录 AOM 控制台查看监控指标

1. 登录控制台，搜索AOM，进入“应用运维管理”控制台。
2. 在左侧导航栏中选择“指标浏览”。
3. 从指标源下拉列表选择“Prometheus_AOM_Default”实例。

图 11-2 选择指标源



4. 通过“全量指标”或“按普罗语句添加”方式选择一个或多个关注的指标。

图 11-3 添加指标

关于更多指标浏览方法请参考华为云帮助中心“**应用运维管理 AOM/ 用户指南（2.0）/ 指标浏览**”。

容器级别的指标介绍

表 11-1 容器级别的指标

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
CPU	CPU使用率	ma_container_cpu_util	该指标用于统计测量对象的CPU使用率。	百分比（Percent）	0~100%	连续2个周期原始值>95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	CPU内核占用量	ma_container_cpu_used_core	该指标用于统计测量对象已经使用的CPU核个数。	核（Core）	≥0	NA	NA	NA
	CPU内核总量	ma_container_cpu_limit_core	该指标用于统计测量对象申请的CPU核总量。	核（Core）	≥1	NA	NA	NA
	CPU显存使用率	ma_container_gpu_mem_util	该指标用于统计测量对象已使用的显存占显存容量的百分比。	百分比（Percent）	0~100%	连续2个周期原始值>95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
内存	内存总量	ma_container_memory_capacity_megabytes	该指标用于统计测量对象申请的物理内存总量。	兆字节（Megabytes）	≥0	NA	NA	NA
	物理内存使用率	ma_container_memory_util	该指标用于统计测量对象已使用内存占申请物理内存总量的百分比。	百分比（Percent）	0～100%	连续2个周期原始值>95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	物理内存使用量	ma_container_memory_used_megabytes	该指标用于统计测量对象实际已经使用的物理内存，对应container_memory_working_set_bytes当前内存工作集（working set）使用量。 工作区内存使用量=活跃的匿名页和缓存，以及file-baked页<=container_memory_usage_bytes。	兆字节（Megabytes）	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
存储	磁盘读取速率	ma_container_disk_read_kilobytes	该指标用于统计每秒从磁盘读出的数据量。	千字节/秒（Kilobytes/Second）	≥0	NA	NA	NA
	磁盘写入速率	ma_container_disk_write_kilobytes	该指标用于统计每秒写入磁盘的数据量。	千字节/秒（Kilobytes/Second）	≥0	NA	NA	NA
GPU显存	GPU显存容量	ma_container_gpu_mem_total_megabytes	该指标用于统计训练作业的显存容量。	兆字节（Megabytes）	>0	NA	NA	NA
	GPU显存使用率	ma_container_gpu_mem_util	该指标用于统计测量对象已使用的显存占显存容量的百分比。	百分比（Percent）	0~100%	NA	NA	NA
	GPU显存使用量	ma_container_gpu_mem_used_megabytes	该指标用于统计测量对象已使用的显存。	兆字节（Megabytes）	≥0	NA	NA	NA
	GPU显存空闲容量	ma_container_gpu_mem_free_megabytes	该指标用于统计测量空闲的显存。	兆字节（Megabytes）	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
GPU	GPU使用率	ma_container_gpu_util	该指标用于统计测量对象的GPU使用率。	百分比（Percent）	0~100%	连续2个周期原始值>95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	GPU内存带宽利用率	ma_container_gpu_mem_copy_util	表示内存带宽利用率。以GPVnt1为例，其最大内存带宽为900 GB/sec，如果当前的内存带宽为450 GB/sec，则内存带宽利用率为50%。	百分比（Percent）	0~100%	NA	NA	NA
	GPU编码器利用率	ma_container_gpu_enc_util	表示编码器利用率。	百分比（Percent）	%	NA	NA	NA
	GPU解码器利用率	ma_container_gpu_dec_util	表示解码器利用率。	百分比（Percent）	%	NA	NA	NA
	GPU温度	DCGM_FI_DEV_GPU_TEMP	表示GPU温度。	摄氏度（℃）	自然数	NA	NA	NA
	GPU功率	DCGM_FI_DEV_POWER_USAGE	表示GPU功率。	瓦特（W）	>0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	GPU显存温度	DCGM_FI_DEV_MEMORY_TEMP	表示显存温度。	摄氏度（℃）	自然数	NA	NA	NA
网络IO	下行速率	ma_container_network_receive_bytes	该指标用于统计测试对象的入方向网络流速。	字节/秒（Bytes/Second）	≥0	NA	NA	NA
	接收包速率	ma_container_network_receive_packets	每秒网卡接收的数据包个数。	个/秒（Packets/Second）	≥0	NA	NA	NA
	下行错包率	ma_container_network_receive_error_packets	每秒网卡接收的错误包个数。	个/秒（Packets/Second）	≥0	连续2个周期原始值 > 1	紧急告警	网络丢包，建议提工单联系运维支持，排查网络问题。
	上行速率	ma_container_network_transmit_bytes	该指标用于统计测试对象的出方向网络流速。	字节/秒（Bytes/Second）	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	上行错包率	ma_container_network_transmit_error_packets	每秒网卡发送的错误包个数。	个/秒（Packets/Second）	≥0	连续2个周期原始值 > 1	紧急告警	网络丢包，建议提工单联系运维支持，排查网络问题。
	发送包速率	ma_container_network_transmit_packets	每秒网卡发送的数据包个数。	个/秒（Packets/Second）	≥0	NA	NA	NA
NPU	NPU使用率	ma_container_npu_util	该指标用于统计测量对象的NPU使用率。（即将废弃，替代指标为ma_container_npu_ai_core_util）。	百分比（Percent）	0~100%	连续2个周期原始值 > 95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NPU显存使用率	ma_container_npu_memory_util	该指标用于统计测量对象已使用的NPU显存占NPU存储容量的百分比。（即将废弃， Snt3系列替代指标为ma_container_npu_ddr_memory_util，Snt9系列替代指标为ma_container_npu_hbm_util）。	百分比（Percent）	0~100%	连续2个周期原始值>98%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	NPU显存使用量	ma_container_npu_memory_used_megabytes	该指标用于统计测量对象已使用的NPU显存。（即将废弃， Snt3系列替代指标为ma_container_npu_ddr_memory_usage_bytes，Snt9系列替代指标为ma_container_npu_hbm_usage_bytes）。	≥0	兆字节（Megabytes）	NA	NA	NA
	NPU显存容量	ma_container_npu_memory_total_megabytes	该指标用于统计测量对象的NPU显存容量。（即将废弃， Snt3系列替代指标为ma_container_npu_ddr_memory_bytes，Snt9系列替代指标为ma_container_npu_hbm_bytes）。	>0	兆字节（Megabytes）	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	AI处理器错误码	ma_container_npu_ai_core_error_code	昇腾系列AI处理器错误码。	-	-	连续3个周期原始值 > 0	紧急告警	卡异常，建议提工单联系运维支持。
	AI处理器健康状态	ma_container_npu_ai_core_health_status	昇腾系列AI处理器健康状态。	-	• 1：健康 • 0：不健康	连续2个周期原始值为0	紧急告警	卡异常，建议提工单联系运维支持。
	AI处理器功耗	ma_container_npu_ai_core_power_usage_watts	昇腾系列AI处理器功耗。	瓦特（W）	>0	NA	NA	NA
	AI处理器温度	ma_container_npu_ai_core_temperature_celsius	昇腾系列AI处理器温度。	摄氏度（℃）	自然数	NA	NA	NA
	AI处理器AI CORE利用率	ma_container_npu_ai_core_util	昇腾系列AI处理器AI Core利用率。	百分比（Percent）	0～100%	连续2个周期原始值 > 95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NPU整体利用率	ma_container_npu_general_util	昇腾系列AI处理器NPU整体利用率（驱动版本24.1.RC2及其以后支持）。	百分比（Percent）	0～100%	NA	NA	NA
	AI处理器AI CORE时钟频率	ma_container_npu_ai_core_frequency_hertz	昇腾系列AI处理器AI Core时钟频率。	赫兹（Hz）	>0	NA	NA	NA
	AI处理器电压	ma_container_npu_ai_core_voltage_volts	昇腾系列AI处理器电压。	伏特（V）	自然数	NA	NA	NA
	AI处理器DDR内存总量	ma_container_npu_ddr_memory_bytes	昇腾系列AI处理器DDR内存总量。	字节（Byte）	>0	NA	NA	NA
	AI处理器DDR内存使用量	ma_container_npu_ddr_memory_usage_bytes	昇腾系列AI处理器DDR内存使用量。	字节（Byte）	>0	NA	NA	NA
	AI处理器DDR内存利用率	ma_container_npu_ddr_memory_util	昇腾系列AI处理器DDR内存利用率。 Snt9C无此指标。	百分比（Percent）	0～100%	连续2个周期原始值>95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	AI处理器HBM内存总量	ma_container_npu_hbm_bytes	昇腾系列AI处理器HBM总内存（Snt9 AI处理器专属）。	字节（Byte）	>0	NA	NA	NA
	AI处理器HBM内存使用量	ma_container_npu_hbm_usage_bytes	昇腾系列AI处理器HBM内存使用量（Snt9 AI处理器专属）。	字节（Byte）	>0	NA	NA	NA
	AI处理器HBM内存利用率	ma_container_npu_hbm_util	昇腾系列AI处理器HBM内存利用率（Snt9 AI处理器专属）。	百分比（Percent）	0～100%	连续2个周期原始值>95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	AI处理器HBM内存带宽利用率	ma_container_npu_hbm_bandwidth_util	昇腾系列AI处理器HBM内存带宽利用率（Snt9 AI处理器专属）。	百分比（Percent）	0～100%	连续2个周期原始值>95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	AI处理器HBM内存时钟频率	ma_container_npu_hbm_frequency_hertz	昇腾系列AI处理器HBM内存时钟频率（Snt9 AI处理器专属）。	赫兹（Hz）	>0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	AI处理器HBM内存温度	ma_container_npu_hbm_temperature_celsius	昇腾系列AI处理器HBM内存温度（Snt9 AI处理器专属）。	摄氏度（℃）	自然数	NA	NA	NA
	AI处理器AI CPU利用率	ma_container_npu_ai_cpu_util	昇腾系列AI处理器AI CPU利用率。	百分比（Percent）	0～100%	NA	NA	NA
	AI处理器控制CPU利用率	ma_container_npu_ctrl_cpu_util	昇腾系列AI处理器控制CPU利用率。	百分比（Percent）	0～100%	NA	NA	NA
	AI处理器控制CPU频率	ma_node_npu_ctrl_cpu_frequency_hertz	昇腾系列AI处理器控制CPU频率。	赫兹（Hz）	>0 系统态（专属池用户态）	NA	NA	NA
	AI处理器Vector CORE利用率	ma_container_npu_vector_core_util	昇腾系列AI处理器Vector Core利用率。	百分比（Percent）	0～100%	连续2个周期原始值>95%	建议	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
NPU RoCE网络	NPU RoCE网络上行速率	ma_container_npu_roce_tx_rate_bytes_per_second	容器所使用的NPU网络模块上行速率。	字节/秒（Bytes/Second）	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NPU RoCE网络下行速率	ma_container_npu_roce_rx_rate_bytes_per_second	容器所使用的NPU网络模块下行速率。	字节/秒（Bytes/Second）	≥0	NA	NA	NA
Notebook业务指标	Notebook cache目录大小	ma_container_notebook_cache_dir_size_bytes	GPU和NPU类型的Notebook会在“/cache”目录上挂载一块高速本地磁盘，该指标描述该目录的总大小。	字节（Bytes）	≥0	NA	NA	NA
	Notebook cache目录利用率	ma_container_notebook_cache_dir_util	GPU和NPU类型的Notebook会在“/cache”目录上挂载一块高速本地磁盘，该指标描述该目录的利用率。	百分比（Percent）	0~100%	连续2个周期原始值>90%	重要	磁盘使用率过高时，会导致Notebook实例重启。

节点级别的指标介绍

表 11-2 节点指标（仅专属池上会收集）

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
CPU	CPU内核总量	ma_node_cpu_limit_core	该指标用于统计测量对象申请的CPU核总量。	核（Core）	≥1	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	CPU内核占用	ma_node_cpu_used_core	该指标用于统计测量对象已经使用的CPU核数。	核（Core）	≥0	NA	NA	NA
	CPU使用率	ma_node_cpu_util	该指标用于统计测量对象的CPU使用率。	百分比（Percent）	0~100%	连续2个周期原始值 > 95%	重要	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	CPU IO等待时间	ma_node_cpu_iowait_counter	从系统启动开始累计到当前时刻，硬盘IO等待时间。	jiffies	≥0	NA	NA	NA
内存	物理内存使用率	ma_node_memory_util	该指标用于统计测量对象已使用内存占申请物理内存总量的百分比。	百分比（Percent）	0~100%	连续2个周期原始值 > 95%	重要	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	物理内存容量	ma_node_memory_total_megabytes	该指标用于统计测量申请的物理内存总量。	兆字节（Megabytes）	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
网络IO	下行Bps	ma_node_network_receive_rate_bytes_seconds	该指标用于统计测试对象的入方向网络流速。	字节/秒（Bytes/Second）	≥0	NA	NA	NA
	上行Bps	ma_node_network_transmit_rate_bytes_seconds	该指标用于统计测试对象的出方向网络流速。	字节/秒（Bytes/Second）	≥0	NA	NA	NA
存储	磁盘读取速率	ma_node_disk_read_rate_kilobytes_seconds	该指标用于统计每秒从磁盘读出的数据量。只考虑被容器使用的数据盘。	千字节/秒（Kilobytes/Second）	≥0	NA	NA	NA
	磁盘写入速率	ma_node_disk_write_rate_kilobytes_seconds	该指标用于统计每秒写入磁盘的数据量。只考虑被容器使用的数据盘。	千字节/秒（Kilobytes/Second）	≥0	NA	NA	NA
	cache空间的总量	ma_node_cache_space_capacity_megabytes	该指标用于统计k8s空间的总容量。	兆字节（Megabytes）	≥0	NA	NA	NA
	cache空间的使用量	ma_node_cache_space_used_capacity_megabytes	该指标用于统计k8s空间的使用量。	兆字节（Megabytes）	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	cache空间的使用率	ma_node_cache_space_used_percent	该指标用于统计k8s空间的使用率。	百分比（Percent）	≥0	连续2个周期原始值 > 90%	紧急	请及时检查，防止磁盘写满影响业务。推荐清理计算节点无效数据。
	容器空间的总量	ma_node_container_space_capacity_megabytes	该指标用于统计容器空间的总容量。	兆字节（Megabytes）	≥0	NA	NA	NA
	容器空间的使用量	ma_node_container_space_used_capacity_megabytes	该指标用于统计容器空间的使用量。	兆字节（Megabytes）	≥0	NA	NA	NA
	容器空间的使用率	ma_node_container_space_used_percent	该指标用于统计容器空间的使用率。	百分比（Percent）	≥0	连续2个周期原始值 > 90%	紧急	请及时检查，防止磁盘写满影响业务。推荐清理计算节点无效数据。

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	磁盘信息	ma_node_disk_info	该指标用于展示磁盘的基础信息。	-	≥0	NA	NA	NA
	读取次数	ma_node_disk_reads_completed_total	成功完成的读取总次数。	-	≥0	NA	NA	NA
	合并读取的次数	ma_node_disk_reads_merged_total	合并读取的次数。	-	≥0	NA	NA	NA
	读取字节数	ma_node_disk_read_bytes_total	成功读取的总字节数。	字节（Bytes）	≥0	NA	NA	NA
	读取花费秒数	ma_node_disk_read_time_seconds_total	所有读取所花费的总秒数。	秒（Seconds）	≥0	NA	NA	NA
	写入次数	ma_node_disk_writes_completed_total	成功完成的写入总数。	-	≥0	NA	NA	NA
	合并写入的次数	ma_node_disk_writes_merged_total	合并写入的次数。	-	≥0	NA	NA	NA
	写入字节数	ma_node_disk_written_bytes_total	成功写入的总字节数。	字节（Bytes）	≥0	NA	NA	NA
	写入花费秒数	ma_node_disk_write_time_seconds_total	所有写入操作花费的总秒数。	秒（Seconds）	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	当前IO数量	ma_node_disk_io_now	当前正在进行的I/O数量。	-	≥0	NA	NA	NA
	IO花费总秒数	ma_node_disk_io_time_seconds_total	执行I/O所花费的总秒数。	秒（Seconds）	≥0	NA	NA	NA
	IO花费加权秒数	ma_node_disk_io_time_weighted_seconds_tota	执行I/O所花费的加权秒数。	秒（Seconds）	≥0	NA	NA	NA
GPU	GPU使用率	ma_node_gpu_util	该指标用于统计测量对象的GPU使用率。	百分比（Percent）	0~100%	NA	NA	NA
	GPU显存容量	ma_node_gpu_mem_total_megabytes	该指标用于统计测量对象的显存容量。	兆字节（Megabytes）	>0	NA	NA	NA
	GPU显存使用率	ma_node_gpu_mem_util	该指标用于统计测量对象已使用的显存占显存容量的百分比。	百分比（Percent）	0~100%	连续2个周期原始值 > 97%	提示	排查是否符合业务资源使用预期，如果业务无问题，无需处理。

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	GPU显存使用量	ma_node_gpu_mem_used_megabytes	该指标用于统计测量对象已使用的显存。	兆字节（Megabytes）	≥0	NA	NA	NA
	GPU显存空闲容量	ma_node_gpu_mem_free_megabytes	该指标用于统计测量空闲的显存。	兆字节（Megabytes）	>0	NA	NA	NA
	共享GPU任务运行数据	node_gpu_share_job_count	针对一个GPU卡，当前运行的共享资源使用的任务数量。	个	≥0	NA	NA	NA
	GPU温度	DCGM_FI_DEV_GPU_TEMP	表示GPU温度。	摄氏度（℃）	自然数	NA	NA	NA
	GPU功率	DCGM_FI_DEV_POWER_USAGE	表示GPU功率。	瓦特（W）	>0	NA	NA	NA
	GPU显存温度	DCGM_FI_DEV_MEMORY_TEMP	表示显存温度。	摄氏度（℃）	自然数	NA	NA	NA
NPU	NPU使用率	ma_node_npu_util	该指标用于统计测量对象的NPU使用率。（即将废弃，替代指标为ma_node_npu_ai_core_util）。	百分比（Percent）	0~100%	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NPU显存使用率	ma_node_npu_memory_util	该指标用于统计测量对象已使用的NPU显存占NPU存储容量的百分比。（即将废弃， Snt3系列替代指标为ma_node_npu_ddr_memory_util，Snt9系列替代指标为ma_node_npu_hbm_util）。	百分比（Percent）	0~100%	连续2个周期原始值 > 97%	提示	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	NPU显存使用量	ma_node_npu_memory_used_megabytes	该指标用于统计测量对象已使用的NPU显存。（即将废弃， Snt3系列替代指标为ma_node_npu_ddr_memory_usage_bytes，Snt9系列替代指标为ma_node_npu_hbm_usage_bytes）。	≥0	兆字节（Megabytes）	NA	NA	NA
	NPU显存容量	ma_node_npu_memory_total_megabytes	该指标用于统计测量对象的NPU显存容量。（即将废弃， Snt3系列替代指标为ma_node_npu_ddr_memory_bytes，Snt9系列替代指标为ma_node_npu_hbm_bytes）。	>0	兆字节（Megabytes）	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	AI处理器错误码	ma_node_npu_ai_core_error_code	昇腾系列AI处理器错误码。	-	-	NA	NA	NA
	AI处理器健康状态	ma_node_npu_ai_core_health_status	昇腾系列AI处理器健康状态。	-	• 1：健康 • 0：不健康	连续2周期 值为0	紧急	建议参考**故障列表**，或者提工单咨询。
	AI处理器功耗	ma_node_npu_ai_core_power_usage_watts	昇腾系列AI处理器功耗。	瓦特（W）	>0	NA	NA	NA
	AI处理器温度	ma_node_npu_ai_core_temperature_celsius	昇腾系列AI处理器温度。	摄氏度（℃）	自然数	NA	NA	NA
	AI处理器AI CORE利用率	ma_node_npu_ai_core_util	昇腾系列AI处理器AI Core利用率。	百分比（Percent）	0~100%	NA	NA	NA
	NPU整体利用率	ma_node_npu_general_util	昇腾系列AI处理器NPU整体利用率（驱动版本24.1.RC2及其以后支持）。	百分比（Percent）	0~100%	NA	NA	NA
	AI处理器AI CORE时钟频率	ma_node_npu_ai_core_frequency_hertz	昇腾系列AI处理器AI Core时钟频率。	赫兹（Hz）	>0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	AI处理器电压	ma_node_npu_ai_core_voltage_volts	昇腾系列AI处理器电压。	伏特（V）	自然数	NA	NA	NA
	AI处理器DDR内存总量	ma_node_npu_ddr_memory_bytes	昇腾系列AI处理器DDR内存总量。 Snt9C无此指标。	字节（Byte）	>0	NA	NA	NA
	AI处理器DDR内存使用量	ma_node_npu_ddr_memory_usage_bytes	昇腾系列AI处理器DDR内存使用量	字节（Byte）	>0	NA	NA	NA
	AI处理器DDR内存利用率	ma_node_npu_ddr_memory_util	昇腾系列AI处理器DDR内存利用率。	百分比（Percent）	0~100%	连续2个周期原始值 > 90%	提示	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	AI处理器HBM内存总量	ma_node_npu_hbm_bytes	昇腾系列AI处理器HBM总内存（Snt9 AI处理器专属）。	字节（Byte）	>0	NA	NA	NA
	AI处理器HBM内存使用量	ma_node_npu_hbm_usage_bytes	昇腾系列AI处理器HBM内存使用量（Snt9 AI处理器专属）。	字节（Byte）	>0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	AI处理器HBM内存利用率	ma_node_npu_hbm_util	昇腾系列AI处理器HBM内存利用率（Snt9 AI处理器专属）。	百分比（Percent）	0~100%	连续2个周期原始值 > 97%	提示	排查是否符合业务资源使用预期，如果业务无问题，无需处理。
	AI处理器HBM内存带宽利用率	ma_node_npu_hbm_bandwidth_util	昇腾系列AI处理器HBM内存带宽利用率（Snt9 AI处理器专属）。	百分比（Percent）	0~100%	NA	NA	NA
	AI处理器HBM内存时钟频率	ma_node_npu_hbm_frequency_hertz	昇腾系列AI处理器HBM内存时钟频率（Snt9 AI处理器专属）。	赫兹（Hz）	>0	NA	NA	NA
	AI处理器HBM内存温度	ma_node_npu_hbm_temperature_celsius	昇腾系列AI处理器HBM内存温度（Snt9 AI处理器专属）。	摄氏度（℃）	自然数	NA	NA	NA
	AI处理器AI CPU利用率	ma_node_npu_ai_cpu_util	昇腾系列AI处理器AI CPU利用率。	百分比（Percent）	0~100%	NA	NA	NA
	AI处理器控制CPU利用率	ma_node_npu_ctrl_cpu_util	昇腾系列AI处理器控制CPU利用率。	百分比（Percent）	0~100%	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	AI处理器控制CPU频率	ma_node_npu_ctrl_cpu_frequency_hertz	昇腾系列AI处理器控制CPU频率。	赫兹（Hz）	>0 系统态（专属池用户可见）	NA	NA	NA
	HBM ECC检测开关	ma_node_npu_hbm_ecc_enable	0表示ecc检测未使能；1表示ecc检测使能。	-	• 1：使能 • 0：未使能	NA	NA	NA
	HBM单比特当前错误计数	ma_node_npu_hbm_single_bit_error_total	HBM单比特当前错误计数。	个	≥0	NA	NA	NA
	HBM多比特当前错误计数	ma_node_npu_hbm_double_bit_error_total	HBM多比特当前错误计数。	个	≥0	NA	NA	NA
	HBM生命周期内所有单比特错误数量	ma_node_npu_hbm_total_single_bit_error_total	HBM生命周期内所有单比特错误数量。	个	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	HBM生命周期内所有多比特错误数量	ma_node_npu_hbm_total_double_bit_error_total	HBM生命周期内所有多比特错误数量。	个	≥0	NA	NA	NA
	HBM单比特错误隔离内存页数量	ma_node_npu_hbm_single_bit_isolated_pages_total	HBM单比特错误隔离内存页数量。	个	≥0	NA	NA	NA
	HBM多比特错误隔离内存页数量	ma_node_npu_hbm_double_bit_isolated_pages_total	HBM多比特错误隔离内存页数量。	个	≥0	连续2个周期原始值 >= 64	严重	若此计数达到64及以上，请提交工单，切换NPU机器。
	AI处理器Vector CORE利用率	ma_node_npu_vector_core_util	昇腾系列AI处理器Vector Core利用率。	百分比（Percent）	0~100%	NA	NA	NA
NPU RoCE网络	NPU RoCE网络上行速率	ma_node_npu_roce_tx_rate_bytes_per_second	NPU RoCE网络上行速率。	字节/秒（Bytes/Second）	≥0	NA	NA	NA
	NPU RoCE网络下行速率	ma_node_npu_roce_rx_rate_bytes_per_second	NPU RoCE网络下行速率。	字节/秒（Bytes/Second）	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	MAC上行pause帧总数	ma_node_npu_roce_mac_tx_pause_packets_total	NPU RoCE网络MAC发送的pause帧总报文数。	个	≥0	NA	NA	NA
	MAC下行pause帧总数	ma_node_npu_roce_mac_rx_pause_packets_total	NPU RoCE网络MAC接收的pause帧总报文数。	个	≥0	NA	NA	NA
	MAC上行pfc帧总数	ma_node_npu_roce_mac_tx_pfc_packets_total	NPU RoCE网络MAC发送的PFC帧总报文数。	个	≥0	delta(ma_node_npu_roce_mac_tx_pause_packets_total[1m]) > 0	重要	建议提交工单处理。
	MAC下行pfc帧总数	ma_node_npu_roce_mac_rx_pfc_packets_total	NPU RoCE网络MAC接收的PFC帧总报文数。	个	≥0	delta(ma_node_npu_roce_mac_rx_pause_packets_total[1m]) > 0	重要	建议提交工单处理。
	MAC上行坏包总数	ma_node_npu_roce_mac_tx_bad_packets_total	NPU RoCE网络MAC发送的坏包总报文数。	个	≥0	delta(ma_node_npu_roce_mac_tx_pfc_packets_total[1m]) > 0	重要	建议提交工单处理。

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	MAC下行坏包总数	ma_node_npu_roce_mac_rx_bad_packets_total	NPU RoCE网络MAC接收的坏包总报文数。	个	≥0	delta(ma_node_npu_roce_mac_rx_pfc_packets_total[1m]) > 0	重要	建议提交工单处理。
	RoCE上行坏包总数	ma_node_npu_roce_tx_err_packets_total	NPU ROCE发送的坏包总报文数。	个	≥0	delta(ma_node_npu_roce_mac_tx_bad_packets_total[1m]) > 0	重要	建议提交工单处理。
	RoCE下行坏包总数	ma_node_npu_roce_rx_err_packets_total	NPU ROCE接收的坏包总报文数。	个	≥0	delta(ma_node_npu_roce_mac_rx_bad_packets_total[1m]) > 0	重要	建议提交工单处理。
	RoCE上行包总数	ma_node_npu_roce_tx_all_packets_total	NPU ROCE发送的总报文数。	个	≥0	delta(ma_node_npu_roce_tx_err_packets_total[1m]) > 0	重要	建议提交工单处理。
	RoCE下行包总数	ma_node_npu_roce_rx_all_packets_total	NPU ROCE接收的总报文数。	个	≥0	delta(ma_node_npu_roce_rx_err_packets_total[1m]) > 0	重要	建议提交工单处理。

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
NPU光模块（Snt9B/C风冷组网具备该指标）	光模块温度	ma_node_npu_optical_temperature	光模块温度。	摄氏度 C	≥0	NA	NA	NA
	光模块电源电压	ma_node_npu_optical_vcc	光模块电源电压。	毫伏 mV	≥0	NA	NA	NA
	光模块发送功率0	ma_node_npu_optical_tx_power0	光模块发送功率0。	毫瓦 mW	≥0	NA	NA	NA
	光模块发送功率1	ma_node_npu_optical_tx_power1	光模块发送功率1。	毫瓦 mW	≥0	NA	NA	NA
	光模块发送功率2	ma_node_npu_optical_tx_power2	光模块发送功率2。	毫瓦 mW	≥0	NA	NA	NA
	光模块发送功率3	ma_node_npu_optical_tx_power3	光模块发送功率3。	毫瓦 mW	≥0	NA	NA	NA
	光模块接收功率0	ma_node_npu_optical_rx_power0	光模块接收功率0。	毫瓦 mW	≥0	NA	NA	NA
	光模块接收功率1	ma_node_npu_optical_rx_power1	光模块接收功率1。	毫瓦 mW	≥0	NA	NA	NA
	光模块接收功率2	ma_node_npu_optical_rx_power2	光模块接收功率2。	毫瓦 mW	≥0	NA	NA	NA
	光模块接收功率3	ma_node_npu_optical_rx_power3	光模块接收功率3。	毫瓦 mW	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
infiniband或RoCE网络	网卡接收数据总量	ma_node_infiniband_port_received_data_bytes_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), received on all VLs from the port.	(counting in double words, 32 bits	≥0	NA	NA	NA
	网卡发送数据总量	ma_node_infiniband_port_transmitted_data_bytes_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), transmitted on all VLs from the port.	(counting in double words, 32 bits	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
NFS挂载状态	NFS检索文件属性操作拥塞时间	ma_node_mountstats_getattr_backlog_wait	Getattr is an NFS operation that retrieves the attributes of a file or directory, such as size, permissions, owner, etc. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NFS检索文件属性操作往返时间	ma_node_mountstats_getattr_rtt	Getattr is an NFS operation that retrieves the attributes of a file or directory, such as size, permissions, owner, etc. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NFS检查文件权限操作拥塞时间	ma_node_mountstats_access_backlog_wait	Access is an NFS operation that checks the access permissions of a file or directory for a given user. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NFS检查文件权限操作往返时间	ma_node_mountstats_access_rtt	Access is an NFS operation that checks the access permissions of a file or directory for a given user. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NFS解析文件句柄操作拥塞时间	ma_node_mountstats_lookup_backlog_wait	Lookup is an NFS operation that resolves a file name in a directory to a file handle. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NFS解析文件句柄操作往返时间	ma_node_mountstats_lookup_rtt	Lookup is an NFS operation that resolves a file name in a directory to a file handle. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NFS读文件操作拥塞时间	ma_node_mountstats_read_backlog_wait	Read is an NFS operation that reads data from a file. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NFS读文件操作往返时间	ma_node_mountstats_read_rtt	Read is an NFS operation that reads data from a file. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NFS写文件操作拥塞时间	ma_node_mountstats_write_backlog_wait	Write is an NFS operation that writes data to a file. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0	NA	NA	NA

分类	名称	指标	指标含义	单位	取值范围	告警阈值	告警级别	处理建议
	NFS写文件操作往返时间	ma_node_mountstats_write_rtt	Write is an NFS operation that writes data to a file. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0	NA	NA	NA

网络相关指标

表 11-3 Diagnos（IB，仅专属池上会收集）

分类	名称	指标	指标含义	单位	取值范围
infiniband或RoCE网络	PortXmitData	infiniband_port_xmit_data_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), transmitted on all VLs from the port.	计数值	自然数
	PortRcvData	infiniband_port_rcv_data_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), received on all VLs from the port.	计数值	自然数
	SymbolErrorCounter	infiniband_symbol_error_counter_total	Total number of minor link errors detected on one or more physical lanes.	计数值	自然数
	LinkErrorRecoveryCounter	infiniband_link_error_recovery_counter_total	Total number of times the Port Training state machine has successfully completed the link error recovery process.	计数值	自然数

分类	名称	指标	指标含义	单位	取值范围
	PortRcvErrors	infiniband_port_rcv_errors_total	Total number of packets containing errors that were received on the port including: Local physical errors (ICRC, VCRC, LPCRC, and all physical errors that cause entry into the BAD PACKET or BAD PACKET DISCARD states of the packet receiver state machine) Malformed data packet errors (LVer, length, VL) Malformed link packet errors (operand, length, VL) Packets discarded due to buffer overrun (overflow)	计数值	自然数
	LocalLinkIntegrityErrors	infiniband_local_link_integrity_errors_total	This counter indicates the number of retries initiated by a link transfer layer receiver.	计数值	自然数
	PortRcvRemotePhysicalErrors	infiniband_port_rcv_remote_physical_errors_total	Total number of packets marked with the EBP delimiter received on the port.	计数值	自然数
	PortRcvSwitchRelayErrors	infiniband_port_rcv_switch_relay_errors_total	Total number of packets received on the port that were discarded when they could not be forwarded by the switch relay for the following reasons: DLID mapping VL mapping Looping (output port = input port)	计数值	自然数

分类	名称	指标	指标含义	单位	取值范围
	PortXmitWait	infiniband_port_transmit_wait_total	The number of ticks during which the port had data to transmit but no data was sent during the entire tick (either because of insufficient credits or because of lack of arbitration).	计数值	自然数
	PortXmitDiscards	infiniband_port_xmit_discards_total	Total number of outbound packets discarded by the port because the port is down or congested.	计数值	自然数

Label 相关指标介绍

表 11-4 Label 名字栏

指标对象	Label名字	Label描述
容器级别指标	modelarts_service	容器属于哪个服务，包含notebook，train和infer。
	instance_name	容器所属pod的名字。
	service_id	页面展示的实例或者job id。如开发环境为：cf55829e-9bd3-48fa-8071-7ae870dae93a， 训练作业为：9f322d5a-b1d2-4370-94df-5a87de27d36e
	node_ip	容器所属的节点IP值。
	container_id	容器ID。
	cid	集群ID。
	container_name	容器名称。
	project_id	用户所属的账号的project id。
	user_id	提交作业的用户所属的账号的user id。
	pool_id	物理专属池对应的资源池id。
	pool_name	物理专属池对应的资源池name。
	logical_pool_id	逻辑子池的id。

指标对象	Label名字	Label描述
	logical_pool_name	逻辑子池的name。
	gpu_uuid	容器使用的GPU的UUID。
	gpu_index	容器使用的GPU的索引。
	gpu_type	容器使用的GPU的型号。
	account_name	训练、推理或开发环境任务创建者的账号名。
	user_name	训练、推理或开发环境任务创建者的用户名。
	task_creation_time	训练、推理或开发环境任务的创建时间。
	task_name	训练、推理或开发环境任务的名称。
	task_spec_code	训练、推理或开发环境任务的规格。
	cluster_name	CCE集群名称。
node级别指标	cid	该node所属CCE集群的ID。
	node_ip	节点的IP。
	host_name	节点的主机名。
	pool_id	物理专属池对应的资源池ID。
	project_id	物理专属池的用户的project id。
	gpu_uuid	节点上GPU的UUID。
	gpu_index	节点上GPU的索引。
	gpu_type	节点上GPU的型号。
	device_name	infiniband或RoCE网络网卡的设备名称。
	port	IB网卡的端口号。
	physical_state	IB网卡每个端口的状态。
	firmware_version	IB网卡的固件版本。
	filesystem	NFS挂载的文件系统。
	mount_point	NFS的挂载点。
Diagnos	cid	GPU所在节点所属的CCE集群ID。
	node_ip	GPU所在节点的IP。
	pool_id	物理专属池对应的资源池ID。
	project_id	物理专属池的用户的project id。
	gpu_uuid	GPU的UUID。

指标对象	Label名字	Label描述
	gpu_index	节点上GPU的索引。
	gpu_type	节点上GPU的型号。
	device_name	网络设备或磁盘设备的名称。
	port	IB网卡的端口号。
	physical_state	IB网卡每个端口的状态。
	firmware_version	IB网卡的固件版本。

11.4 使用 Grafana 查看 AOM 中的监控指标

11.4.1 安装配置 Grafana

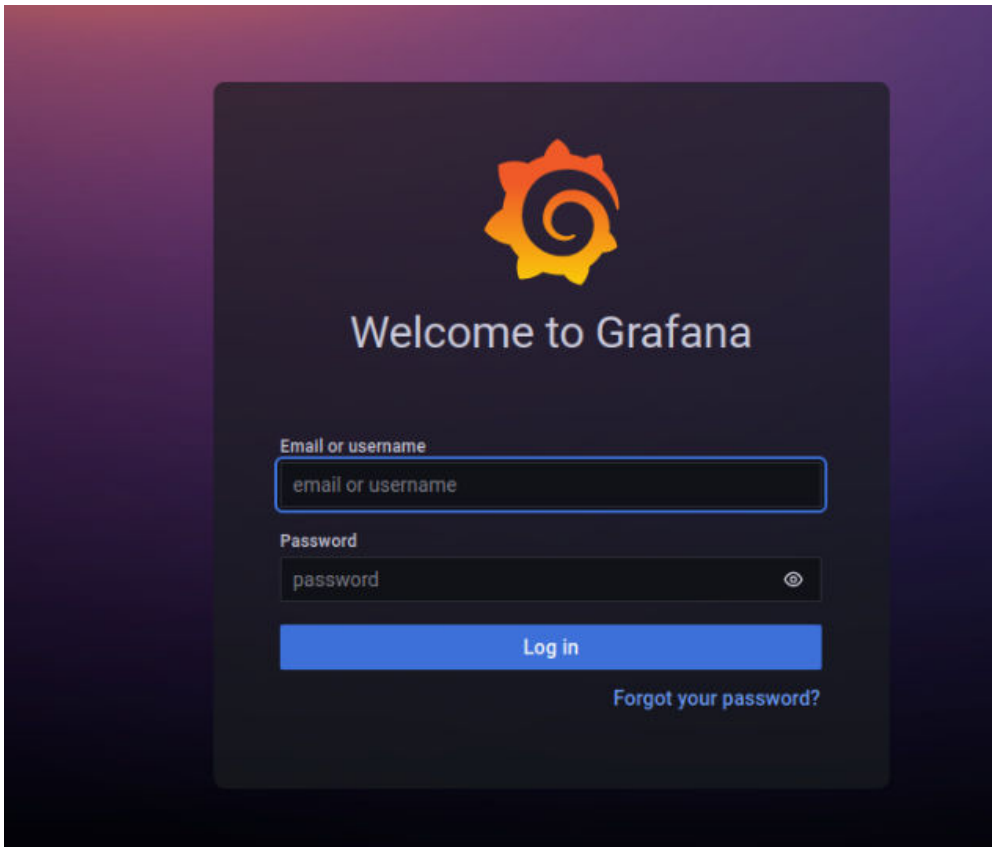
11.4.1.1 在 Windows 上安装配置 Grafana

适用场景

本章节适用于在Windows操作系统的PC中安装配置Grafana。

操作步骤

1. 下载Grafana安装包。
   进入**下载链接**，单击Download the installer，等待下载成功即可。
2. 安装Grafana。
   双击安装包，按照指示流程安装完成即可。
3. 在Windows的“服务”中，找到Grafana，将其开启，如果已经开启，则直接进入**4**。
4. 登录Grafana。
   Grafana默认在本地的3000端口启动，打开链接http://localhost:3000，出现Grafana的登录界面。首次登录用户名和密码为admin，登录成功后请根据提示修改密码。



11.4.1.2 在 Linux 上安装配置 Grafana

适用场景

本章节适用于在Linux操作系统的PC中安装配置Grafana。

前提条件

- 一台可访问外网的Ubuntu服务器。如果没有请具备以下条件：
- 准备一台ECS服务器（建议规格选8U或者以上，镜像选择Ubuntu，建议选择22.04版本，本地存储100G），具体操作请参考**《购买弹性云服务器》**。
- 购买弹性公网IP，并绑定到购买的弹性云服务器ECS上，具体操作请参见**《弹性公网IP快速入门》**。

操作步骤

1. 登录弹性云服务器。根据需要选择登录方式，具体操作请参考。
2. 执行如下命令安装libfontconfig1。
   ```
   sudo apt-get install -y adduser libfontconfig1
   ```
   回显如下代表执行成功：


```
root@ecs-9ec3:~# sudo apt-get install -y adduser libfontconfig1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
adduser is already the newest version (3.118ubuntu5).
adduser set to manually installed.
libfontconfig1 is already the newest version (2.13.1-4.2ubuntu5).
libfontconfig1 set to manually installed.
The following packages were automatically installed and are no longer required:
  eatmydata libeatmydata1 libflashrom1 libftdi1-2 python-babel-localedata python3-babel python3-certifi python3-jinja2
  python3-json-pointer python3-jsonpatch python3-jsonschema python3-markupsafe python3-pyrsistent python3-requests python3-tz
  python3-urllib3
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
```

3. 执行如下命令下载Grafana安装包。

   wget **https://dl.grafana.com/oss/release/grafana_9.3.6_amd64.deb** --no-check-certificate

   下载完成：

```
root@ecs-9ec3:~# wget https://dl.grafana.com/oss/release/grafana_9.3.6_amd64.deb --no-check-certificate
--2023-03-07 10:22:12--  https://dl.grafana.com/oss/release/grafana_9.3.6_amd64.deb
Resolving dl.grafana.com (dl.grafana.com)... 151.101.42.217
Connecting to dl.grafana.com (dl.grafana.com)|151.101.42.217|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 89252050 (85M) [application/octet-stream]
Saving to: 'grafana_9.3.6_amd64.deb'

grafana_9.3.6_amd64.deb        100%[=====================================================>]  85.12M   379KB/s    in 2m 21s

2023-03-07 10:24:36 (617 KB/s) - 'grafana_9.3.6_amd64.deb' saved [89252050/89252050]
```

4. 执行如下命令安装Grafana。

   sudo dpkg -i grafana_9.3.6_amd64.deb

```
root@ecs-9ec3:~# sudo dpkg -i grafana_9.3.6_amd64.deb
Selecting previously unselected package grafana.
(Reading database ... 80788 files and directories currently installed.)
Preparing to unpack grafana_9.3.6_amd64.deb ...
Unpacking grafana (9.3.6) ...
Setting up grafana (9.3.6) ...
Adding system user `grafana' (UID 116) ...
Adding new user `grafana' (UID 116) with group `grafana' ...
Not creating home directory `/usr/share/grafana'.
### NOT starting on installation, please execute the following statements to configure grafana to start automatically using syst
emd
 sudo /bin/systemctl daemon-reload
 sudo /bin/systemctl enable grafana-server
### You can start grafana-server by executing
 sudo /bin/systemctl start grafana-server
```

5. 执行命令启动Grafana。

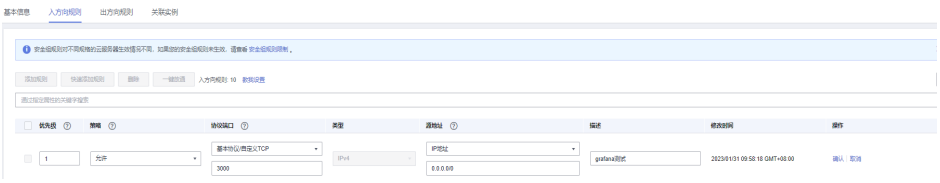
   sudo /bin/systemctl start grafana-server

6. 在本地PC访问Grafana配置。

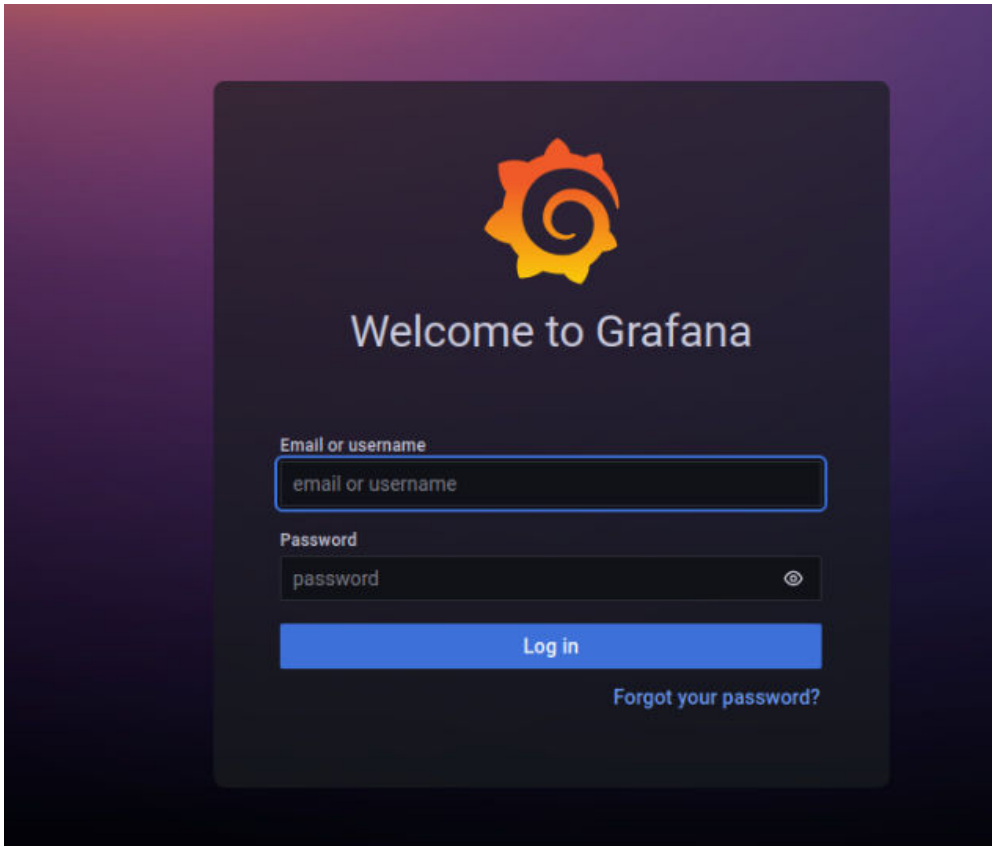
   确保ECS绑定了弹性公网IP，且对应**安全组**配置正确（入方向放开TCP协议的3000端口，出方向全部放通）。设置如下：

   a. 单击ECS服务器名称进入详情页，单击“安全组”页签，单击“配置规则”。

   b. 单击“入方向规则”，入方向放开TCP协议的3000端口，出方向默认全部放通。



7. 在浏览器中输入“http://{弹性公网IP}:3000”，即可进行访问。首次登录用户名和密码为admin，登录成功后请根据提示修改密码。



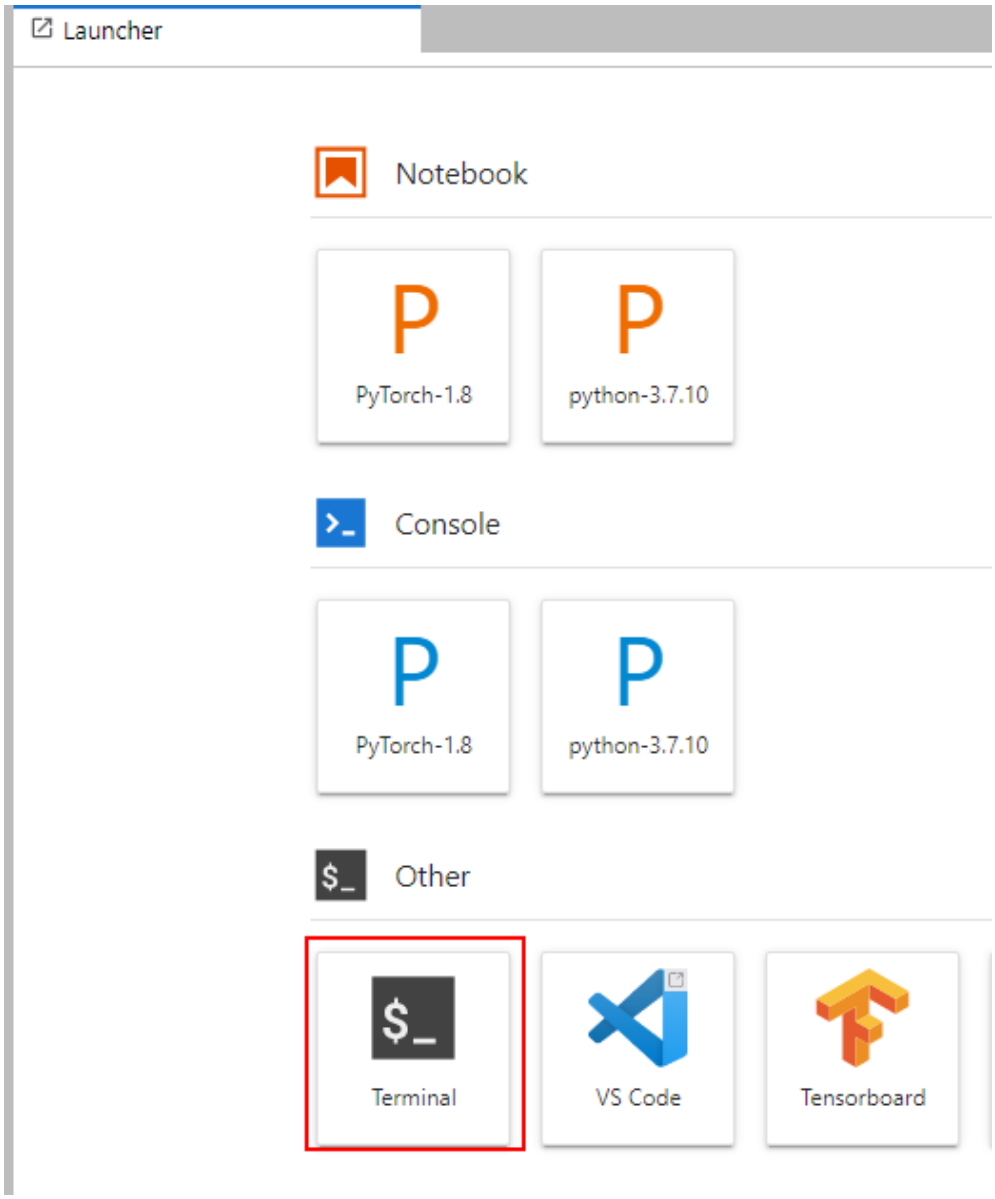
11.4.1.3 在 Notebook 上安装配置 Grafana

适用场景

本章节适用于在ModelArts Standard的Notebook中安装配置Grafana。

前提条件

- 已创建CPU或GPU类型的Notebook实例，并处于运行中。
- 打开Terminal。



操作步骤

1. 在Terminal中依次执行以下命令，下载并安装Grafana。

```
mkdir -p /home/ma-user/work/grf
cd /home/ma-user/work/grf
wget https://dl.grafana.com/oss/release/grafana-9.1.6.linux-amd64.tar.gz
tar -zxvf grafana-9.1.6.linux-amd64.tar.gz
```

2. 将Grafana注册到jupyter-server-proxy。
   a. 在JupyterLab Terminal中执行以下命令：

```
mkdir -p /home/ma-user/.local/etc/jupyter
vi /home/ma-user/.local/etc/jupyter/jupyter_notebook_config.py
```

```
(PyTorch-1.8) [ma-user grf]$mkdir -p /home/ma-user/.local/etc/jupyter
(PyTorch-1.8) [ma-user grf]$vi /home/ma-user/.local/etc/jupyter/jupyter_notebook_config.py
```

b. 在打开的juputer_notebook_config.py中，增加以下代码后按ESC退出然后输入:wq保存。

```
c.ServerProxy.servers = {
  'grafana': {
    'command': ['/home/ma-user/work/grf/grafana-9.1.6/bin/grafana-server', '--homepath', '/home/ma-user/work/grf/grafana-9.1.6', 'web'],
    'timeout': 1800,
    'port': 3000
  }
}
```

说明

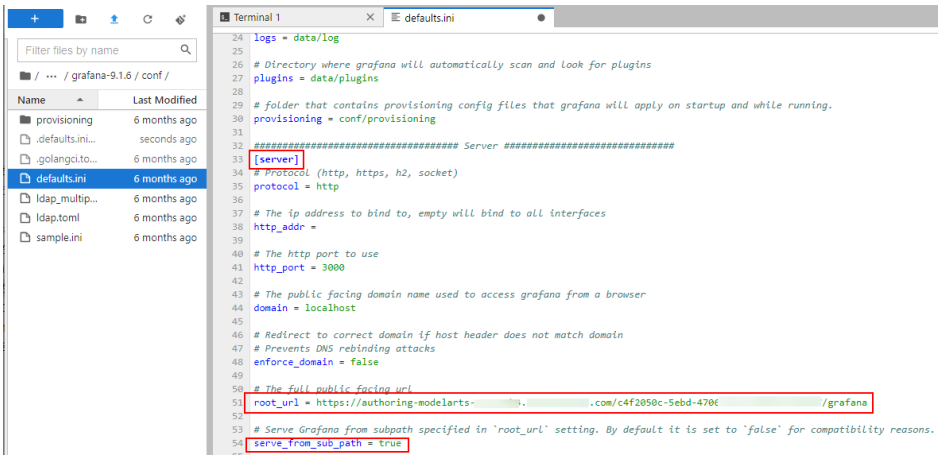
如果“/home/ma-user/.local/etc/jupyter/jupyter_notebook_config.py”文件中已有“c.ServerProxy.servers”字段，新增对应的key-value键值对即可。

3. 适配JupyterLab访问地址。

a. 在左侧导航打开“ vi /home/ma-user/work/grf/grafana-9.1.6/conf/defaults.ini”文件。

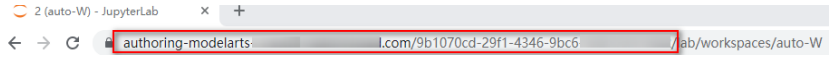
b. 修改[server]中的“root_url”和“serve_from_sub_path”字段。

图 11-4 修改 defaults.ini 文件



其中：

- root_url的组成为：https:{jupyterlab域名}/{INSTANCE_ID}/grafana。域名和INSTANCE_ID可以从打开的jupyterLab页面地址栏获取，如下：



- Serve_from_sub_path设置为true

4. 保存Notebook镜像。

a. 进入Notebook控制台，单击“开发空间 > Notebook”，在Notebook实例列表里找到对应的实例，选择“更多 > 保存镜像”。

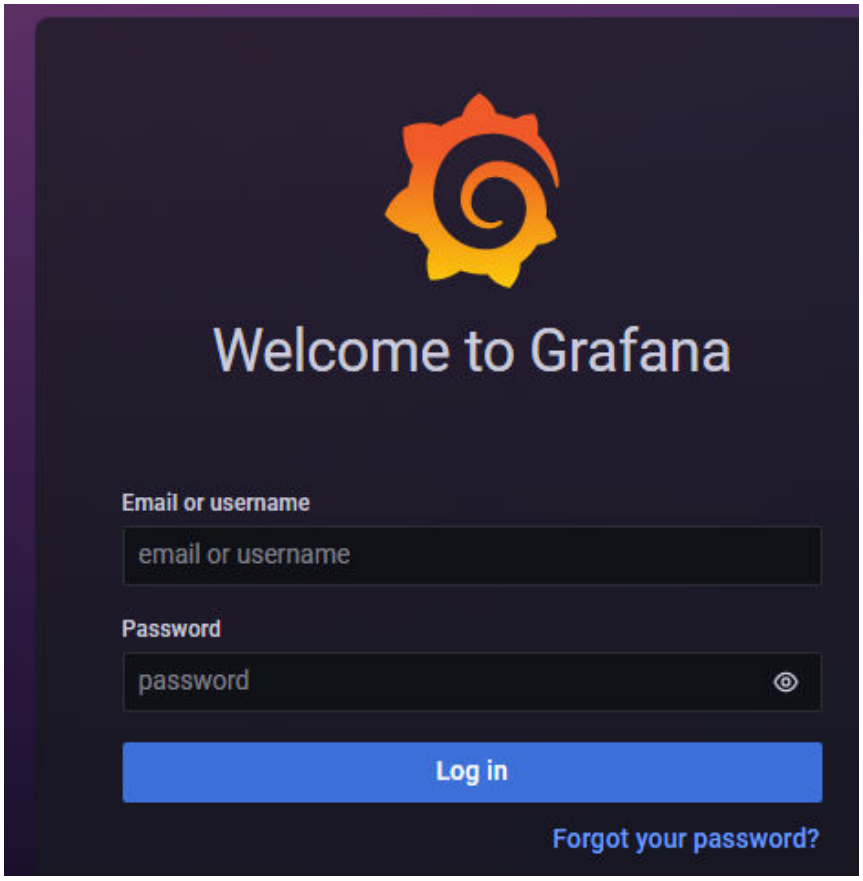
b. 在保存镜像对话框中，设置组织、镜像名称、镜像版本和描述信息。单击“确定”保存镜像。

c. 镜像会以快照的形式保存，保存过程约5分钟，请耐心等待。此时不可再操作实例。

d. 镜像保存成功后，实例状态变为“运行中”，重启Notebook实例。

5. 打开Grafana页面。

新打开一个浏览器窗口，在地址栏输入3中配置的root_url后。出现Grafana登录页面即代表在Notebook中安装和配置Grafana成功。首次登录用户名和密码为admin，登录成功后请根据提示修改密码。



11.4.2 配置 Grafana 数据源

在Grafana配置数据源后，即可通过Grafana查看ModelArts的监控数据。

前提条件

已安装Grafana。

配置 Grafana 数据源

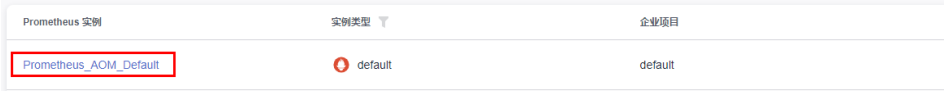
1. 获取Grafana数据源配置代码。

a. 进入AOM管理控制台。

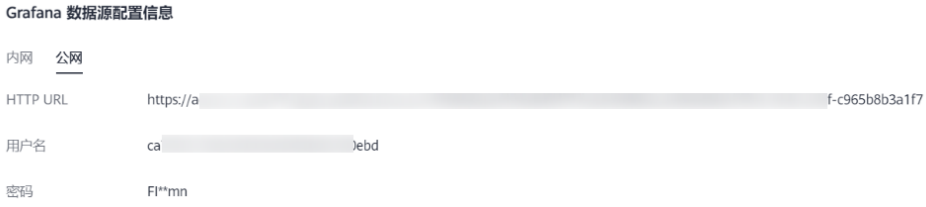
图 11-5 AOM 管理控制台

b. 在左侧导航栏中选择“Prometheus监控 > 实例列表”，在实例列表中单击“Prometheus_AOM_Default”实例。

图 11-6 Prometheus_AOM_Default

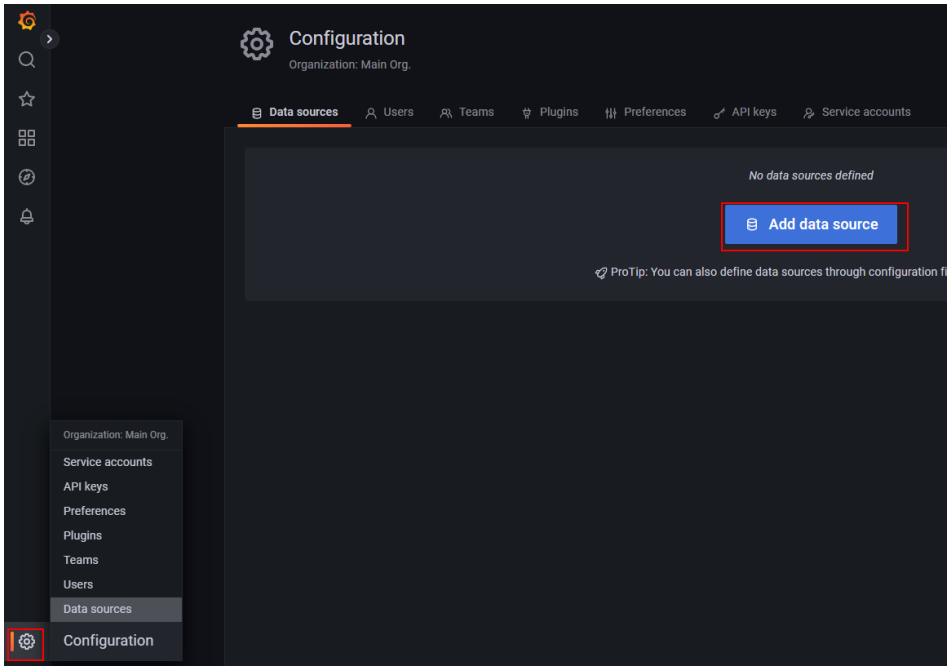


c. 从“设置”页签的“Grafana数据源配置信息”区域，获取当前Prometheus实例的Grafana数据源配置代码。



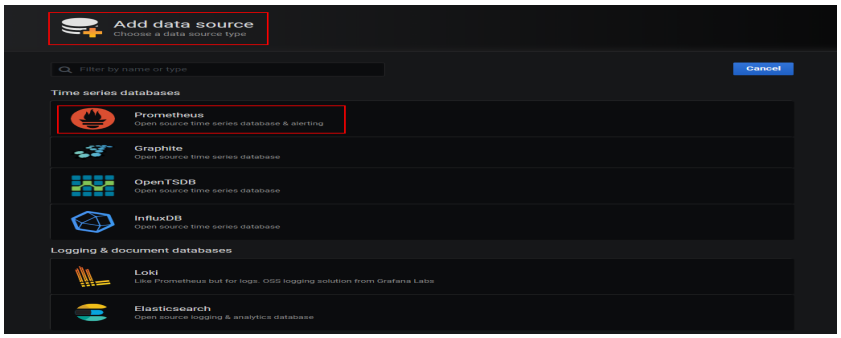
2. 在Grafana中增加数据源。
   a. 登录Grafana。首次登录用户名和密码为admin，登录成功后可根据提示修改密码。
   b. 在左侧菜单栏，选择“Configuration > Data Sources”，单击“Add data source”。

图 11-7 配置 Grafana



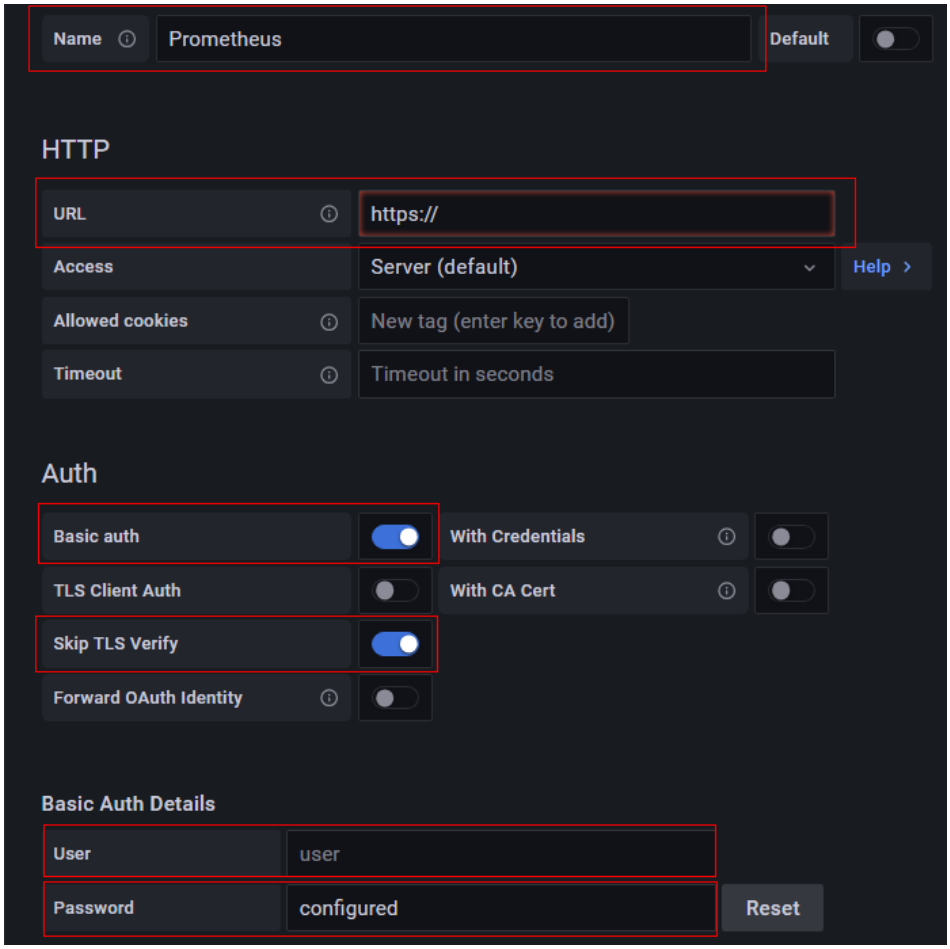
c. 单击“Prometheus”，进入Prometheus配置页面。

图 11-8 进入 Prometheus 配置页面



d. 参考下图进行配置。

图 11-9 配置 Grafana 数据源



说明

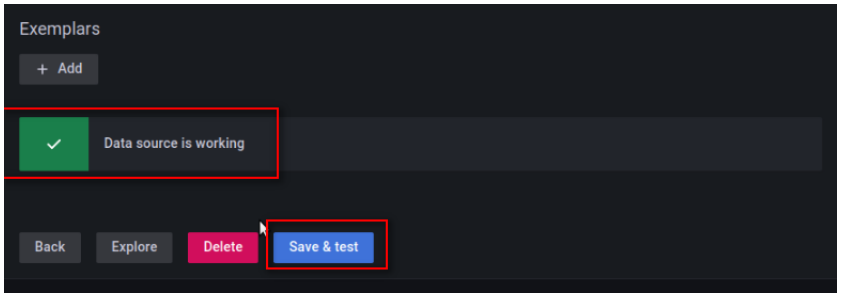
Grafana安装方式不同，Grafana版本也可能不同，图11-9仅为示例，请以实际配置界面为准。

表 11-5 参数配置说明

参数名称	配置说明
Name	自定义名称。
URL	设置为从**c.从“设置”页签的“Grafana数据源配置信...**获取的HTTP URL信息。
Basic auth	建议开启。
Skip TLS Verify	建议开启。
User	设置为从**c.从“设置”页签的“Grafana数据源配置信...**获取的用户名信息。
Password	设置为从**c.从“设置”页签的“Grafana数据源配置信...**获取的密码信息。

e. 配置完成后，单击下方的“Save & test”，展示“Data source is working”代表配置数据源成功。

图 11-10 配置数据源成功



11.4.3 配置仪表盘查看指标数据

Grafana中可以自定义配置各种视图的仪表盘，ModelArts也提供了针对集群的配置模板。本章节通过使用ModelArts提供的模板查看指标和创建Dashboards查看指标的方式，说明如何进行仪表盘配置。Grafana的更多使用请参考**Grafana官方文档**。

准备工作

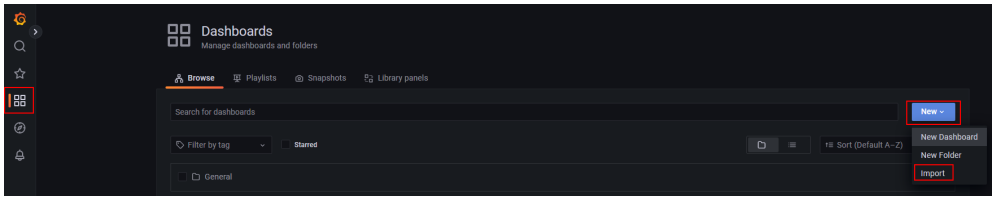
ModelArts提供了集群视图、节点视图、用户视图、任务视图和任务详细视图这5个模板，这些模板在Grafana官方文档可以搜索下载，您导入模板配置Dashboards时，可直接使用。

表 11-6 模板下载地址

模板名称	下载地址
集群视图	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-Cluster-View.json
节点视图	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-Node-View.json
用户视图	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-User-View.json
任务视图	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-Task-View.json
任务详细视图	https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/metrics/grafana/dashboards/ModelArts-Task-Detail-View.json

使用 ModelArts 提供的模板查看指标

1. 打开“DashBoards”，选择“New”>“Import”。



2. 导入DashBoards模板。

   复制**准备工作**提供的模板的下载地址到浏览器中打开，复制JSON文件的内容。粘贴到DashBoards模板里，如**图11-12**所示，最后单击“Load”。

图 11-11 复制 JSON 文件的内容

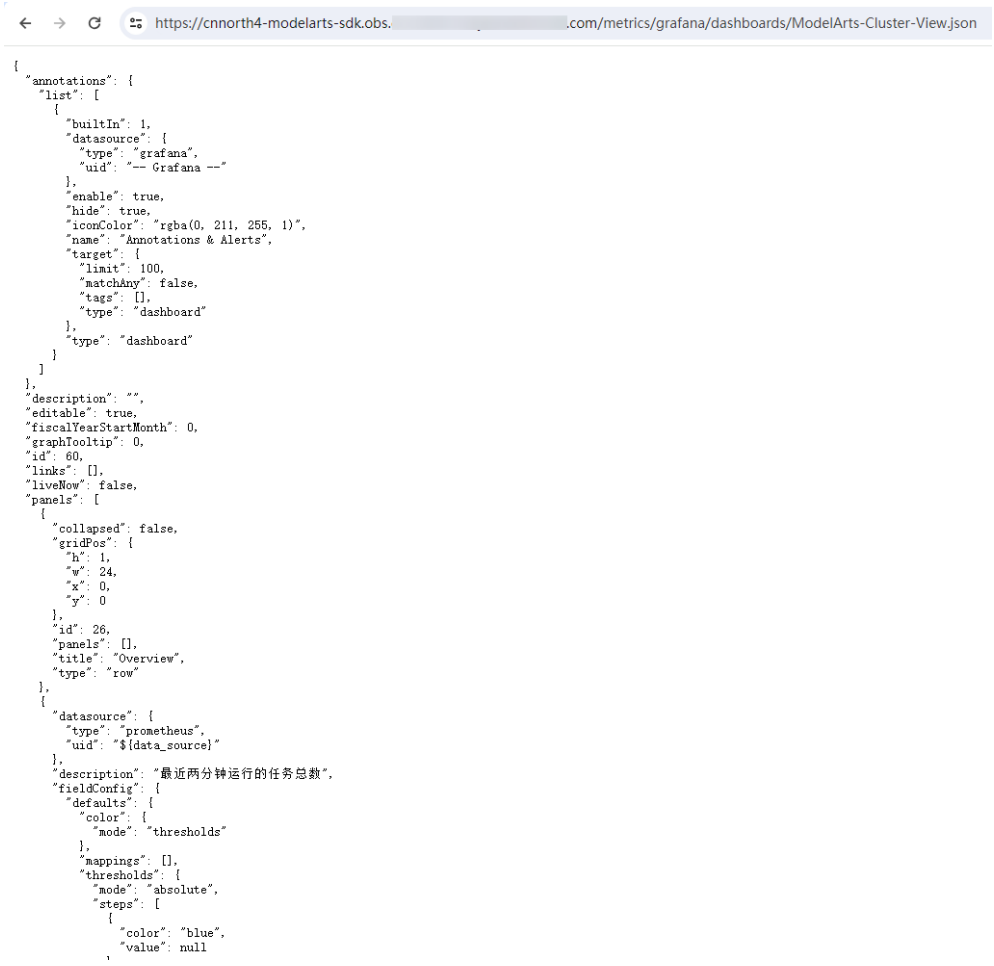
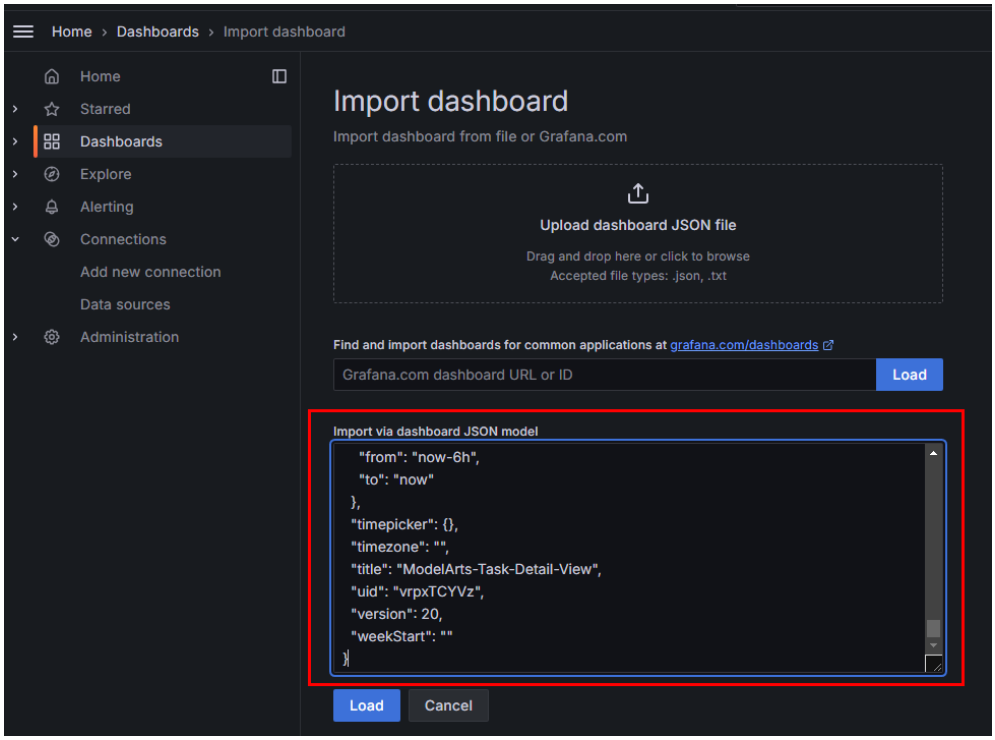
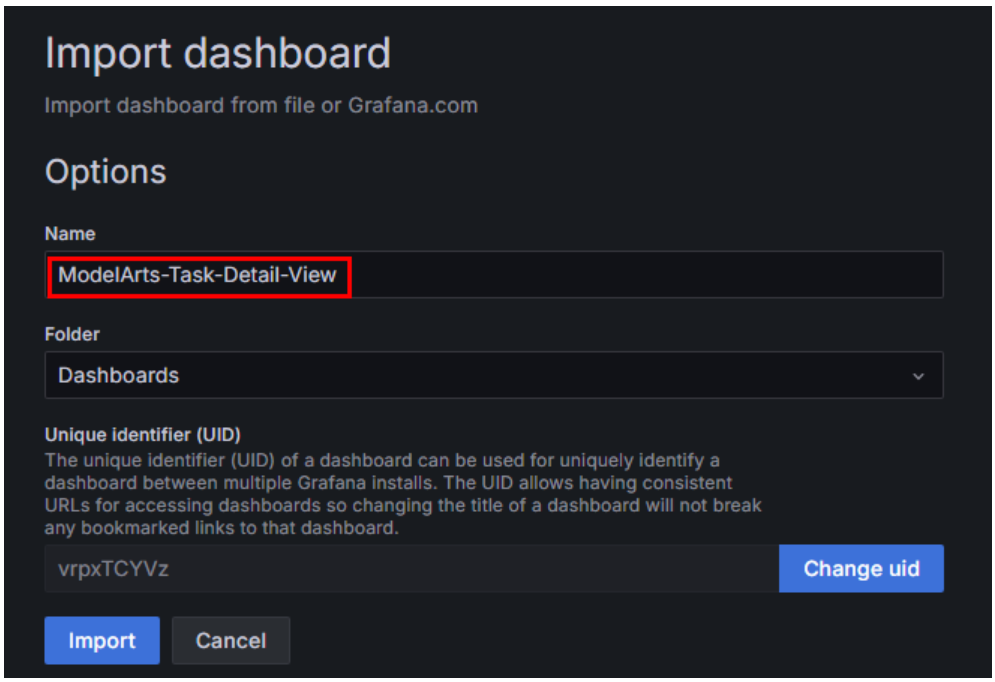


图 11-12 粘贴 JSON 文件的内容到 DashBoards 模板



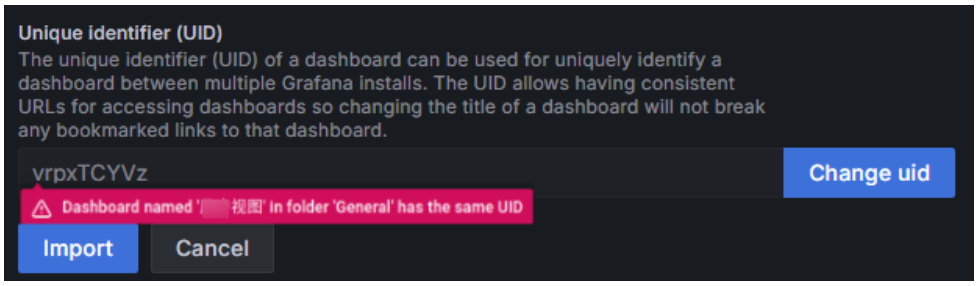
3. 修改视图名称，单击Import。

图 11-13 修改视图名称

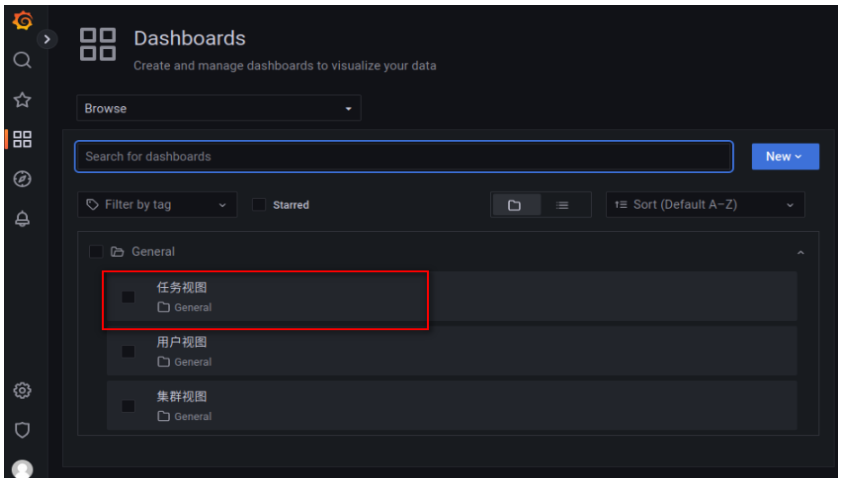


注意：如提示uid重复，则单击“Change uid”，修改json中的uid后单击“Import”。

图 11-14 修改 uid



4. 导入成功后，在Dashboards下，即可看到导入的视图，单击视图即可打开监控。

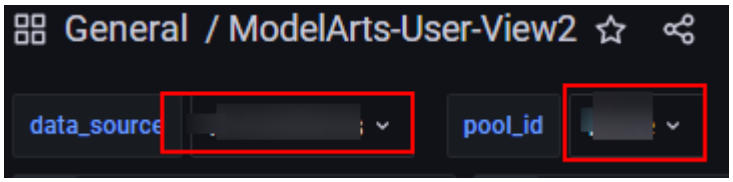


5. 模板使用

   导入成功后，单击想查看的模板即可查看响应内容。这里介绍一些常用功能的使用。

   - 切换数据源和资源池

     图 11-15 切换数据源和资源池

     单击红框中相应位置，即可出现下拉框，修改响应的数据源和资源池。

   - 刷新数据

     单击右上角的图标，即可刷新整个DashBoard的所有数据，各panel也会更新

   - 修改自动刷新时间

     模板的默认刷新时间是15分钟，如果觉得该时间不合适，可在右上角下拉选择修改，修改后，单击保存即可生效。

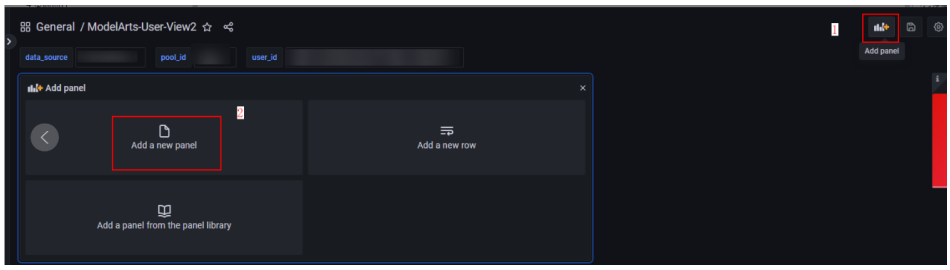
   - 修改DashBoard查询数据时间范围

图 11-16 修改查询数据时间范围

单击右上角图标，即可修改DashBoard整体的数据查询时间。除固定查询时间外的其他panel，都会应用该数据查询时间范围。

- 增加新panel

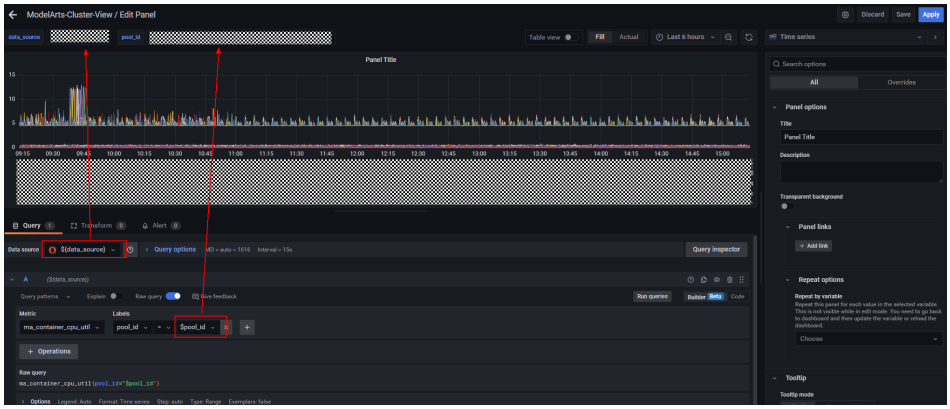
图 11-17 新增一个 panel



单击右上角的'+'图标，即可新增一个panel。

新增一个panel后，即可在其中查询相应的数据。将数据源和资源池进行如下的相应选择，即可应用当前DashBoard的对应配置。

图 11-18 使用当前 DashBoard 的配置



创建 Dashboards 查看指标

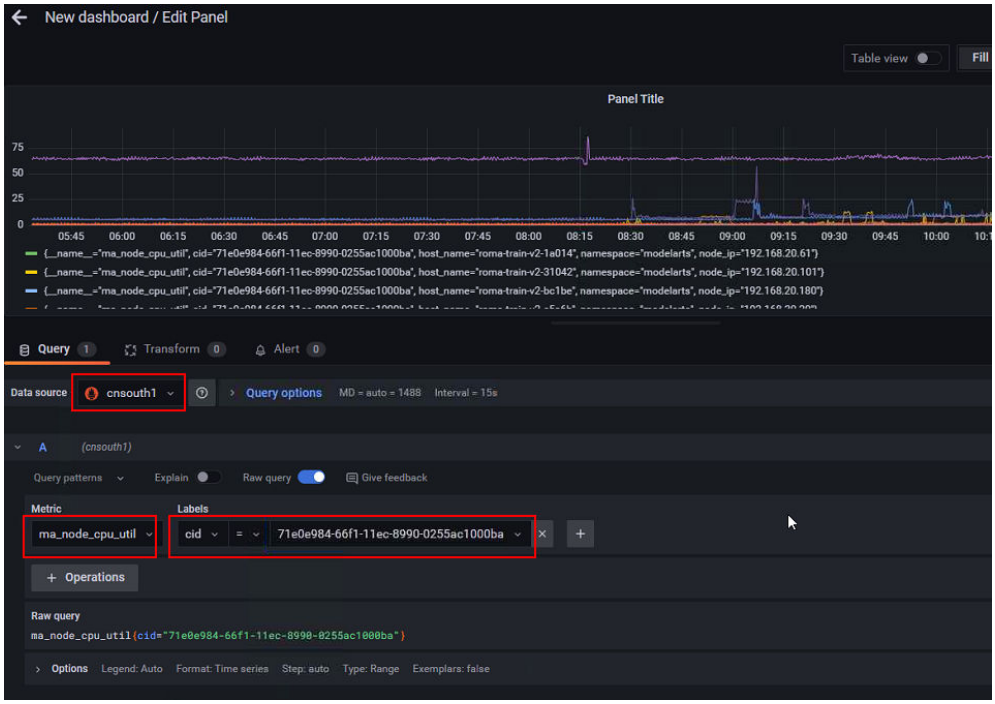
1. 打开“DashBoards”，单击“New”，选择“New Dashboards”。
2. 在New Dashboards界面，单击“Add a new panel”。
3. 在New dashboard /Edit Panel界面，填写如下参数。

   Data source：**已配置Grafana数据源**；

   Metric：指标名称，可参考**表11-1**、**表11-2**、**表11-3**获取想要查询的指标；

   Labels：填写过滤该指标的标签，请参考**表11-4**。

图 11-19 创建 Dashboards 查看指标



12 使用 CTS 审计 ModelArts 服务

12.1 ModelArts 支持云审计的关键操作

通过云审计服务，您可以记录与ModelArts相关的操作事件，便于日后的查询、审计和回溯。

前提条件

已开通云审计服务。

数据管理支持审计的关键操作列表

表 12-1 数据管理支持审计的关键操作列表

操作名称	资源类型	事件名称
创建数据集	dataset	createDataset
删除数据集	dataset	deleteDataset
更新数据集	dataset	updateDataset
发布数据集版本	dataset	publishDatasetVersion
删除数据集版本	dataset	deleteDatasetVersion
同步数据源	dataset	syncDataSource
导出数据集	dataset	exportDataFromDataset
创建自动标注任务	dataset	createAutoLabelingTask
创建自动分组任务	dataset	createAutoGroupingTask
创建自动部署任务	dataset	createAutoDeployTask

操作名称	资源类型	事件名称
导入样本到数据集	dataset	importSamplesToDataset
创建数据集标签	dataset	createLabel
更新数据集标签	dataset	updateLabel
删除数据集标签	dataset	deleteLabel
删除数据集标签和对应的样本	dataset	deleteLabelWithSamples
添加样本	dataset	uploadSamples
删除样本	dataset	deleteSamples
停止自动标注任务	dataset	stopTask
创建团队标注任务	dataset	createWorkforceTask
删除团队标注任务	dataset	deleteWorkforceTask
启动团队标注验收的任务	dataset	startWorkforceSamplingTask
通过/驳回/取消验收任务	dataset	updateWorkforceSamplingTask
提交验收任务的样本评审意见	dataset	acceptSamples
给样本添加标签	dataset	updateSamples
发送邮件给团队标注任务的成员	dataset	sendEmails
接口人启动团队标注任务	dataset	startWorkforceTask
更新团队标注任务	dataset	updateWorkforceTask
给团队标注样本添加标签	dataset	updateWorkforceTaskSamples
团队标注审核	dataset	reviewSamples
创建标注成员	workforce	createWorker
更新标注成员	workforce	updateWorker
删除标注成员	workforce	deleteWorker
批量删除标注成员	workforce	batchDeleteWorker
创建标注团队	workforce	createWorkforce
更新标注团队	workforce	updateWorkforce
删除标注团队	workforce	deleteWorkforce

操作名称	资源类型	事件名称
自动创建IAM委托	IAM	createAgency
标注成员登录labelConsole标注平台	labelConsoleWorker	workerLoginLabelConsole
标注成员登出labelConsole标注平台	labelConsoleWorker	workerLogOutLabelConsole
标注成员修改labelConsole平台密码	labelConsoleWorker	workerChangePassword
标注成员忘记labelConsole平台密码	labelConsoleWorker	workerForgetPassword
标注成员通过url重置labelConsole标注密码	labelConsoleWorker	workerResetPassword

开发环境支持审计的关键操作列表

表 12-2 开发环境支持审计的关键操作列表

操作名称	资源类型	事件名称
创建Notebook	Notebook	createNotebook
删除Notebook	Notebook	deleteNotebook
打开Notebook	Notebook	openNotebook
启动Notebook	Notebook	startNotebook
停止Notebook	Notebook	stopNotebook
更新Notebook	Notebook	updateNotebook
删除NotebookApp	NotebookApp	deleteNotebookApp
切换CodeLab规格	NotebookApp	updateNotebookApp

训练作业支持审计的关键操作列表

表 12-3 训练作业支持审计的关键操作列表

操作名称	资源类型	事件名称
创建训练作业	ModelArtsTrainJob	createModelArtsTrainJob
创建训练作业版本	ModelArtsTrainJob	createModelArtsTrainVersion

操作名称	资源类型	事件名称
停止训练作业	ModelArtsTrainJob	stopModelArtsTrainVersion
更新训练作业描述	ModelArtsTrainJob	updateModelArtsTrainDesc
删除训练作业版本	ModelArtsTrainJob	deleteModelArtsTrainVersion
删除训练作业	ModelArtsTrainJob	deleteModelArtsTrainJob
创建训练作业参数	ModelArtsTrainConfig	createModelArtsTrainConfig
更新训练作业参数	ModelArtsTrainConfig	updateModelArtsTrainConfig
删除训练作业参数	ModelArtsTrainConfig	deleteModelArtsTrainConfig
创建可视化作业	ModelArtsTensorboardJob	createModelArtsTensorboardJob
删除可视化作业	ModelArtsTensorboardJob	deleteModelArtsTensorboardJob
更新可视化作业描述	ModelArtsTensorboardJob	updateModelArtsTensorboardDesc
停止可视化作业	ModelArtsTensorboardJob	stopModelArtsTensorboardJob
重启可视化作业	ModelArtsTensorboardJob	restartModelArtsgTensorboardJob

模型管理支持审计的关键操作列表

表 12-4 模型管理支持审计的关键操作列表

操作名称	资源类型	事件名称
创建模型	model	addModel
更新模型	model	updateModel
删除模型	model	deleteModel
添加转换任务	convert	addConvert
更新转换任务	convert	updateConvert
删除转换任务	convert	deleteConvert

服务管理支持审计的关键操作列表

表 12-5 服务管理支持审计的关键操作列表

操作名称	资源类型	事件名称
部署服务	service	addService
删除服务	service	deleteService
更新服务	service	updateService
启停服务	service	startOrStopService
添加用户访问密钥	service	addAkSk
删除用户访问密钥	service	deleteAkSk
创建专属资源池	cluster	createCluster
删除专属资源池	cluster	deleteCluster
添加专属资源池节点	cluster	addClusterNode
删除专属资源池节点	cluster	deleteClusterNode
获取专属资源池创建结果	cluster	createClusterResult

AI Gallery 支持审计的关键操作列表

表 12-6 AI Gallery 支持审计的关键操作列表

操作名称	资源类型	事件名称
发布资产	ModelArts_Market	create_content
修改资产信息	ModelArts_Market	modify_content
发布资产新版本	ModelArts_Market	add_version
订阅资产	ModelArts_Market	subscription_content
取消收藏资产	ModelArts_Market	cancel_star_content
点赞资产	ModelArts_Market	like_content
取消点赞资产	ModelArts_Market	cancel_like_content
发布实践	ModelArts_Market	publish_activity
报名实践	ModelArts_Market	regist_activity
修改个人资料	ModelArts_Market	update_user

资源管理支持审计的关键操作列表

表 12-7 资源管理支持审计的关键操作列表

操作名称	资源类型	事件名称
创建资源池	PoolV2	CreatePoolV2
删除资源池	PoolV2	DeletePoolV2
更新资源池	PoolV2	UpdatePoolV2
创建网络	NetworksV1	CreateNetworksV1
删除网络	NetworksV1	DeleteNetworksV1
更新网络	NetworksV1	UpdateNetworksV1

12.2 查看 ModelArts 相关审计日志

在您开启了云审计服务后，系统会记录ModelArts的相关操作，且控制台保存最近7天的操作记录。本节介绍如何在云审计服务管理控制台查看最近7天的操作记录。

操作步骤

1. 登录云审计服务管理控制台。
2. 在管理控制台左上角单击 图标，选择区域。
3. 在左侧导航栏中，单击“事件列表”，进入“事件列表”页面。
4. 事件列表支持通过筛选来查询对应的操作事件。当前事件列表支持四个维度的组合查询，详细信息如下：
   - 事件来源、资源类型和筛选类型。

     在下拉框中选择查询条件。

     其中筛选类型选择事件名称时，还需选择某个具体的事件名称。

     选择资源ID时，还需输入某个具体的资源ID。

     选择资源名称时，还需选择或手动输入某个具体的资源名称。
   - 操作用户：在下拉框中选择某一具体的操作用户，此操作用户指用户级别，而非租户级别。
   - 事件级别：可选项为“所有事件级别”、“normal”、“warning”、“incident”，只可选择其中一项。
   - 时间范围：可选择查询最近七天内任意时间段的操作事件。
5. 在需要查看的事件左侧，单击 展开该事件的详细信息。
6. 单击需要查看的事件“操作”列的“查看事件”，可以在弹窗中查看该操作事件结构的详细信息。

   更多关于云审计服务事件结构的信息，请参见**云审计服务用户指南**。