ModelArts

模型开发

文档版本 01

发布日期 2026-08-03

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 使用 Notebook 进行 AI 开发调试

1.1 Notebook 使用说明

在软件开发领域，随着技术的发展，降低开发者成本和提升开发体验一直是追求的目标。然而，在AI开发阶段，开发者们面临着工具链复杂、资源管理困难等挑战，这与传统软件开发形成了鲜明对比。如何解决这些问题，提升AI开发的效率和体验？ModelArts提供灵活开放的开发环境，通过云原生的资源使用和开发工具链的集成，旨在为不同类型AI开发、探索、教学用户提供更好的云化AI开发体验，降低开发门槛。

警告

Notebook主要用于代码开发与业务调测，不支持长时间稳定运行，不得用于生产类业务。

控制台导航栏说明

为了提升创建Notebook的效率，ModelArts对导航栏进行了一系列的易用性改进。现推出新版导航栏入口样式，旨在简化操作流程并增强界面的直观性。

Notebook导航栏入口：

- 新版控制台：在**ModelArts管理控制台**左侧菜单栏选择“模型开发与训练 > Notebook”，进入“Notebook”页面。

  图 1-1 新版控制台入口

- 旧版控制台：在**ModelArts管理控制台**左侧菜单栏选择“开发空间 > Notebook”，进入“Notebook”页面。

图 1-2 旧版控制台入口

使用 Notebook

创建Notebook

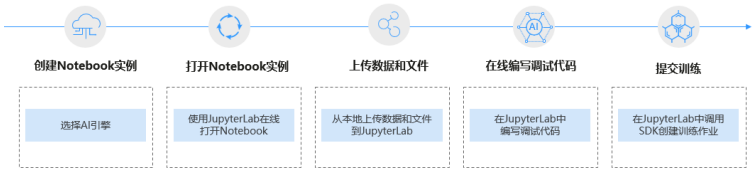
ModelArts提供了云化版本的Notebook，无需关注安装配置，即开即用，具体参见**创建Notebook实例**。

打开Notebook

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。

- JupyterLab：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**。

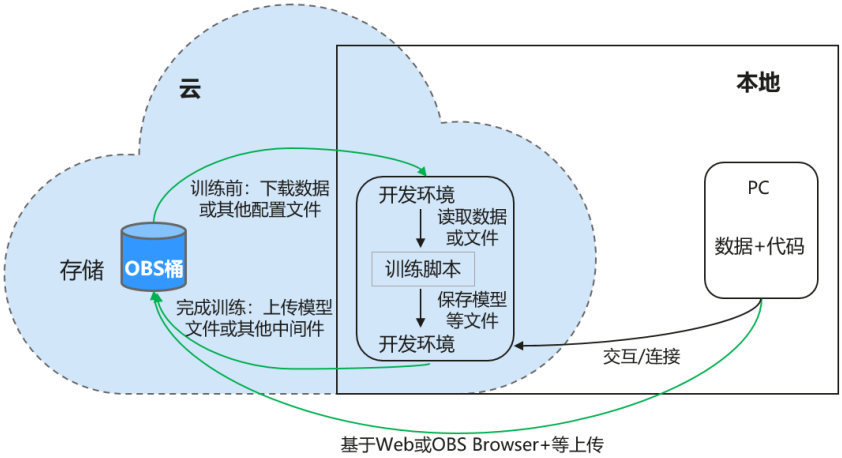
  图 1-3 使用 JupyterLab 在线开发调试代码

- Code Server：**Code Server**是一个开源项目，它将Visual Studio Code完整运行在远程服务器上，并通过浏览器提供访问能力，具体请参见**使用Code Server进行代码开发**。
- 本地IDE：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、SSH工具。VS Code有专门的插件VS Code Toolkit，让远程连接操作更便捷。具体参见**通过VS Code远程使用Notebook实例**、**通过SSH工具远程使用Notebook**。

图 1-4 本地 IDE 远程访问 Notebook



管理Notebook

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。

表 1-1 Notebook 相关操作说明

操作	说明	相关文档
查看Notebook实例详情	查看Notebook实例的详细信息，包括基础信息、存储信息、事件信息等。	**查看Notebook实例详情**
查找Notebook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。	**查找Notebook实例**
接入环境	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。	**打开Notebook**
保存镜像	通过预置镜像创建Notebook实例，在基础镜像上安装对应的自定义软件和依赖，将运行的实例环境以容器镜像的方式保存下来。	**保存Notebook实例**

操作	说明	相关文档
停止/启动/删除Notebook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。 针对不再使用的Notebook实例，可以删除以释放资源。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。	启动/停止/删除Notebook实例
设置自动停止	开启后，可设置Notebook实例自动停止方式及对应时长，超出您预设的时长，将自动停止运行 (可能存在2-5分钟的延迟，此过程正常计费)。	启动/停止/删除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。	变更镜像
变更实例规格	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。	变更Notebook实例运行规格
节点亲和性调度	通过节点亲和性，可以确保Pod被调度到满足特定条件的节点上，从而实现更细粒度的资源管理和优化。	设置节点亲和性调度
配置Notebook SSH远程连接	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。“SSH远程开发”开关打开后不可关闭。	配置Notebook SSH远程连接
编辑标签	标签可用于标识、分类、搜索和管理Notebook实例。	编辑标签
动态扩充云硬盘EVS容量	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。	Notebook动态扩充云硬盘EVS容量
动态挂载OBS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。	Notebook存储挂载方式说明

操作	说明	相关文档
查看Notebook实例资源使用情况	使用Notebook过程中，可以在Notebook详情页了解资源使用情况。	查看Notebook实例资源使用情况
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。	查看Notebook实例事件
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。	Notebook Cache盘告警上报
升级JupyterLab版本	为了提高Notebook的易用性，现将JupyterLab升级至4.4.10版本（2025年稳定版），实现从JupyterLab 3.2.3（2021年老版）的跨版本升级。此次升级对Notebook的性能和启动速度进行了优化，重构了编辑器，强化了调试和协作功能，扩展了更稳定的生态系统，并全面改善了交互体验和内存占用。	升级JupyterLab版本

上传文件至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**。

ModelArts CLI

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**。

MoXing

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**ModelArts MoXing命令参考**。

相关文档

- **开发环境计费项**
- **开发环境最佳实践**
- **开发环境故障排除**
- **开发环境常见问题**

1.2 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。

创建Notebook实例有以下两种方式：

- 通过**ModelArts管理控制台**进行创建，下文将详细介绍创建Notebook实例的控制台操作。
- 通过ModelArts提供的API接口进行创建，详细操作请参见**创建Notebook实例**。

约束限制

- 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。
- 默认每个IAM用户最多可创建10个Notebook实例。
- Snt9b2x（例如Snt9b23等）资源池或D310P-300资源池单卡的实例规格不支持创建挂载EVS（“存储配置”选择“云硬盘EVS”）的Notebook实例。
- Notebook不支持开放端口对外提供服务。
- 当专属资源池的节点被设置为热备节点时，该节点将不支持运行Notebook实例，例如创建、启动Notebook实例等。关于如何关闭热备节点，请参见**修复专属资源池故障节点**。

注意事项

Notebook定位为调测环境，虽然提供了公网下载能力，但使用的公网代理是局点的公共代理。因此，不建议在Notebook中下载大文件，尤其是超过10GB的文件。此外，Notebook提供的公网访问带宽有限，仅能保证网络连通性，不保证下载速度。

计费说明

Notebook使用涉及计费，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。更多信息，请参见**开发环境计费项**。

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”，在“委托列表”页面检查是否配置了委托。如果未配置，请先配置委托，详情请参见**快速配置ModelArts委托**。

   图 1-5 查看委托配置信息

2. 登录**ModelArts管理控制台**，在左侧导航栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：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，参照如下参数说明配置相关信息。
   - **基础信息**

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：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。
添加描述	单击“添加描述”，可以对Notebook进行自定义描述，长度不能超过512位。
标签	如果您需要使用同一标签标识多种云资源，即所有服务均可在标签输入框下拉选择同一标签，建议在TMS中创建预定义标签，具体操作，请参见**创建预定义标签**。 单击“添加标签”，按需输入或选择标签的键，输入标签值。最多可添加20个标签。 添加标签后，您可以在“Notebook”页面或者Notebook实例详情页查看标签内容，也可以按需修改、增加或删除标签。具体操作，请参见**编辑标签**。 **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。

- **自动停止**

参数名称	说明
自动停止	默认开启，当Notebook实例运行时开始计时，运行时间超出您预设的时长时，将自动停止运行Notebook实例。 “停止方式”支持“定时停止”和“空闲停止”。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。支持选择“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。 **注意** 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，方便您进行续约。 “空闲停止”：开启空闲停止功能后，该Notebook实例资源如果空闲时长超出您所设置的时长后，自动停止。支持选择“15分钟”、“30分钟”、“1小时”、“2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。 **说明** 空闲停止当前处于受限使用阶段，如果有试用需求，请提工单申请权限。

- **环境配置**

参数名称	说明
选择镜像	按需选择预置镜像或自定义镜像，然后单击 图标，在“选择镜像”页面，选择目标镜像，单击“确定”。 ● 预置镜像：即预置在ModelArts内部的AI引擎。 ● 自定义镜像：用户创建的自定义镜像。您可以任选以下方式制作自定义镜像。 – 将基于预置镜像创建的实例保存下来，作为自定义镜像使用，详情请参见**保存Notebook实例**。 – 基于基础镜像、企业镜像或第三方镜像制作自定义镜像。制作自定义镜像需要遵循镜像规范，构建完成后需要在ModelArts“镜像管理”页面注册，才能在Notebook中使用，详情请参见**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。具体操作，请参见**更新Notebook实例**。

参数名称	说明
WebIDE	支持JupyterLab和Code Server。在卡片右下角，单击图标，可以切换版本。关于使用Code Server的约束限制及操作指导，请参见**约束限制**。 ● JupyterLab：ModelArts Notebook支持以下两个版本的JupyterLab，默认为JupyterLab 4版本。 – JupyterLab 4：在用户体验、功能完善和性能提升方面均有显著改进。关于JupyterLab 4版本的详细说明，请参见**升级JupyterLab版本**。 – JupyterLab 3：JupyterLab 3版本计划于2026年10月停止支持新的Notebook实例创建，并且不再提供相应的技术支持，包括新特性更新、漏洞/问题修复、补丁升级以及工单指导和在线排查等客户支持服务，这些将不再适用于ModelArts服务的运维保障。建议您使用JupyterLab 4版本。 ● Code Server：Code-Server 4基于VS Code 1.85，支持完整的远程开发体验、内置终端、Git版本控制、调试器和丰富的扩展市场。

– **资源配置**

参数名称		说明
资源配置	资源池类型	支持公共资源池和专属资源池。专属资源池支持CPU、NPU和GPU异构资源池混部能力，例如当节点规格支持GPU和CPU时，“实例规格”可以选择GPU或CPU。 ● “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”：专属资源池不与其他用户共享，资源更可控。在“资源池”区域，单击“选择资源池”，在“选择专属资源池”页面按实际情况选择专属资源池，单击“确定”。如果您没有专属资源池，可以在“选择专属资源池”页面下方单击“购买专属资源池”，按需购买专属资源池后使用。关于购买专属资源池的参数说明，请参见**创建专属资源池**。 **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。
	实例规格	系统会默认选择一个实例规格，您可以按需更改。

参数名称		说明
	规格类型	支持预置规格和自定义规格。 仅“资源池类型”选择“专属资源池”，且节点池规格为GPU、CPU或者异构资源池（节点池规格为CPU+CPU、GPU+CPU等）时，支持自定义规格。公共资源池和NPU类型的专属资源池不支持自定义规格。 **说明** 当异构资源池有NPU节点时，仅NPU节点不支持自定义规格，其他节点可支持自定义规格。 ● 预置规格：ModelArts预置的实例规格，您可以按需选择实例规格下拉框中的规格。 ● 自定义规格：请根据资源池内节点的情况，自定义GPU/CPU和内存的规格。 您可以在**ModelArts管理控制台**的“资源管理 > 专属算力资源池 > 资源池”或者“资源管理 > 专属资源池”页面查看资源池节点详情。 自定义规格取值说明如下： – GPU（卡） 使用GPU虚拟化功能，需满足以下条件。关于使用GPU虚拟化的约束限制，请参见**使用GPU虚拟化**。 – 节点为H20机型。 查看机型：您可以在**ModelArts管理控制台**的“资源管理 > 专属算力资源池 > 资源池”或者“资源管理 > 专属资源池”页面，单击资源池名称，在“节点”页签查看实例规格中是否存在“h20”，如果存在，表示该节点为H20机型。 **图 1-6** H20 机型  – AI套件（NV GPU）插件版本为2.12.0。查看或升级AI套件（NV GPU）插件版本，请参见**AI套件（NV GPU）**。 – 已安装Volcano调度器（Volcano Scheduler）。关于如何安装Volcano调度器，请参见**Volcano调度器（Volcano Scheduler）**。 取值说明： – 当取值＜1时，步长为0.1，即取值范围为[0.0,0.9]。 – 当取值≥1时，步长为1，即取值为整数。

参数名称		说明
		– CPU（vCPUs） 取值≥0.4，单位为核，步长为0.01。 – 内存（MiB） 取值≥513，单位为MiB，步长为1。
	节点亲和性调度	仅“资源池类型”选择“专属资源池”时，支持设置此参数。 通过节点亲和性，可以确保Pod被调度到满足特定条件的节点上，从而实现更细粒度的资源管理和优化。ModelArts支持精细控制Pod的部署策略：严格部署（强亲和）、尽量部署（弱亲和）、禁止部署（强反亲和）、避免部署（弱反亲和）。 选中“节点亲和性调度”，在“亲和调度策略”对话框，按需选择亲和类型、强度和节点，单击“确定”。 ● “亲和类型”选择“节点亲和”，“强度”选择“弱”：尽量将Pod调度到指定节点，不保证成功。 ● “亲和类型”选择“节点亲和”，“强度”选择“强”：严格将Pod调度到指定节点，否则不执行调度。 ● “亲和类型”选择“节点反亲和”，“强度”选择“弱”：避免将Pod调度到指定节点，不保证成功。 ● “亲和类型”选择“节点反亲和”，“强度”选择“强”：禁止将Pod调度到指定节点，否则不执行调度。选择节点时，不支持全部勾选，需要至少保留1个可用节点，否则无法执行调度策略。

参数名称		说明
存储配置	存储类型	包括“云硬盘 EVS”、“对象存储 OBS - 并行文件系统”、“对象存储 OBS - 对象桶”、“高性能弹性文件服务 SFS Turbo”。请根据界面实际情况和需要选择。关于存储类型的介绍，请参见**Notebook存储类型说明**。 在SFS Turbo与OBS桶进行存储联动时，如果将其挂载到Notebook中进行使用，会存在一些使用限制，详情请参见**管理SFS Turbo文件系统与OBS桶的存储联动**。 **说明** “对象存储 OBS - 对象桶”、“对象存储 OBS - 并行文件系统”当前处于受限使用阶段，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。 **注意：** – 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。 – 从Notebook实例创建成功起，直至删除成功，每GB按照规定费用收费。 ● 选择“高性能弹性文件服务 SFS Turbo”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或者关联SFS Turbo，才能生效。 关于如何打通VPC，请参见**ModelArts网络**；关于如何关联SFS Turbo，请参见**关联SFS Turbo**。 **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。 – “文件系统”：选择已创建的SFS Turbo。如果未创建，可以登录在**弹性文件服务控制台**创建SFS Turbo。 – “文件系统目录”：输入文件系统的目录。 **注意：** – 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。 – 高性能弹性文件服务需要在专属资源池对应的网络关联SFS Turbo才能生效。生效后，请您重新进入创建Notebook页面。 ● 选择“对象存储 OBS - 对象桶”或“对象存储 OBS - 并行文件系统 ”作为存储位置。 在“存储地址”下方直接输入存储地址（例如obs://bucketname/path/），或者单击图标，在“选择存储地址”面板，选择用于存储Notebook数据的OBS路径，单击“确定”。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

参数名称		说明
		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。 **注意：** – 挂载在/home/ma-user/work目录下，Notebook实例中对其的增删改文件操作会同步到“对象存储 OBS - 对象桶”或“对象存储 OBS - 并行文件系统”上。 – 该目录下的内容在实例停止后会被保留。 “云硬盘 EVS”、“高性能弹性文件服务 SFS Turbo”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**Notebook存储挂载方式说明**操作来增加数据存储路径。 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。

参数名称		说明
扩展存储		单击“添加扩展存储”，可按需增加不同类型的扩展存储。最多可添加30个扩展存储。对于不需要的扩展存储，可单击“删除”。 关于存储类型的介绍，请参见**Notebook存储类型说明**。 在SFS Turbo与OBS桶进行存储联动时，如果将其挂载到Notebook中进行使用，会存在一些使用限制，详情请参见**管理SFS Turbo文件系统与OBS桶的存储联动**。 ● “存储类型”选择“对象存储 OBS - 对象桶”或“对象存储 OBS - 并行文件系统”： – 存储地址：输入存储地址，例如obs://bucketname/path/。存储地址必须以obs://或/开头，以/结尾，且除前缀外不得出现//。 – 挂载路径：输入挂载路径，例如/temp/。挂载路径不能为空，不允许挂载到黑名单目录，并且以斜杠（/）开头和结尾，仅包含字母、数字、下划线和中划线。 ● “存储类型”选择“高性能弹性文件服务 SFS Turbo ”（仅专属资源池支持）： – 文件系统：按需选择文件系统。 – 文件系统目录：输入文件系统目录。挂载路径不能为空，不允许挂载到黑名单目录。 – 挂载路径：输入挂载路径。挂载路径不能为空，不允许挂载到黑名单目录，并且以斜杠（/）开头和结尾，仅包含字母、数字、下划线和中划线。 **说明** 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实例详情页的“存储配置”页签，查看或编辑扩展存储信息。在存储个数未达到最大个数时，也可以单击“添加扩展存储”。具体操作，请参见**Notebook存储挂载方式说明**。
认证信息	凭据	当“存储类型”选择“对象存储 OBS - 并行文件系统”或“对象存储 OBS - 对象桶”时，需要设置此参数。 选择已有的凭据或单击右侧的“创建凭据”，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。

– **更多配置**

参数名称	说明
SSH远程开发	开启后，可在本地IDE通过VS Code远程登录Notebook实例。该功能存在使用约束与连接上限，使用前请查看**SSH使用约束**，避免Notebook实例响应超时或中断，影响正常使用。 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。 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。 您可以选择已有密钥对，也可以单击密钥对右侧的“创建密钥对”，跳转到**数据加密控制台**，在“密钥对管理”页面的“账号密钥对”或者“私有密钥对”页签，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。
配置网络	开启后，可配置VPC相关信息使该Notebook实例接入网络。 开启此功能后，实例能够挂载在用户的VPC下，实现多网络平面接入。 使用此功能前，您需要参考**创建IAM用户并授权使用ModelArts**配置VPC细粒度访问授权。如果您有“VPC Administrator”权限，则无需配置。 ● 虚拟私有云：从下拉菜单中选择已有的VPC，或根据需求单击“新建虚拟私有云”，在“创建虚拟私有云”面板配置相关信息，单击“确定”，在下拉菜单中选择新建的VPC。 ● 子网：选择VPC后此处会显示默认的子网。您也可以从下拉菜单中选择已有的子网，或根据需求单击“新建子网”，在“新建子网”面板配置相关信息，单击“确定”，在下拉菜单中选择新建的子网。 ● 安全组：选择已有的安全组，或单击“新建安全组”，在“使用预设规则创建安全组”面板配置相关信息，单击“确定”，在下拉菜单中选择新建的安全组。

参数名称	说明
指定运行用户	在启动Notebook实例时，ModelArts支持以下两种运行用户配置。不同资源配置支持的运行用户配置可能不同，请以实际环境为准。 仅新网络模式下专属资源池支持root启动Notebook实例，详情请参见**使用root启动Notebook实例**。关于如何基于新网络创建专属资源池，请参见**创建专属资源池**。 • **ma-user/ma-group**：Notebook公共镜像默认的非特权用户配置（安全模式）。使用时需满足以下条件： – 用户身份：ma-user（UID: 1000） – 用户组：ma-group（GID: 100） **注意**：如果使用自定义镜像，需提前在镜像中预置上述用户和用户组，否则容器启动时可能因权限不足导致服务异常。关于添加指定的用户和用户组的具体操作，请参见**Dockerfile文件（基础镜像为非ModelArts提供）**。 • **root/root**：以最高权限运行Notebook，适用于需要访问系统级资源的场景，但需注意潜在的安全风险。选择root/root时，系统将强制绑定以下用户和用户组。 – 用户身份：root（UID: 0） – 用户组：root（GID: 0） **注意**：不允许修改root的UID/GID或所属用户组，否则可能引发容器权限冲突或安全漏洞。

4. 参数填写完成后，页面右侧会显示配置概要信息，页面下方会显示配置费用，请确认无误后，单击“立即创建”。

   实际扣费请以账单为准。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。

   如果创建或使用Notebook过程中出现问题，您可以参考**故障排除**和**常见问题**进行解决。

   在Notebook页面，您可以进行以下操作。

   表 1-2 Notebook 操作说明

操作	说明	相关文档
查看Notebook实例详情	查看Notebook实例的详细信息，包括基础信息、存储信息、事件信息等。	**查看Notebook实例详情**
查找Notebook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。	**查找Notebook实例**
接入环境	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。	**打开Notebook实例**

操作	说明	相关文档
保存镜像	通过预置镜像创建Notebook实例，在基础镜像上安装对应的自定义软件和依赖，将运行的实例环境以容器镜像的方式保存下来。	保存Notebook实例
停止/启动/删除Notebook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。 针对不再使用的Notebook实例，可以删除以释放资源。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。	启动/停止/删除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。	变更镜像
变更实例规格	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。	变更Notebook实例运行规格
节点亲和性调度	通过节点亲和性，可以确保Pod被调度到满足特定条件的节点上，从而实现更细粒度的资源管理和优化。	设置节点亲和性调度
设置自动停止	开启后，可设置Notebook实例自动停止方式及对应时长，超出您预设的时长，将自动停止运行 (可能存在2-5分钟的延迟，此过程正常计费)。	启动/停止/删除Notebook实例
编辑标签	标签可用于标识、分类、搜索和管理Notebook实例。	编辑标签

查看 Notebook 实例详情

1. 登录ModelArts管理控制台，在左侧导航栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 在Notebook列表，单击实例名称，进入实例详情页。

图 1-7 Notebook 实例详情

表 1-3 Notebook 实例详情说明

页签名称	说明	相关文档
基本信息	展示Notebook实例的基础信息（名称、ID、标签、描述、自动停止等）、环境信息（镜像、JupyterLab版本、虚拟私有云、子网、安全组等）、资源信息（资源池、实例规格、节点亲和性调度、存储类型、存储容量等）、SSH远程开发信息。	• 配置Notebook SSH远程连接 • 设置节点亲和性调度
存储配置	展示Notebook实例的动态存储和扩展存储信息，您也可以按需进行添加、卸载等操作。 • 动态存储：支持在实例运行期间动态挂载存储，无需停止实例，实例启动后自动重新挂载，数据持久保留。适合需要运行时灵活挂载或临时扩展存储的场景。 • 扩展存储：基于云原生持久化存储方案，提供稳定可靠的存储管理，但挂载或变更存储配置需停止实例。适用于规划好的、长期使用的数据卷。	• Notebook存储类型说明 • Notebook存储挂载方式说明 • Notebook动态扩充云硬盘EVS容量
数据服务	仅专属资源池的Notebook实例显示该页签。展示支持的数据服务、资源名称及状态。 单击“状态同步”，可同步数据服务的状态。	-
事件	展示Notebook实例的事件信息，包括事件名称、事件级别、事件信息、首次发生时间等。	查看Notebook实例事件
监控	展示Notebook实例的监控指标信息，包括CPU使用率、物理内存使用率、GPU显卡使用率、GPU显存使用率、磁盘写入等。	查看Notebook实例资源使用情况

Notebook实例详情页右上角的操作与Notebook页面操作类似，您可以参考**表1-2**，了解操作的说明及指导文档。

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**。
- 通过Code Server在线使用Notebook实例，具体参见**使用Code Server进行代码开发**。
- 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实例默认是以ma-user启动。用户进入实例后，工作目录默认是/home/ma-user/work。

图 1-8 工作目录示例

专属池的部分Notebook实例以root用户身份启动。具体说明如下：

- 当以root用户身份登录终端时，系统会自动执行source /home/ma-user/.bashrc命令，以同步ma-user用户的环境变量。如果需要禁用此功能，可以通过在自定义镜像中设置环境变量export DISABLE_MA_USER_BASHRC=true，即可阻止加载/home/ma-user/.bashrc文件。
- 以root用户启动的实例，仅支持使用root用户进行SSH远程连接。在Notebook实例详情页的“基本信息”页签可以查看SSH远程开发地址。

图 1-9 使用 root 用户进行 SSH 远程连接

Notebook 容器挂载目录说明

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1-4**所示。

表 1-4 Notebook 挂载目录说明

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	客户并行文件系统的挂载目录。
/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。

常见问题

- **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**Notebook动态扩充云硬盘EVS容量**。
- **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**Notebook存储挂载方式说明**。
- **使用JupyterLab 4版本时启动出现问题，如何切换回JupyterLab 3版本?**

  在Notebook实例列表的“操作”列，单击目标实例对应的“启动”，在弹出的对话框中，选择JupyterLab 3版本，单击“确定”后即可启动JupyterLab 3版本。
- **可以在一个项目中同时使用JupyterLab 3和JupyterLab 4版本吗?**

  不建议在同一个项目中同时使用两个版本。每个JupyterLab实例独立运行，因此需要为每个版本分别创建实例。如果您希望尝试不同版本，可以在不同的容器或环境中分别启动它们，但请注意以下几点：
  - 不同版本的配置文件和数据路径可能不同，需确保数据和配置的独立性。
  - 同时运行多个版本可能会导致端口冲突或其他资源竞争问题。
- **已有Notebook实例为JupyterLab 3版本，怎么升级至JupyterLab 4版本?**

  您可以任选以下方式升级至JupyterLab 4版本。具体操作，请参见**升级JupyterLab版本**。
  - 方式一：停止Notebook实例并升级JupyterLab版本
  - 方式二：保存Notebook实例并升级JupyterLab版本
- **Notebook是否支持使用gdb工具?**

Notebook目前不支持gdb工具的使用。gdb工具的运行依赖开启特权容器（privileged container）的Docker，而开发环境的容器出于安全考虑，无法开启特权容器，因此不支持在Notebook中使用gdb工具。

1.3 通过 Code Server 在线使用 Notebook 实例

1.3.1 使用 Code Server 进行代码开发

Code Server是一个开源项目，它将Visual Studio Code完整运行在远程服务器上，并通过浏览器提供访问能力。主要支持以下能力：

- **完整VS Code体验：**支持扩展市场、调试器、终端、Git等几乎所有原生功能。
- **跨平台访问：**只需浏览器即可在任何设备上编写代码。
- **统一开发环境：**代码、配置和依赖项均部署在服务器上，确保环境一致性。
- **私有部署**：数据完全由用户掌控。

本文介绍如何在Notebook实例使用Code Server进行代码开发。

注意事项

- 在使用Code Server AI助手前，请仔细阅读**Code Server AI助手风险说明及安全建议**。
- 企业用户使用Code Server时，需满足一定的网络要求，详情请参见**企业用户使用Code Server网络要求**。

约束限制

您创建的Notebook实例需满足如下条件：

- 已配置Code Server。
  - 未创建Notebook实例：**创建Notebook实例**时，“环境配置”中需选中Code Server。

    图 1-10 选中 Code Server

  - 已创建Notebook实例：在Notebook实例详情页的“基本信息”页签，可查看是否配置Code Server。

图 1-11 Notebook 实例详情页

- 部分容器镜像版本的操作系统暂不支持Code Server。具体限制如下：
  查看容器镜像的操作系统：假设预置镜像的地址为pytorch_cuda: pytorch_2.7.0-cuda_12.8-py_3.11.10-ubuntu_22.04-x86_64-20251215163925-4e5422a，其中“ubuntu_22.04”为镜像的操作系统。

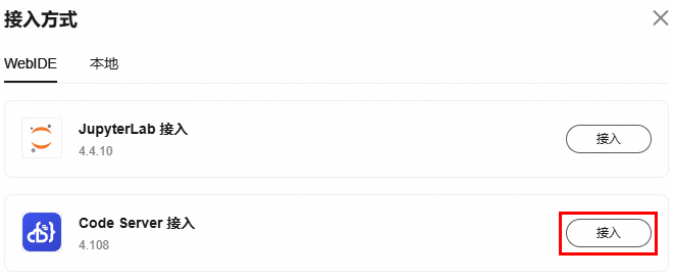
  表 1-5 操作系统支持情况

容器镜像的操作系统	架构	是否支持Code Server
ubuntu_18.04	x86	不支持
ubuntu_20.04	x86	支持
ubuntu_22.04	x86	支持
hce_2.0/hce2.0	ARM	支持
euler_2.8.3/euler2.8.3	ARM	支持
euler_2.10/euler2.10	ARM	支持

启动 Code Server

1. 登录**ModelArts管理控制台**，在左侧导航栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 在“状态”为“运行中”的Notebook实例的操作列，单击“接入环境”，在“接入方式”对话框的“WebIDE”页签，单击“Code Server 接入”右侧的“接入”。

图 1-12 Code Server 接入



接入成功后，可以看到其界面布局、操作逻辑和功能都和桌面版VS Code高度一致。

Code Server 功能介绍

下文介绍Code Server的界面布局及功能、核心功能特性、插件扩展市场。

界面布局及功能

Code Server的界面分为以下几个主要区域：

表 1-6 Code Server 区域说明

区域	说明
顶部标题栏	显示当前打开的文件或文件夹名称，以及窗口控制按钮。
左侧活动栏	包含多个图标按钮，用于切换不同的视图，支持如下功能： • 文件资源管理器（Explorer）：浏览和管理项目文件。 • 搜索（Search）：在项目中搜索文件内容和符号。 • 源代码管理（Source Control）：集成Git操作，支持提交、分支管理、查看差异等。 • 调试（Run and Debug）：配置和运行调试任务。 • 扩展（Extensions）：管理插件安装和配置。 • 测试（Testing）：配置和运行测试用例。 • GitLens：强大的Git操作插件，支持代码历史、差异查看等。 • Cline：AI编程助手，基于大语言模型的智能开发插件。
左侧边栏	显示活动栏选中视图的具体内容，如文件树、搜索结果、插件列表等。

区域	说明
编辑器区域	代码编辑的主区域，支持多标签页同时打开、代码高亮、语法提示等功能。
底部状态栏	显示当前文件的语言模式、行号、列号，以及Git分支、错误警告等信息。
集成终端	位于底部，支持创建多个终端实例，可以同时运行多个命令。

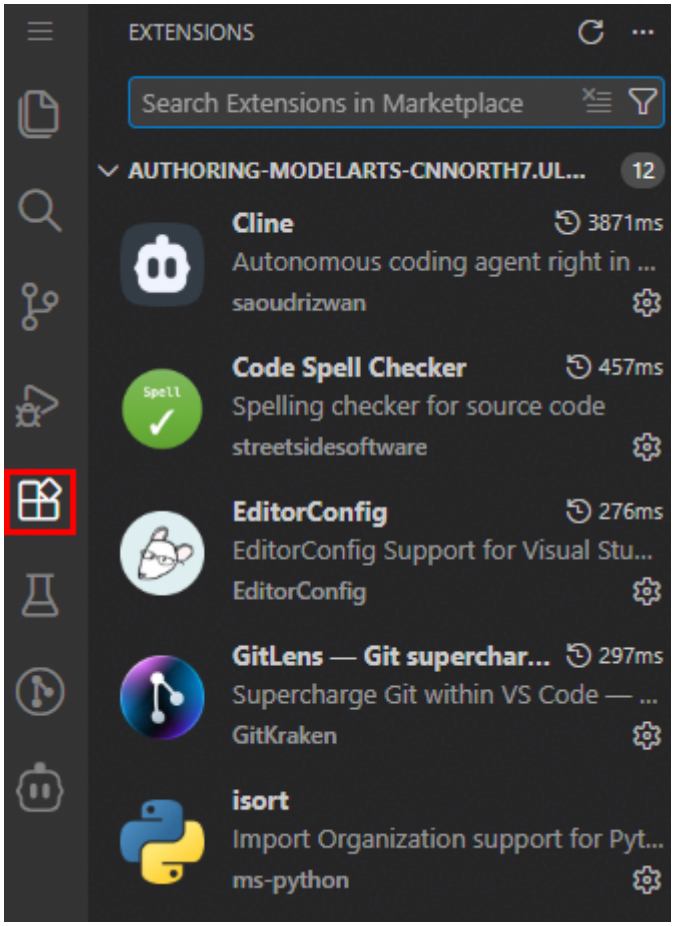
核心功能特性

- **完整的扩展生态**：Code Server默认使用Open-VSX扩展市场，您可以通过左侧的  图标搜索并安装绝大多数流行的开源扩展，例如Python、GitLens、Prettier等。
- **内置终端与Git**：集成的终端是其一大亮点，您可以在浏览器中完成从编码到构建、测试、部署的全部流程，体验如同本地开发。同时，源代码管理视图集成了Git操作，方便您进行代码的提交、分支管理和查看差异。
- **便捷的端口转发**：当您开发Web应用或需要访问服务器上的特定服务端口时，Code Server内置了端口转发功能。您可以在端口 (PORTS)面板中轻松地将远程端口映射到本地，方便调试。
- **设置同步与持久化**：Code Server的配置、主题、快捷键和已安装的扩展都保存在服务器的配置目录中。这意味着您无论在哪台设备上登录，都能获得完全一致的开发环境。

插件扩展市场

Code Server的插件市场提供了丰富的开发工具，Notebook实例预装了一些常用插件，您也可以按需下载其他插件。

图 1-13 插件市场



Notebook实例预装的常用插件及说明如下：

表 1-7 预装插件说明

插件名称	版本	说明
Python	2025.6.1	Python语言支持，提供代码补全、语法检查等功能。
Python Debugger	2025.18.0	Python调试器，支持断点调试、变量查看等。
Python Environments	1.10.0	自动检测、管理并激活不同的Python解释器和虚拟环境。
Code Spell Checker	4.1.0	实时检测源代码、注释和文档中的英文拼写错误，遵循驼峰命名法（camelCase）和蛇形命名法（snake_case）。

插件名称	版本	说明
EditorConfig for VS Code	0.16.6	团队定义和维护统一的编码样式。
GitLens	17.1.1	强大的Git操作插件，支持代码历史、差异查看等。
isort	2023.11.0-dev	对代码中的import语句进行自动排序、分组和格式化。
Jupyter	2024.8.1	Jupyter Notebook支持。
Jupyter Cell Tags	0.1.9	Jupyter单元格标签管理。
Jupyter Keymap	1.1.2	Jupyter键盘快捷键。
Jupyter Notebook Renderers	1.0.19	Jupyter Notebook渲染器。
Jupyter Slide Show	0.1.6	Jupyter幻灯片演示。
Pyright	1.1.400	标准的Python静态类型检查器，旨在提升大型Python代码库的类型安全性、开发效率和代码质量。
Cline	3.75.0	AI编程助手，基于大语言模型的智能开发插件。

您也可以在插件市场下载所需的其他插件。以Markdown相关插件为例，在搜索栏中输入Markdown，选中需要下载的插件，单击“Install”。

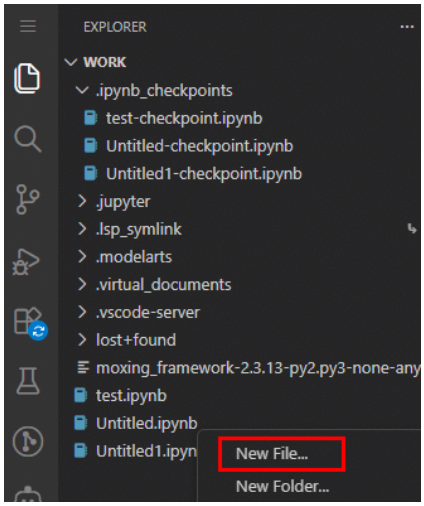
Python 代码调试

Notebook已在Code Server中预装了Python相关的依赖插件，您可以直接进行Python代码调试。

代码准备

1. 在工作区右键新建Python文件，文件名称后缀为.py，例如test.py。

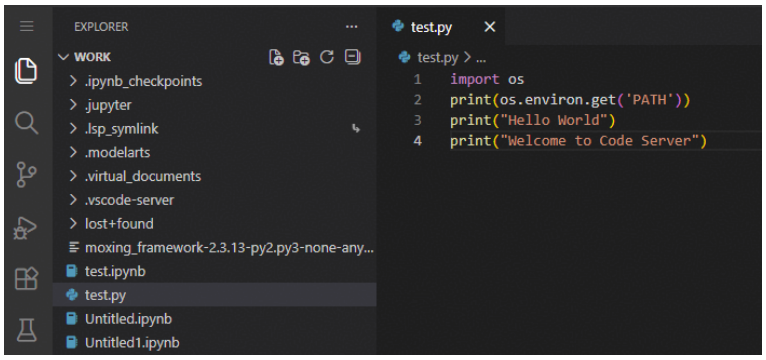
图 1-14 新建 Python 文件



2. 将如下Python代码拷贝至Python文件中。

```
import os
print(os.environ.get('PATH'))
print("Hello World")
print("Welcome to Code Server")
```

图 1-15 拷贝 Python 代码



运行 Python 文件

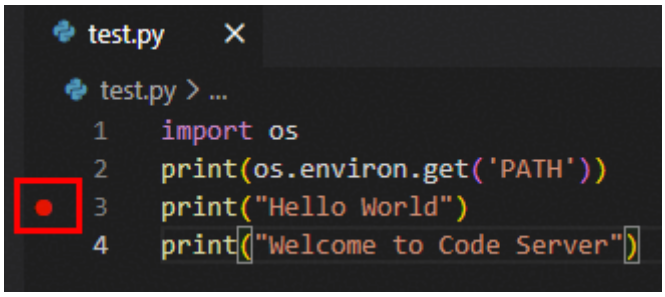
在Python文件右上角，单击  图标，下方“TERMINAL”页签会展示运行的结果。

图 1-16 TERMINAL 运行结果

调试 Python 文件

1. 在Python文件中添加断点。在文件行号左侧，单击鼠标左键添加或删除断点。

图 1-17 添加断点



2. 在左侧活动栏，单击  图标，单击“Run and Debug”，选择“Python Debugger”，在“Select a debug configuration”弹窗，选择“Python File Debug the currently active Python file”，进入调试模式。
3. 代码运行至第一个断点处停止，可以在左侧变量区查看变量，也可以在上方操作栏执行重试、继续等操作。

项目开发

Nanochat是AI领域的Andrej Karpathy发布的一个开源项目，是一套极简、完整且低成本的训练框架，可以用不到100美元的成本，在几个小时内从零训练出一个属于自己的、能对话的迷你版ChatGPT。

本节以nanochat项目为例，介绍如何在Code Server中快速开发，包括文件上传、环境准备、项目运行等能力的演示。

演示环境配置

- 实例规格：8 vCPUs | 32 GB (modelarts.vm.cpu.8u)
- 演示镜像：pytorch_2_1:pytorch_2.1.0-cuda_12.1-py_3.9.11-ubuntu_22.04-x86_64-20240313165241-219655b

项目加载

您可以任选以下方式加载项目。

- 方式一：Git命令下载
  a. 打开Terminal。

  图 1-18 New Terminal

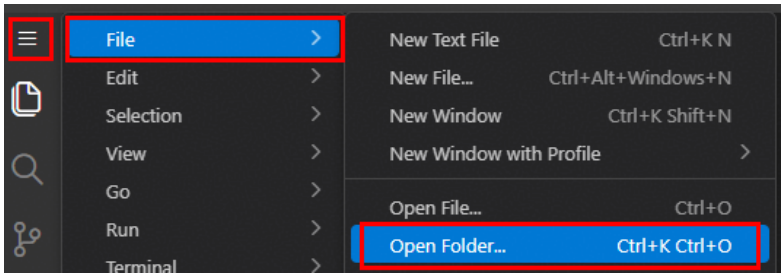
  b. 使用命令下载。

```
git clone https://github.com/karpathy/nanochat.git
cd nanochat
```

- 方式二：拖拽上传

a. 进入工作目录，打开文件夹。

图 1-19 Open Folder



b. 将本地文件夹拖到Code Server工作目录下，自动上传文件夹下内容。

nanochat 环境准备

1. 安装uv。

   uv是一个用Rust编写的极速Python包和项目管理工具，旨在成为Python生态系统中一个统一、高效的工具链。nanochat项目中使用uv来管理Python环境，所以需先安装uv。

   ```
   curl -LsSf https://astral.sh/uv/install.sh | sh
   source $HOME/.local/bin/env
   ```

2. 安装nanochat环境。

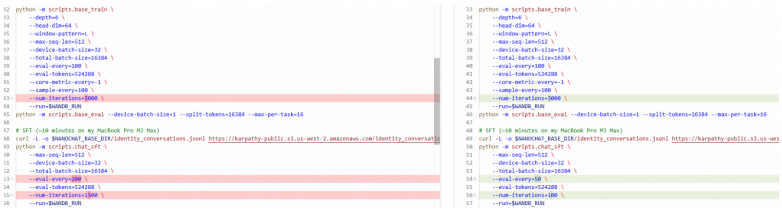
   本教程基于CPU环境演示，所以执行如下命令安装nanochat。

   ```
   export UV_INDEX_URL="https://pypi.tuna.tsinghua.edu.cn/simple
   uv sync --extra cpu
   ```

   注意：如果无法访问HuggingFace服务，需将dataset.py代码中硬编码数据地址切换为国内源。

在 CPU 环境执行模型训练

本文档主要为演示Code Server功能，非大模型的训练调试，所以将训练脚本中的迭代步数减少以缩短训练耗时。对模型的训练结果也未做调参处理，如果想获得更好的训练效果，可调整参数或在GPU环境运行本项目。



启动训练脚本。

```
# 启动运行脚本
bash runs/runcpu.sh
```

之后TERMINAL上即可显示大模型的主要训练流程日志，主要包括如下步骤：

1. 训练数据下载：数据集默认下载至 /home/ma-user/.cache/nanochat路径下。
2. Tokenizer训练。


```
max_chars: 2,000,000,000
doc_cap: 10,000
vocab_size: 32,768
Training time: 49.07s
Saved tokenizer encoding to /home/ma-user/.cache/nanochat/tokenizer/tokenizer.pkl
Saved token_bytes to /home/ma-user/.cache/nanochat/tokenizer/token_bytes.pt
```

3. 模型初始化。

```
Model config:
{
  "sequence_len": 512,
  "vocab_size": 32768,
  "n_layer": 6,
  "n_head": 6,
  "n_kv_head": 6,
  "n_embd": 384,
  "window_pattern": "L"
}
```

4. 自监督训练过程：可以观察到Loss逐步降低，模型从随机开始，逐渐学会预测下一个Token。在第100步时，模型输出重复的无意义结果；第3000步，模型输出具备一定的逻辑。

```
Step 00000 | Validation bpb: 3.194289
step 00000/03000 (0.00%) | loss: 10.396702 | lrm: 0.03 | dt: 39486.18ms | tok/sec: 414 | bf16_mfu: 0.00 | epoch: 1 pq: 0 rg: 1 | total time: 0.00m
step 00001/03000 (0.03%) | loss: 10.396128 | lrm: 0.05 | dt: 7665.87ms | tok/sec: 2,137 | bf16_mfu: 0.00 | epoch: 1 pq: 0 rg: 1 | total time: 0.00m
step 00002/03000 (0.07%) | loss: 10.394956 | lrm: 0.07 | dt: 7383.87ms | tok/sec: 2,218 | bf16_mfu: 0.00 | epoch: 1 pq: 0 rg: 1 | total time: 0.00m
step 00003/03000 (0.10%) | loss: 10.392873 | lrm: 0.10 | dt: 7422.76ms | tok/sec: 2,207 | bf16_mfu: 0.00 | epoch: 1 pq: 0 rg: 1 | total time: 0.00m
step 00004/03000 (0.13%) | loss: 10.390493 | lrm: 0.12 | dt: 7387.94ms | tok/sec: 2,217 | bf16_mfu: 0.00 | epoch: 1 pq: 0 rg: 1 | total time: 0.00m
step 00005/03000 (0.17%) | loss: 10.387613 | lrm: 0.15 | dt: 7112.41ms | tok/sec: 2,303 | bf16_mfu: 0.00 | epoch: 1 pq: 0 rg: 1 | total time: 0.00m
...
step 02995/03000 (99.83%) | loss: 5.779074 | lrm: 0.05 | dt: 6889.23ms | tok/sec: 2,378 | bf16_mfu: 0.00 | epoch: 1 pq: 1 rg: 46 | total time: 352.97m | eta: 0.6m
step 02996/03000 (99.87%) | loss: 5.778528 | lrm: 0.05 | dt: 6904.46ms | tok/sec: 2,372 | bf16_mfu: 0.00 | epoch: 1 pq: 1 rg: 46 | total time: 353.09m | eta: 0.5m
step 02997/03000 (99.90%) | loss: 5.777136 | lrm: 0.05 | dt: 6885.48ms | tok/sec: 2,379 | bf16_mfu: 0.00 | epoch: 1 pq: 1 rg: 46 | total time: 353.20m | eta: 0.4m
step 02998/03000 (99.93%) | loss: 5.786403 | lrm: 0.05 | dt: 6911.86ms | tok/sec: 2,370 | bf16_mfu: 0.00 | epoch: 1 pq: 1 rg: 46 | total time: 353.32m | eta: 0.2m
step 02999/03000 (99.97%) | loss: 5.795797 | lrm: 0.05 | dt: 6898.11ms | tok/sec: 2,375 | bf16_mfu: 0.00 | epoch: 1 pq: 1 rg: 46 | total time: 353.43m | eta: 0.1m
...
Step 00100 | Validation bpb: 2.238087
<|bos|>The capital of France is to to the the of of, and the. of, and the. of
<|bos|>The chemical symbol of gold is. to, and the. of, and the. of, and the.
<|bos|>If yesterday was Friday, then tomorrow will be to to the the of of, and the. of, and the. of
<|bos|>The opposite of hot is. to, and the. of, and the. of, and the.
<|bos|>The planets of the solar system are: to is the to of the, and the. of, and the. of
<|bos|>My favorite color is the to of the, and the. of, and the. of, and
<|bos|>If 5*x + 3 = 13, then x is. to, 200.2, 200.2,
...
Step 03000 | Validation bpb: 1.782582
<|bos|>The capital of France is The largest city of France, which, in, is, is, in,
<|bos|>The chemical symbol of gold is, in, the form of a symbol of the earth. surface, the symbol
<|bos|>If yesterday was Friday, then tomorrow will be the first time of the year. the first time of the year was the first
<|bos|>The opposite of hot is, the hot of the cold is, the cold is, the cold is,
<|bos|>The planets of the solar system are: to is the most of the solar of the solar of the solar of the solar
<|bos|>My favorite color is, I'm not sure what I'm doing., but I'm not sure
<|bos|>If 5*x + 3 = 13, then x is, 1 = 1 = 1 = 1 = 1 =
```

5. SFT过程：可以看到SFT过程出现Loss异常，说明CPU上训练SFT需要进行更专业的调参。

```
...
1206   Step 00000 | Validation bpb: 1.9417
1207   step 00001 (2.00%) | loss: 3.332982 | lrm: 0.08 | dt: 36312.24ms | tok/sec: 451 | mfu: 0.00 | epoch: 1 | total time: 0.00m
1208   step 00002 (3.00%) | loss: 4.404898 | lrm: 0.12 | dt: 19955.22ms | tok/sec: 821 | mfu: 0.00 | epoch: 1 | total time: 0.00m
1209   step 00003 (4.00%) | loss: 4.884014 | lrm: 0.16 | dt: 16627.26ms | tok/sec: 985 | mfu: 0.00 | epoch: 1 | total time: 0.00m
1210   step 00004 (5.00%) | loss: nan | lrm: 0.20 | dt: 9377.18ms | tok/sec: 1,747 | mfu: 0.00 | epoch: 1 | total time: 0.00m
1211   step 00005 (6.00%) | loss: nan | lrm: 0.24 | dt: 9815.09ms | tok/sec: 1,669 | mfu: 0.00 | epoch: 1 | total time: 0.00m
```

6. 模型对话。

   执行以下脚本 ，可以获得模型的对话输出。

```
python -m scripts.chat_cli -p "What is the capital of France?"
```

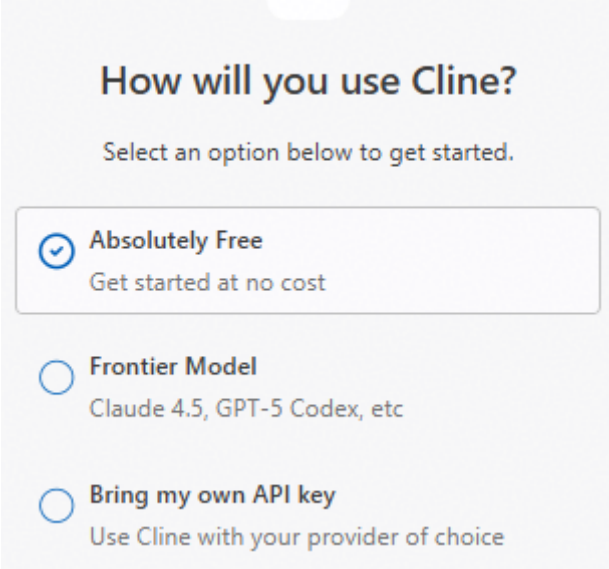
AI 助手

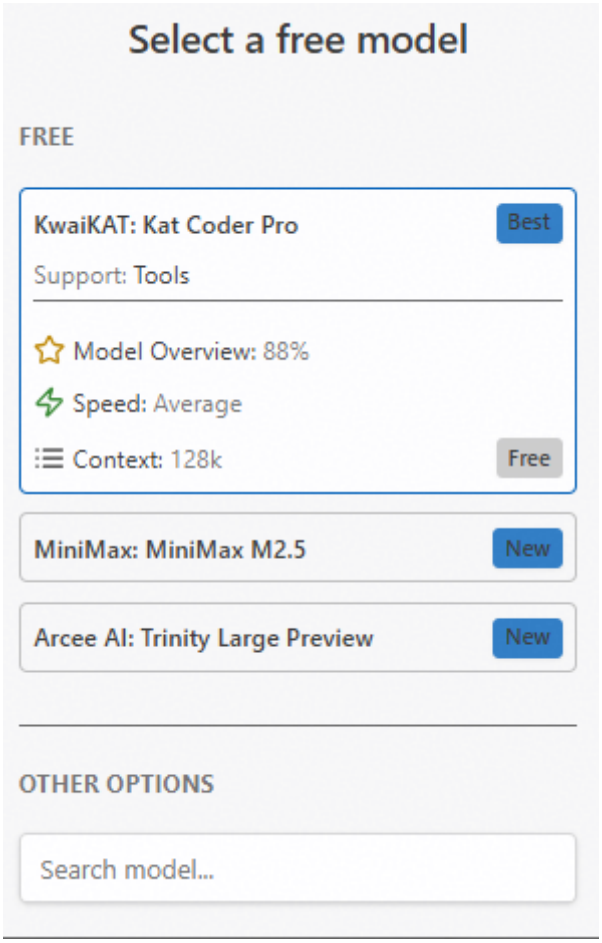
Cline是一款基于大语言模型的Code Server插件，能够理解自然语言指令，自动完成代码生成、文件编辑、终端命令执行等复杂开发任务。您可以借助Cline作为AI编程助手，自动化完成各种项目的环境配置、依赖安装及使用，大幅提升开发效率。关于AI助手的风险说明，请参见**Code Server AI助手风险说明及安全建议**。

在 Cline 配置模型服务信息

- 免费模型

  Cline提供免费的模型调用，选择Absolutely Free，之后选择一个免费的模型，登录Cline账号使用。





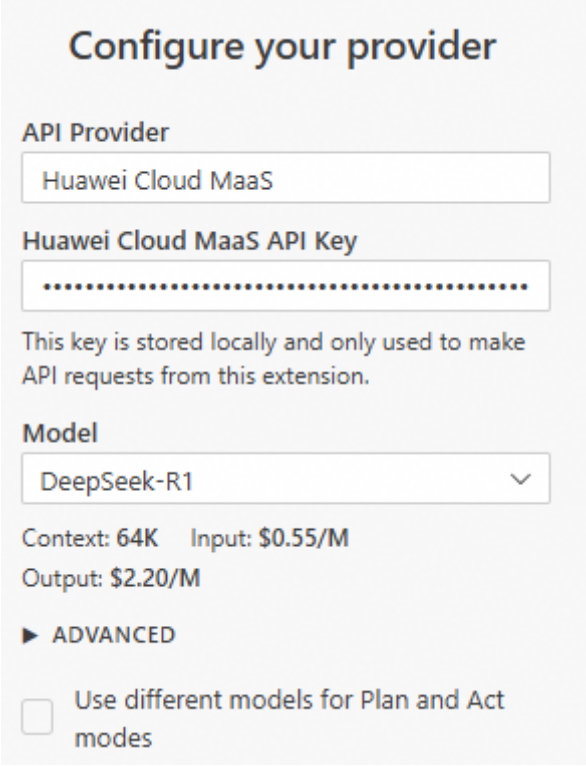
- 第三方模型厂商

  Cline支持多种模型接入方式，覆盖面非常广泛，包括Huawei Cloud MaaS、Claude Code、DeepSeek、OpenAI等。

  以Huawei Cloud MaaS为例：

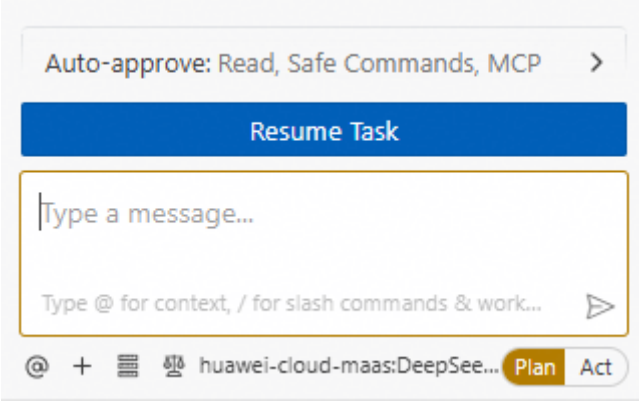
  a. 在API Provider中搜索Huawei Cloud MaaS。
  b. API Key输入在华为云已创建的API Key。关于如何创建API Key，请参见**创建API Key**。

图 1-20 配置模型信息



配置完成后，在对话框即可与大模型对话。

图 1-21 与大模型对话

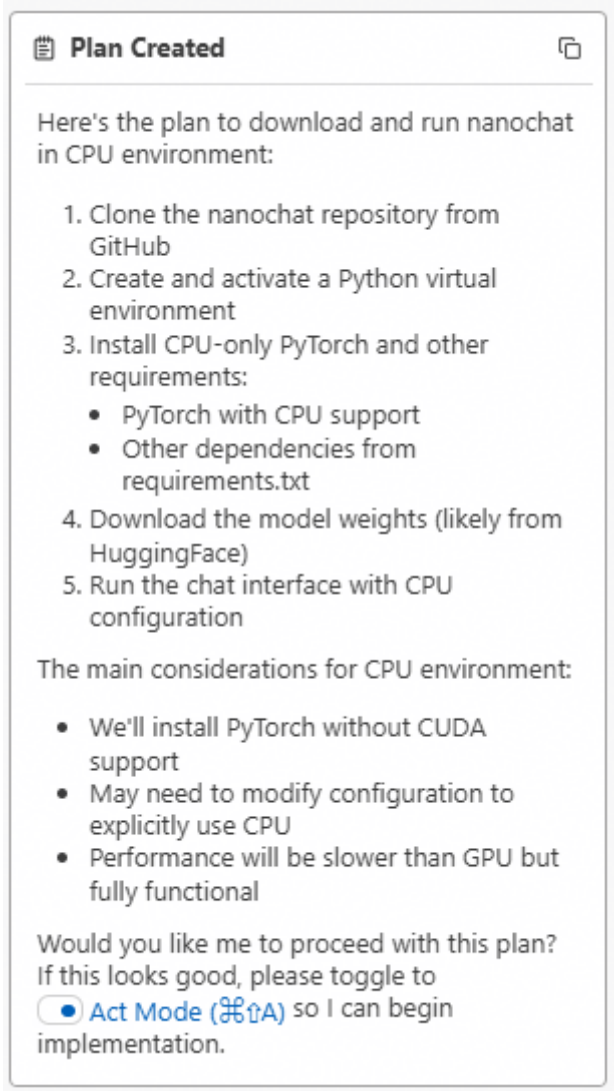


项目实践

本节以项目开发中nanochat项目为例，介绍如何使用AI助手Cline，提升开发效率。

1. 使用Plan模型设定计划。
   在Cline对话框中输入问题，Cline会制定一个执行计划。

图 1-22 制定计划



2. 使用Act模式执行计划。单击“Act”切换为Act模式，Cline会帮助用户执行对应的计划。

   Cline执行需要一定的操作权限，可以通过对话框上方 > 图标设置。

图 1-23 Act 模型

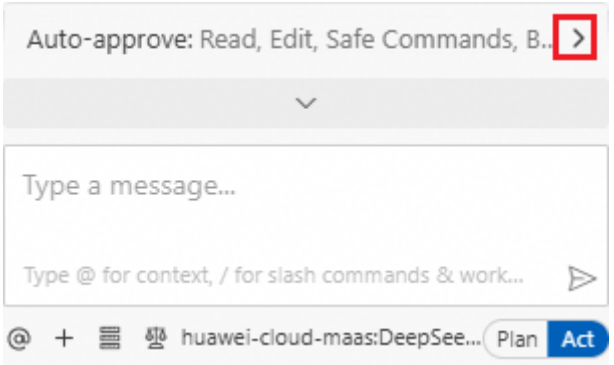
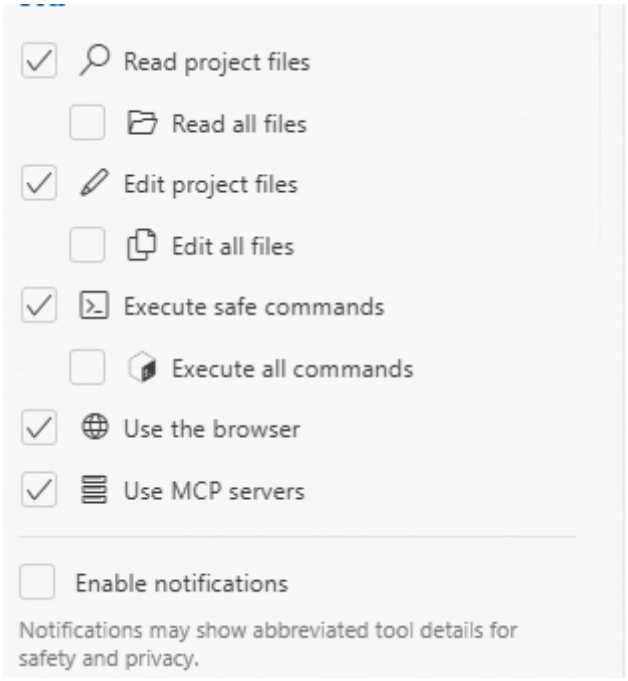
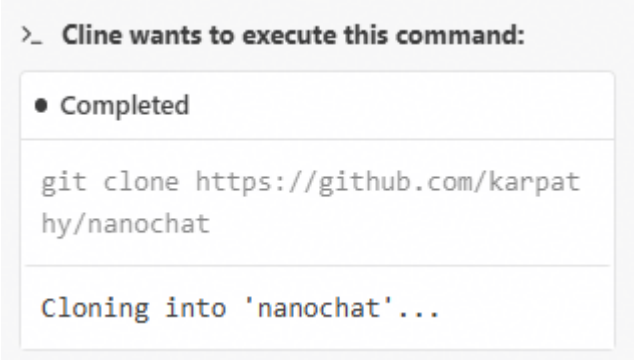


图 1-24 设置操作权限

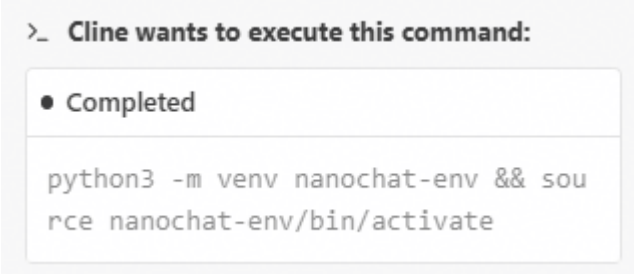


Cline根据Plan指定的计划，自动执行如下流程：

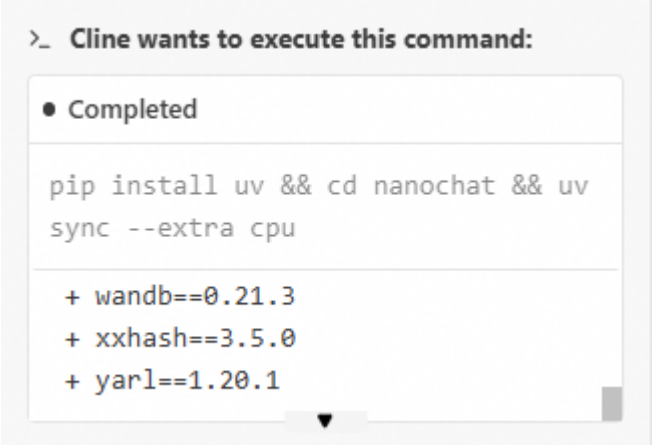
a. 下载项目。



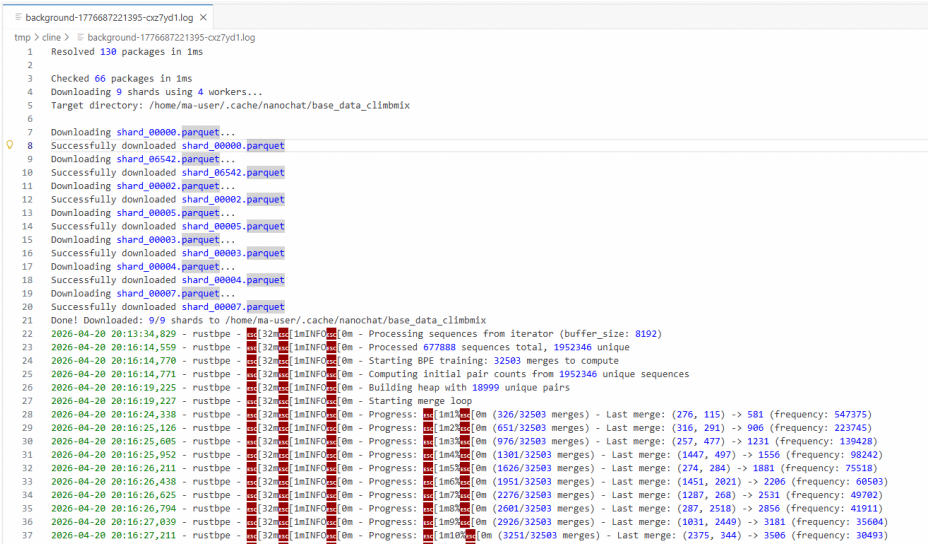
b. nanochat环境初始化。



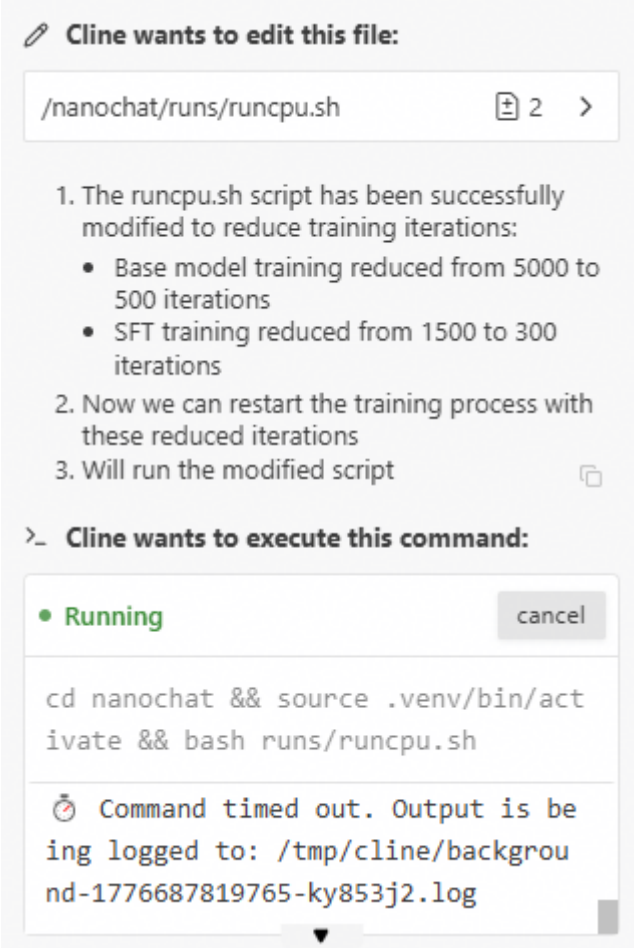
c. 运行nanochat。



d. 打开日志文件，可以看到脚本在正常运行。



e. 修改训练步数后重新训练。



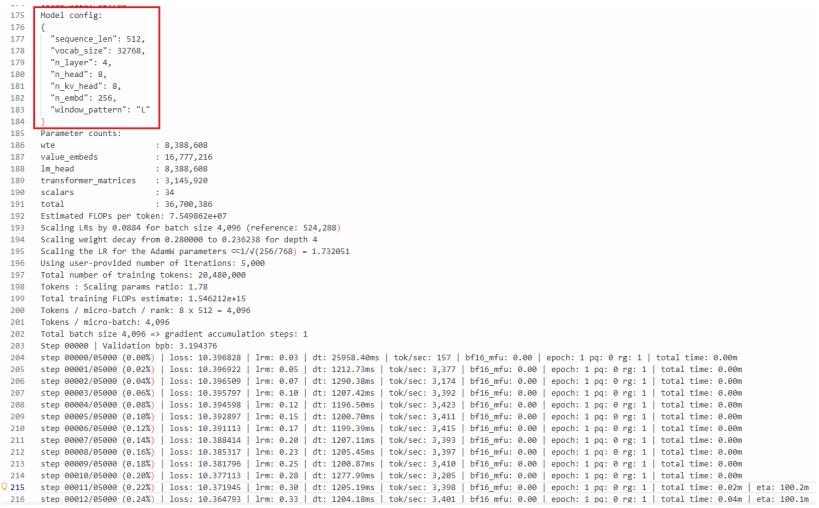
可以看到Cline停止了刚才的训练脚本，自动修改了步数后再次触发训练脚本。

f. 问题定位。

i. 脚本运行一段时间后异常停止，使用Cline进行问题定位，识别到为OOM错误，使用Cline解决后重新运行。

ii. 让Cline修改脚本，减小模型大小，再次训练。

iii. Cline自动修改模型大小后，成功运行项目。



1.3.2 Code Server AI 助手风险说明及安全建议

在使用Code Server AI助手时，存在一定的安全风险。本文以Cline为例，介绍可能出现的风险、影响及应对策略。

表 1-8 主要风险说明

风险类型	风险描述	影响分析	策略建议
敏感凭证明文存储	Cline的Code Server设置中，API Key（OpenAI/Anthropic/OpenRouter等）等凭据以明文形式保存在~/.cline/data/secrets.json中，任何能访问该文件的进程均可读取。	明文存储的API Key一旦被恶意扩展或攻击者获取，可直接调用大模型接口，产生未授权消费或数据泄露。	• 使用环境变量而非直接填入API Key。 • 定期轮换API Key。 • 镜像保存时，请及时清理~/.cline目录下的配置，避免敏感信息泄露。 • 限制Code Server扩展安装来源，避免恶意扩展读取配置。
工作区上下文泄露	Cline在执行任务时会读取项目文件、终端输出等上下文，可能将包含敏感信息（数据库连接串、内部API地址、业务逻辑）的代码片段发送至第三方LLM服务，并且有日志记录。	代码和业务敏感信息被传输至外部服务，存在数据泄露风险。部分LLM提供商可能使用交互数据用于模型训练。	• 审查.clineignore配置，排除敏感文件。 • 优先选择支持数据不用于训练的LLM提供商。 • 对核心敏感模块避免使用AI辅助编辑。

风险类型	风险描述	影响分析	策略建议
终端命令执行风险	Cline具备执行终端命令的能力，如果提示词被注入恶意指令，可能导致非预期的系统操作（如删除文件、安装恶意包等）。	攻击者通过prompt注入诱导Cline执行危险命令，造成文件损坏、系统被入侵或供应链攻击。	• 始终开启Cline的命令确认机制，逐条审核后再执行。 • 避免在不受信任的代码库中自动批准命令。 • 定期审查终端执行历史。
MCP工具调用风险	Cline支持Model Context Protocol，可连接外部工具服务器，扩展了攻击面。恶意的MCP服务器可能返回注入内容或执行未授权操作。	恶意MCP服务器可通过返回内容实施prompt注入，或利用工具权限执行越权操作。	• 仅连接可信的MCP服务器。 • 审查MCP工具的权限范围。 • 对MCP返回内容保持警惕，避免盲目信任。

表 1-9 攻击手法与策略建议

攻击手法	策略建议
供应链投毒	• 对于任何敏感操作（删文件、发邮件、转账），必须设置人工确认步骤。 • 遵循“最小权限原则”，只授予AI完成任务所必需的权限，切勿图省事一键授权。
间接提示注入攻击	• 对于任何敏感操作（删文件、发邮件、转账），必须设置人工确认步骤。 • 遵循“最小权限原则”，只授予AI完成任务所必需的权限，切勿图省事一键授权。

1.3.3 企业用户使用 Code Server 网络要求

为确保Code Server在企业内网环境中的稳定性，请满足以下要求：

- 所有依赖域名已放行，详情请参见**Code Server依赖域名**。
- 企业网络设备（防火墙、CDN、代理）支持WebSocket协议透传。如遇问题，请参考**Code Server网络故障排查**。

Code Server 依赖域名

为确保Code Server正常运行，需开放以下域名的网络访问权限。这些域名涉及插件市场、资源加载、远程开发等功能，禁用将影响相关功能的使用。

域名	用途
authoring-modelarts-cnnorth4.huaweicloud.com	Modelarts平台域名（示例为华北-北京四，请根据实际区域修改）。
vscode-remote+authoring-002dmodelarts-002dcnnorth4-002ehuaweicloud-002ecom.vscode-resource.vscode-cdn.net	VS Code远程开发资源加速CDN（Modelarts平台）。
openvsx.eclipsecontent.org	Open VSX内容分发/镜像域名，用于加速插件文件（.vsix）下载和静态资源分发。
img.shields.io	定期（或实时）从Visual Studio Marketplace的API拉取下载量数据，生成项目徽章（如下载量、版本号等），实时拉取数据并渲染为图片。
github.com	GitHub代码托管平台与原始文件访问入口。
raw.githubusercontent.com	GitHub原始文件直链服务（用于直接访问代码文件）。
badges.gitter.im	生成Gitter聊天室徽章图标。
open-vsx.org	VS Code插件市场与扩展注册表（开源替代方案）。
code.visualstudio.com	VS Code官方网站（文档、下载、社区等）。

Code Server 网络故障排查

在企业内网环境下，Code Server可能出现以下网络故障及解决方案：

故障一：WebSocket 连接异常

用户浏览器与Code Server服务端建立WebSocket连接时，相关插件未显示。

图 1-25 插件未显示

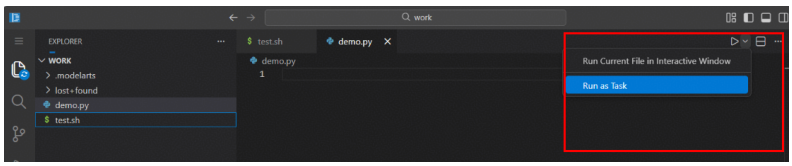
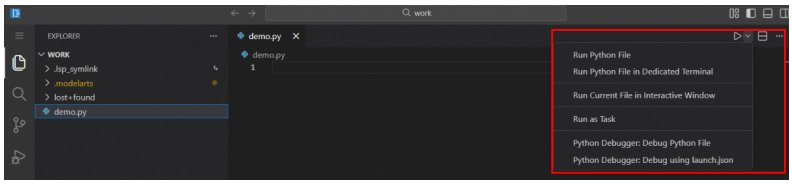


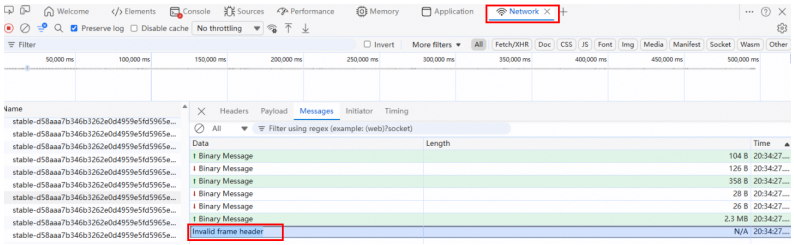
图 1-26 插件正常显示



打开浏览器调试窗口出现以下报错：

```
Invalid frame header
```

图 1-27 浏览器报错



故障二：创建或打开 Jupyter 文件报错

在Code Server中创建或打开Jupyter文件时出现以下报错。

- 创建Jupyter文件：
  ```
  Command 'Create: New Jupyter Notebook' resulted in an error
  ```

  图 1-28 创建 Jupyter 文件

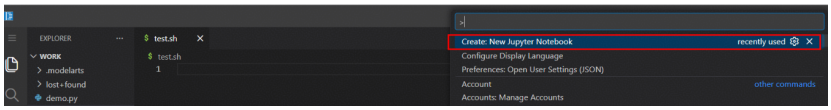
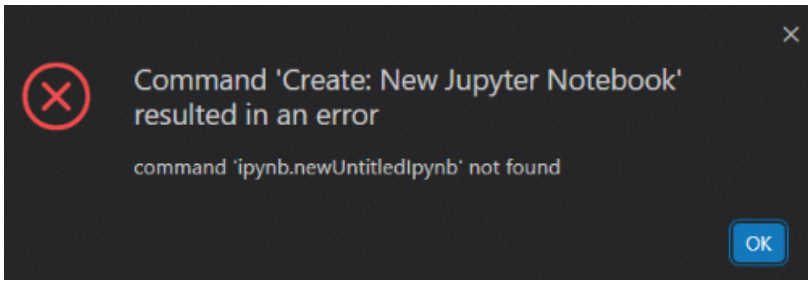
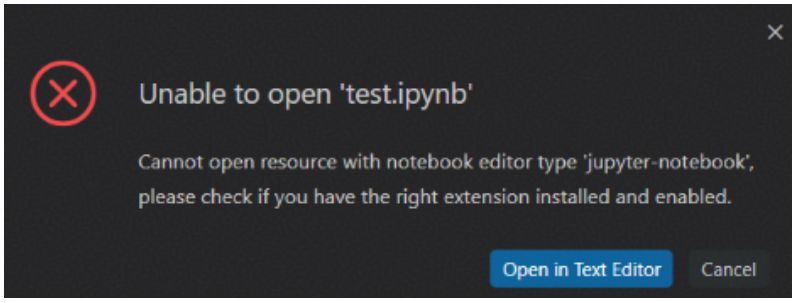
  

  图 1-29 创建 Jupyter 文件报错

- 打开Jupyter文件：
  ```
  Unable to open 'xxx.ipynb'
  ```

  图 1-30 打开 Jupyter 文件报错

故障可能原因及解决方案

上述故障的可能原因及解决方案如下。

可能原因	说明	解决方案
协议不匹配	WebSocket连接要求服务器严格遵循**RFC 6455协议**。错误提示表明： • 服务器返回的数据帧头部格式不符合WebSocket标准。 • 代理服务器/TLS层可能修改了数据包（常见于企业网络环境）。	检查企业代理服务器或TLS中间件（如Nginx、Squid）是否对WebSocket流量进行了非标准处理，需关闭相关修改功能。
企业防火墙/CDN干扰	企业网络中的防火墙或CDN可能对WebSocket流量进行拦截或优化，导致连接异常。	• **防火墙配置**： – 确认防火墙已放行WebSocket流量（协议类型ws或wss）。 – 检查是否拦截了Upgrade: websocket请求头。 • **CDN配置**： 1. 关闭CDN的“自动压缩”功能（如Gzip、Brotli），避免修改WebSocket数据帧。 2. 确保CDN支持WebSocket协议透传。 • **NAT超时调整**：将NAT超时时间调整为≥1800秒（30分钟），避免连接因超时中断。
代理网络异常	代理服务器可能未正确转发WebSocket请求，或存在中间人（MITM）加密导致协议冲突。	确保代理服务器支持WebSocket协议透传。

1.4 通过 JupyterLab 在线使用 Notebook 实例

1.4.1 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 1-31 使用 JupyterLab 在线开发调试代码

约束限制

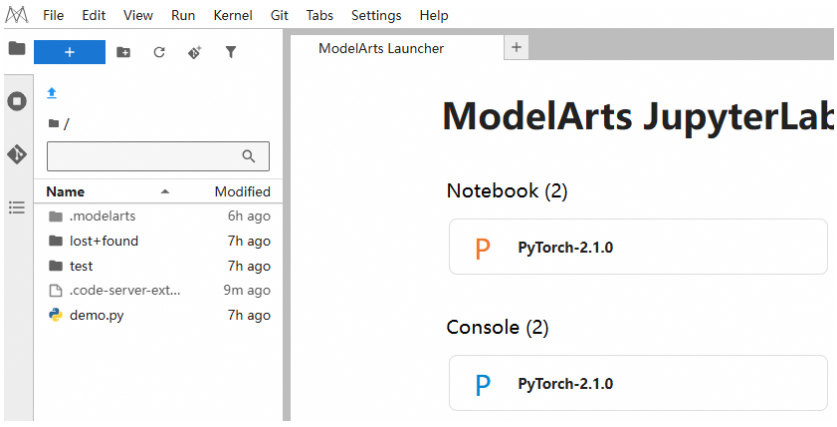
- 在JupyterLab中加载文件并无严格的固定大小限制，但其实际处理能力受系统资源及配置影响。在JupyterLab左侧浏览器打开较大文本文件（如超过100MB）可能导致内存不足、界面响应迟缓或内核中断等问题。为保证使用流畅性与稳定性，建议打开的单个文件大小不超过100MB。
- 系统长期平均每秒可处理最多40个JupyterLab请求，并允许在短时间内突发处理最多10个JupyterLab请求。通常情况下，如果在短时间内对同一资源池中的JupyterLab进行超过10次操作，将会触发流量控制。

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。具体操作，请参见**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“接入环境”，在“接入方式”对话框，单击JupyterLab接入右侧的“接入”，访问JupyterLab。
3. 进入JupyterLab页面后，自动打开ModelArts 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**。

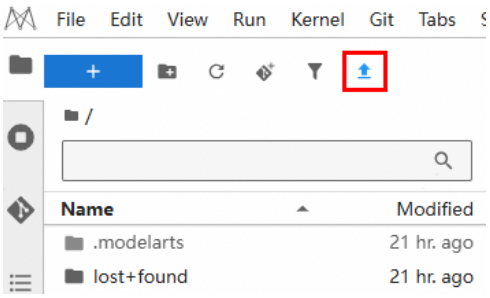
   图 1-32 JupyterLab 主页

   说明

   不同AI引擎的Notebook，打开后ModelArts Launcher页面呈现的Notebook和Console内核及版本均不同，**图1-32**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 1-33 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。

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**。

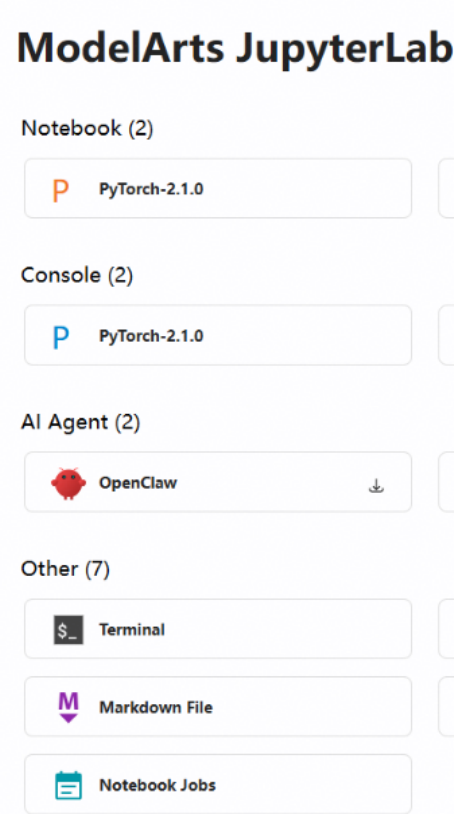
1.4.2 JupyterLab 常用功能介绍

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**，按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：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“接入环境”，在“接入方式”对话框，单击JupyterLab接入右侧的“接入”，访问JupyterLab。
3. 进入JupyterLab页面后，自动打开ModelArts 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 1-34 JupyterLab 主页



说明

不同AI引擎的Notebook，打开后ModelArts Launcher页面呈现的Notebook和Console内核及版本均不同，图1-34仅作为示例，请以实际控制台为准。

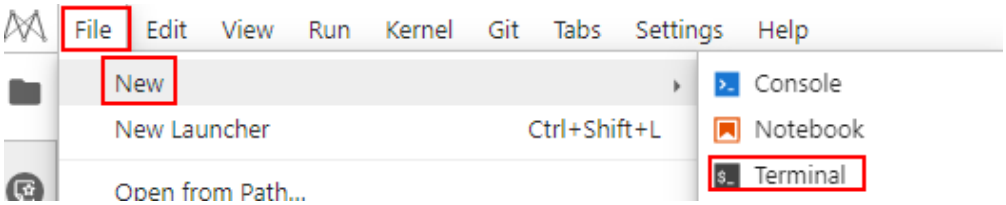
- Notebook：选择运行Notebook的一个内核，例如TensorFlow、Python。
- Console：可调出终端进行命令控制。
- AI Agent：支持一键部署OpenClaw等AI Agent开发助手，帮您自动完成环境配置、代码编写、问题排查，大幅提升开发效率（当前功能仅专属资源池支持）。
- Other：可编辑其他文件。

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“接入环境”，在“接入方式”对话框，单击JupyterLab接入右侧的“接入”，进入“JupyterLab”开发页面。具体操作，请参见**使用JupyterLab在线开发和调试代码**。
2. 选择“Files > New > Terminal”，进入到Terminal界面。

图 1-35 进入 Terminal 界面



在Terminal中，您可以看到操作相关提示。请您谨慎使用明文密码、硬编码AK/SK，并且不要直接在命令行中输入敏感信息。

图 1-36 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。

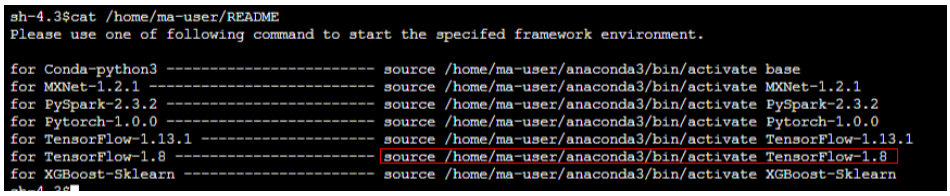
   cat /home/ma-user/README

   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 1-37 激活环境



在代码输入栏输入以下命令安装Shapely。

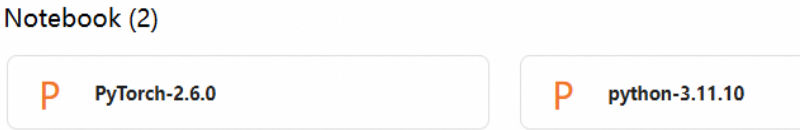
pip install Shapely

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。

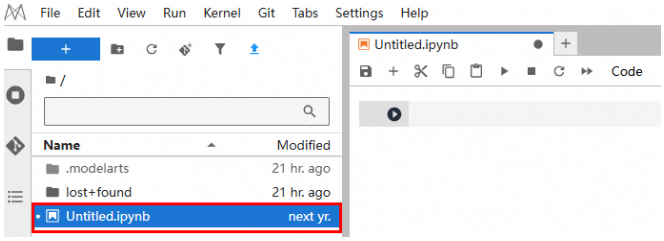
由于每个Notebook实例选择的工作环境不同，其支持的AI框架也不同，下图仅为示例，请根据实际显示界面选择AI框架。

图 1-38 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 1-39 新建文件



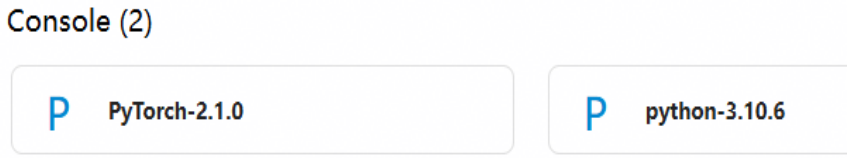
新建文件并打开 Console

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。

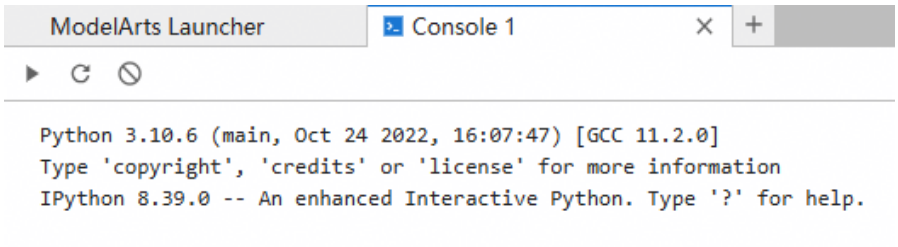
由于每个Notebook实例选择的工作环境不同，其支持的AI框架也不同，下图仅为示例，请根据实际显示界面选择AI框架。

图 1-40 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 1-41 新建文件（Console）

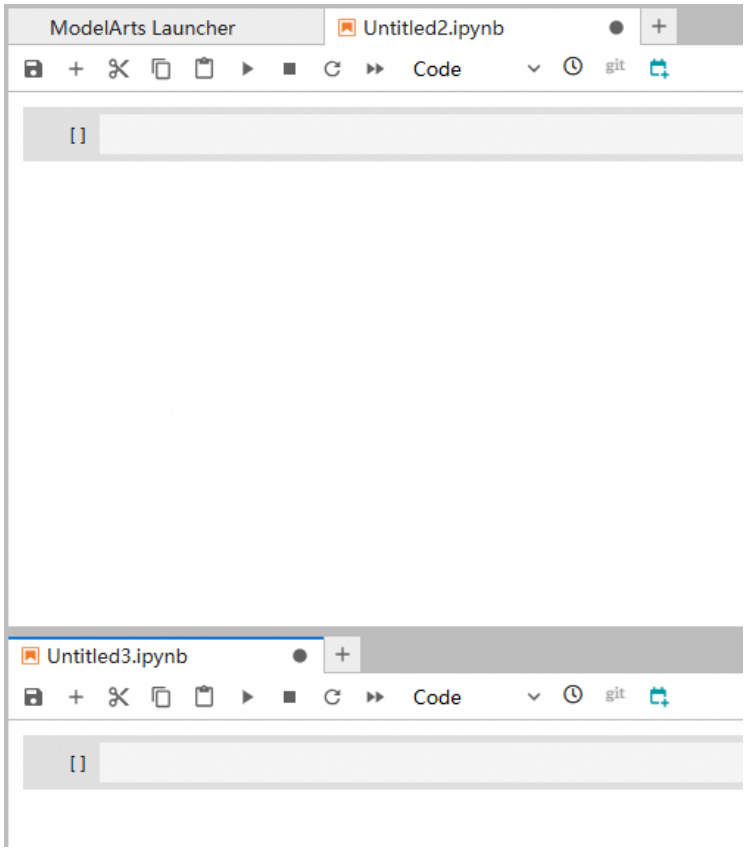


在 JupyterLab 中编辑文件

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。

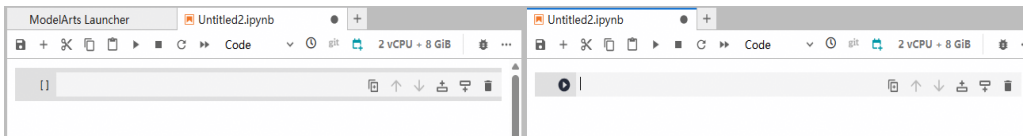
图 1-42 多文件任意编排



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 1-43 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。

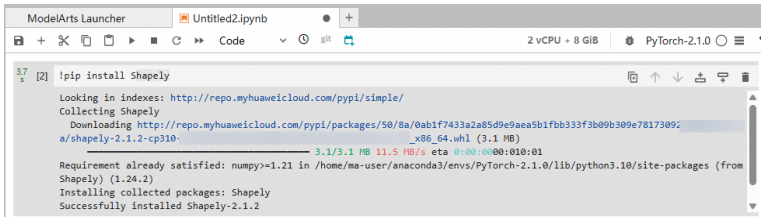
例如：安装外部库Shapely

```
!pip install Shapely
```

例如：查看PythonPath

```
!echo $PYTHONPATH
```

图 1-44 运行代码

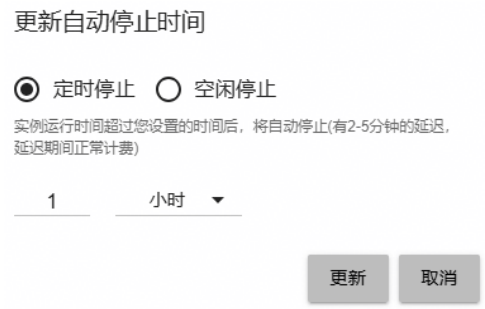


自动停止及续期

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 1-45 自动停止

图 1-46 续期



JupyterLab 常用快捷键和插件栏

图 1-47 JupyterLab 常用快捷键和插件栏

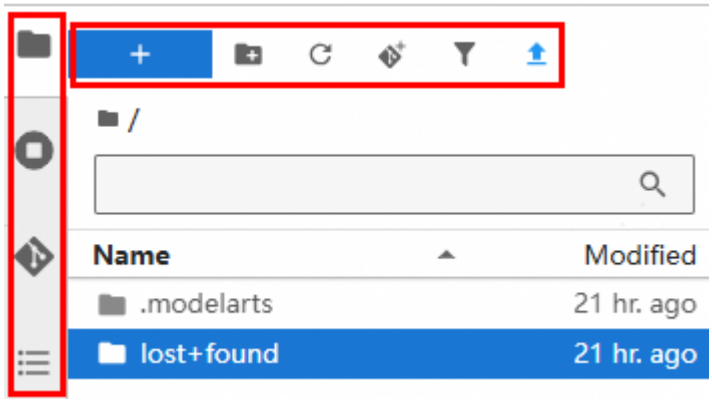


表 1-10 快捷键说明

快捷键	说明
	快速打开Notebook、Terminal。或打开ModelArts Launcher页面，可快速创建新的Notebook、Console或其他文件。
	创建文件夹。
	刷新文件目录。
	Git插件，可连接此Notebook实例关联的Github代码库。
	切换文件过滤器。
	上传文件。

表 1-11 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。
	文档结构图。

图 1-48 导航栏按钮

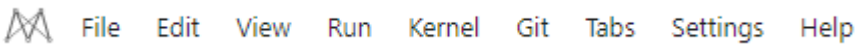


表 1-12 导航栏按钮介绍

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。

按钮	说明
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 1-49 ipynb 文件菜单栏中的快捷键

表 1-13 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。
	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。

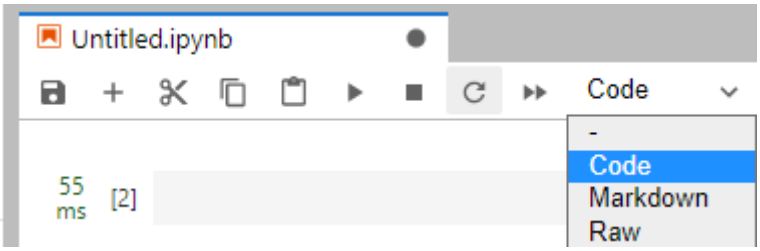
快捷键	说明
Code	此处下拉框有4个选项，分别是： Code（写python代码），Markdown（写Markdown代码，通常用于注释），Raw（一个转换工具），-（不修改）。
	查看代码历史版本。
git	git插件，图标显示灰色表示当前Region不支持。
2 vCPU + 4 GiB	当前的资源规格。
PyTorch-1.4	单击可以选择Kernel。
	表示代码运行状态，变为实心圆时，表示代码在运行中。

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。

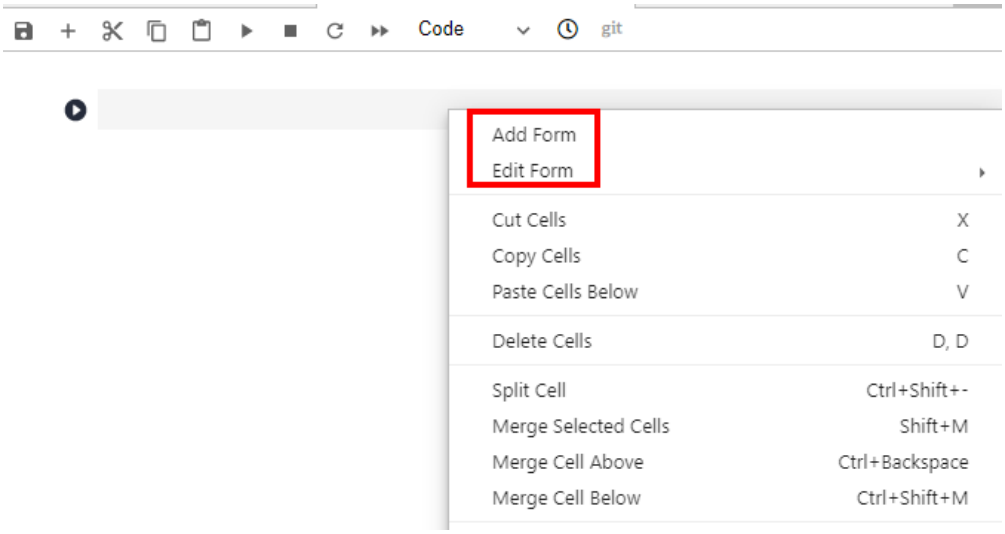
- 仅对Code cell类型新增了Edit Form和Add Form功能，如果cell类型是Markdown或者Raw类型则不支持。如下图所示：

  图 1-50 查看 Code cell

- 打开新的代码后，需先Add Form，再Edit Form。

图 1-51 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 1-52 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 1-14 “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1-53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1-54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。
Show all	同时展示code和form区域。

图 1-53 “dropdown”，“input”，“slider”的表单样式

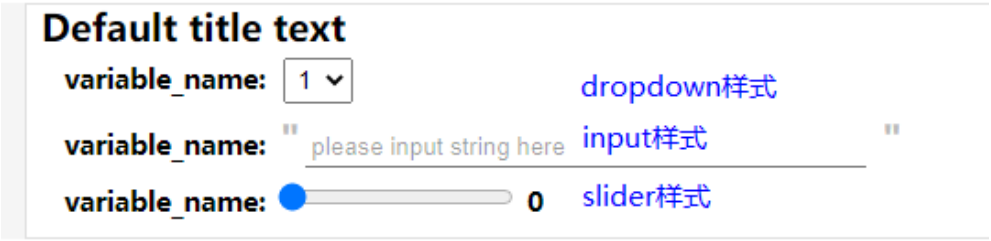


图 1-54 创建“dropdown”类型的表单

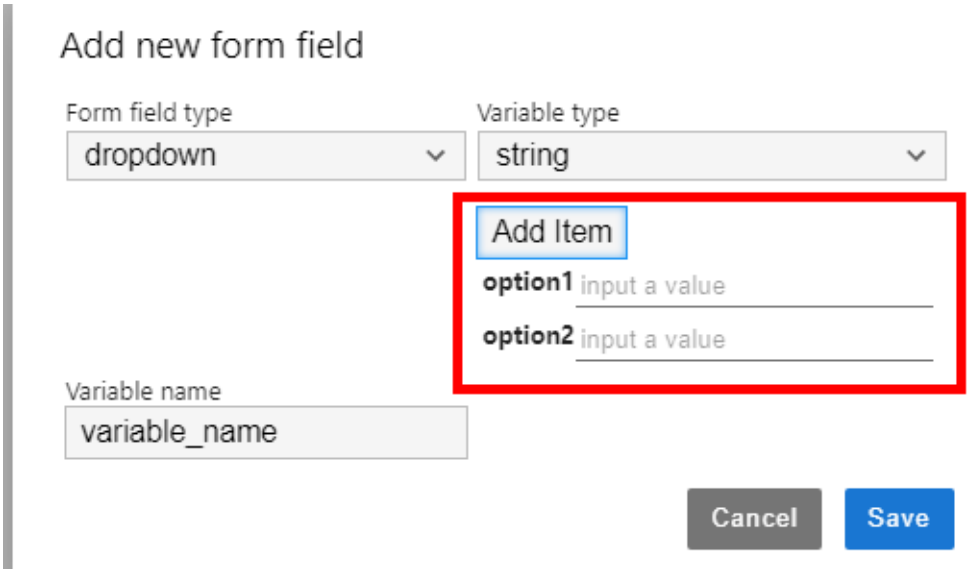
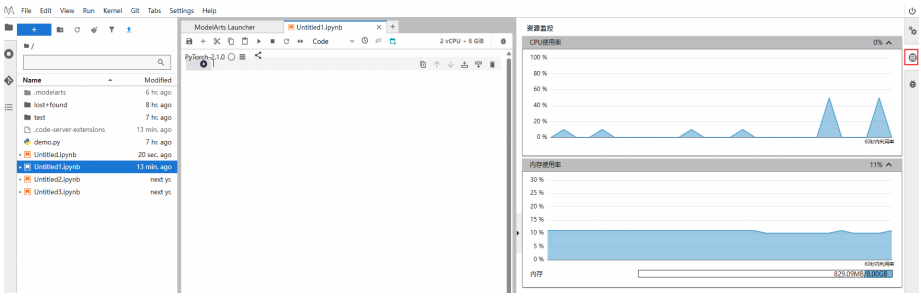


图 1-55 删除表单

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”，显示“CPU使用率”和“内存使用率”。

图 1-56 资源监控



1.4.3 升级 JupyterLab 版本

为了提高Notebook的易用性，现将JupyterLab升级至4.4.10版本（2025年稳定版），实现从JupyterLab 3.2.3（2021年老版）的跨版本升级。此次升级对Notebook的性能和启动速度进行了优化，重构了编辑器，强化了调试和协作功能，扩展了更稳定的生态系统，并全面改善了交互体验和内存占用。欢迎您使用。

注意事项

JupyterLab 3.2.3版本计划于2026年10月停止支持新的Notebook实例创建，并且不再提供相应的技术支持，包括新特性更新、漏洞/问题修复、补丁升级以及工单指导和在线排查等客户支持服务，这些将不再适用于ModelArts服务的运维保障。

JupyterLab 4 版本更新说明

JupyterLab 4版本在用户体验、功能完善和性能提升方面均有显著改进，以下是该版本的主要更新内容。关于JupyterLab 4版本的更多功能信息，请参见**JupyterLab 4官方文档**。

表 1-15 功能更新说明

功能模块	功能描述
工作区（Workspaces）特性	支持多个工作区，您可以在不同的工作区中组织和管理不同的项目和文件，提升项目管理效率。
新增Launcher创建工具栏	添加新的工具栏，提供了更多的功能访问方式，使操作更加便捷。
主题框架优化	改进主题框架，增强主题的灵活性和兼容性。您可以根据需求进行个性化的界面调整。
设置界面优化	优化设置界面的用户体验，使配置更加直观和易于使用。
性能优化和调试功能增强	对系统性能进行全面优化，调试功能也得到增强，提升开发体验。
代码编辑器增强	对代码编辑器进行改进，增加更多功能和性能优化，提升代码编写和编辑的体验。
搜索功能增强	增强搜索功能，您可以更快速、更准确地查找所需内容。

功能模块	功能描述
性能增强	整体性能得到提升，系统的响应速度和稳定性都有显著提高。
支持自定义CSS样式表	支持自定义CSS样式表，以便调整界面外观，满足个性化需求。
Markdown支持图表	在Markdown文档中支持图表功能，使图文并茂的文档编写变得更加方便。
虚拟滚动条	引入虚拟滚动条，改善大文件或大量内容的滚动体验。
Workspace UI	改进工作区用户界面，提供了更好的可视化效果和操作体验，增强界面友好性。
文件访问记录	支持查看最近打开和关闭文件，便于快速访问常用文件，提升工作效率。
键盘快捷键改进	改进键盘快捷键，提高操作的效率和便捷性。

升级 JupyterLab 版本

您可以任选以下方式升级JupyterLab版本。

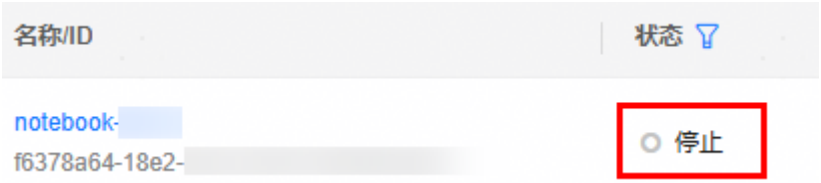
方式一：停止 Notebook 实例并升级 JupyterLab 版本

Notebook实例停止后，/home/ma-user/work目录下的数据会保存，其余目录下内容会被清理。如果需要保存开发环境，可以通过保存镜像的方式保留开发环境设置。具体操作，请参见**方式二：保存Notebook实例并升级JupyterLab版本**。

当Notebook实例存储类型为云硬盘EVS时，实例停止后，EVS仍将按照每GB规定费用持续计费，直至实例删除。关于计费详情，请参见**开发环境计费项**。

1. 停止Notebook实例。
   a. 登录**ModelArts管理控制台**，在左侧导航栏按需选择以下操作。
      - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
      - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
   b. 在“Notebook”页面的“操作”列，单击目标实例对应的“停止”。
   c. 在“停止实例”对话框，阅读提示信息，单击“确定”。
      当Notebook实例的“状态”为“停止”，表示停止成功。

      图 1-57 Notebook 实例状态

2. 升级JupyterLab版本。
   a. 在“Notebook”页面的“操作”列，单击停止实例对应的“启动”。
   b. 在“启动Notebook实例”对话框，选择JupyterLab 4版本，按需配置“自动停止”，单击“确定”。
3. 查看JupyterLab版本是否升级成功。
   在“Notebook”页面，单击实例名称，在“基本信息”页签的“环境信息”区域，在WebIDE下方可以看到JupyterLab 4版本，表示JupyterLab版本升级成功。

方式二：保存 Notebook 实例并升级 JupyterLab 版本

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。

1. 保存镜像。具体操作，请参见**保存Notebook实例**。
2. 升级JupyterLab版本。
   a. 在左侧导航栏按需选择以下操作。
      - 新版控制台：选择“资产管理 > 镜像”。
      - 旧版控制台：选择“资产管理 > 镜像管理”。
   b. 在“已注册镜像”页签，单击**步骤1**保存的镜像名称。
   c. 在“操作”列，单击“创建Notebook”。
   d. 在“创建Notebook”页面，选择JupyterLab 4版本，按需修改其他参数，单击“立即创建”。
      当Notebook实例的“状态”为“运行中”，表示Notebook创建成功。
3. 查看JupyterLab版本是否升级成功。
   在“Notebook”页面，单击实例名称，在“基本信息”页签的“环境信息”区域，在WebIDE下方可以看到JupyterLab 4版本，表示JupyterLab版本升级成功。

1.4.4 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

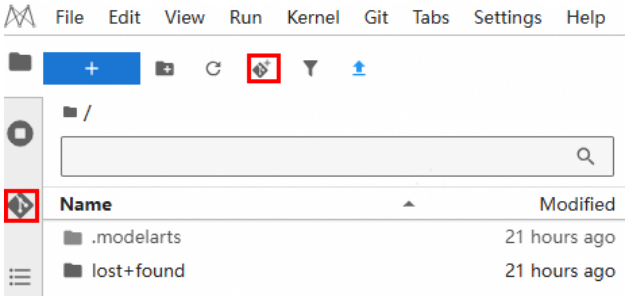
Notebook处于运行中状态。

打开 JupyterLab 的 git 插件

在Notebook列表中，选择一个实例，单击右侧的“接入环境”，在“接入方式”对话框，单击JupyterLab接入右侧的“接入”，进入“JupyterLab”页面。

图1-58所示图标，为JupyterLab的Git插件。

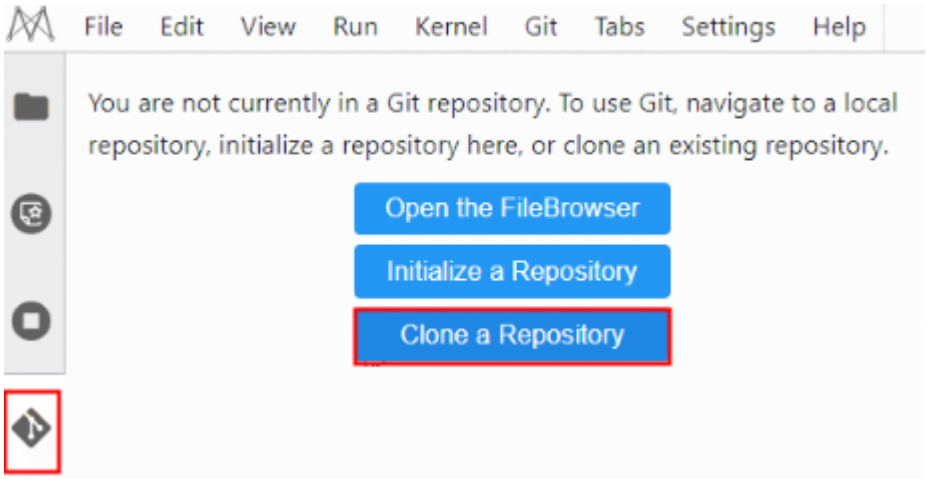
图 1-58 Git 插件



克隆 GitHub 的开源代码仓库

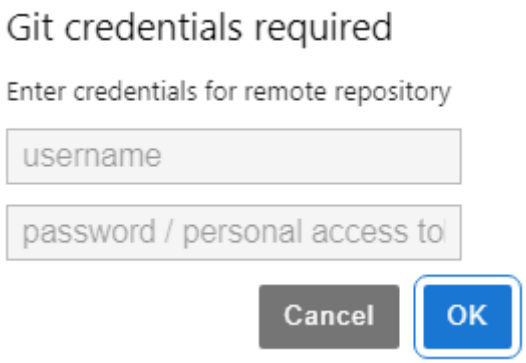
GitHub开源仓库地址：https://github.com/jupyterlab/extension-examples，单击 ，输入仓库地址，单击确定后即开始克隆，克隆完成后，JupyterLab左侧导航出现代码库文件夹。

图 1-59 使用 git 插件克隆 GitHub 的开源代码仓库



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。

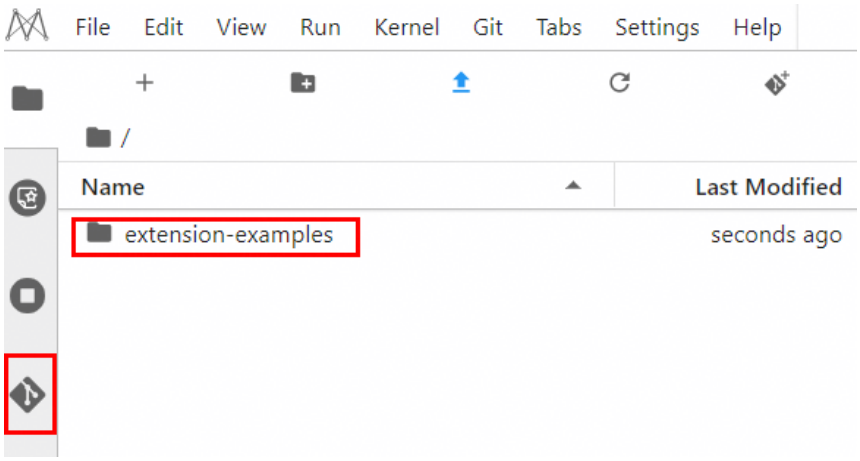
须知

- 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。

查看代码库信息

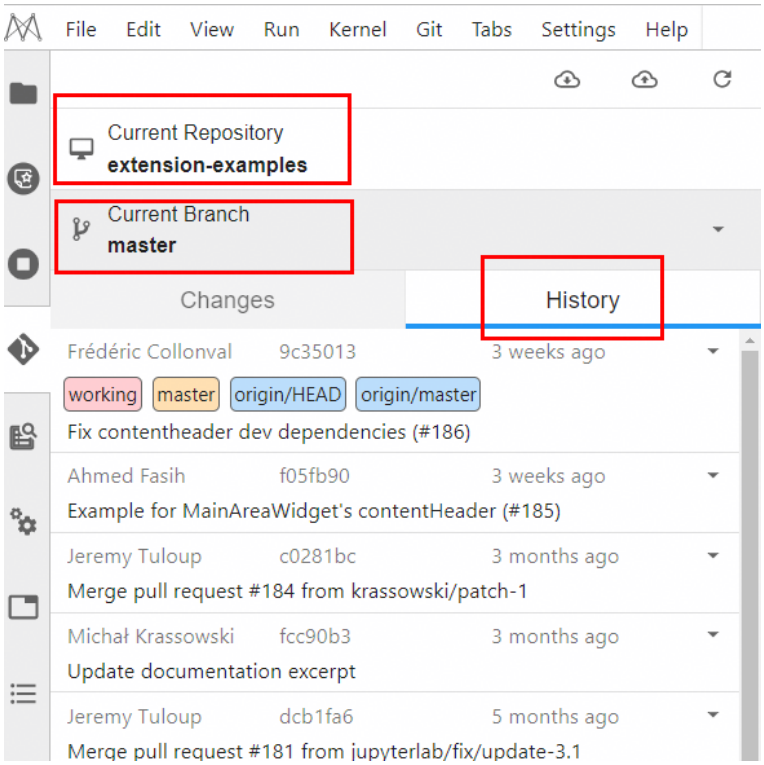
在Name下方列表中，选中您希望使用的文件夹，双击打开，然后单击左侧git插件图标进入此文件夹对应的代码库。

图 1-60 打开文件夹后打开 git 插件



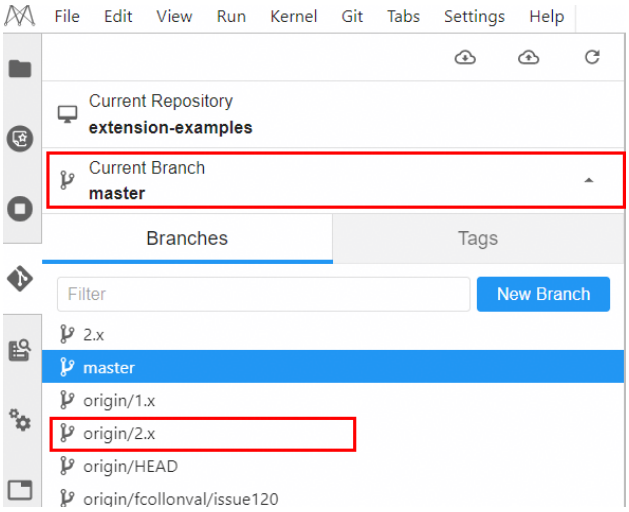
即可看到当前代码库的信息，如仓库名称、分支、历史提交记录等。

图 1-61 查看代码库信息



说明

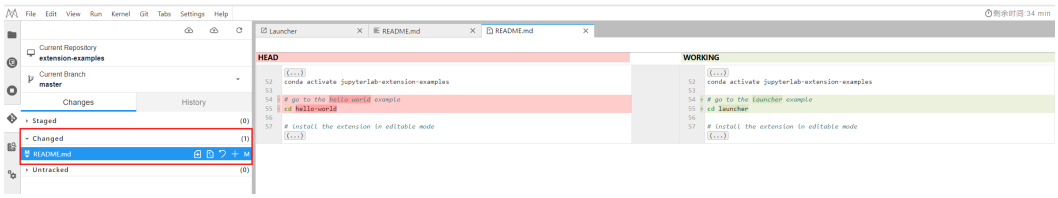
Git插件一般默认克隆master分支，如果要切换分支可单击Current Branch展开所有分支，单击相应分支名称可完成切换。



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。

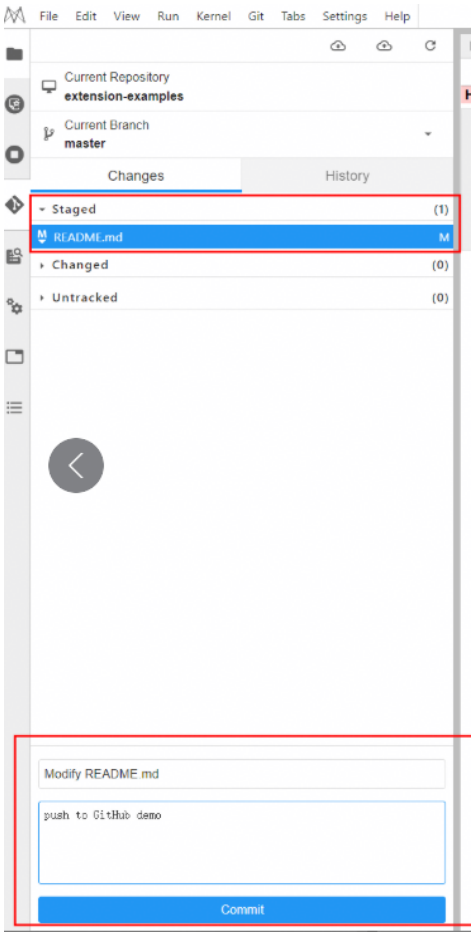
图 1-62 查看修改的内容



提交修改的内容

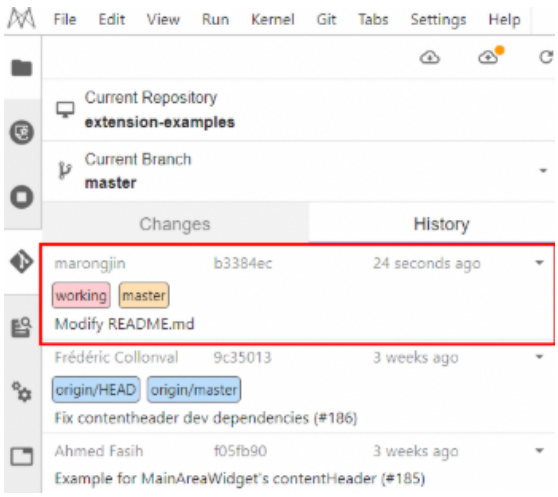
确认修改无误后，单击修改文件名称右侧的“Stage this change”，文件将进入Staged状态，相当于执行了git add命令。在左下方输入本次提交的Message，单击“Commit”，相当于执行了git commit命令。

图 1-63 提交修改内容



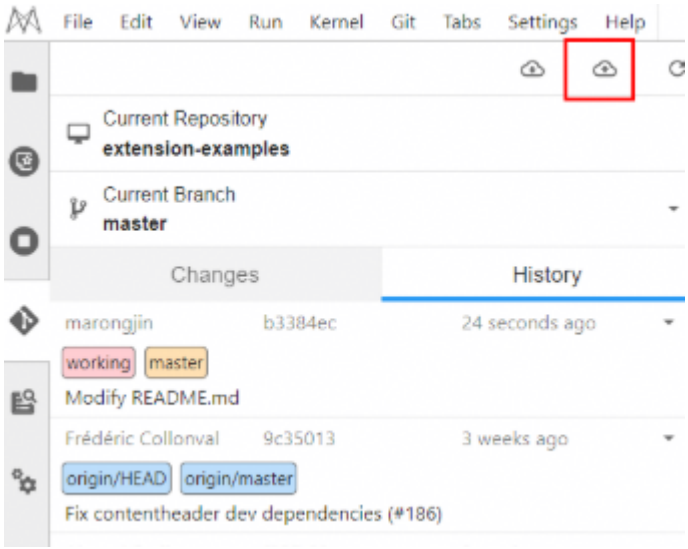
此时，可以在“History”页签下看到本地提交已成功。

图 1-64 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 1-65 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“origin/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。

1.4.5 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。

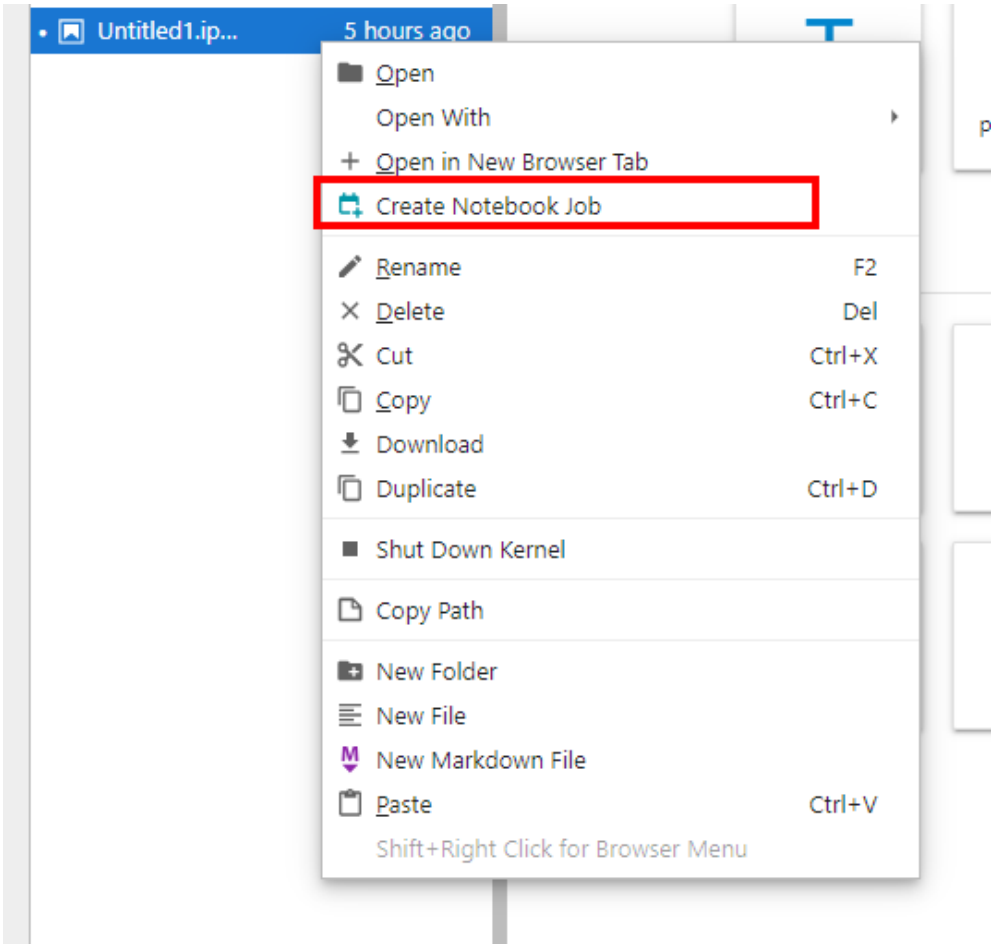
功能亮点

- **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

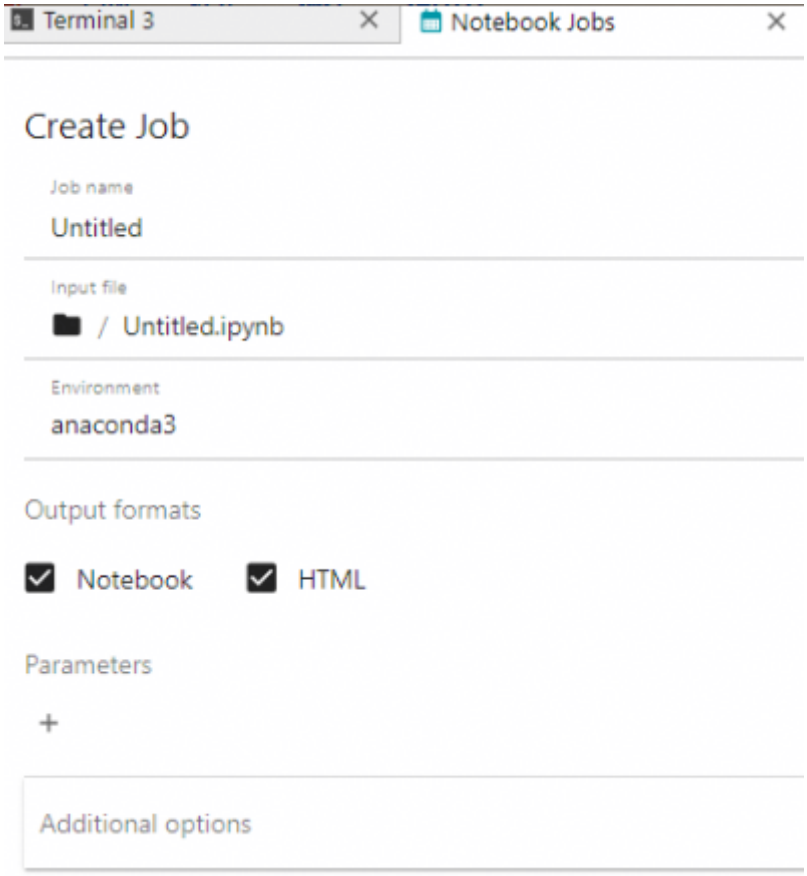
1. 打开ModelArts Notebook。
2. 选中Notebook文件（ipynb文件），创建定时任务。

图 1-66 打开 Notebook Jobs



3. 在Create Job界面，填写参数后单击“create”。

图 1-67 创建定时任务参数填写



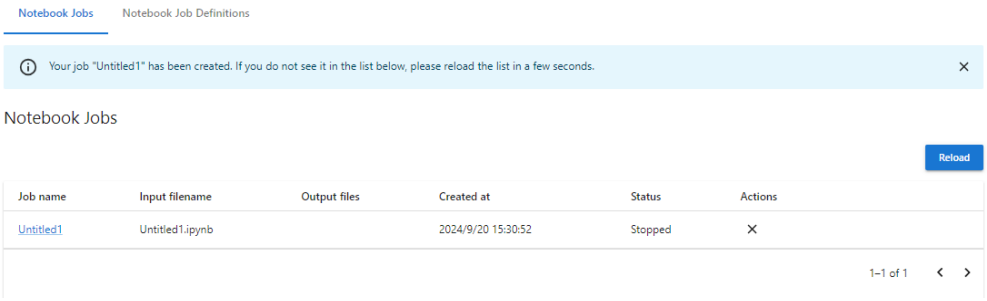
- Job name：定时任务名称。
- Environment：要运行该Notebook的python环境。
- Output formats：执行结果的输出文件类型。
- Parameter：单击+，手动设置运行Notebook的python变量。
- Schedule：任务执行策略，可以立即运行；也可以设置定时策略运行，支持cron表达式。

  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 1-68 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 1-69 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 1-70 在 Notebook Job Definitions 页签单击任务名称

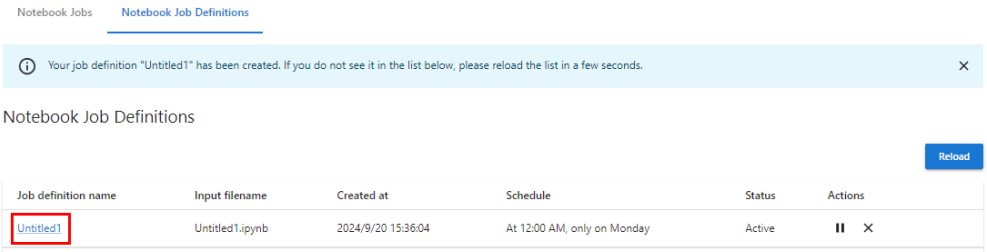
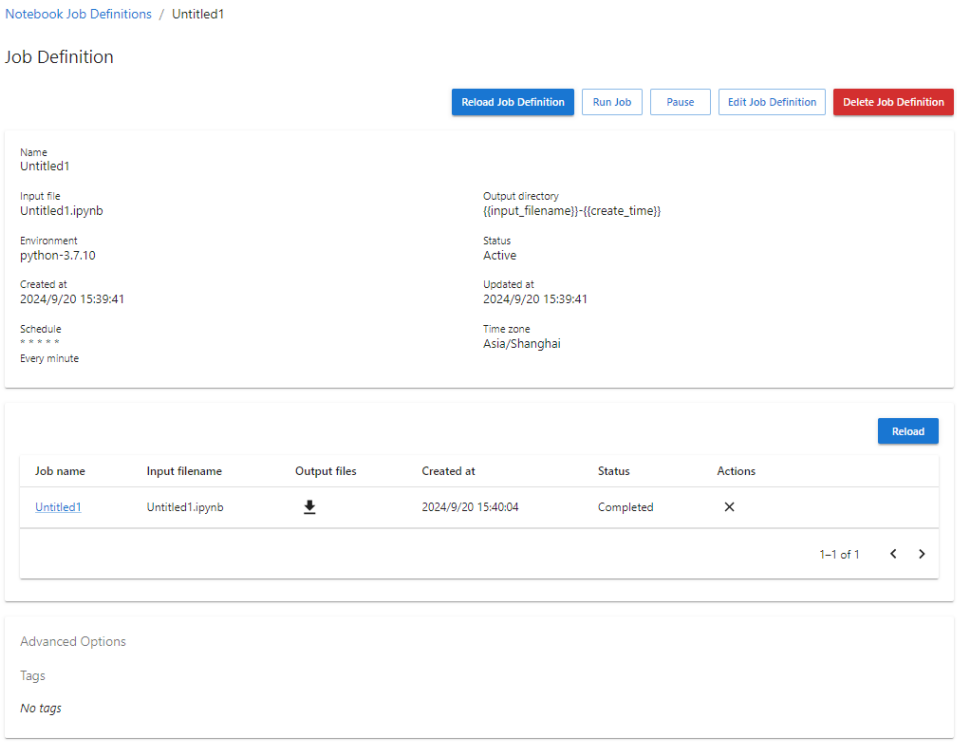


图 1-71 设置定时任务



1.4.6 上传文件至 JupyterLab

1.4.6.1 上传本地文件至 JupyterLab

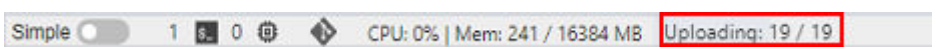
Notebook的JupyterLab中提供了多种方式上传文件。

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5GB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。
- 5GB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。
  ```

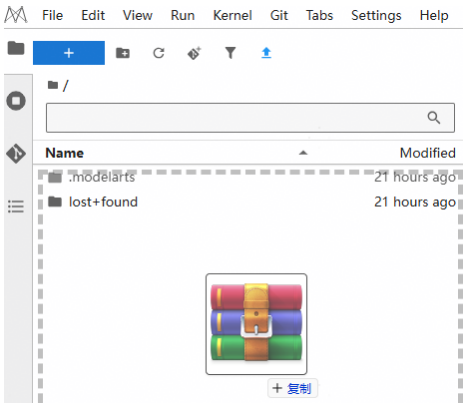
  解压命令的更多使用说明可以在主流搜索引擎中查找Linux解压命令操作。
- 多个文件同时上传时，JupyterLab窗口最下面会显示上传文件总数和已上传文件数。



上传文件入口

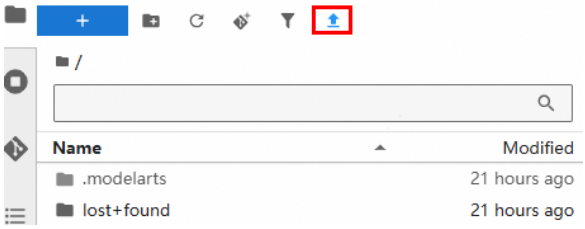
方式一：使用JupyterLab打开一个运行中的Notebook环境。直接将文件拖拽到JupyterLab窗口左边的空白处上传。

图 1-72 拖拽文件至 JupyterLab



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。

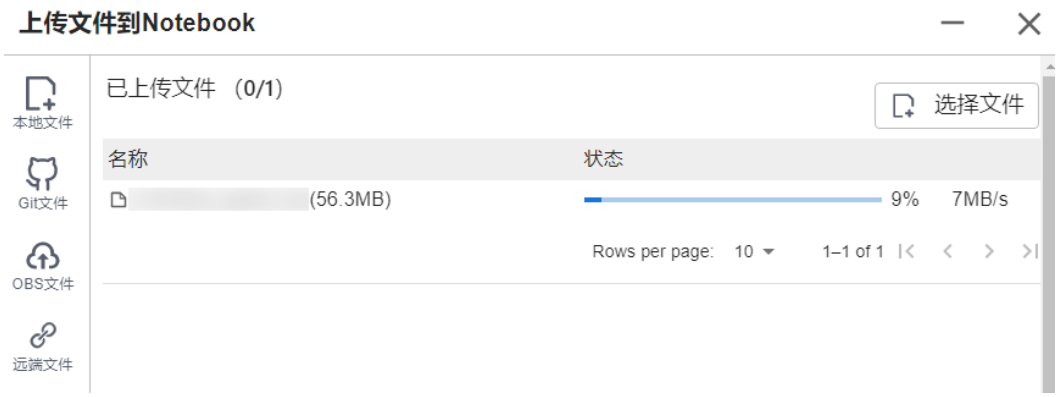
图 1-73 单击窗口上方导航栏的 ModelArts Upload Files 按钮



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 1-74 上传 100MB 以下小文件



文件上传完成后给出提示。

图 1-75 上传成功

上传本地大文件（100MB~5GB）至 JupyterLab

对于大小超过100MB不超过5GB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 1-76 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：

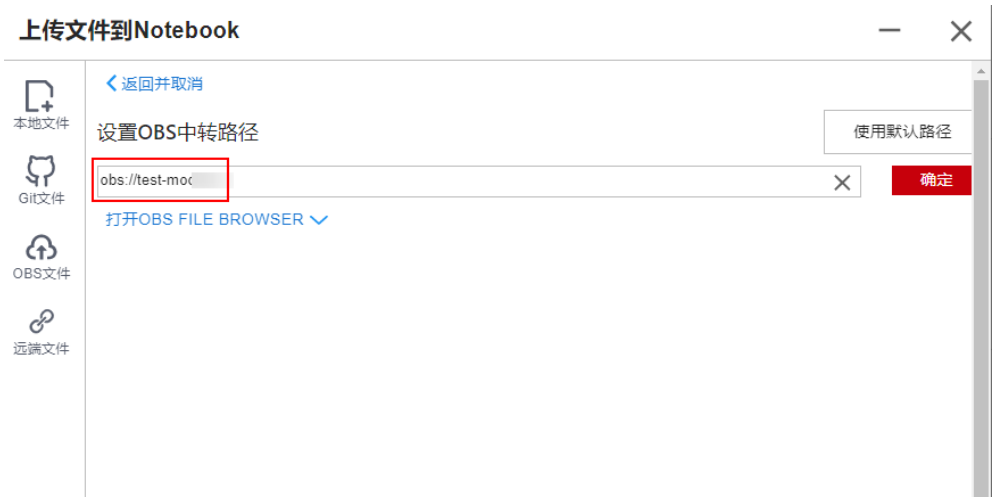
图 1-77 通过 OBS 中转路径上传

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1-81所示。

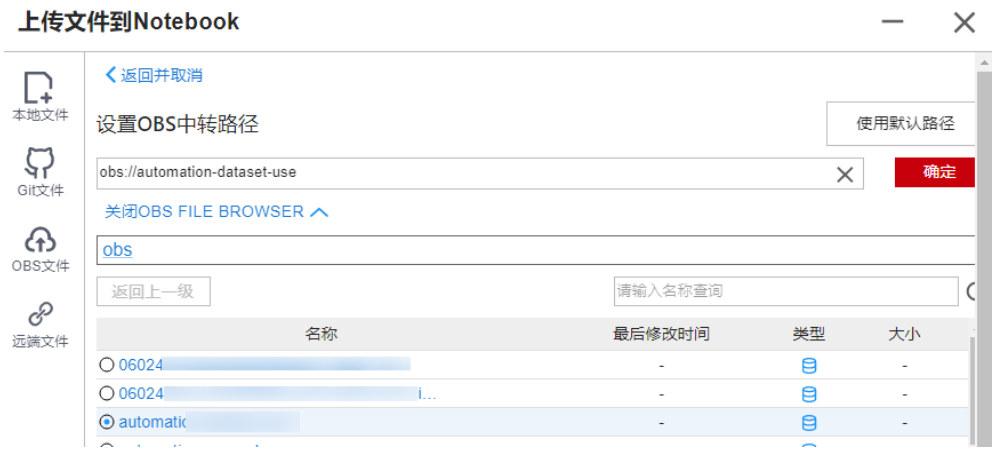
- 方式一：在输入框中直接输入有效的OBS中转路径，然后单击“确定”完成。

图 1-78 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 1-79 打开 OBS File Browser



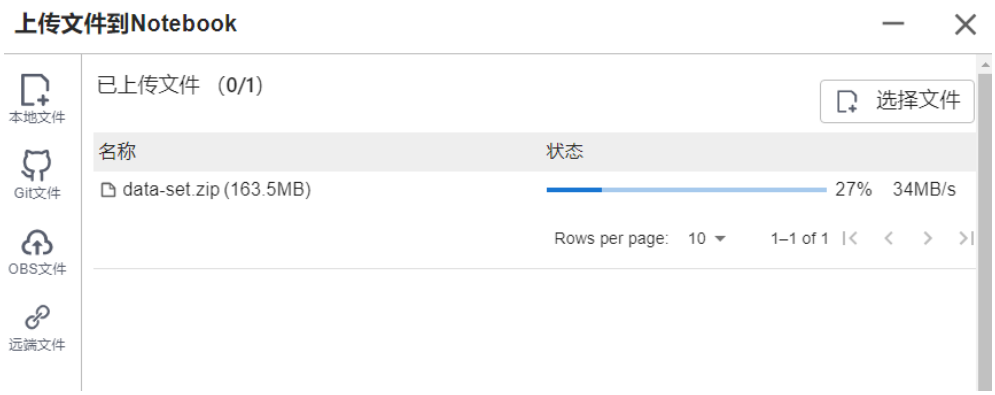
- 方式三：单击“使用默认路径”完成。

图 1-80 使用默认路径上传文件

图 1-81 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 1-82 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
```

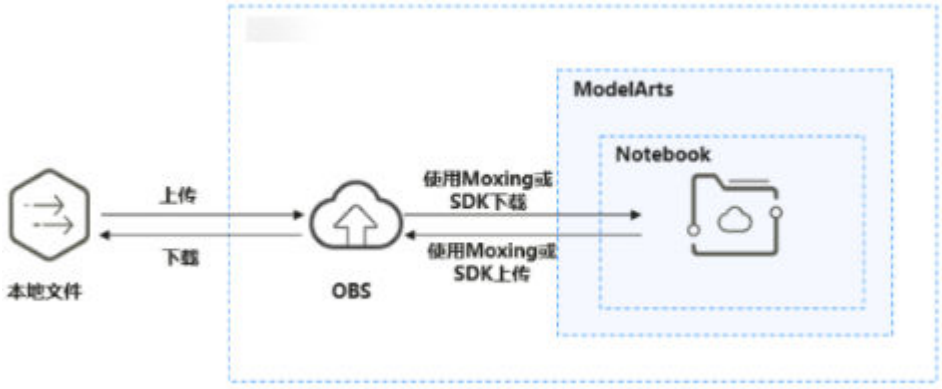
解压命令的更多使用说明可以在主流搜索引擎中查找Linux解压命令操作。

上传本地超大文件（5GB 以上）至 JupyterLab

不支持在Notebook的JupyterLab中直接上传大小超过5GB的文件。

5GB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 1-83 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1-84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1-85所示。

图 1-84 打开 Terminal

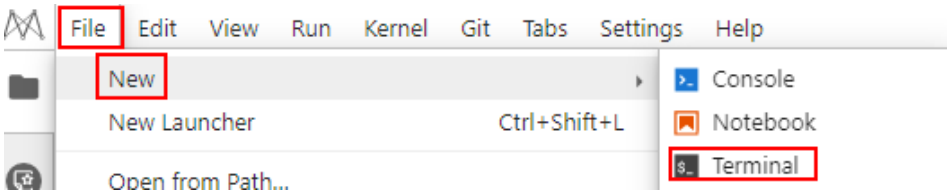
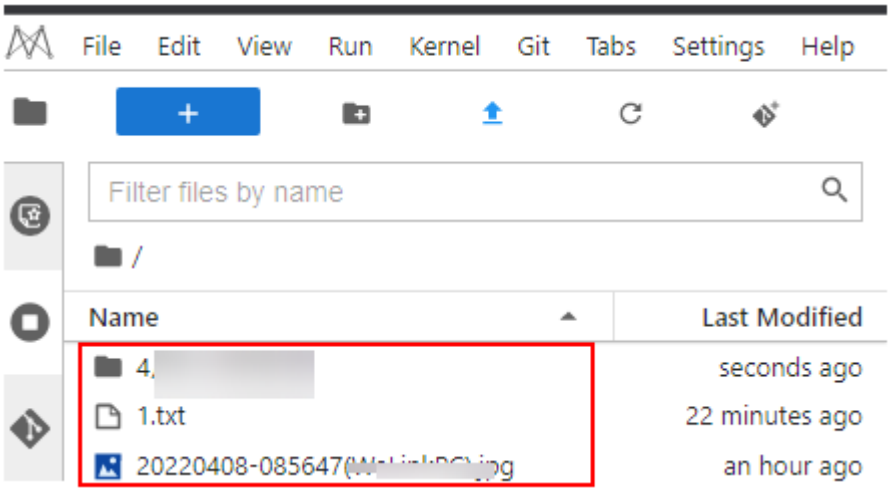


图 1-85 查看下载到 Notebook 中的文件



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**。

1.4.6.2 克隆 GitHub 开源仓库文件到 JupyterLab

在Notebook的JupyterLab中，支持从GitHub开源仓库克隆文件。

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开源仓库克隆界面。

图 1-86 上传文件图标

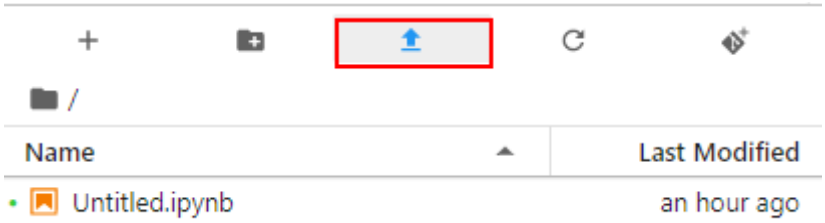
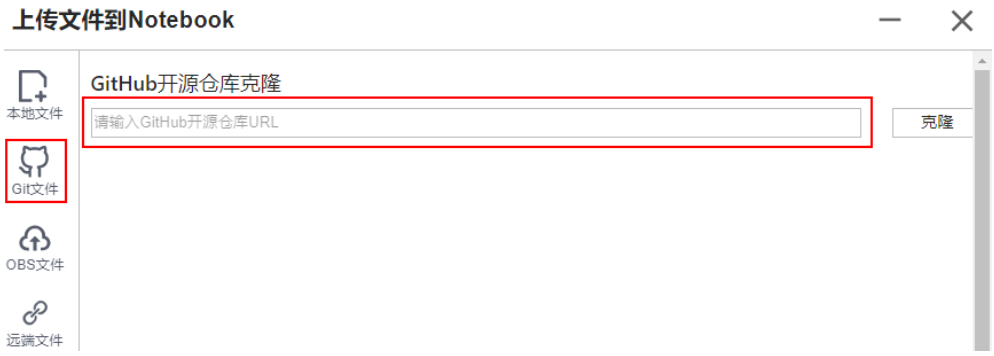


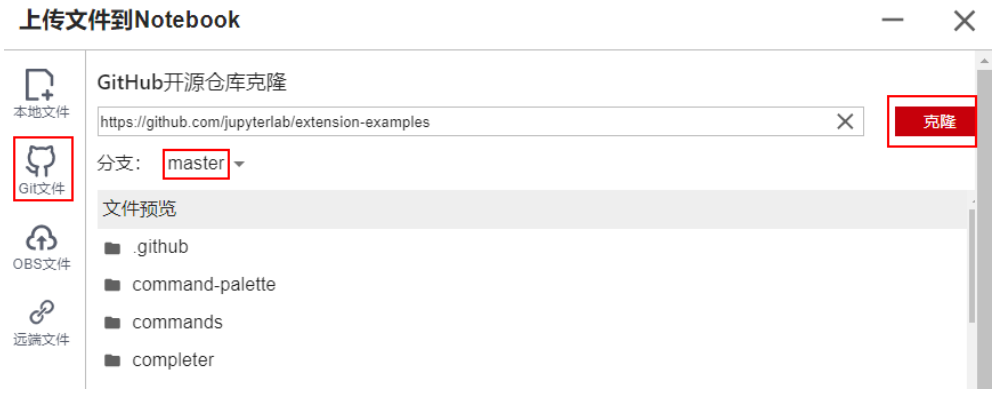
图 1-87 进入 GitHub 开源仓库克隆界面



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”开始克隆仓库。

   GitHub开源仓库地址：https://github.com/jupyterlab/extension-examples

图 1-88 输入有效的 GitHub 开源仓库地址

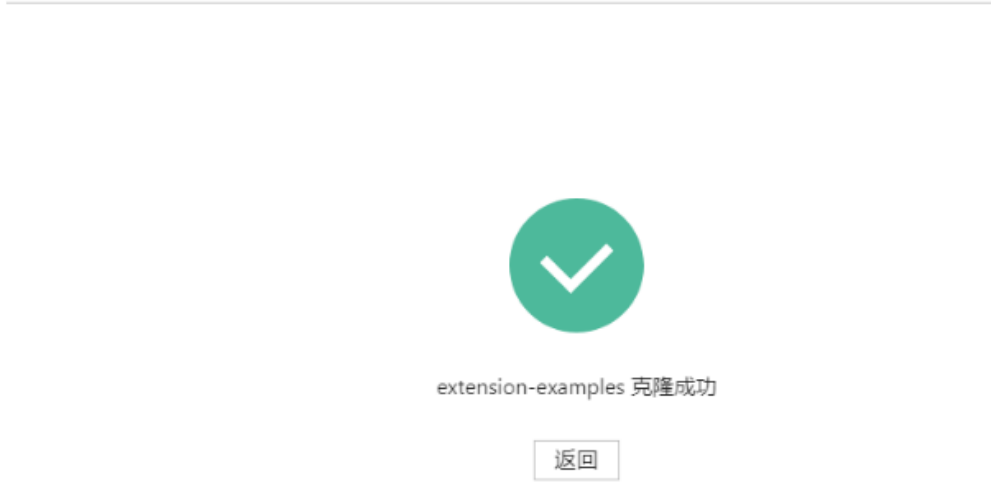


4. Clone仓库的过程中会将进度展示出来。

图 1-89 Clone 仓库的过程

5. Clone仓库成功。

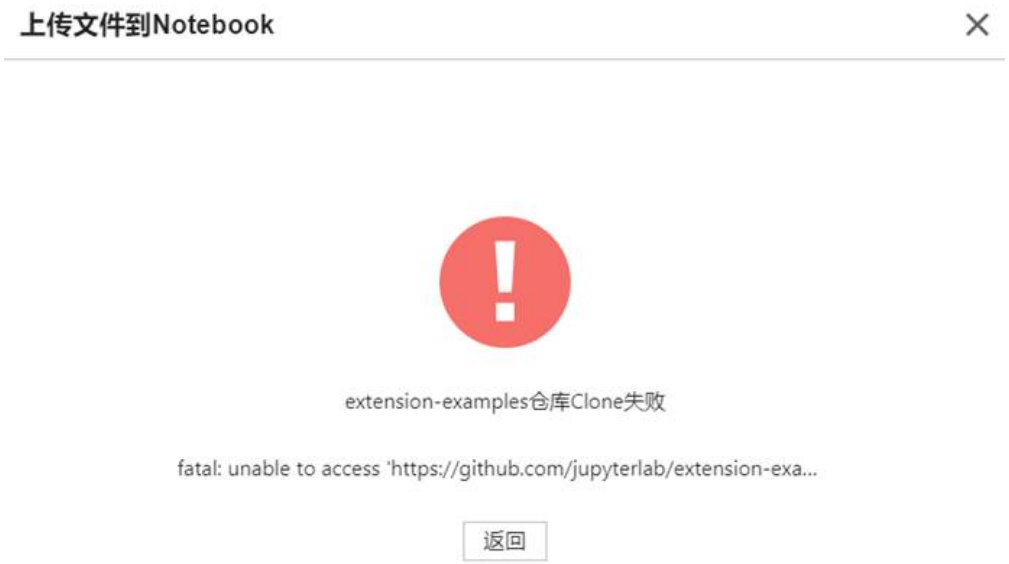
图 1-90 Clone 仓库成功



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。

图 1-91 Clone 仓库失败



- 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 图标取消。

1.4.6.3 上传 OBS 文件到 JupyterLab

在Notebook的JupyterLab中，支持将OBS中的文件上传到Notebook。注意：文件大小不能超过50GB，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 1-92 上传文件图标

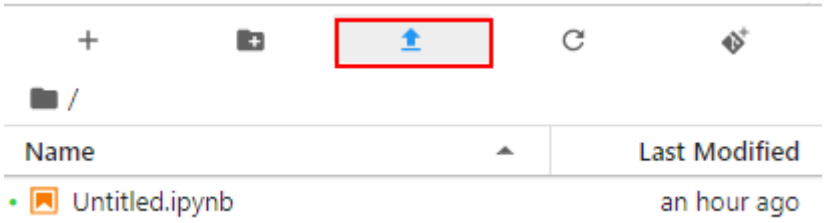
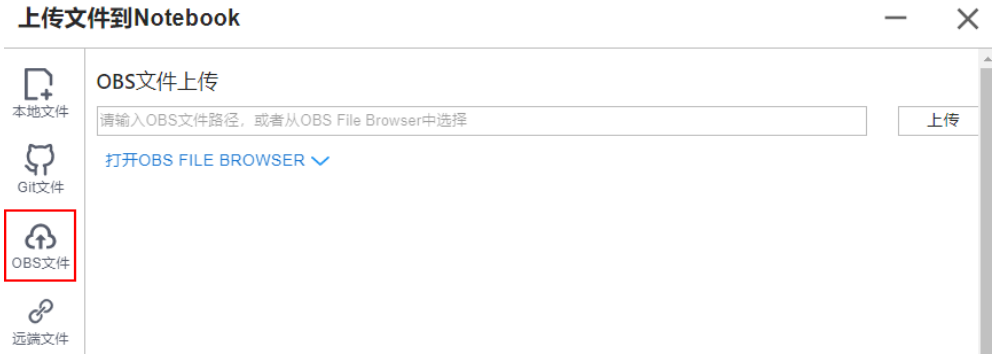
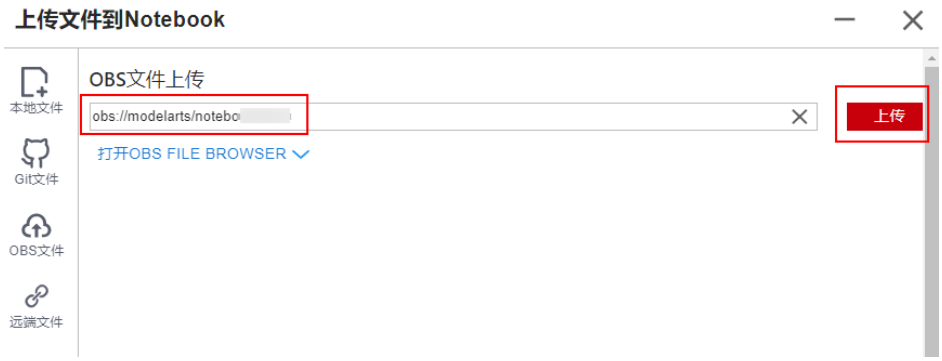


图 1-93 OBS 文件上传界面



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”开始上传文件。

     图 1-94 输入有效的 OBS 文件路径

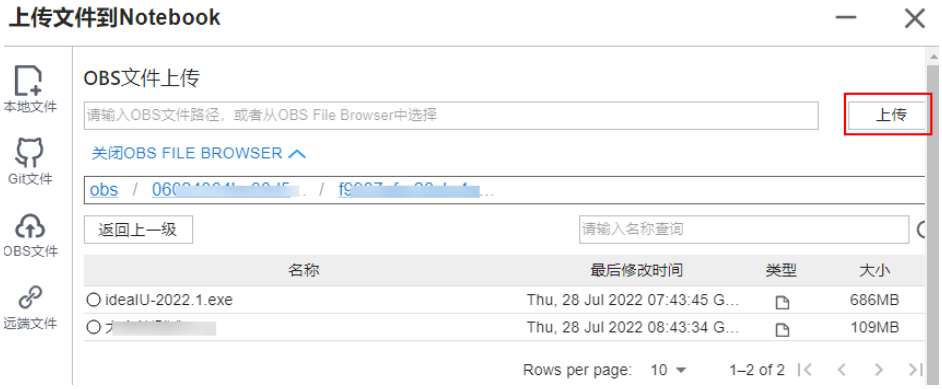
     

     说明

     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 1-95 上传 OBS 文件

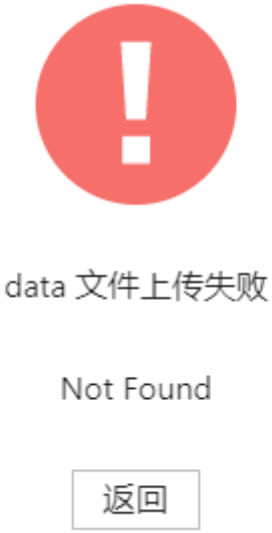
     

异常处理

提示文件上传失败，有以下三种常见场景。

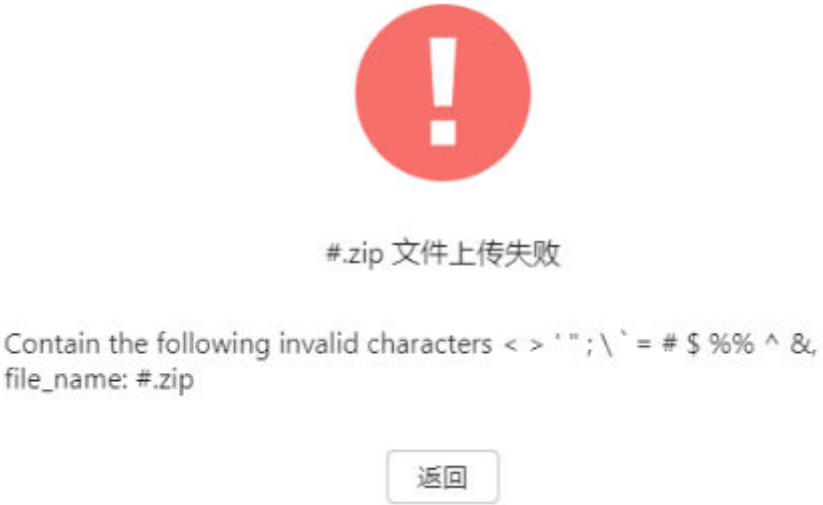
- **异常场景1**

  图 1-96 文件上传失败

  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 1-97 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 1-98 文件上传失败

可能原因：

文件大小超过50GB导致上传失败。

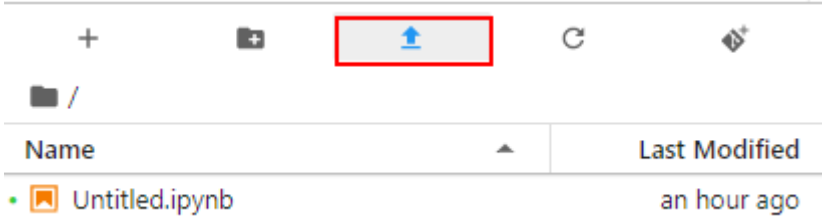
1.4.6.4 上传远端文件至 JupyterLab

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。

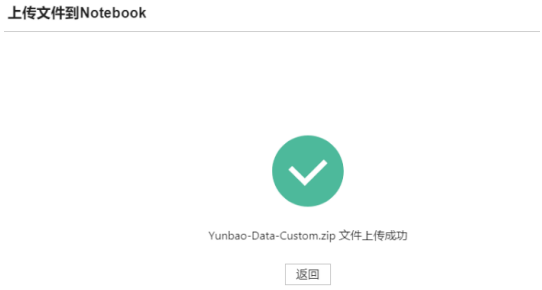
1. 通过JupyterLab打开一个运行中的Notebook。
2. 单击JupyterLab窗口上方导航栏的 ModelArts Upload Files按钮，打开文件上传窗口，选择左侧的 进入远端文件上传界面。

图 1-99 上传文件图标



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。

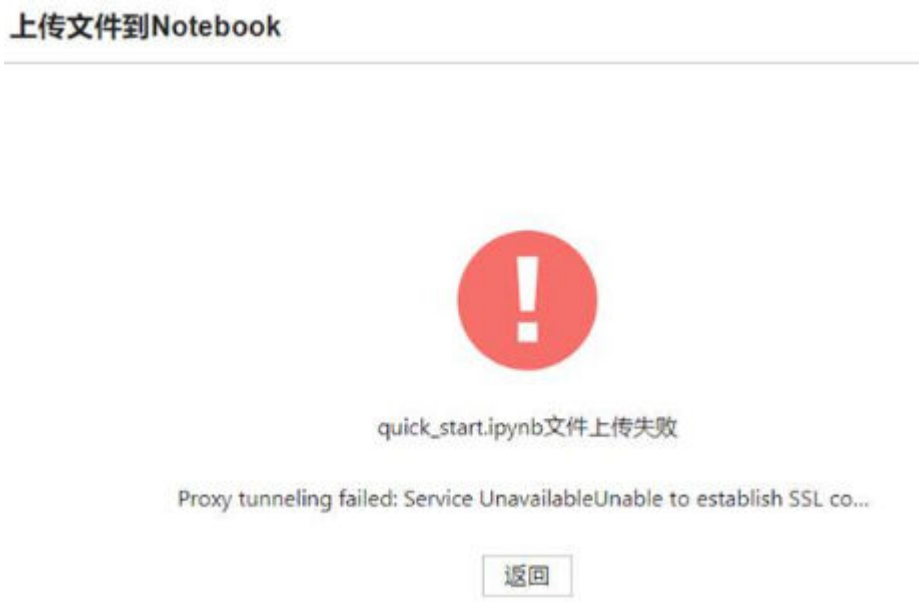
图 1-100 远端文件上传成功



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。

图 1-101 远端文件上传失败



1.4.7 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**。

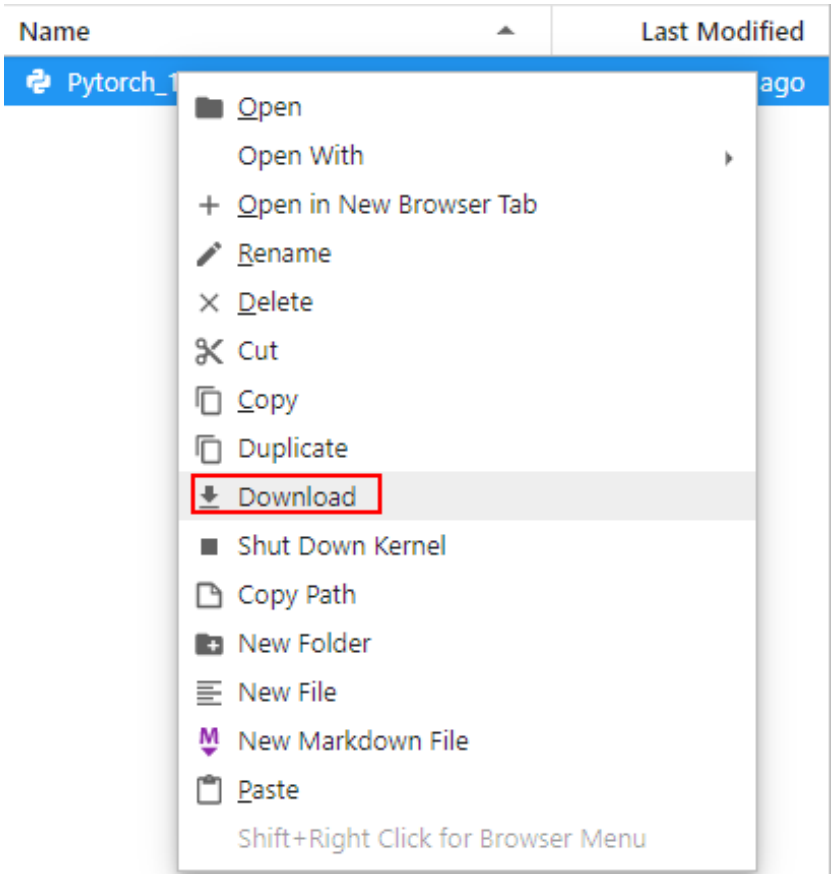
- 不大于100MB的文件，可以直接从JupyterLab中下载到本地，具体操作请参见**从JupyterLab中下载不大于100MB的文件至本地**。
- 大于100MB的文件，需要先从JupyterLab上传到OBS，再通过OBS下载到本地，具体操作请参见**从JupyterLab中下载大于100MB的文件到本地**。

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 1-102 下载文件



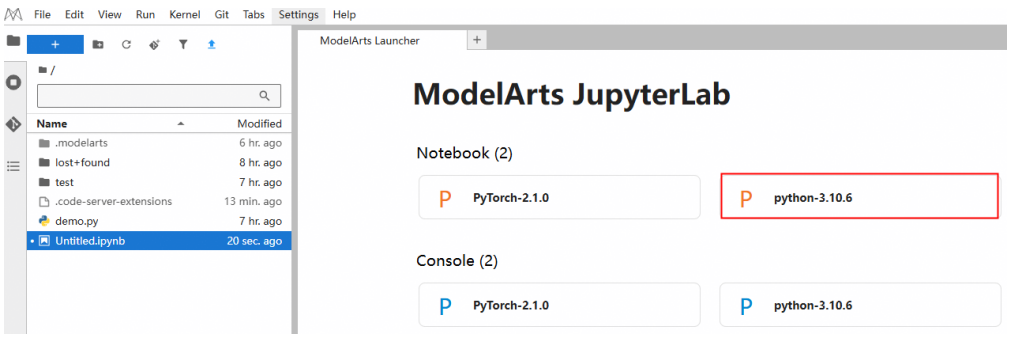
从 JupyterLab 中下载大于 100MB 的文件到本地

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ModelArts Launcher页面的Notebook区域，单击“python”。请您以实际环境为准。

   图 1-103 打开 Python 运行环境

2. 使用MoXing将目标文件从Notebook上传到OBS中。

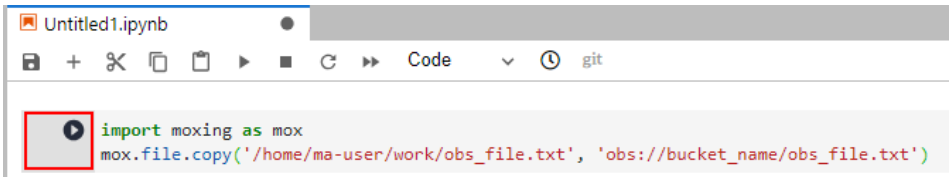
上传txt、压缩后文件夹的Python示例代码如下。代码中的“/home/ma-user/work/xxx”为文件在Notebook中的存储路径，“obs://bucket_name/xxx”为该文件上传到OBS的存储路径，请您按照实际路径进行修改。

- 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 1-104 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")
     ```

1.4.8 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能够有效地展示训练过程中的计算图、各种指标随时间的变化趋势以及训练中使用到的数据信息，相关概念请参考**TensorBoard官网**。

TensorBoard可视化工具当前仅支持在PyTorch和TensorFlow引擎中使用，不支持在MindSpore引擎或其他AI引擎中使用。

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**。

PyTorch引擎的训练脚本中添加Summary代码，具体方式请参见**PyTorch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

步骤一：创建开发环境并在线打开

步骤二：生成Summary数据

步骤三：上传Summary数据

步骤四：启动TensorBoard

步骤五：查看看板中的可视化数据

步骤一：创建开发环境并在线打开

1. 登录**ModelArts管理控制台**，在左侧导航栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 在“Notebook”页面右上角，单击“创建”，创建TensorFlow或者PyTorch镜像的开发环境实例。具体操作，请参见**创建Notebook实例**。
3. 创建成功后，单击开发环境实例“操作”列的“接入环境”，在“接入方式”对话框，单击JupyterLab接入右侧的“接入”，在线打开运行中的开发环境。

说明

如果待可视化数据目录下内容较多（例如总量为百M级别）时，2U8G的CPU规格实例可能会出现CPU或Memory不足的情况，导致Notebook实例卡顿无法正常工作，此时建议使用更大规格（例如4U16G）的实例进行可视化操作。具体规格以实际局点支持的资源规格为准。

步骤二：生成 Summary 数据

在开发环境中使用TensorBoard可视化功能，需要生成Summary数据。以下是一个简单的线性回归训练示例，展示如何生成和记录Summary数据。如果需要了解如何生成Summary数据，请参考**PyTorch官方文档**。

```
import torch
from torch.utils.tensorboard import SummaryWriter
```

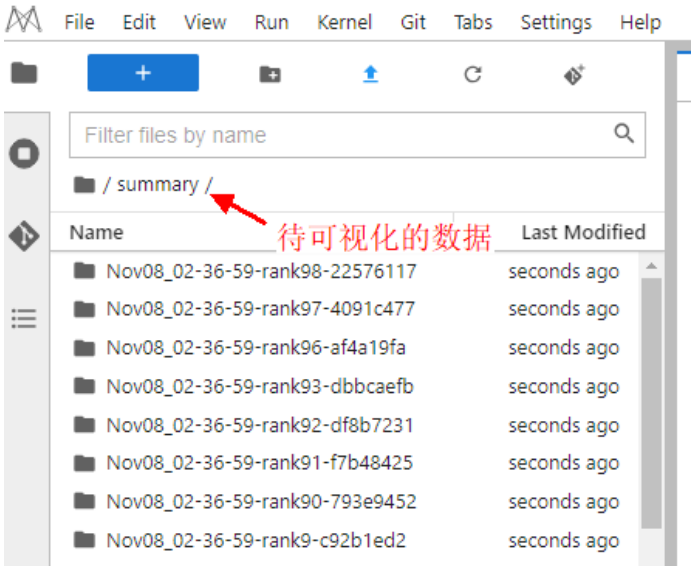
```
writer = SummaryWriter()

x = torch.arange(-5, 5, 0.1).view(-1, 1)
y = -5 * x + 0.1 * torch.randn(x.size())
model = torch.nn.Linear(1, 1)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.1)
def train_model(iter):
    for epoch in range(iter):
        y1 = model(x)
        loss = criterion(y1, y)
        writer.add_scalar("Loss/train", loss, epoch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
train_model(10)
writer.flush()
```

步骤三：上传 Summary 数据

您可以通过如下方式将本地Summary数据上传到Notebook中：

在Notebook左侧文件导航栏下新建目录，并通过拖拽等方式将本地数据上传到Notebook该新建目录中，具体请参见**上传本地文件至JupyterLab**。

图 1-105 Notebook 中的待可视化的数据



说明

建议将需要可视化的数据上传到单独目录下，如果使用OBS挂载功能，建议只挂载可视化数据所在层级的OBS路径到Notebook，而不是将整个桶都挂载上来进行可视化，避免混杂其他数据导致TensorBoard加载变慢，甚至无法正常可视化。

步骤四：启动 TensorBoard

1. 在开发环境的JupyterLab中打开ModelArts Launcher页面，然后单击TensorBoard图标。

图 1-106 打开 ModelArts Launcher 界面

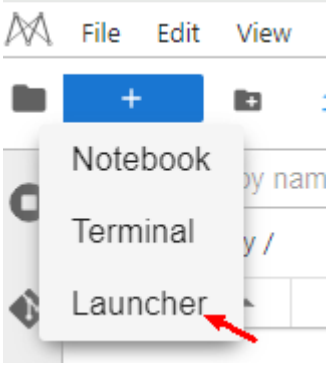
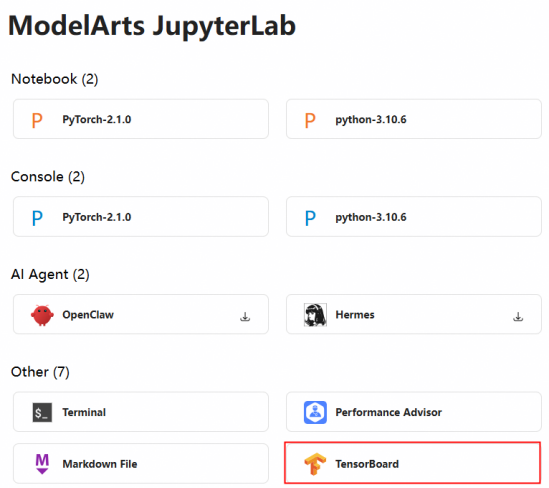


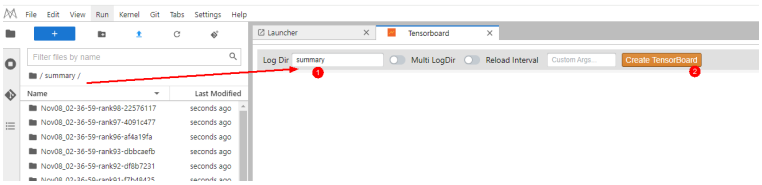
图 1-107 在 Launcher 中打开 TensorBoard



2. 首次单击TensorBoard会进入到一个默认的初始化面板，可以从该面板创建TensorBoard实例。

   在Log Dir中填写可视化数据所在的目录，该路径为/home/ma-user/work下的相对路径，填写好后单击右侧“Create TensorBoard”按钮，如**图2 打开Launcher界面**所示。非首次进入则会直接进入到第一个活跃的TensorBoard实例。

图 1-108 TensorBoard 插件界面



参数介绍如下：

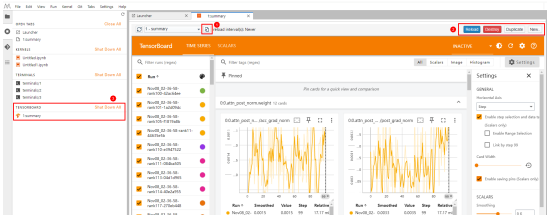
- Log Dir：默认是单击TensorBoard时当前侧边栏的目录，也可以手动填写对应目录，这里建议目录尽可能细化，目录内容比较少的话会提高初始化速度。
- Multi LogDir：支持输入多个目录参数，用逗号分隔，对应原生TensorBoard的--logdir_spec参数。TensorBoard官方对该功能支持不完善，不推荐使用。
- Reload Interval：TensorBoard多久对相应目录进行一次重新扫描，这个选项是默认是关闭的，日常使用选择手动Reload即可（设置Reload Interval之

后，TensorBoard后端持续扫描目录会对JupyterLab的稳定性和文件系统都产生一定的影响）。

关于该插件的更多功能介绍请参考**JupyterLab TensorBoard Pro官网**。

3. 创建好后的Tensorboard可视化展示面板如**图1-109**所示。
   - 单击**图1-109**中数字1所示按钮，可以在单独的浏览器页签展示可视化面板。
   - **图1-109**数字2中提供了实例管理相关功能，包括重启Tensorboard实例（Reload），关闭Tensorboard实例（Destroy），复制一个前端可视化面板（Duplicate），创建一个新的Tensorboard实例（New）。
   - **图1-109**数字3处，可以使用JupyterLab的Kernel管理面板管理实例，提供跳转至对应实例和删除等功能。

图 1-109 创建后的可视化界面



说明

不建议打开Reload Interval，避免后台频繁刷新导致Notebook实例产生卡顿甚至无法正常使用，需要查看新数据时单击右上角刷新按钮即可。

不建议使用New按钮创建多个TensorBoard实例，该操作可能会导致CPU/Memory占用过大，导致Notebook实例卡顿甚至无法正常使用等问题。当需要可视化新的目录时，可以先将当前TensorBoard实例关闭，再指定新目录创建一个新的TensorBoard实例。

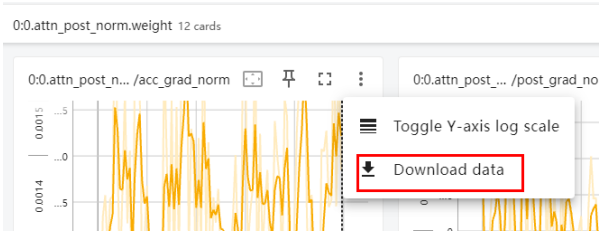
步骤五：查看看板中的可视化数据

可视化看板是TensorBoard的可视化组件的重要组成部分，看板标签包含：标量可视化、图像可视化和计算图可视化等。

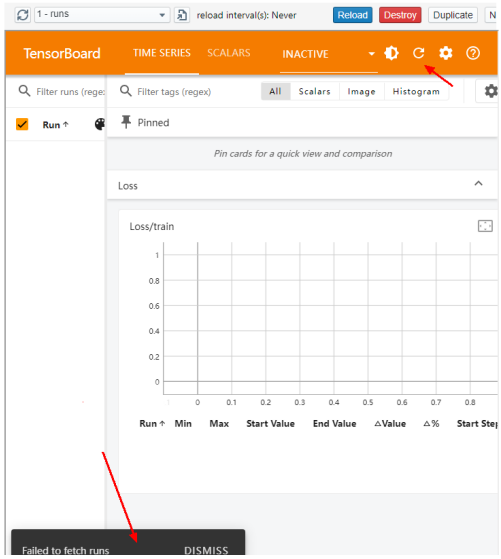
更多功能介绍和用法请参见**TensorBoard官网资料**。

说明

1. 由于Chrome等浏览器的安全限制导致无法从iframe下载数据，当前不支持可视化数据的Download data按钮。



2. 如果可视化面板一直未展示数据，请检查可视化数据目录是否选择正确以及目录下的数据是否完整（请勿随意手工调整或删除部分训练打点产生的summary数据，否则可能无法正常可视化）。
3. 如果数据正确但一直未成功展示数据（右上角刷新按钮一直在转圈），请打开一个terminal窗口，使用top命令查看CPU占用率等信息，可能是由于数据量过大导致CPU/Memory超负荷，建议减少数据量或者更换更大的Notebook实例规格。
4. 如果偶尔显示数据未成功加载，可稍等片刻后单击右上角的刷新按钮，通常情况下可再次正常显示相关数据。



1.4.9 在 JupyterLab 中预览 Markdown、PDF、数学公式

在JupyterLab中预览Markdown、PDF、数学公式功能默认关闭。预览功能存在跨站脚本（XSS）安全风险。为了保护您的系统安全，建议不要打开不受信任的文件。如果必须使用这些功能，请谨慎操作，并确保文件来源可靠。

开启/关闭预览功能

1. 打开Notebook的Terminal。具体操作，请参见**JupyterLab常用功能介绍**。
2. 按需执行以下命令，开启或关闭所需功能的预览。
   - 开启PDF文件预览功能
     ```
     $CONDA_BIN/jupyter labextension enable @jupyterlab/pdf-extension
     ```
   - 关闭PDF文件预览功能
     ```
     $CONDA_BIN/jupyter labextension disable @jupyterlab/pdf-extension
     ```
   - 开启Markdown文件预览功能

```
$CONDA_BIN/jupyter labextension enable @jupyterlab/markdownviewer-extension:plugin
```

- 关闭Markdown文件预览功能
  ```
  $CONDA_BIN/jupyter labextension disable @jupyterlab/markdownviewer-extension:plugin
  ```
- 开启数学公式预览功能
  ```
  $CONDA_BIN/jupyter labextension enable @jupyterlab/mathjax-extension:plugin
  $CONDA_BIN/jupyter labextension enable @jupyterlab/mathjax2-extension:plugin
  ```
- 关闭数学公式预览功能
  ```
  $CONDA_BIN/jupyter labextension disable @jupyterlab/mathjax-extension:plugin
  $CONDA_BIN/jupyter labextension disable @jupyterlab/mathjax2-extension:plugin
  ```

3. 开启或关闭所需功能的预览后，刷新网页。

1.5 通过 VS Code 远程使用 Notebook 实例

1.5.1 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：

- **使用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。
- **在VS Code中手动连接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推荐版本、约束限制和安装指导如下。

- **VS Code推荐版本及约束限制**：
  - **VS Code 1.86版本**：暂无约束。

    图 1-110 VS Code 1.86 版本下载位置

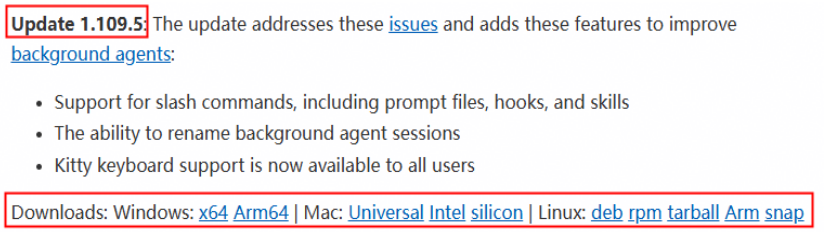
  - **VS Code 1.109.5版本**：
    - Notebook实例使用的镜像**GLIBC≥2.28**。

      您可以通过ldd --version命令查看GLIBC版本。例如，ubuntu18.04镜像的GLIBC<2.28，则只能使用VS Code 1.86及以下版本。

- Notebook通公网。

  如果Notebook不通公网，建议使用VS Code 1.86版本。如需使用VS Code 1.109.5版本，则需手动安装VS Code Server包。具体操作，请参见**如何在Notebook中离线安装本地VS Code对应版本的VS Code Server**。

图 1-111 VS Code 1.109.5 版本下载位置



- **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 <VS Code安装包名>**进行安装。

  例如，VS Code安装包名为code_1.86.2-1707854558_amd64.deb，命令示例如下：

  ```
  sudo dpkg -i code_1.86.2-1707854558_amd64.deb
  ```

  说明

  Linux系统用户，需要在非root用户进行VS Code安装。

相关文档

远程主机不满足VS Code服务器对GLIBC和libstdc++的版本要求怎么办?

1.5.2 使用 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。

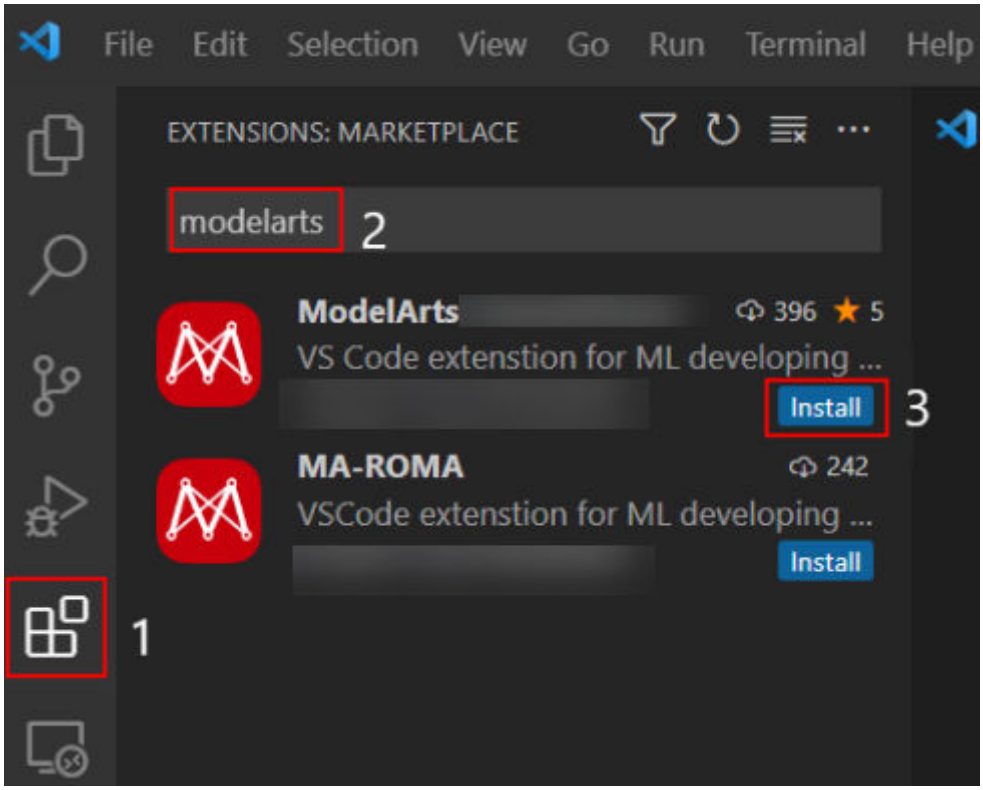
前提条件

已下载并安装VS Code。详细操作请参考**VS Code连接Notebook方式介绍**。

Step1 安装 VS Code 插件

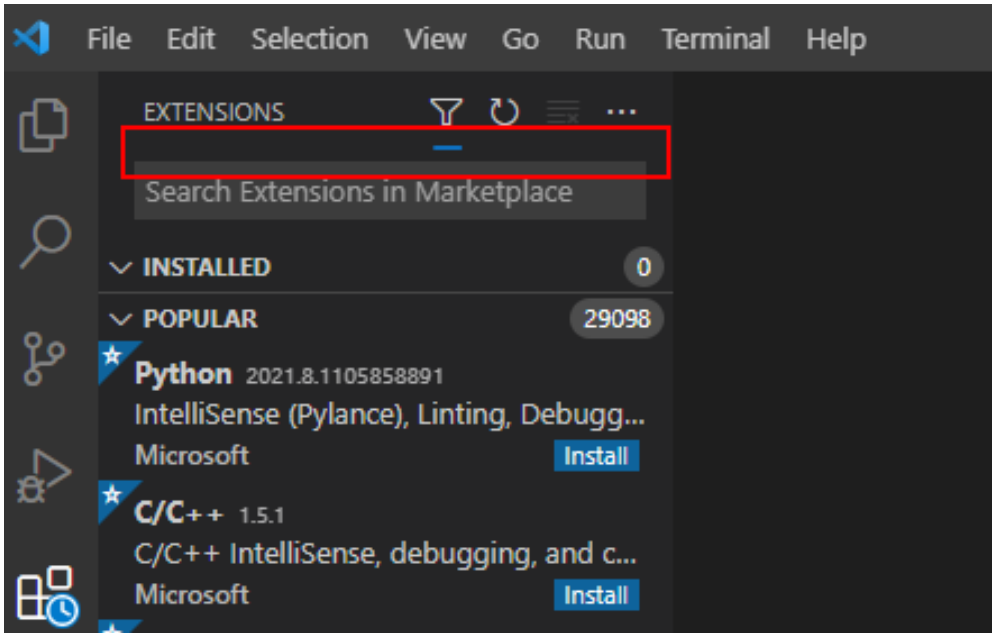
1. 在本地的VS Code开发环境中，如**图1-112**所示，在VS Code扩展中搜索“ModelArts-HuaweiCloud”并单击“安装”。

图 1-112 安装 VS Code 插件



2. 安装过程预计1~2分钟，如图1-113所示，请耐心等待。

图 1-113 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 1-114 安装完成提示

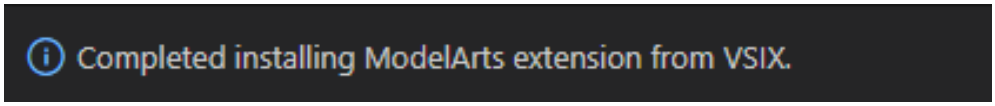
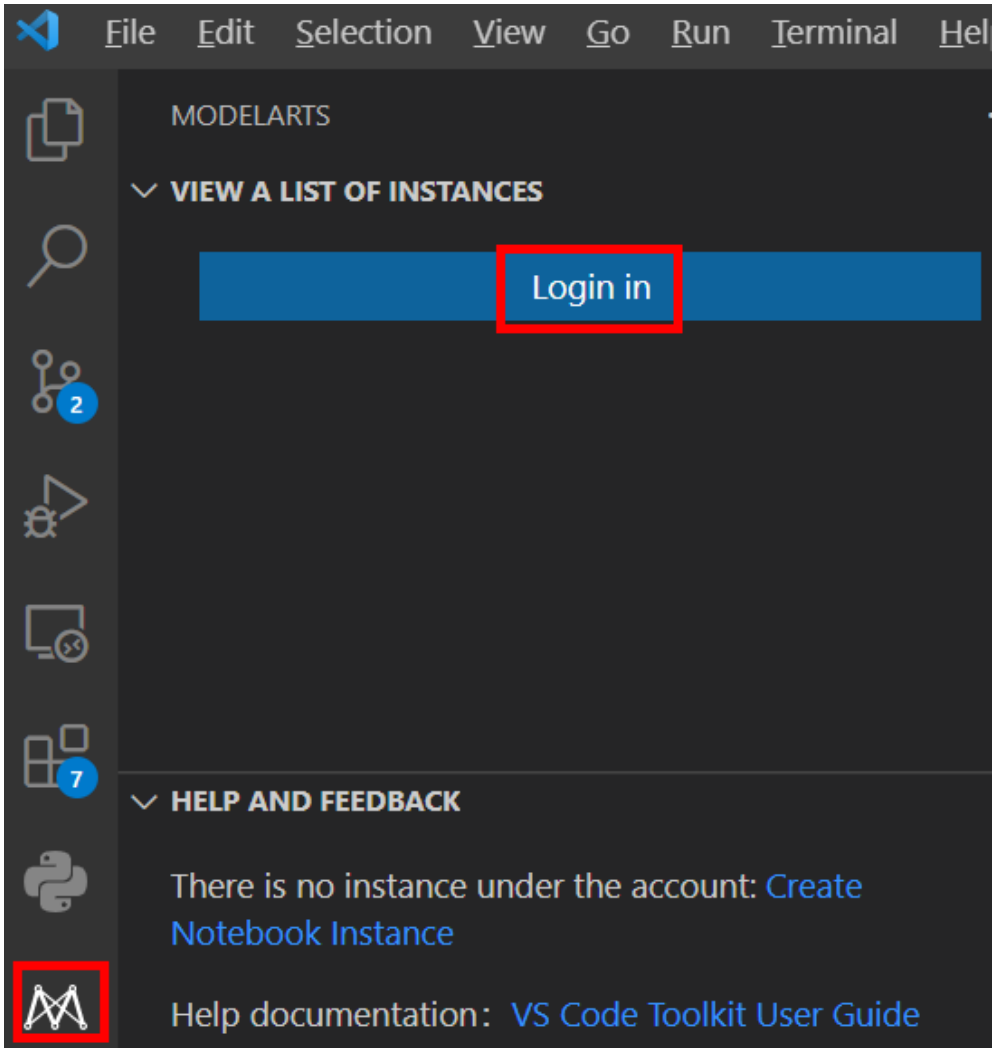
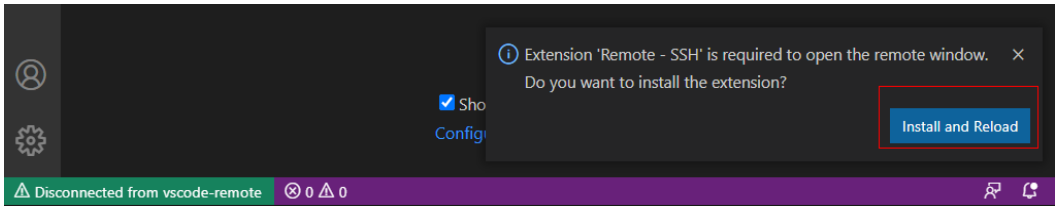


图 1-115 安装完成



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。

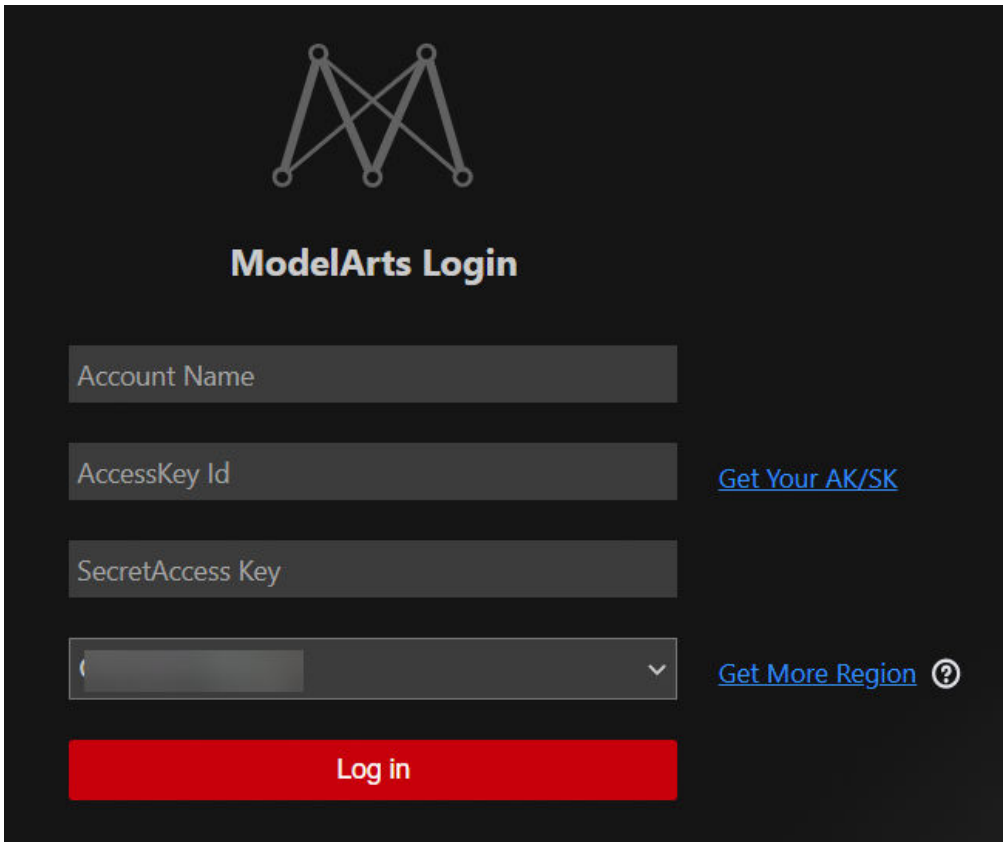
图 1-116 重新连接远程 SSH



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 1-117 登录插件

   输入如下用户登录信息，单击“登录”。

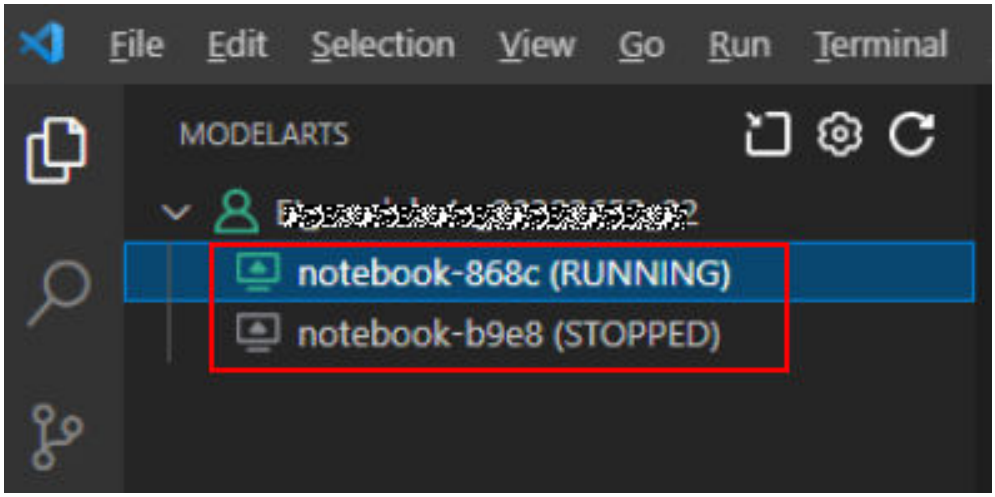
   - Name：自定义用户名，仅用于VS Code页面展示，不与任何华为云用户关联。
   - AK、SK：在“账号中心 > 我的凭证 > 访问密钥”中创建访问密钥，获取AK、SK（**参考链接**）。
   - 选择站点：此处的站点必须和远程连接的Notebook在同一个站点，否则会导致连接失败。

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 1-118 登录成功



Step3 创建 Notebook 实例

注意

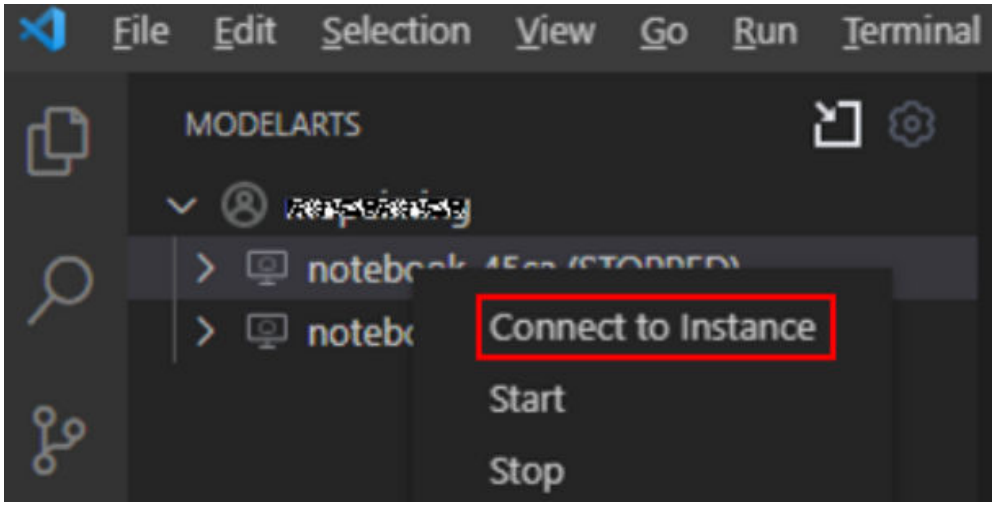
- 创建实例时，需开启“SSH远程开发”，并下载保存密钥对至本地如下目录。
  Windows：C:\Users\{{user}}
  macOS/Linux: ~
- 密钥对在用户第一次创建时自动下载，之后使用相同的密钥时不会再有下载界面（请妥善保管），或者每次都使用新的密钥对。

创建一个Notebook实例，并开启SSH远程开发，具体参见**创建Notebook实例**。

Step4 连接 Notebook 实例

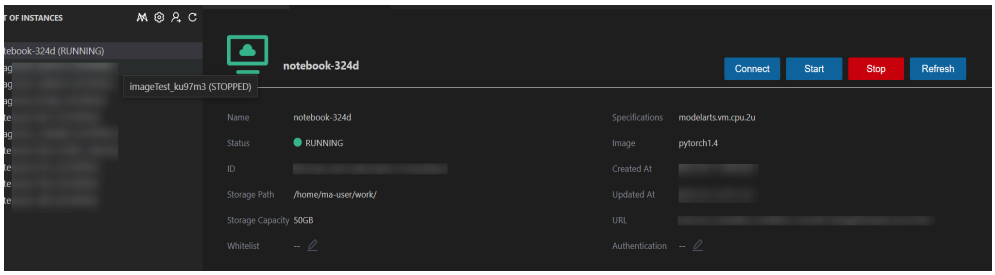
1. 在本地的VS Code开发环境中，右键单击实例名称，单击“Connect to Instance”，启动并连接Notebook实例。
   Notebook实例状态处于“运行中”或“停止”状态都可以，如果Notebook实例是停止状态，连接Notebook时，VS Code插件会先启动实例再去连接。

图 1-119 连接 Notebook 实例



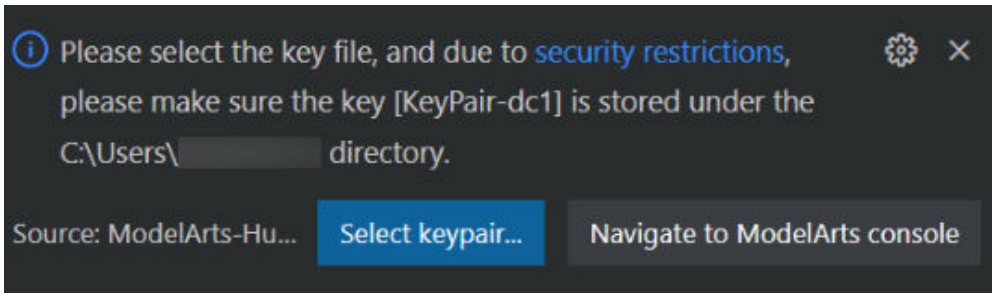
或者单击实例名称，在VS Code开发环境中显示Notebook实例详情页，单击“连接”，系统自动启动该Notebook实例并进行远程连接。

图 1-120 查看 Notebook 实例详情页



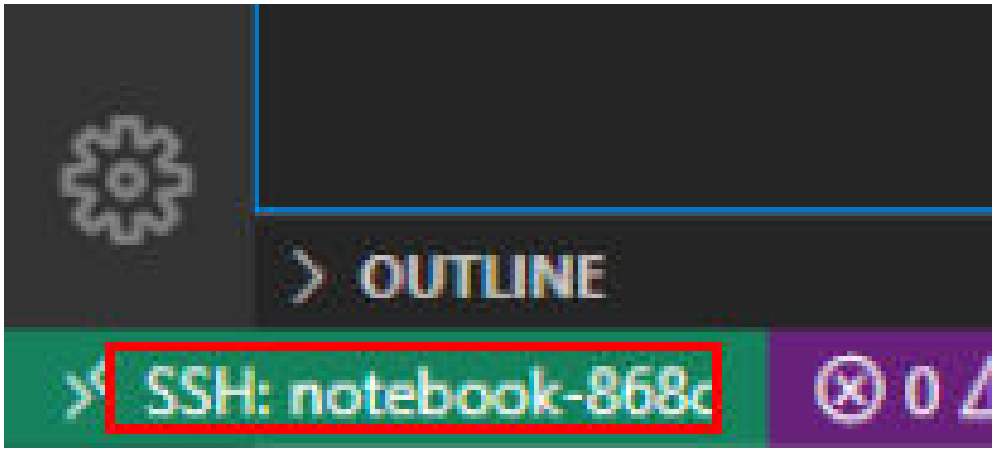
2. 第一次连接Notebook时，系统右下角会提示需要先配置密钥文件。选择本地密钥pem文件，根据系统提示单击“OK”。

图 1-121 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 1-122 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 1-123 Open Folder

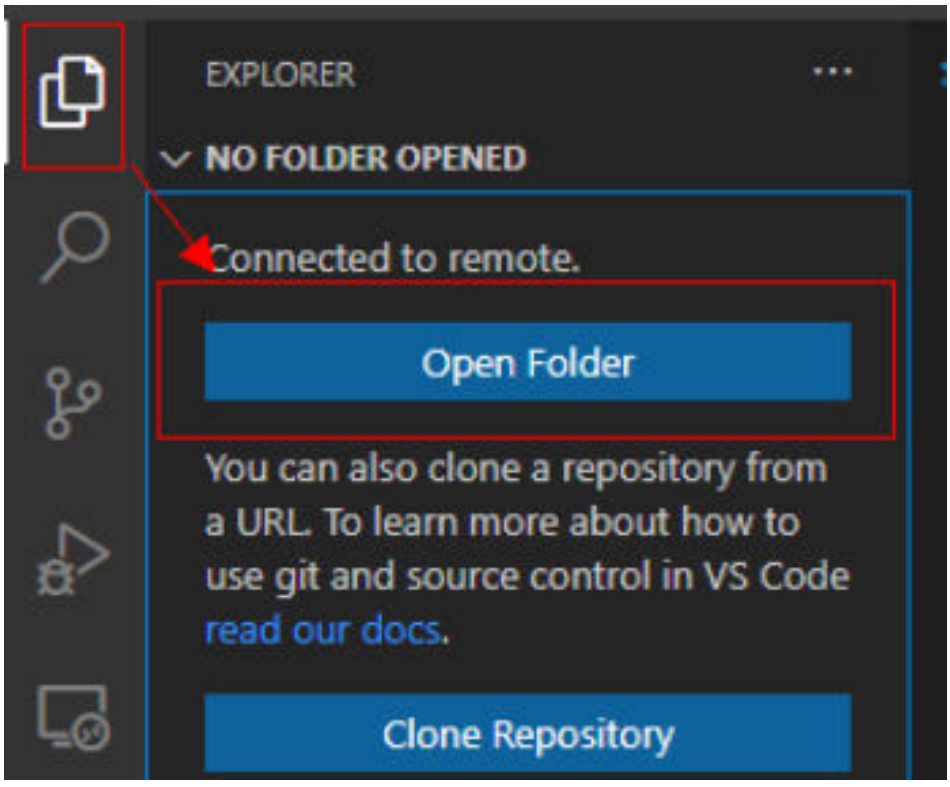
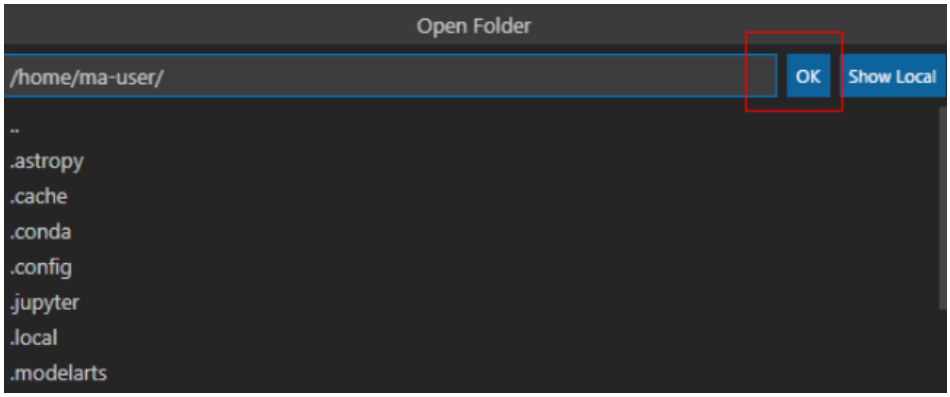


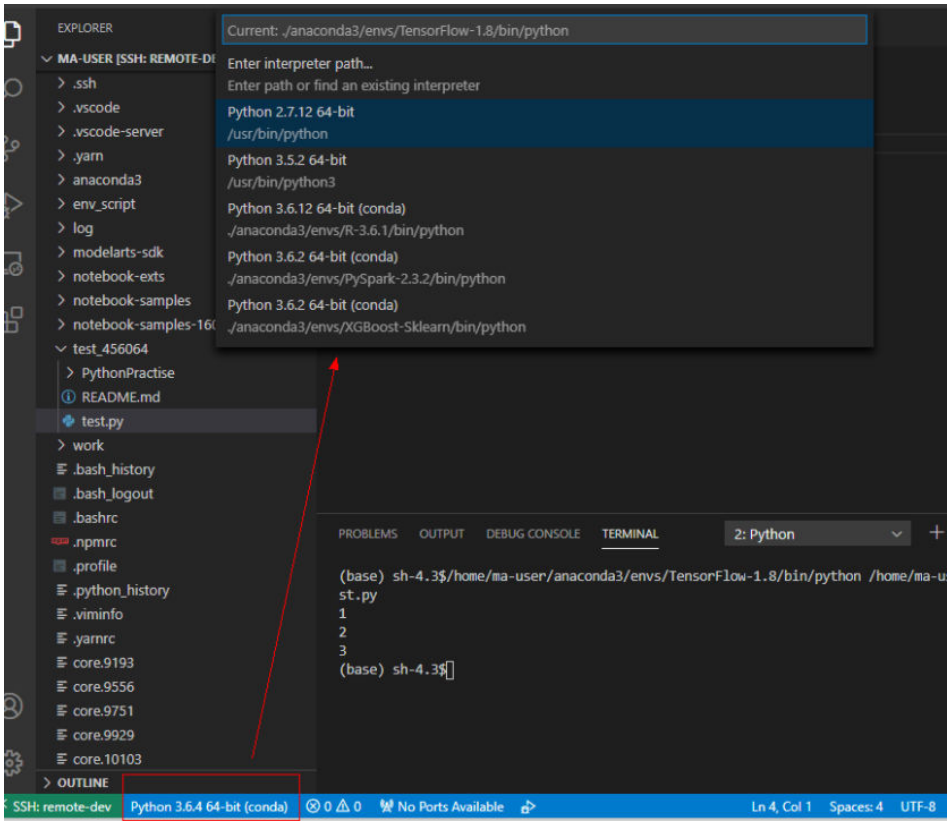
图 1-124 选择文件路径



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 1-125 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：

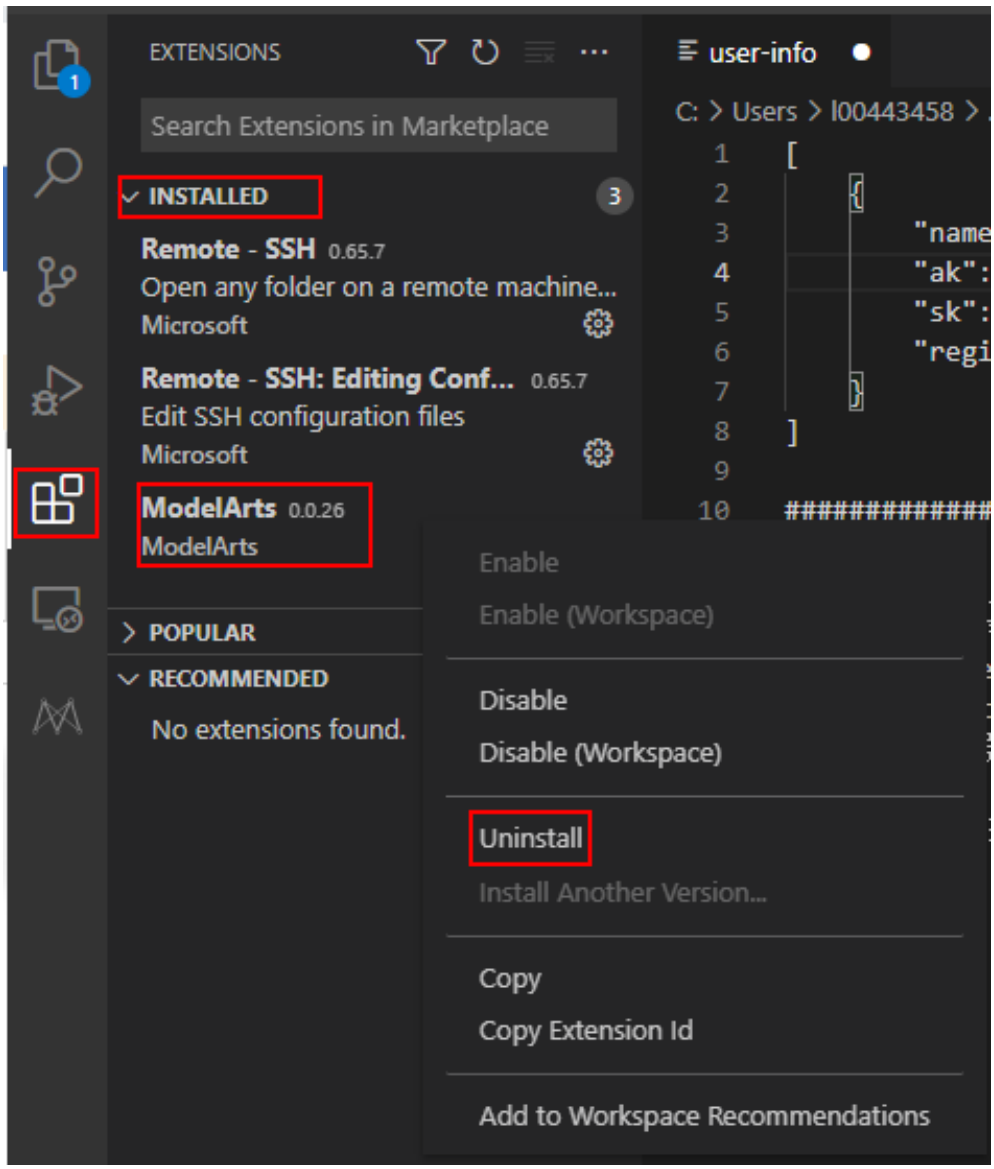
     ```
     nohup your_train_job.sh > output.log 2>&1  &  tail -f  output.log
     ```

- 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进行调试，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1-126所示。

图 1-126 卸载 VS Code 插件



1.5.3 在 VS Code 中手动连接 Notebook

本地IDE环境支持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

- 已安装VS Code推荐版本并满足约束限制，详情请参见**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，详情请参见**VS Code官方指导**。
- 已在**DEW管理控制台**创建账号密钥对或私有密钥对，并将密钥文件并存放在指定目录下。具体操作，请参见**创建密钥对**。
  - Windows：C:\Users\{{user}}
  - macOS/Linux：~

注意

如果创建账号密钥对或私有密钥对时，勾选“我同意将密钥对私钥托管”，则密钥可重复下载（在“账号密钥对”或“私有密钥对”页面的操作列单击“导出私钥”）；如果未勾选“我同意将密钥对私钥托管”，则仅创建时会自动下载，后续无法下载，请妥善保存。

图 1-127 我同意将密钥对私钥托管

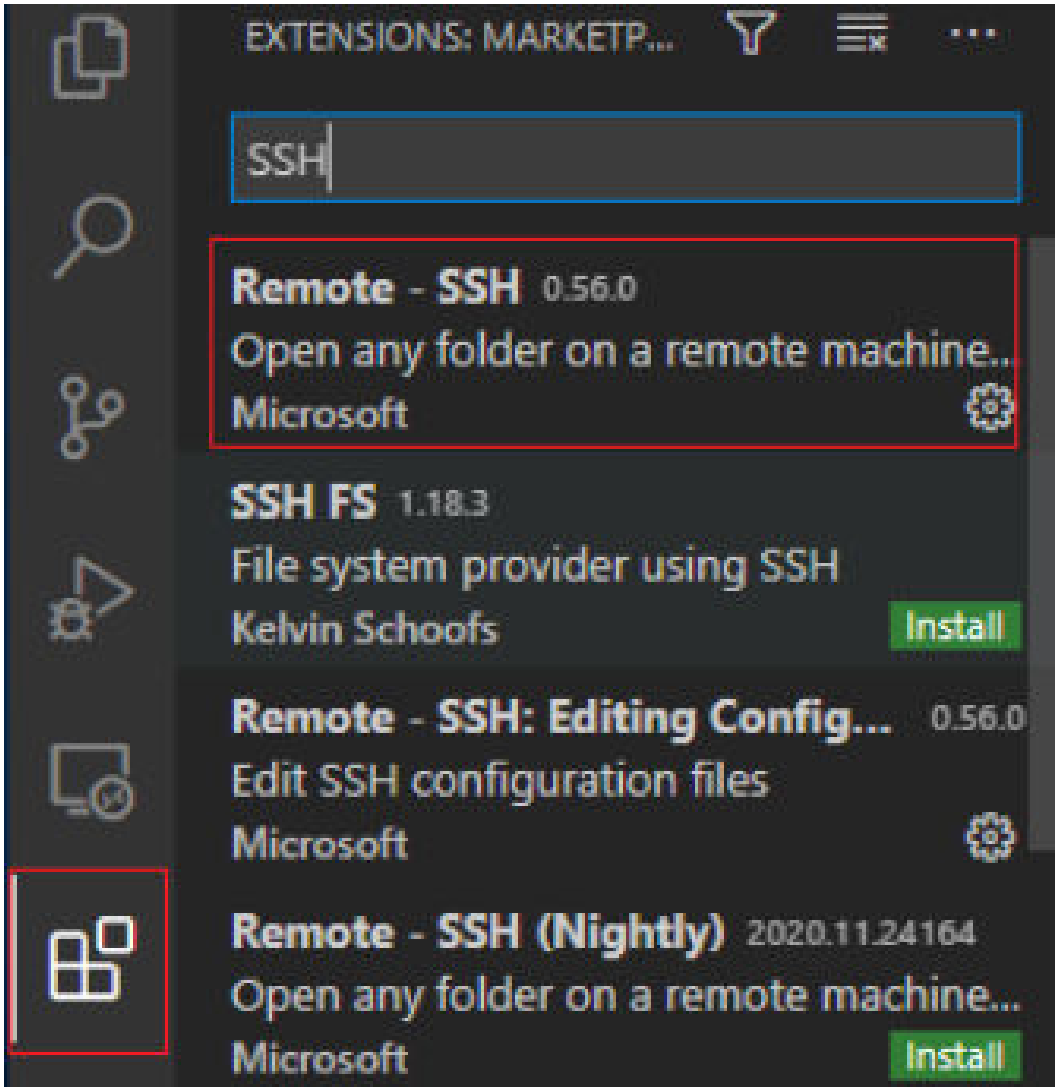
- 已创建一个Notebook实例，并开启SSH远程开发（密钥对选择已创建的账号密钥对或私有密钥对）。该实例状态必须处于“运行中”，详情请参见**创建Notebook实例**。
- 在Notebook实例详情页面的“基本信息”页签，获取SSH远程开发地址。

图 1-128 Notebook 实例详情页面

步骤一：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。

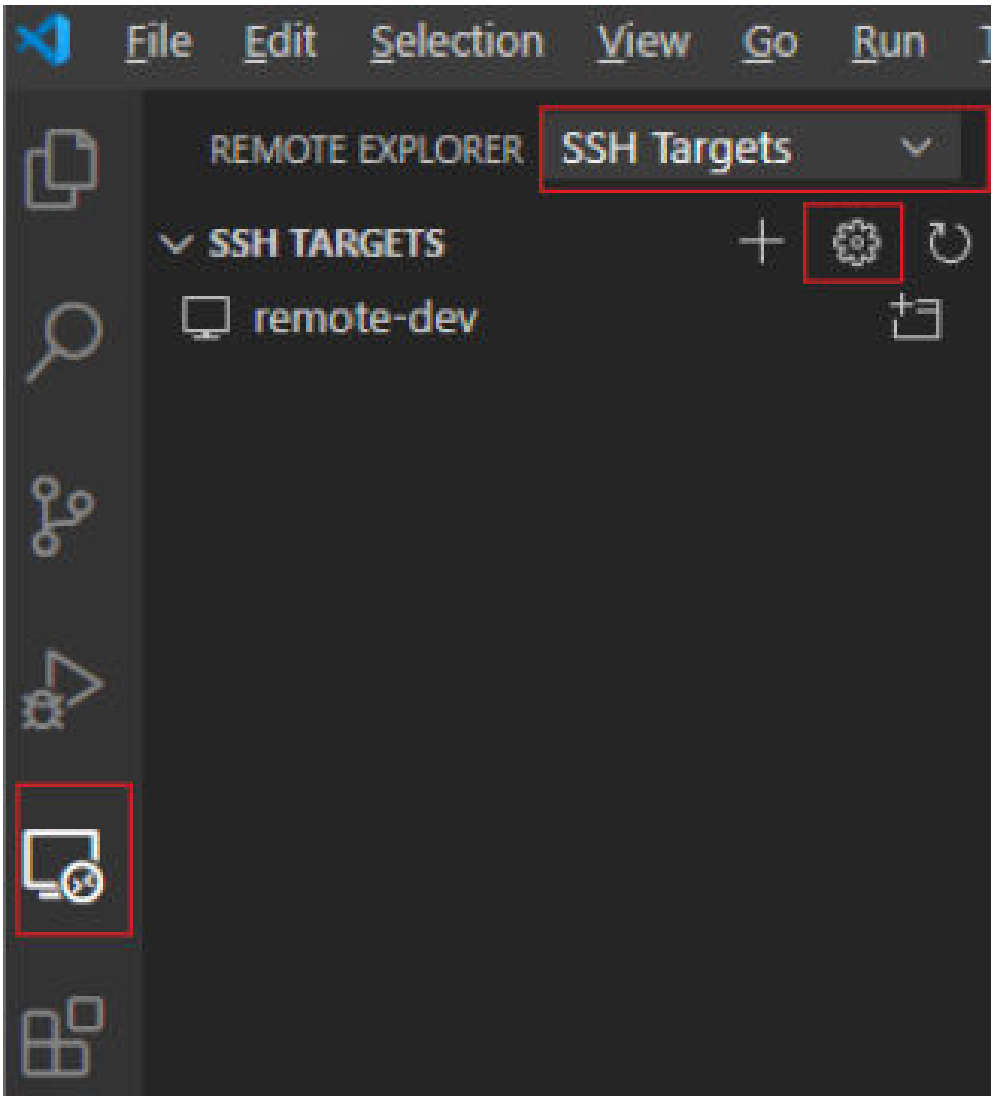
图 1-129 添加 Remote-SSH 插件



步骤二：配置 SSH

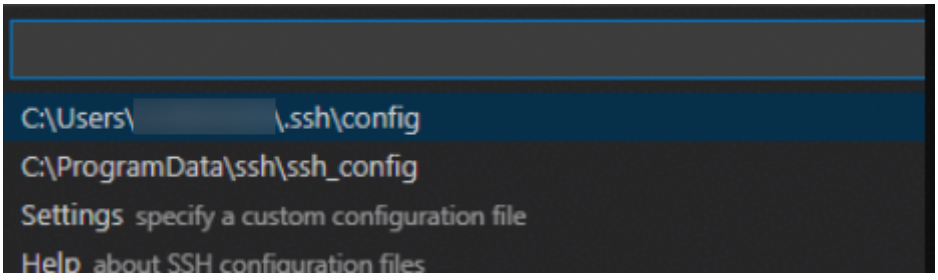
1. 在本地的VS Code开发环境中，单击左侧Remote Explorer按钮，在上方的下拉列表中选择“SSH Target”，再单击页面上的设置按钮，此时会出现SSH配置文件路径。

图 1-130 配置 SSH Targets 页面



2. 单击列表中出现的SSH路径按钮，打开并配置config文件。

图 1-131 配置 SSH Config 文件



config文件配置格式示例如下。

```
HOST remote-dev
    hostname <instance connection host>
    port <instance connection port>
    user <instance connection user>
    IdentityFile <instance connection IdentityFile>
    UserKnownHostsFile=/dev/null
```

```
    StrictHostKeyChecking no
    ForwardAgent yes
```

- 运行用户为ma-user/ma-group：假设SSH远程开发的地址为ssh://ma-user@authoring-ssh-modelarts-*****.huawei.com:31092，其中authoring-ssh-modelarts-*****.huawei.com为hostname，31092为port，ma-user为user。SSH远程开发地址从**前提条件**获取。配置示例如下：

```
HOST remote-dev
    hostname authoring-ssh-modelarts-*****.huawei.com
    port 31092
    user ma-user
    IdentityFile C:\Users\Administrator\KeyPair-1234-test.pem
    UserKnownHostsFile=/dev/null
    StrictHostKeyChecking no
    ForwardAgent yes
```

- 运行用户为root/root：假设SSH远程开发的地址为ssh://root@authoring-ssh-modelarts-*****.huawei.com:31092，其中authoring-ssh-modelarts-*****.huawei.com为hostname，31092为port，root为user。SSH远程开发地址从**前提条件**获取。配置示例如下：

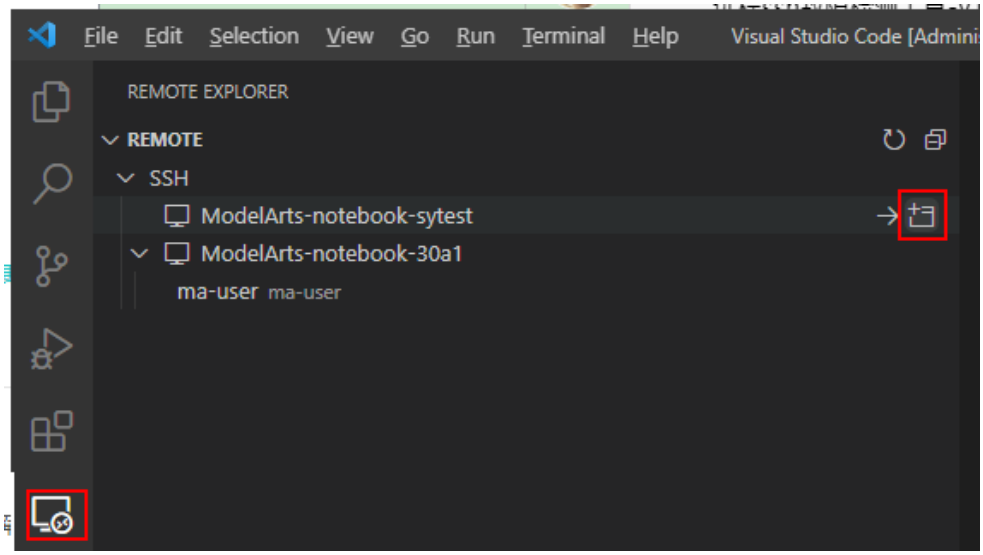
```
HOST remote-dev
    hostname authoring-ssh-modelarts-*****.huawei.com
    port 31092
    user root
    IdentityFile C:\Users\Administrator\KeyPair-1234-test.pem
    UserKnownHostsFile=/dev/null
    StrictHostKeyChecking no
    ForwardAgent yes
```

表 1-16 config 文件参数说明

参数	说明
HOST	自定义连接配置名称。
hostname	云上开发环境的访问地址，从**前提条件**SSH远程开发地址中获取。
port	云上开发环境的端口，从**前提条件**SSH远程开发地址中获取。
user	用于连接云上开发环境的用户名，从**前提条件**SSH远程开发地址中获取。
IdentityFile	存放在本地的云上开发环境私钥文件，即**前提条件**存放在指定目录下的密钥对。
UserKnownHostsFile	指定已知主机文件的路径。
StrictHostKeyChecking	是否严格检查云上开发环境的公钥。设置为no表示即使云上开发环境的公钥不匹配，也会继续连接。
ForwardAgent	支持SSH免密连接第三方远程服务器。

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 1-132 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 1-133 选择 Notebook 运行环境

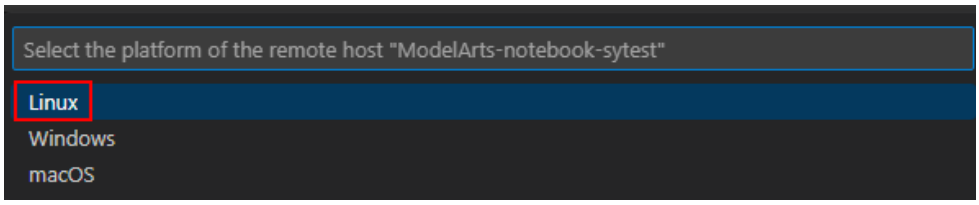
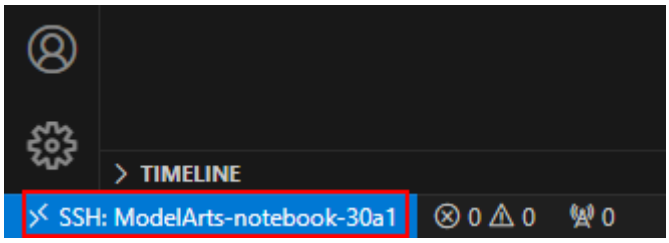


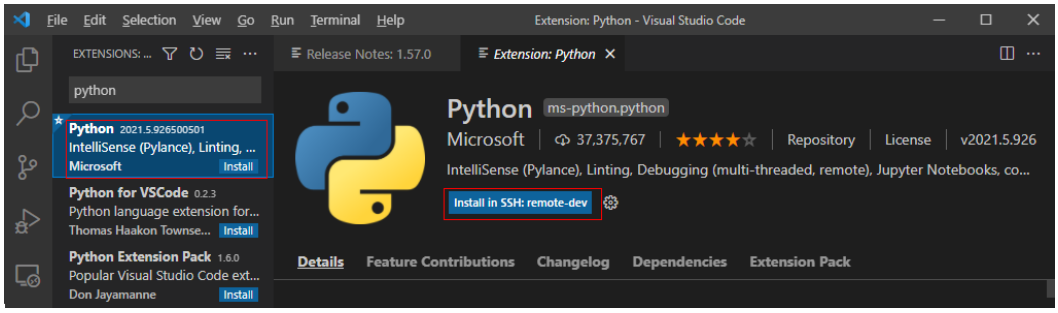
图 1-134 开发环境远程连接成功



步骤三：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 1-135 安装云端 Python 插件



如果安装远端的Python插件不成功时，建议通过离线包的方式安装。

步骤四：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
```

1.5.4 在 VS Code 中上传下载文件

在 VS Code 中上传数据至 Notebook

小于或等于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

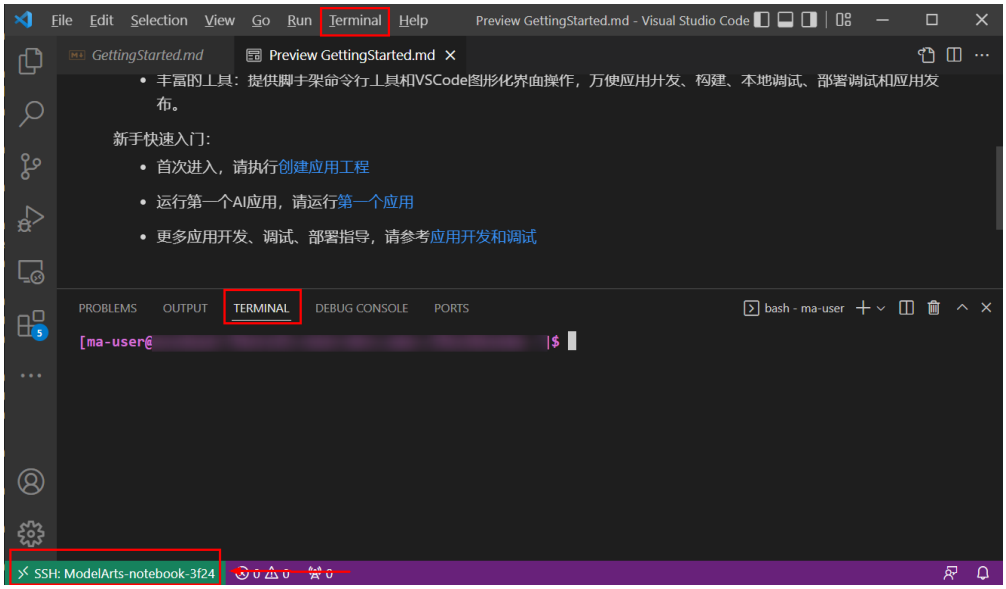
1. 上传数据至OBS。具体操作请参见**上传文件至OBS桶**。

   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 1-136 远程连接 VS Code 环境开启 Terminal



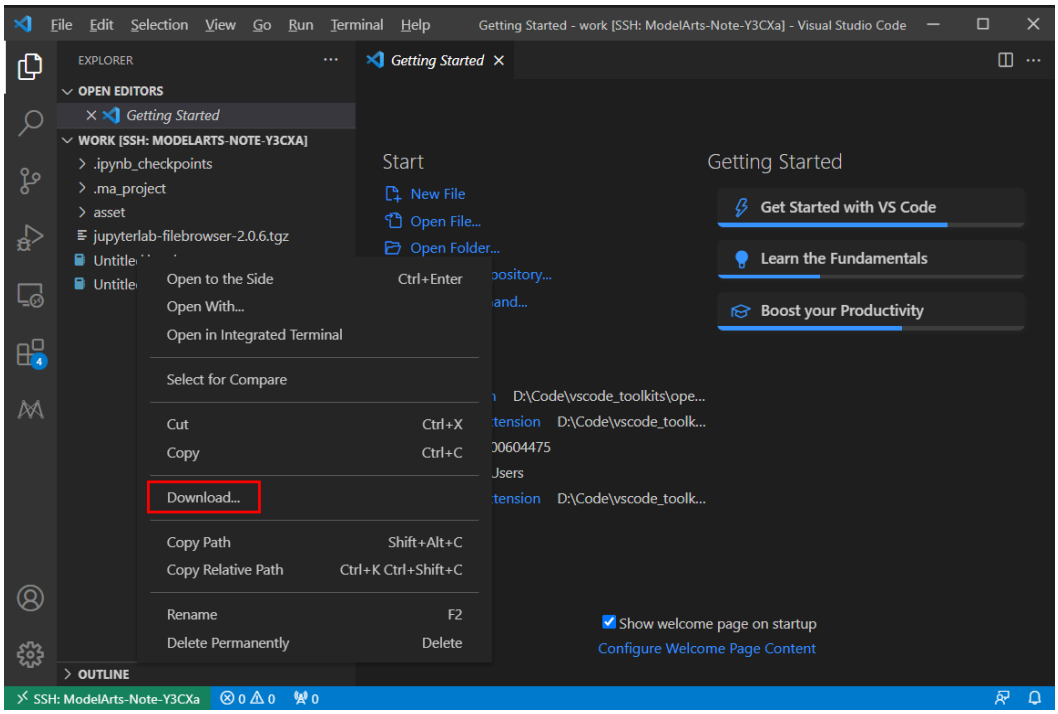

```
#手动source进入开发环境
cat /home/ma-user/README
#然后选择要source的环境
source /home/ma-user/miniconda3/bin/activate MindSpore-python3.7-aarch64
#输入python并回车，进入python环境
python
```

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 1-137 VS Code 环境下载 Notebook 中的文件至本地



1.6 通过 SSH 工具远程使用 Notebook

本文介绍在Windows环境中使用CMD和MobaXterm工具通过SSH远程登录Notebook实例的操作步骤。

约束限制

使用SSH连接Notebook实例的约束限制如下：

- 本地用户密钥和权限必须匹配。
- 本地用户密钥需要存放在指定目录下。
  - Windows：C:\Users\{{user}}
  - macOS/Linux：~

    说明

    在macOS和Linux系统中，~ 表示当前用户的主目录。
- 远端镜像中的ma-user或root用户不能被锁定。
- 远端的~/.ssh目录权限建议设置为750或755。
- 本地或远端的OpenSSH版本不能低于8.0。
- 建议同时建立的连接数不超过10个。

如果遇到SSH连接问题，请参考**本地通过SSH连接Notebook实例的故障排查**进行解决。

前提条件

- 已在**DEW管理控制台**创建账号密钥对或私有密钥对，并将密钥文件并存放在指定目录下。具体操作，请参见**创建密钥对**。
  - Windows：C:\Users\{{user}}
  - macOS/Linux：~

注意

如果创建账号密钥对或私有密钥对时，勾选“我同意将密钥对私钥托管”，则密钥可重复下载（在“账号密钥对”或“私有密钥对”页面的操作列单击“导出私钥”）；如果未勾选“我同意将密钥对私钥托管”，则仅创建时会自动下载，后续无法下载，请妥善保存。

图 1-138 我同意将密钥对私钥托管

- 已创建一个Notebook实例，并开启SSH远程开发（密钥对选择已创建的账号密钥对或私有密钥对）。该实例状态必须处于“运行中”。

图 1-139 创建 Notebook-SSH 远程开发

  - 关于如何新建Notebook实例，请参见**创建Notebook实例**。
  - 关于如何在已有Notebook实例开启SSH远程开发，请参见**配置Notebook SSH远程连接**。
- 在Notebook实例详情页面的“基本信息”页签，获取SSH远程开发的访问地址。

图 1-140 Notebook 实例详情页面

方式一：使用 CMD 连接 Notebook 实例

下文以Windows系统为例进行说明。

1. 打开CMD。
   - 方式一：使用开始菜单
     i. 在电脑左下角，单击“开始”，或者按键盘上的Windows键。
     ii. 在开始菜单中，输入CMD，单击“命令提示符”。
   - 方式二：使用运行对话框
     i. 在键盘按“Windows+R”键，打开“运行”对话框。
     ii. 在“运行”对话框中输入CMD，按Enter键或单击“确定”。
2. 在CMD中，进入密钥所在目录并执行以下命令，连接Notebook实例。

```
ssh -o StrictHostKeyChecking=no -i {密钥文件名} {启动用户}@{域名} -p {端口}
```

命令中的{密钥文件名}、{启动用户}、{域名}、{端口}请替换为您实际的信息，从**前提条件**获取。

例如，密钥文件名为KeyPair-1234-test.pem，SSH远程开发的访问地址为ssh://ma-user@authoring-ssh-modelarts-*****.huawei.com:31092，其中ma-user为启动用户，authoring-ssh-modelarts-*****.huawei.com为域名，31092为端口。命令示例如下：

```
ssh -o StrictHostKeyChecking=no -i KeyPair-1234-test.pem ma-user@authoring-ssh-modelarts-*****.huawei.com -p 31092
```

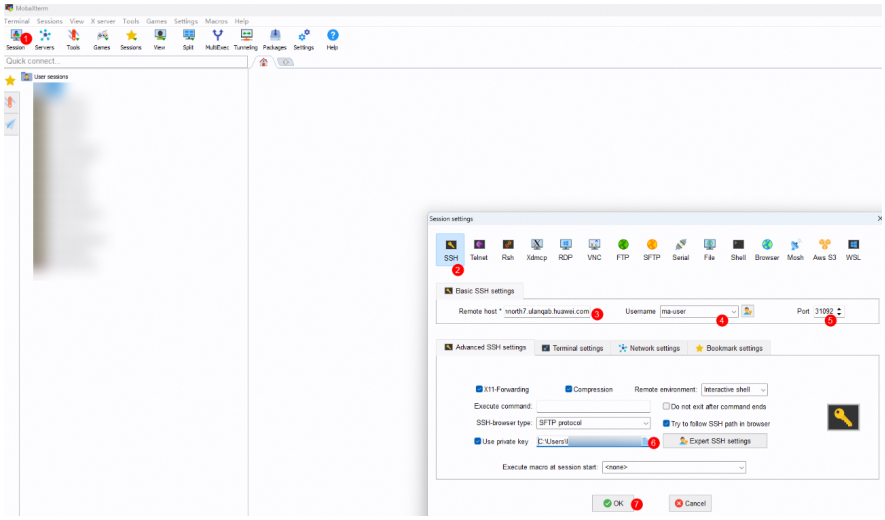
如下图所示，出现交互界面，表示连接成功。

图 1-141 使用 CMD 连接 Notebook 实例

方式二：使用 MobaXterm 连接 Notebook 实例

1. 下载并安装**MobaXterm**。
2. 打开MobaXterm，在左上角单击“Session”，在“Session settings”对话框的SSH页签，配置Remote host、Username、Port和Use private key，单击“OK”。

图 1-142 配置 Session



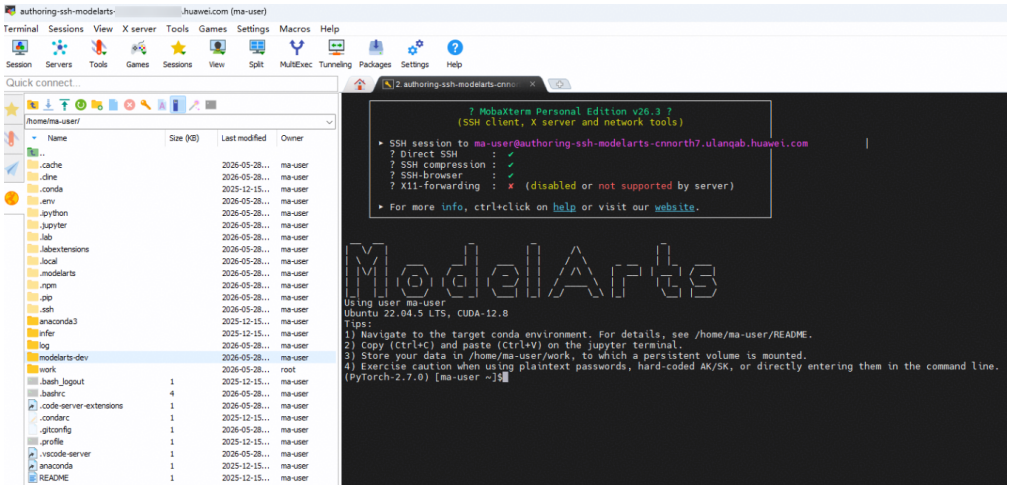
SSH远程开发的访问地址、密钥文件的路径从**前提条件**获取。例如，SSH远程开发的访问地址为ssh://ma-user@authoring-ssh-modelarts-*****.huawei.com:31092，配置示例如下。

表 1-17 配置说明

配置项		说明	示例值
Basic SSH settings	Remote host	远程主机的地址，从SSH远程开发的访问地址中获取。	authoring-ssh-modelarts-*****.huawei.com
	Username	用于连接远程主机的用户名，从SSH远程开发的访问地址中获取。	ma-user
	Port	用于连接远程主机的端口号，从SSH远程开发的访问地址中获取。	31092
Advanced SSH settings	Use private key	密钥文件的路径。从**前提条件**获取。	C:\Users***\KeyPair-1234-test.pem

如下图所示，出现交互界面，表示连接成功。

图 1-143 使用 MobaXterm 连接 Notebook 实例



1.7 Notebook 存储配置

1.7.1 Notebook 存储类型说明

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。

在SFS Turbo与OBS桶进行存储联动时，如果将其挂载到Notebook中进行使用，会存在一些使用限制，详情请参见**管理SFS Turbo文件系统与OBS桶的存储联动**。

表 1-18 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
对象存储 OBS - 并行文件系统	**说明** • 并行文件系统当前处于受限使用阶段，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并行文件系统。 适合直接使用并行文件系统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。将存储在并行文件系统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。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**存储挂载方式说明**。 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。	并行文件系统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，推荐使用并行文件系统挂载。 **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，可能导致Notebook卡顿。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** 并行文件系统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 OBS - 对象桶	**说明** • 对象存储 OBS - 对象桶当前处于受限使用阶段，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。 小文件频繁读写性能较差，可能导致Notebook卡顿。模型训练和大文件解压等场景慎用。
高性能弹性文件服务 SFS Turbo	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。
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。

相关文档

- ModelArts支持动态存储和扩展存储两种方式，详情请参见**存储挂载方式说明**。
- 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， 例如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，详情请参见**动态扩充云硬盘EVS容量**。

1.7.2 Notebook 存储挂载方式说明

ModelArts支持动态存储和扩展存储两种方式。

- 动态存储：支持在实例运行期间动态挂载存储，无需停止实例，实例启动后自动重新挂载，数据持久保留。适合需要运行时灵活挂载或临时扩展存储的场景。
- 扩展存储：基于云原生持久化存储方案，提供稳定可靠的存储管理，但挂载或变更存储配置需停止实例。适用于规划好的、长期使用的数据卷。

关于存储类型的详细说明，请参见**Notebook存储类型说明**。

约束限制

- 动态存储：
  - 仅支持专属资源池Notebook实例。
  - 动态存储仅支持实例运行时操作。
- 扩展存储：
  - 仅支持实例启动失败和停止时操作。
  - 最多可添加30个扩展存储。
  - 在SFS Turbo与OBS桶进行存储联动时，如果将其挂载到Notebook中进行使用，会存在一些使用限制，详情请参见**管理SFS Turbo文件系统与OBS桶的存储联动**。

添加动态存储

1. 登录**ModelArts管理控制台**，在左侧导航栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 在Notebook页面，单击实例名称，进入Notebook实例详情页面。
3. 单击“存储配置”页签，在“动态存储”区域单击“添加动态存储”。
4. 在“添加动态存储”面板，配置相关信息，单击“确定”。

表 1-19 “存储类型”选择“对象存储 OBS - 并行文件系统”

参数	说明	示例值
存储地址	直接输入地址，或者单击图标选择存储地址。 存储地址必须以obs://或/开头，以/结尾，且除前缀外不得出现//。	obs://bucketname/path/
挂载路径	仅支持被挂载至/data/子目录下。	/data/test1/

说明

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/

5. 当存储的状态显示为“已挂载”，表明存储挂载成功。

   当存储不再需要时，可单击操作列的“卸载”，在“卸载”对话框单击“确定”。

卸载不影响存储中的数据，如果要再次在Notebook中使用，重新挂载即可。

添加扩展存储

1. 登录**ModelArts管理控制台**，在左侧导航栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 在Notebook页面，单击实例名称，进入Notebook实例详情页面。
3. 单击“存储配置”页签，在“扩展存储”区域单击“添加扩展存储”。
4. 在“添加扩展存储”面板，配置相关信息，单击“确定”。

表 1-20 “存储类型”选择“对象存储 OBS - 对象桶”或“对象存储 OBS - 并行文件系统”

参数	说明	示例值
凭据	挂载存储场景使用DEW凭据时，需要包含“accessKeyId”和“secretAccessKey”，分别对应用户的AK、SK信息。请确保填写内容正确有效，否则会导致功能异常。	AKSK_05
存储地址	直接输入地址，或者单击 图标选择存储地址。 存储地址必须以obs://或/开头，以/结尾，且除前缀外不得出现//。	obs://bucketname/path/
挂载路径	挂载路径不能为空，不允许挂载到黑名单目录, 并且以斜杠（/）开头和结尾，仅包含字母、数字、下划线和中划线。	/temp/

表 1-21 “存储类型”选择“高性能弹性文件服务 SFS Turbo”

参数	说明	示例值
文件系统	在下拉框选择文件系统。	sfs-turbo
文件系统目录	文件系统目录不能为空，首尾必须为/ ，且仅支持字母、数字、下划线、中划线。	/demo/
挂载路径	挂载路径不能为空，不允许挂载到黑名单目录, 并且以斜杠（/）开头和结尾，仅包含字母、数字、下划线和中划线。	/temp/

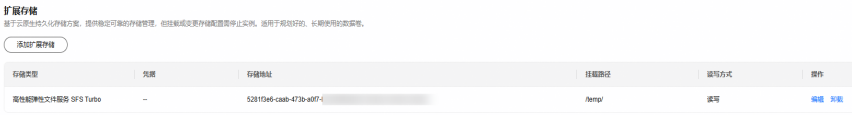
说明

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/

5. 扩展存储添加后，可在“扩展存储”区域看到新增的扩展存储。

图 1-144 扩展存储



- 如果需要修改扩展存储信息，可单击操作列的“编辑”，修改相关配置，单击“确定”。
- 当扩展存储不再需要时，可单击操作列的“卸载”，在“卸载扩展存储”对话框单击“确定”。

1.7.3 Notebook 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 1-145 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**，在左侧导航栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：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。
3. 在“基本信息”页签的“资源信息”区域，单击“存储容量”下方的“扩容”。
4. 设置目标容量，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。

相关文档

- **Notebook存储类型说明**
- **Notebook存储挂载方式说明**

1.8 管理 Notebook 实例

1.8.1 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“Notebook”页面，选择“全部”，可以看到IAM项目下所有子账号创建的Notebook实例。

  图 1-146 查看所有

- 筛选Notebook实例：
  - 在页面上方，单击“计费中”或“运行中”，可以筛选对应的Notebook实例。鼠标悬浮“计费中”，可以查看涉及处于计费中的计算资源和存储资源的Notebook实例个数。
  - 搜索框默认按照名称搜索，您也可以按需选择名称、ID、资源池ID、状态、计费资源类型或资源标签进行筛选。

给子账号配置操作所有 Notebook 实例的权限

当子账号被授予“modelarts:notebook:listAllNotebooks”和“iam:users:listUsers”权限时，在Notebook页面上，单击“查看所有”，可以看到并打开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。

⚠ 注意

主账号给子用户配置“modelarts:notebook:listAllNotebooks”后，子用户默认会具有查看并打开其他子用户创建的Notebook的权限，请慎重配置。

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，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。

   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。

1.8.2 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。

注意事项：

- 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。
- 更换镜像后挂载目录下的数据不会丢失（例如/home/ma-user/work 、/data 等），非挂载目录下的会重置（例如/home/ma-user）。

操作步骤：

1. 登录**ModelArts管理控制台**，在左侧菜单栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 在Notebook列表，单击目标Notebook实例操作列的“更多 > 变更镜像”。
3. 在“变更镜像”页面，按需选择预置镜像或自定义镜像，勾选“我已阅读并同意《镜像免责声明》《云商店通用商品用户协议》”，单击“确定”。

   镜像列表中的镜像会按照硬件机型维度进行匹配，如果不匹配则无法选择，请以实际环境为准。

   变更成功后，在Notebook页面的“镜像”列，可以查看变更后的镜像。

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**，在左侧菜单栏按需选择以下操作。

- 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
- 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。

2. 在Notebook列表，单击目标Notebook实例“操作”列的“更多 > 变更实例规格”。
3. 在“变更实例规格”页面，按需选择目标规格，勾选“我已知晓变更可能存在的风险，同意进行变更。”，单击“确认”。

   变更成功后，在Notebook页面的“实例规格”列，可以查看变更后的规格。

   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。

配置 Notebook SSH 远程连接

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。“SSH远程开发”开关打开后不可关闭。

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**，在左侧菜单栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 在Notebook列表，进入Notebook实例详情页。在“基本信息”页签打开“SSH远程开发”开关，更新密钥对，单击“确定”。

设置节点亲和性调度

仅允许停止或启动失败状态且“资源池类型”为“专属资源池”的Notebook实例开启、修改或关闭节点亲和性调度。关于停止Notebook实例的操作步骤，请参见**启动/停止/删除Notebook实例**。

开启节点亲和性调度

1. 登录**ModelArts管理控制台**，在左侧导航栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 任选以下方式开启节点亲和性调度。
   - 方式一：
     i. 在“Notebook”页面的“操作”列，单击已停止或启动失败的Notebook实例对应的“更多 > 节点亲和性调度”。
     ii. 在“提示”对话框，单击“配置”，在“亲和调度策略”对话框，按需选择亲和类型、强度和节点，单击“确定”。
     - “亲和类型”选择“节点亲和”，“强度”选择“弱”：尽量将Pod调度到指定节点，不保证成功。

- “亲和类型”选择“节点亲和”，“强度”选择“强”：严格将Pod调度到指定节点，否则不执行调度。
- “亲和类型”选择“节点反亲和”，“强度”选择“弱”：避免将Pod调度到指定节点，不保证成功。
- “亲和类型”选择“节点反亲和”，“强度”选择“强”：禁止将Pod调度到指定节点，否则不执行调度。选择节点时，不支持全部勾选，需要至少保留1个可用节点，否则无法执行调度策略。

iii. 在“Notebook”页面，单击已停止或启动失败的Notebook实例名称，在“基本信息”页签的“资源信息”区域，可以看到节点亲和性调度的配置信息，说明节点亲和性调度已开启成功。

- 方式二：

i. 在“Notebook”页面，单击已停止或启动失败的Notebook实例名称。

ii. 在“基本信息”页签的“资源信息”区域，单击“节点亲和性调度”下方的“开启”，在“亲和调度策略”对话框，按需选择亲和类型、强度和节点，单击“确定”。

iii. 在“节点亲和性调度”下方，可以看到节点亲和性调度的配置信息，说明节点亲和性调度已开启成功。

修改节点亲和性调用

1. 登录**ModelArts管理控制台**，在左侧菜单栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 任选以下方式修改节点亲和性调度。
   - 方式一：
     i. 在“Notebook”页面的“操作”列，单击已停止或启动失败的Notebook实例对应的“更多 > 节点亲和性调度”。
     ii. 在“节点亲和性调度”对话框，选择“修改亲和策略”，单击“确定”。
     iii. 在“亲和调度策略”对话框，按需修改亲和类型、强度或节点，单击“确定”。
     iv. 在“Notebook”页面，单击已停止或启动失败的Notebook实例名称，在“基本信息”页签的“资源信息”区域，可以看到节点亲和性调度的配置信息已更新，说明节点亲和性调度修改成功。
   - 方式二：
     i. 在“Notebook”页面，单击已停止或启动失败的Notebook实例名称。
     ii. 在“基本信息”页签的“资源信息”区域，在“节点亲和性调度”下方，单击图标，在“节点亲和性调度”对话框，选择“修改亲和策略”，单击“确定”。
     iii. 在“亲和调度策略”对话框，按需修改亲和类型、强度或节点，单击“确定”。
     iv. 在“节点亲和性调度”下方，可以看到节点亲和性调度的配置信息已更新，说明节点亲和性调度修改成功。

关闭节点亲和性调度

节点亲和性调度关闭后，将根据系统调度规则进行智能调度。

1. 登录**ModelArts管理控制台**，在左侧菜单栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
1. 任选以下方式关闭节点亲和性调度。
   a. 方式一：
      i. 在“Notebook”页面的“操作”列，单击已停止或启动失败的Notebook实例对应的“更多 > 节点亲和性调度”。
      ii. 在“节点亲和性调度”对话框，选择“关闭亲和策略”，单击“确定”，在“提示”对话框，单击“确定”。
      iii. 在“Notebook”页面的“操作”列，单击已停止或启动失败的Notebook实例对应的“更多 > 节点亲和性调度”，出现“提示”对话框，提示当前暂未开启节点亲和性调度，说明节点亲和性调度关闭成功。
   b. 方式二：
      i. 在“Notebook”页面，单击已停止或启动失败的Notebook实例名称。
      ii. 在“基本信息”页签的“资源信息”区域，在“节点亲和性调度”下方，单击图标，在“节点亲和性调度”对话框，选择“关闭亲和策略”，单击“确定”。
      iii. 在“提示”对话框，单击“确定”。
      iv. 在“节点亲和性调度”下方，出现“开启”按钮，说明节点亲和性调度关闭成功。

编辑标签

您可以对Notebook实例的标签进行修改、新增或删除操作。

1. 登录**ModelArts管理控制台**，在左侧导航栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 任选以下方式编辑标签。
   a. 方式一：在“Notebook”页面的“标签”列，鼠标悬浮至Notebook实例对应的图标，单击“编辑标签”。

图 1-147 编辑标签

b. 方式二：在“Notebook”页面，单击Notebook实例名称，在“基本信息”页签的“基础信息”区域，鼠标悬浮至“标签”下的图标，单击“编辑标签”。

3. 在“编辑标签”面板，您可以对标签进行修改、新增或删除操作。
   - 修改标签：在已有的标签中，按需修改或选择标签键，修改标签值，单击“确定”。
   - 新增标签：单击“添加标签”，按需输入或选择标签键、标签值，单击“确定”。一个Notebook实例最多可添加20个标签。
   - 删除标签：在已有的标签右侧，单击“删除”，然后单击“确定”。

   如果您需要使用同一标签标识多种云资源，即所有服务均可在标签输入框下拉选择同一标签，建议在TMS中创建预定义标签，具体操作，请参见**创建预定义标签**。

   说明

   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。

   在“Notebook”页面的“标签”列或者在Notebook实例详情页面的“基本信息”页签，可以查看编辑后的标签。

1.8.3 启动/停止/删除 Notebook 实例

本文介绍如何启动、停止和删除Notebook实例。

约束限制

当专属资源池的节点被设置为热备节点时，该节点将不支持运行Notebook实例，例如创建、启动Notebook实例等。关于如何关闭热备节点，请参见**修复专属资源池故障节点**。

启动/停止 Notebook 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**，在左侧菜单栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 执行如下操作启动或停止Notebook。

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。

- **启动Notebook**：

  只有处于“停止”状态的Notebook可以执行启动操作。

  i. 单击“操作”列的“启动”。

  ii. 在“启动Notebook实例”对话框，按需选择WebIDE和自动停止，单击“确定”。

- **自动停止Notebook**：

  只有处于“运行中”状态的Notebook可以执行自动停止操作。

  开启后，可设置Notebook实例自动停止方式及对应时长，超出您预设的时长，它将自动停止运行 (可能存在2-5分钟的延迟，此过程正常计费)。

  i. 单击“操作”列的“更多 > 设置自动停止”。

  ii. 在“重置Notebook自动停止时间”对话框，勾选“自动停止”，按需选择停止方式和停止时长，阅读提示信息，单击“确定”。

- **停止Notebook**：

  只有处于“运行中”状态的Notebook可以执行停止操作。

  i. 单击“操作”列的“停止”。

  ii. 在“停止实例”对话框，阅读提示信息，单击“确定”。

删除 Notebook 实例

针对不再使用的Notebook实例，可以删除以释放资源。**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。**

1. 登录**ModelArts管理控制台**，在左侧菜单栏按需选择以下操作
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入Notebook页面。
2. 在Notebook列表中，单击操作列的“更多 > 删除”，在“删除Notebook实例”对话框，确认信息无误，输入“DELETE”，单击“确定”，完成删除操作。

1.8.4 使用 root 启动 Notebook 实例

在ModelArts平台中，默认的用户名为ma-user，用户组为ma-group。如果在使用过程中挂载了SFS Turbo、使用自定义镜像或直接上传文件，需要提前修改文件权限，确保ma-user用户具有读取权限，否则可能会遇到“Permission denied”错误。下文将介绍使用root启动Notebook实例的适用场景及具体操作。

适用场景

仅新网络模式下专属资源池支持root启动Notebook实例。关于如何基于新网络创建专属资源池，请参见**创建专属资源池**。

- 华为云ModelArts针对专属资源池网络进行安全加固和优化，仅新网络模式下专属资源池支持root启动Notebook实例，旧网络模式的专属资源池无法支持此功能，且将于2025年10月15日起逐步限制使用。查看是否为旧网络模式以及公告详情请参见**ModelArts专属资源池网络调整公告**。
- 公共资源池不支持使用root启动Notebook实例。

开启指定运行用户

创建Notebook时，您可以在“更多配置”中，开启“指定运行用户”。在启动Notebook实例时，ModelArts支持以下两种运行用户配置。不同资源配置支持的运行用户配置可能不同，请以实际环境为准。关于创建Notebook的具体操作，请参见**创建Notebook实例**。

- **ma-user/ma-group**：Notebook公共镜像默认的非特权用户配置（安全模式）。使用时需满足以下条件：
  - 用户身份：ma-user（UID: 1000）
  - 用户组：ma-group（GID: 100）

  注意：如果使用自定义镜像，需提前在镜像中预置上述用户和用户组，否则容器启动时可能因权限不足导致服务异常。关于添加指定的用户和用户组的具体操作，请参见**Dockerfile文件（基础镜像为非ModelArts提供）**。
- **root/root**：以最高权限运行Notebook，适用于需要访问系统级资源的场景，但需注意潜在的安全风险。选择root/root时，系统将强制绑定以下用户和用户组。
  - 用户身份：root（UID: 0）
  - 用户组：root（GID: 0）

  注意：不允许修改root的UID/GID或所属用户组，否则可能引发容器权限冲突或安全漏洞。

常见问题

创建Notebook时，“更多配置 > 指定运行用户”无法开启，显示“当前所选资源池不支持开启指定运行用户”，怎么办?

您可以基于新网络创建专属资源池解决此问题。

1. 重新创建网络，并使用该网络创建专属资源池。具体操作，请参见**创建专属资源池**。
2. 使用新创建的专属资源池创建Notebook。具体操作，请参见**创建Notebook实例**。

1.8.5 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。

镜像保存功能通过在原始镜像基础上叠加新层（New Layer）实现版本迭代。由于镜像层具有不可变性，无论执行新增或删除操作，生成的新镜像体积将大于原始镜像。

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“专属资源池扩缩容”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。
- ContainerD镜像保存功能使用说明：
  - 不建议在实例内同时执行文件下载任务和镜像保存任务，以避免资源竞争导致性能下降。
  - 如果实例系统文件夹（如/opt/等）中存在10G以上的大文件，不建议使用镜像保存功能，以免镜像体积过大或保存时间过长。

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”列的“保存镜像”。
2. 在“保存镜像”页面，配置相关参数，勾选“我已知晓变更带来的风险，同意进行镜像保存”，单击“确定”保存镜像。

   保存方式支持“保存至已有”或“新建镜像”。

   - 保存至已有：将镜像保存为当前已注册镜像的新版本，即“已注册镜像”页面已有的镜像文件夹内。
   - 新建镜像：镜像将保存至容器镜像服务SWR，保存完成后会自动注册至ModelArts。

表 1-22 参数说明

保存方式	参数	说明
保存至已有	镜像	单击镜像选择框，在“选择镜像”面板，按需选择基础仓库或企业仓库中的镜像，单击“确定”。
	镜像版本	自定义镜像的版本。支持大小写字母、数字、下划线、中划线和半角句号，最大长度为64个字符。
	描述	自定义镜像的描述，不能包含字符&<>"'/，输入长度范围为0到256个字符。

保存方式	参数	说明
新建镜像	镜像仓库	按需选择SWR基础版或企业版镜像仓库。默认使用SWR基础版仓库。 如果您需要使用企业版镜像仓库，可在SWR进行注册。容器镜像服务企业版的镜像仓库为注册表名称。具体操作，请参见**用户指南（企业版）**。 关于SWR基础版与企业版的区别，请参见**基础版及企业版对比**。
	域名	仅“镜像仓库”选择SWR企业版，需配置此参数。 容器镜像服务企业版的访问地址。
	命名空间	仅“镜像仓库”选择SWR企业版，需配置此参数。 命名空间在容器镜像服务（企业版）中用于隔离镜像仓库。
	所属组织	仅“镜像仓库”选择SWR基础版，需配置此参数。 组织在SWR中用于隔离镜像，每个组织可对应一个公司或部门，将其拥有的镜像集中在该组织下。在不同的组织下，可以有同名的镜像。同一IAM用户可属于不同的组织。同一个组织内的用户可以共享使用该组织内的所有镜像。 如果没有组织，可以单击“前往SWR创建”，创建一个组织。创建组织的详细操作请参见**组织管理**。
	镜像名称	自定义保存的镜像名称。名称只能由小写英文字母、数字及特殊字符_ / . -组成，且必须以小写英文字母或数字开头和结尾，不能连续出现2个以上下划线，长度不能超过128个字符。
	镜像版本	自定义镜像的版本。支持大小写字母、数字、下划线、中划线和半角句号，最长长度为64个字符。
	描述	输入镜像的描述，不能包含字符&<>"'/，输入长度范围为0到256个字符。

3. 镜像会以快照的形式保存，保存过程约3~10分钟，保存期间实例状态将处于“快照中”，请耐心等待。此时不可再操作实例。镜像保存成功后，实例状态变为“运行中”。

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。
- 保存过程中连接可能会暂时中断，镜像保存操作完毕即可恢复。
- 保存的镜像中不会包含持久化存储挂载目录(/home/ma-user/work)下的文件和数据。

4. 查看镜像详情。
   a. 在左侧导航栏，按需选择以下操作。
      - 新版控制台：选择“资产管理 > 镜像”。
      - 旧版控制台：选择“资产管理 > 镜像管理”
   b. 在“已注册镜像”页签，单击镜像的名称，进入镜像详情页，查看镜像详细信息。

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”或者“镜像”页面中查询到，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。具体操作，请参见**创建Notebook实例**。

图 1-148 创建基于自定义镜像的 Notebook 实例

方式二：在“镜像管理”或者“镜像”页面的“已注册镜像”页签，单击某个镜像的镜像详情，在镜像详情页的“版本列表”页签，单击“创建Notebook”，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。

1.9 Notebook 监控审计

1.9.1 查看 Notebook 实例资源使用情况

在使用Notebook实例的过程中，您可以在Notebook详情页查看资源使用情况，例如CPU使用率、物理内存使用率、GPU显卡使用率等。

约束限制

系统会自动保存Notebook实例的资源使用情况30天，过期后将被清除。

查看 Notebook 实例监控信息

1. 登录**ModelArts管理控制台**，在左侧菜单栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 在“Notebook”页面，单击Notebook实例名称，选择“监控”页签，查看Notebook实例资源使用情况。

   您可以查看最近30分钟、1小时、6小时、12小时、24小时、7天内的监控数据，也可自定义时间段，最多可查看1个月内的监控数据。Notebook实例运行时，会定期向后台获取最新的资源使用率数据并刷新，您可以在右侧下拉框选择“手动刷新”，或设置间隔30秒、1分钟、5分钟自动刷新。

   单击图标，可跳转至AOM控制台，添加指标信息后查看实例的监控指标详情。详细信息请参考**在AOM控制台查看ModelArts所有监控指标**。

图 1-149 查看 Notebook 实例监控信息

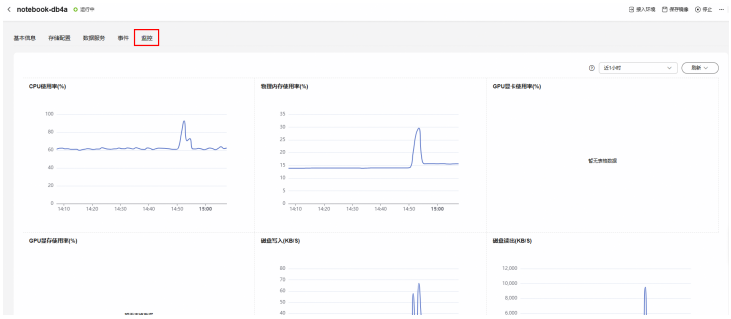


表 1-23 Notebook 实例指标说明

分类	指标名称	指标参数	指标含义
CPU	CPU使用率(%)	ma_container_cpu_util	用于统计测量对象的CPU使用率。
内存	物理内存使用率(%)	ma_container_memory_util	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。
GPU	GPU显卡使用率(%)	ma_container_gpu_util	用于统计测量对象的显卡使用率。
	GPU显存使用率(%)	ma_container_gpu_mem_util	用于统计测量对象已使用的显存占显存容量的百分比。

分类	指标名称	指标参数	指标含义
磁盘	磁盘写入(KB/S)	ma_container_disk_write_kilobytes	用于统计每秒写入磁盘的数据量。
	磁盘读出(KB/S)	ma_container_disk_read_kilobytes	用于统计每秒从磁盘读出的数据量。
网络	网络下行BPS(B/S)	ma_container_network_receive_bytes	网络接收数据速率，该指标用于统计测试对象的入方向网络流速。
	网络上行BPS(B/S)	ma_container_network_transmit_bytes	网络发送数据速率，该指标用于统计测试对象的出方向网络流速。

1.9.2 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 1-24 实例创建过程的事件列表

事件名称	事件描述	事件级别	处理建议
Scheduled	实例被调度成功	提示	正常事件，无需处理。
PullingImage	正在拉取镜像	提示	正常事件，无需处理。
PulledImage	镜像拉取完毕	提示	正常事件，无需处理。
NotebookHealthy	实例运行中，处于健康状态	重要	正常事件，无需处理。

事件名称	事件描述	事件级别	处理建议
CreateNotebookFailed	创建实例失败	紧急	内部服务错误，建议提工单，联系运维人员协助处理。
PullImageFailed	镜像拉取失败	紧急	检查实例创建时所选镜像是否存在，如不存在，重新选择镜像进行实例创建。如存在，建议提工单，联系运维人员协助处理。
FailedCreate	Failed to create notebook container. Please contact SRE to check node {node_name}	紧急	内部服务错误，建议提工单，联系运维人员协助处理。
CreateContainerError	Failed to create container. Please contact SRE to check node {node_name}	紧急	内部服务错误，建议提工单，联系运维人员协助处理。
FailedAttachVolume	Failed to attach volume. Please contact SRE to check node {node_name}	重要	内部服务错误，建议提工单，联系运维人员协助处理。
MountVolumeFailed	Mount volume failed; Check whether the DEW secret is correct if the instance cannot change to running in five minutes	紧急	等待5-10分钟，观察实例状态是否刷新至运行中，如正常刷新无需处理。如未刷新，请检查使用对象存储服务OBS时选择的认证信息是否正确。
	Mount volume failed; Check if vpc of sfs-turbo is interconnected if the instance cannot change to running in five minutes	紧急	等待5-10分钟，观察实例状态是否刷新至运行中，如正常刷新无需处理。如未刷新，请检查使用的SFS是否完成专属资源池的VPC打通。具体操作，请参见**专属资源池VPC打通**。
	Mount volume failed; Please contact SRE to check node {node_name} if the instance cannot change to running in five minutes	紧急	等待5-10分钟，观察实例状态是否刷新至运行中，如正常刷新无需处理。如未刷新，建议提工单，联系运维人员协助处理。

表 1-25 实例停止过程的事件列表

事件名称	事件描述	事件级别	处理建议
StopNotebook	实例停止	重要	正常事件，无需处理。
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	正常事件，无需处理。

表 1-26 更新实例过程的事件列表

事件名称	事件描述	事件级别	处理建议
UpdateName	更新实例名称	提示	正常事件，无需处理。
UpdateDescription	更新实例描述	提示	正常事件，无需处理。
UpdateFlavor	更新实例规格	重要	正常事件，无需处理。
UpdateImage	更新实例镜像	重要	正常事件，无需处理。
UpdateStorageSize	实例存储正在扩容（User %s is updating storage size from %sGB to %sGB）	重要	正常事件，无需处理。
	实例扩容完成（User %s updated storage size successfully）	重要	正常事件，无需处理。
UpdateKeyPair	配置实例密钥对（User %s updated the instance keypair to "{%s}"）	重要	正常事件，无需处理。
	更新实例密钥对（User %s updated the instance keypair from %s to %s）	重要	正常事件，无需处理。
UpdateHook	更新自定义脚本	重要	正常事件，无需处理。
UpdateStorageSizeFailed	资源售罄引起的实例存储扩容失败（The EVS disk is sold out）	紧急	进入需扩容实例详情页面，选择存储配置页签，添加动态存储或者扩展存储实现扩容。

事件名称	事件描述	事件级别	处理建议
	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	内部服务异常，建议提工单，联系运维人员协助处理。

表 1-27 镜像保存过程中的事件列表

事件名称	事件描述	事件级别	处理建议
SaveImage	保存镜像成功	重要	正常事件，无需处理。
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	执行ps -aux命令查找所有D状态进程，执行kill -9 <PID>终止所有D状态进程后重新执行镜像保存操作。
	镜像大小引起的保存镜像失败（Container size %dG is greater than threshold %dG）	紧急	删除实例中/home/ma-user/work/目录之外的其他无用目录文件，减小容器镜像大小至事件描述中阈值后重试。
	层数限制引起的保存镜像失败（Too many layers in your image）	紧急	启动实例使用的镜像层数超过125，重新制作实例启动镜像，制作过程中可通过合并命令和分阶段构建的方式减少镜像层数。
	任务超时引起的保存镜像失败（Image saving failed due to task timeout）	紧急	网络或者依赖服务异常导致任务超时，建议提工单，联系运维人员协助处理。
	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	SWR服务异常，建议提工单，联系运维人员协助处理。

事件名称	事件描述	事件级别	处理建议
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	正常事件，无需处理。
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	正常事件，无需处理。
ContainerCommitStarted	Start to commit notebook container.	提示	正常事件，无需处理。
ContainerCommitSuccess	Notebook container commit successfully.	提示	正常事件，无需处理。
ImagePushStarted	Start to push notebook image.	提示	正常事件，无需处理。
ImagePushSuccess	Notebook image push successfully.	提示	正常事件，无需处理。
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	节点异常或内部服务错误，建议提工单，联系运维人员协助处理。
ImagePushFailed	Failed to push Notebook image. Please contact SRE to check node {node_name}.	提示	推送镜像失败，可重试。如重试无法解决，建议提工单，联系运维人员协助处理。

表 1-28 实例运行过程的事件列表

事件名称	事件描述	事件级别	处理建议
NotebookUnhealthy	实例处于不健康状态	紧急	在实例中启动调试任务，如任务占用CPU，Memory或者IO资源过高，可能触发此事件，实例负载降低后可自动恢复。间隔一定时间后刷新页面，如新增NotebookHealthy事件，表示实例状态已恢复正常，无需处理。如长时间无法恢复，请提工单，联系运维人员进行协助处理。
OutOfMemory	实例占用内存超过规格申请内存导致被驱逐	紧急	实例中进程占用内存超过实例规格申请内存，K8s机制会触发此事件，并重启实例。重启完成后实例状态刷新为正常，后续实例使用过程中，需避免高内存占用任务。
JupyterProcessKilled	Jupyter进程异常终止	紧急	实例中误操作停止jupyter进程或者实例对应容器未知错误，可能触发此事件。此状态实例会自动重启，重启完成后实例状态刷新为正常。
CacheVolumeExceedQuota	/cache目录文件大小超过最大限制	紧急	/cache目录文件超过对应规格分配最大限制会触发此事件。此状态实例会自动重启，重启完成后实例状态刷新为正常。后续实例使用过程中，需关注/cache目录文件大小，此目录分配空间和实例规格对应关系可参考**在ModelArts的Notebook中不同规格资源/cache目录的大小是多少**。
NotebookHealthy	实例从异常状态恢复至正常状态	重要	正常事件，无需处理。
EVSSoldOut	EVS存储售罄	紧急	创建Notebook时， 存储类型选择云硬盘EVS，如云硬盘EVS售罄，可能触发此事件。建议切换存储类型至“对象存储 OBS - 对象桶”或“对象存储 OBS - 并行文件系统 ”等类型。如需继续使用云硬盘EVS，请提工单，联系运维人员进行扩容处理。

表 1-29 OBS 动态挂载产生的事件列表

事件名称	事件描述	事件级别	处理建议
DynamicMountStorage	挂载OBS存储	重要	正常事件，无需处理。
DynamicUnmountStorage	卸载OBS存储	重要	正常事件，无需处理。

表 1-30 用户侧触发的事件

事件名称	事件描述	事件级别	处理建议
RefreshCredentialsFailed	用户鉴权失败	紧急	正常事件，无需处理。

1.9.3 使用 CTS 审计 Notebook 实例

ModelArts支持对接云审计CTS服务，通过云审计服务，您可以记录与ModelArts Notebook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。详细操作请参见**《云审计服务用户指南》**。

开发环境支持审计的关键操作列表

表 1-31 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实例	Notebook	createNotebook
查询Notebook实例详情	Notebook	getNotebook
查询所有的Notebook实例列表	Notebook	listallNotebook
查询Notebook实例列表	Notebook	listNotebook
打开CodeLab实例	Notebook	openNotebook
删除Notebook实例	Notebook	deleteNotebook
更新Notebook实例	Notebook	updateNotebook
启动Notebook实例	Notebook	startNotebook
停止Notebook实例	Notebook	stopNotebook

操作名称	资源类型	事件名称
通过运行的实例保存成容器镜像	NotebookImage	createNotebookImage
查询支持的镜像列表	NotebookImage	listNotebookImage
查询用户镜像信息概览	NotebookImageGroups	listNotebookImageGroups
查询镜像详情	NotebookImage	getNotebookImage
同步镜像状态	NotebookImage	updateNotebookImage
注册自定义镜像	NotebookImage	createNotebookImage
删除镜像	NotebookImage	deleteNotebookImage
创建NotebookApp实例	NotebookApp	createNotebookApp
查询NotebookApp实例列表	NotebookApp	listNotebookApp
查询NotebookApp实例详情	NotebookApp	getNotebookApp
删除NotebookApp实例	NotebookApp	deleteNotebookApp
使用NotebookApp实例	NotebookApp	updateNotebookApp
查询项目下的Notebook标签信息	CombineTmsTag	listCombineTmsTag
查询资源下的标签信息	NotebookTmsTag	listNotebookTmsTag

查看审计事件

1. 登录**CTS云审计服务控制台**。
2. 在管理控制台左上角单击  图标，选择目标区域。
3. 在左侧导航栏，单击“事件列表”，进入事件列表信息页面。
   用户每次登录云审计控制台时，控制台默认显示新版事件列表，单击页面右上方的“返回旧版”按钮，可切换至旧版事件列表页面。
4. 事件列表支持通过筛选来查询对应的操作事件，您可以按需筛选，查看目标事件的相关信息。
   关于事件筛选的参数说明，请参见**在CTS事件列表查看云审计事件**。

1.9.4 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 1-150 监控对象指标设置

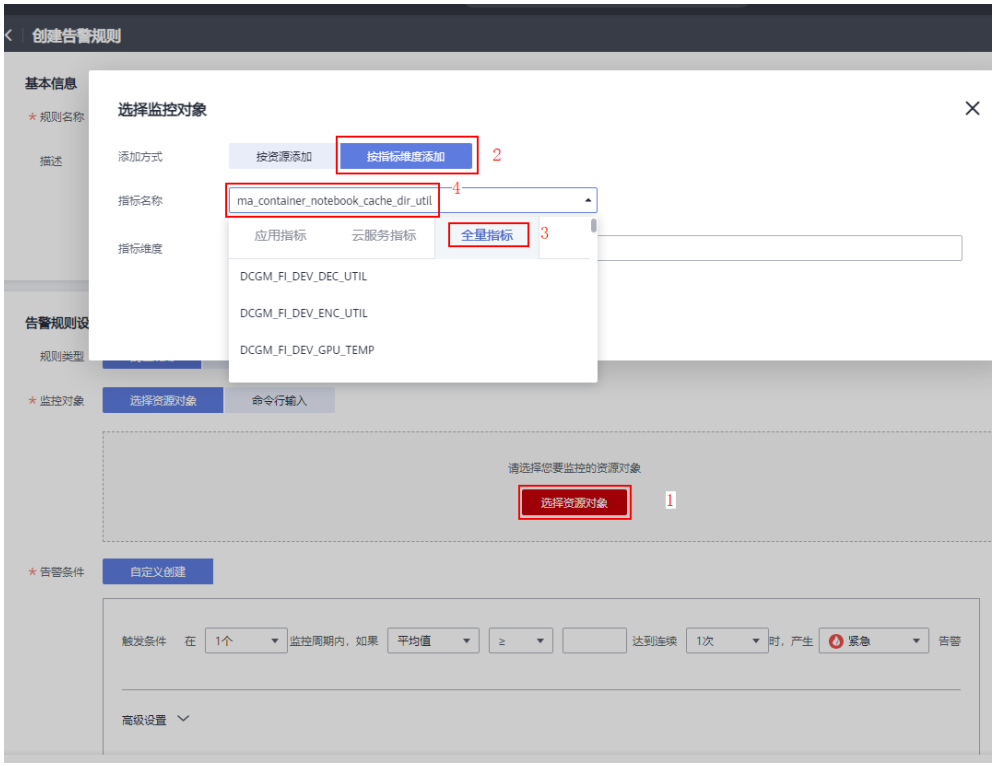


图 1-151 设置指标统计方式

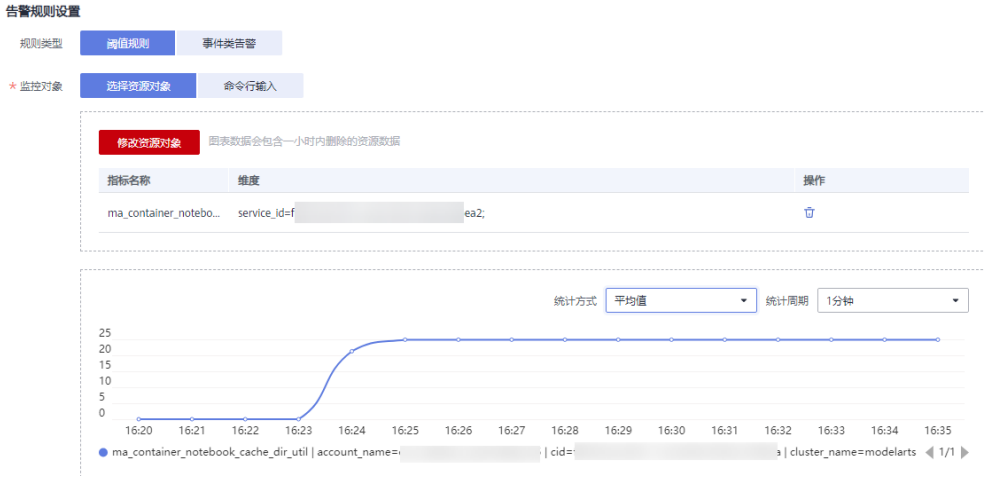


图 1-152 告警条件设置

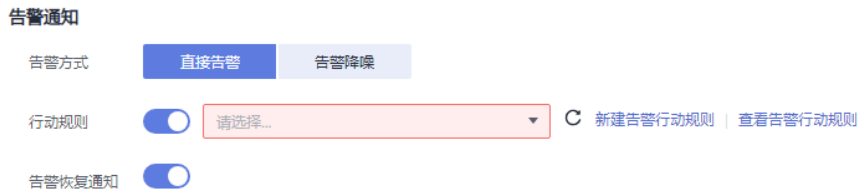
5. 设置告警通知，单击“立即创建”。

   “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 1-153 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 1-154 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，单击“确定”。

     此时订阅总数中会出现一条记录，但是处于未确认的状态。

图 1-155 订阅总数

例如，当收到邮件后单击“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 1-156 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，那么就会收到邮件或其他方式通知。

1.10 ModelArts MoXing 命令参考

1.10.1 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**
>
> 在Notebook中，MoXing已经预先安装好，用户无需自行下载。只需在Notebook中**导入MoXing Framework模块**，即可直接使用其功能。

使用场景

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/')
```

导入 MoXing Framework 模块

使用MoXing Framework前，您需要在代码的开头先导入MoXing Framework模块。

执行如下代码，导入MoXing模块。

```
import moxing as mox
```

导入 MoXing Framework 的相关说明

在导入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)
```

可以在导入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
```

导入 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，可以导入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。

说明

以下操作仅适用于旧版控制台（左侧导航栏为“模型训练 > 训练作业”）。如果您是新版控制台（左侧导航栏为“模型开发与训练 > 模型训练”），请在导航栏最下方，单击“返回旧版”。

1. 登录**ModelArts管理控制台**，在左侧菜单栏选择“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**创建生产训练作业**。 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。

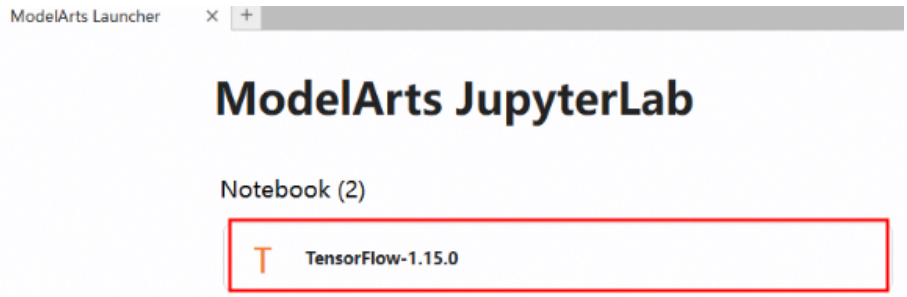
1.10.2 Notebook 中快速使用 MoXing

本文档介绍如何在ModelArts中调用MoXing Framework接口。

进入 ModelArts，创建 Notebook 实例

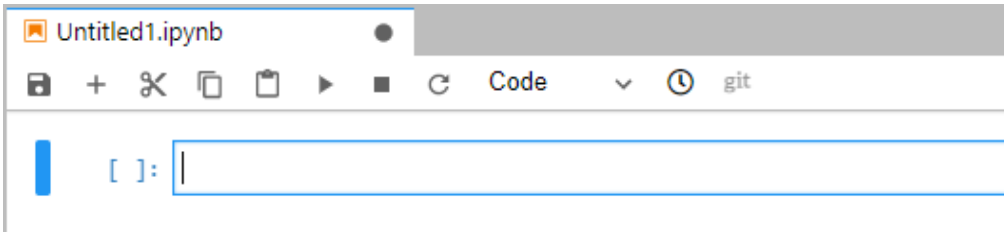
1. 登录**ModelArts管理控制台**，在左侧菜单栏按需选择以下操作。
   - 新版控制台：选择“模型开发与训练 > Notebook”，进入“Notebook”页面。
   - 旧版控制台：选择“开发空间 > Notebook”，进入“Notebook”页面。
2. 在右上角单击“创建”进入“创建Notebook”页面，参考**创建Notebook实例**填写信息并完成Notebook实例创建。
3. 当Notebook实例创建完成后，且状态为“运行中”时，单击“操作”列中的“接入环境”，单击“JupyterLab接入”右侧的“接入”，进入“JupyterLab Notebook”开发页面。
4. 在JupyterLab的“ModelArts 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 1-157 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 1-158 进入编码页面



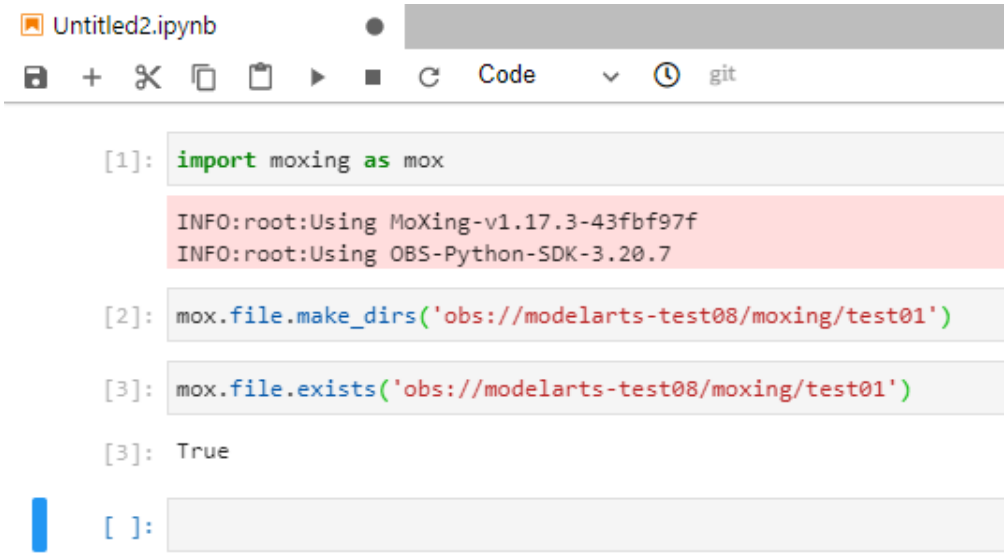
调用 mox.file

输入如下代码，实现如下几个简单的功能。

1. 引入MoXing Framework。
   ```
   import moxing as mox
   ```
2. 在已有的“modelarts-test08/moxing”目录下，创建一个“test01”文件夹。
   ```
   mox.file.make_dirs('obs://modelarts-test08/moxing/test01')
   ```
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。
   ```
   mox.file.exists('obs://modelarts-test08/moxing/test01')
   ```

执行结果如图1-159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 1-159 运行示例



复制数据到 OBS

在Notebook的JupyterLab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')
```

1.10.3 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，与Python接口具有一一对应的关系。
- 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，文件操作接口不会自动切换为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 1-32 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(在低于2.5.0版本的moxing中st_mtime属性存在问题)	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

1.10.4 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')
```

1.10.5 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,)))
```

1.11 ModelArts CLI 命令参考

1.11.1 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管理控制台**，在左侧导航栏按需选择以下操作。关于如何打开Terminal，请参见**在JupyterLab中新建Terminal**。

  - 新版控制台：在“模型开发与训练 > Notebook”中创建Notebook实例，打开Terminal，使用ma-cli命令。
  - 旧版控制台：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 1-33 ma-cli 支持的命令

命令	命令详情
auto-completion	命令自动补全。
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。
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。
obs-copy	本地和OBS文件/文件夹间的相互复制。

1.11.2 （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。

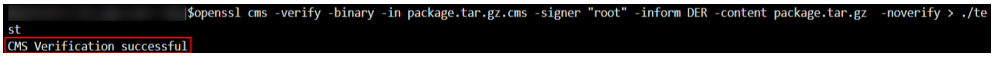
Step2：下载 ma-cli

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。

1.11.3 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 Zsh”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 completion.

[ 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 completion.

[ OK ] eval (env _MA_CLI_COMPLETE=fish_source ma-cli)
```

1.11.4 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 1-34 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。

例如${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。

1.11.5 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 1-35 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。

命令	命令详情
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 1-36 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 1-37 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。
-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.
```

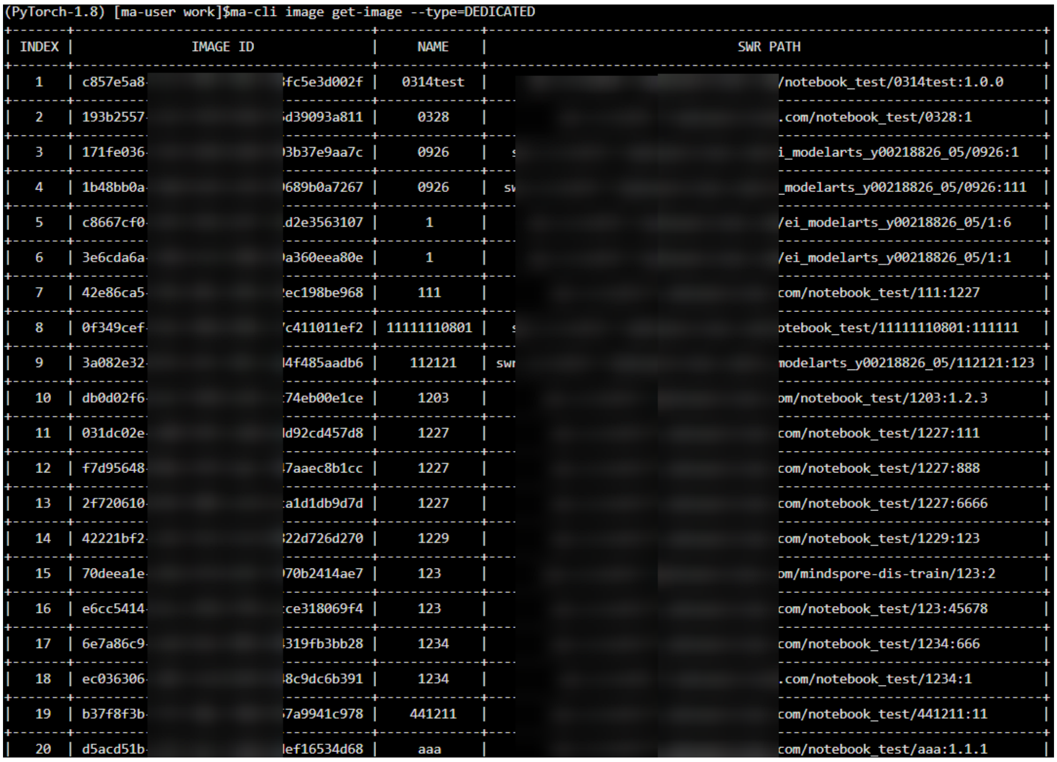
表 1-38 参数说明

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。

参数名	参数类型	是否必选	参数说明
-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	[redacted]i_modelarts_y00218826_05/0926:1
4	1b48bb0a[redacted]689b0a7267	0926	[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	[redacted]otebook_test/11111110801:111111
9	3a082e32[redacted]4f485aadb6	112121	[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 1-39 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。

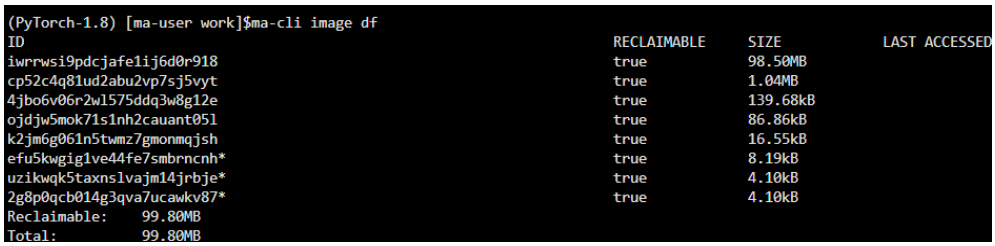
```
$ ma-cli image df -h
Usage: ma-cli image df [OPTIONS]

  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 1-40 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。

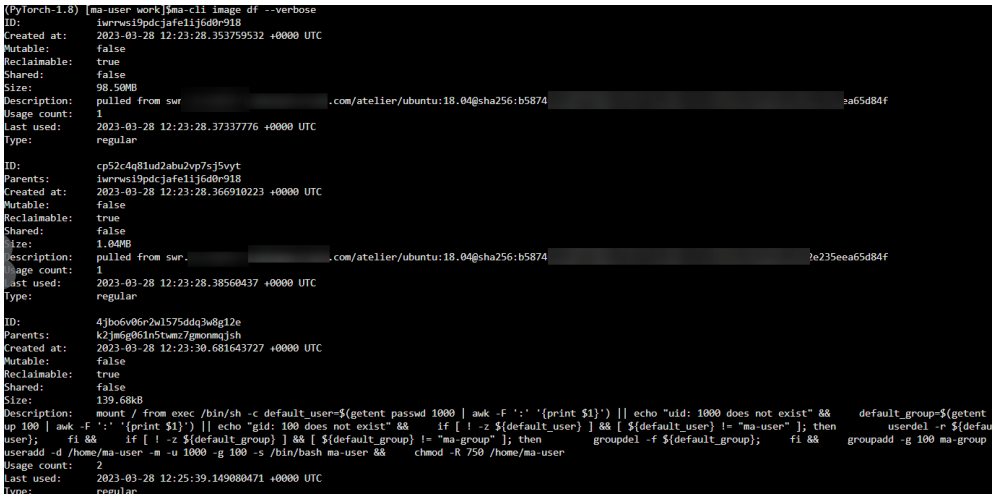
示例：在ModelArts Notebook里查看所有镜像缓存。

```
ma-cli image df
```



示例：显示镜像缓存占用磁盘的详细信息。

```
ma-cli image df --verbose
```

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 1-41 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.
```

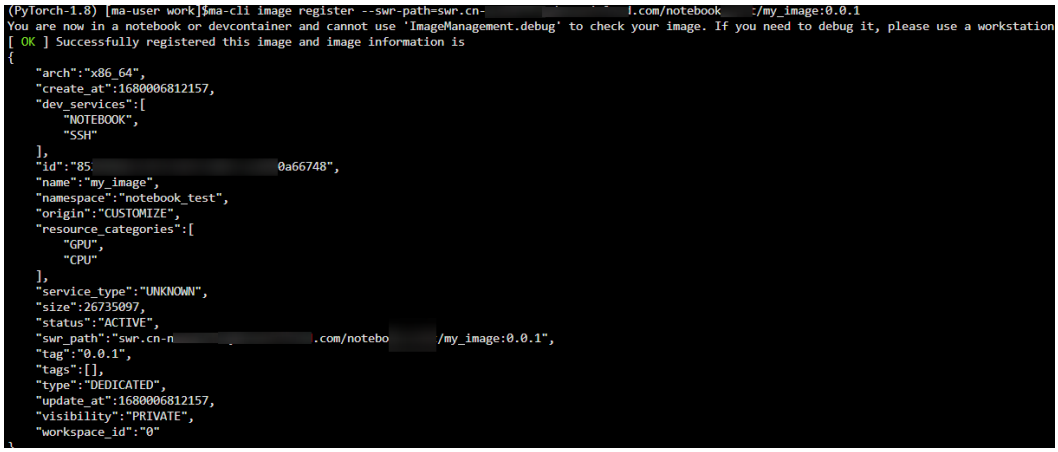
表 1-42 参数说明

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。

参数名	参数类型	是否必选	参数说明
-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 1-43 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 1-44 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。

参数名	参数类型	是否必选	参数说明
-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。

1.11.6 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 job 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 1-45 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

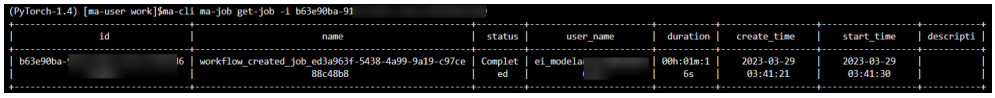
使用**ma-cli ma-job get-job**命令可以查看训练作业列表或某个作业详情。

```
$ 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.
```

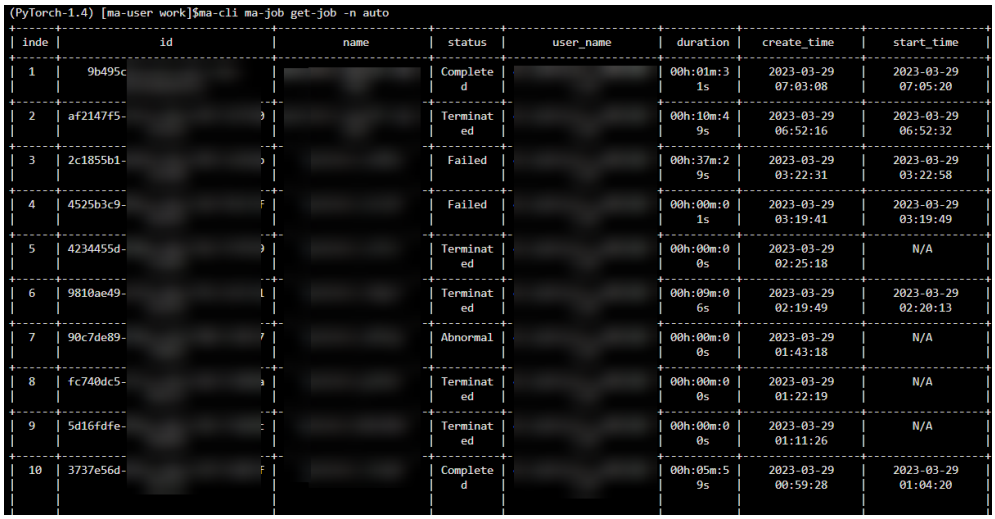
表 1-46 参数说明

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*

- 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		Completed		00h:01m:31s	2023-03-29 07:03:08	2023-03-29 07:05:20
2	af2147f5-		Terminated		00h:10m:49s	2023-03-29 06:52:16	2023-03-29 06:52:32
3	2c1855b1-		Failed		00h:37m:29s	2023-03-29 03:22:31	2023-03-29 03:22:58
4	4525b3c9-		Failed		00h:00m:01s	2023-03-29 03:19:41	2023-03-29 03:19:49
5	4234455d-		Terminated		00h:00m:00s	2023-03-29 02:25:18	N/A
6	9810ae49-		Terminated		00h:09m:06s	2023-03-29 02:19:49	2023-03-29 02:20:13
7	90c7de89-		Abnormal		00h:00m:00s	2023-03-29 01:43:18	N/A
8	fc740dc5-		Terminated		00h:00m:00s	2023-03-29 01:22:19	N/A
9	5d16fdfe-		Terminated		00h:00m:00s	2023-03-29 01:11:26	N/A
10	3737e56d-		Completed		00h:05m:59s	2023-03-29 00:59:28	2023-03-29 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 1-47 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。
--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。

参数名	参数类型	是否必选	参数说明
--pool-id	String	否	训练作业选择的资源池ID。可在**ModelArts管理控制台**，在左侧单击“资源管理 > 专属算力资源 > 资源池”或者“资源管理 > 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
--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。

参数名	参数类型	是否必选	参数说明
--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

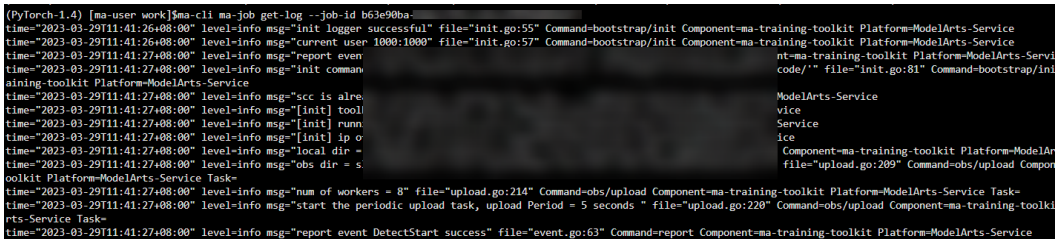
执行**ma-cli ma-job get-log**命令查询ModelArts训练作业日志。

```
$ 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

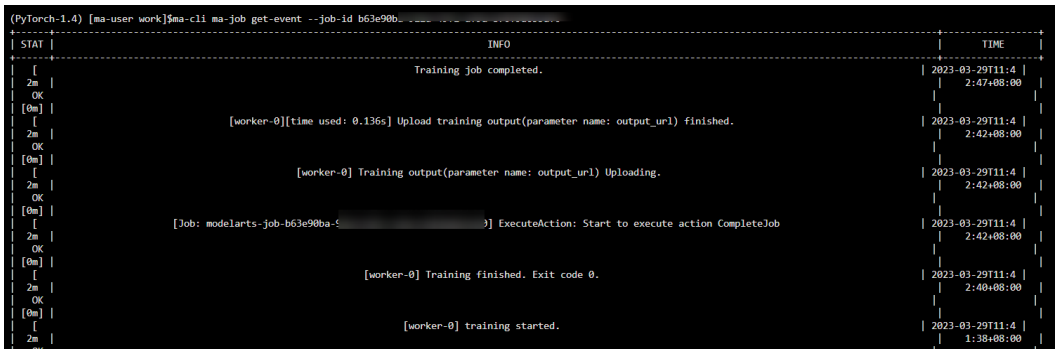
执行**ma-cli ma-job get-event**命令查看ModelArts训练作业事件。

```
$ 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 1-48 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 1-49 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 1-50 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
```

1.11.7 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 1-51 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。
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。

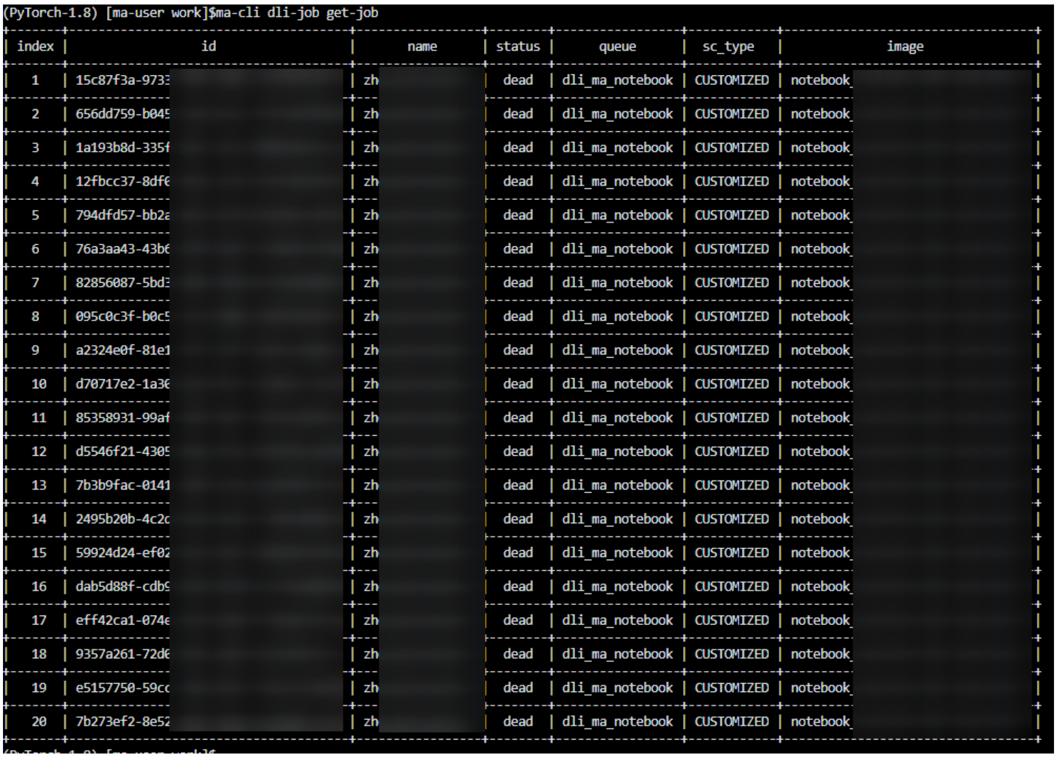
```
ma-cli dli-job get-job -h
Usage: ma-cli dli-job get-job [OPTIONS]

  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 1-52 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-973[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
2	656dd759-b04[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
3	1a193b8d-335[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-bb2[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	85358931-99a[redacted]	zh[redacted]	dead	dli_ma_notebook	CUSTOMIZED	notebook_[redacted]
12	d5546f21-430[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-ef0[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	7b273ef2-8e5[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 1-53 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。

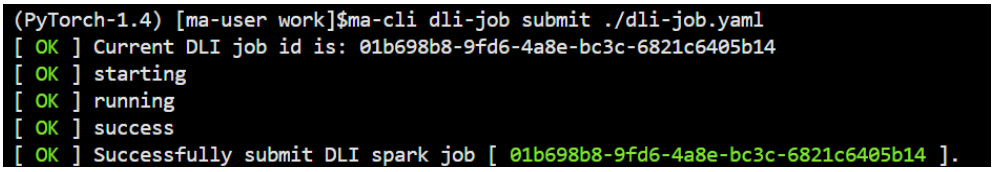
参数名	参数类型	是否必选	参数说明
--queue	String	否	用于指定队列，填写已创建DLI的队列名。必须为通用类型的队列。队列名称的获取请参考**表1-55**。
-ec / --executor-cores	String	否	Spark应用每个Executor的CPU核数。该配置项会替换sc_type中对应的默认参数。
-em / --executor-memory	String	否	Spark应用的Executor内存，参数配置例如2G，2048M。该配置项会替换“**sc_type**”中对应的默认参数，使用时必须带单位，否则会启动失败。
-ne / --num-executors	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

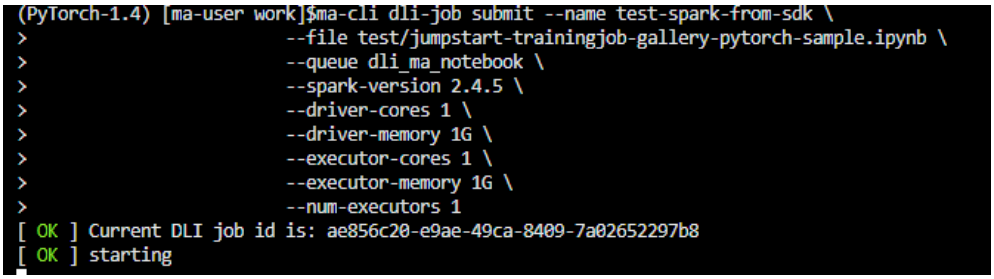
- 通过YAML_FILE文件提交DLI Spark作业。

```
$ma-cli dli-job submit dli_job.yaml
```



- 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
```



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.
```

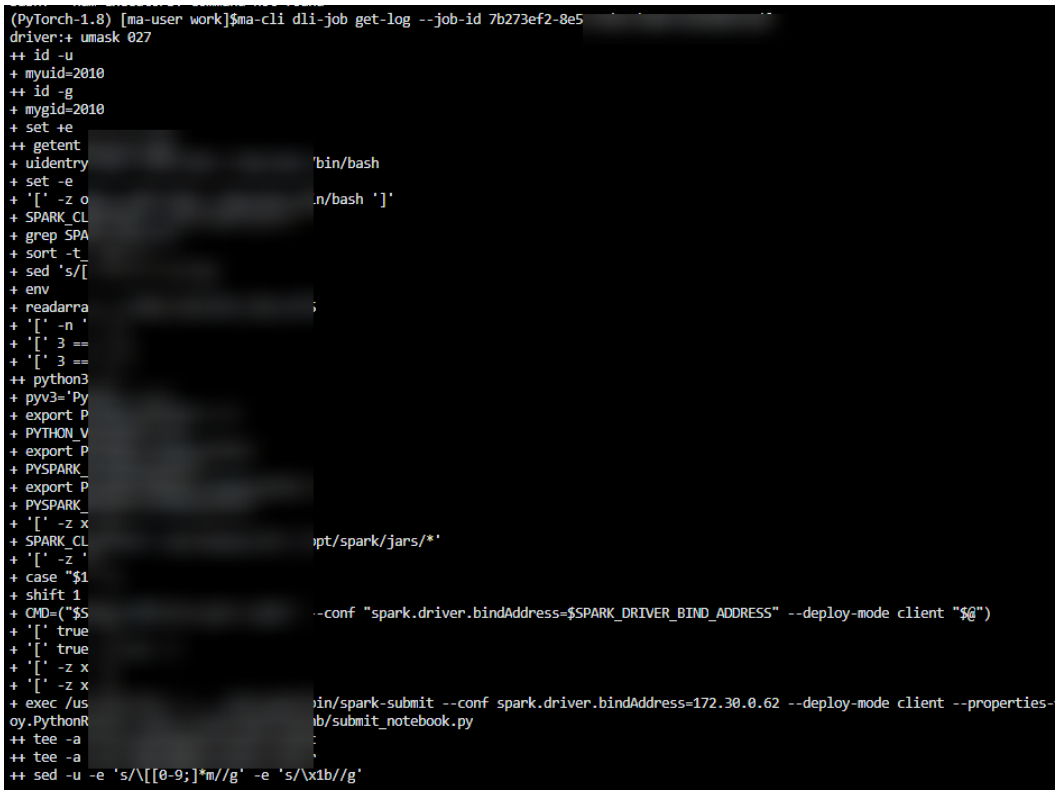
```
-D, --debug           Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT    CLI connection profile to use. The default profile is "DEFAULT".
-H, -h, --help        Show this message and exit.
```

表 1-54 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 1-55 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': '[redacted]',
 'queueName': 'dli_ma_notebook',
 'queueType': 'general',
 'queue_id': 8[redacted],
 'resource_id': '242e7af8-c9d1[redacted]',
 '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 1-56 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 1-57 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 1-58 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}
```

```
(PyTorch-1.8) [ma-user work]$ma-cli dli-job stop -i 4b[redacted]861
[ WARN ] Spark job 4b[redacted]1 will be stopped, do you want to continue (y for confirm)? [y/N]: y
[ OK ] Successfully stop spark batch job [ 4b[redacted]1 ].
```

1.11.8 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 1-59 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/ ]
```

1.12 历史待下线

1.12.1 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。

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

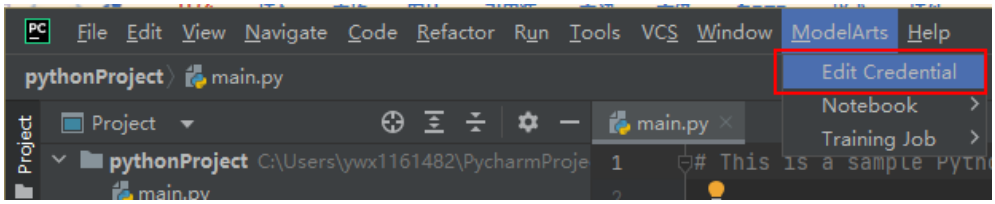
创建一个Notebook实例，并开启SSH远程开发。该实例状态必须处于“运行中”，具体参见**创建Notebook实例**。

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 1-160 Edit Credential

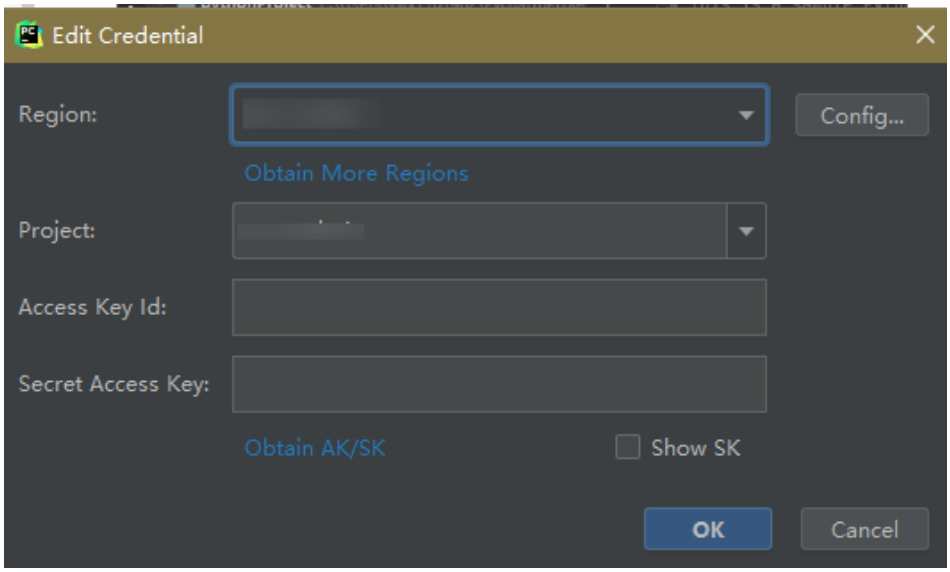
   

   说明

   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 1-161 填写区域和访问密钥



3. 查看认证结果。

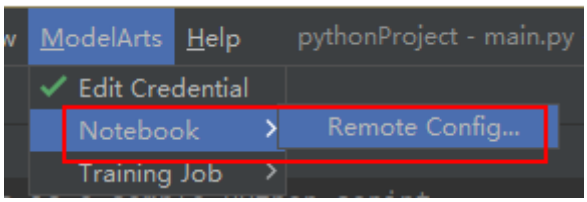
   在Event Log区域中，当提示如下类似信息时，表示访问密钥添加成功。

   16:01Validate Credential Success: The HUAWEI CLOUDcredential is valid.

Step4 插件自动化配置

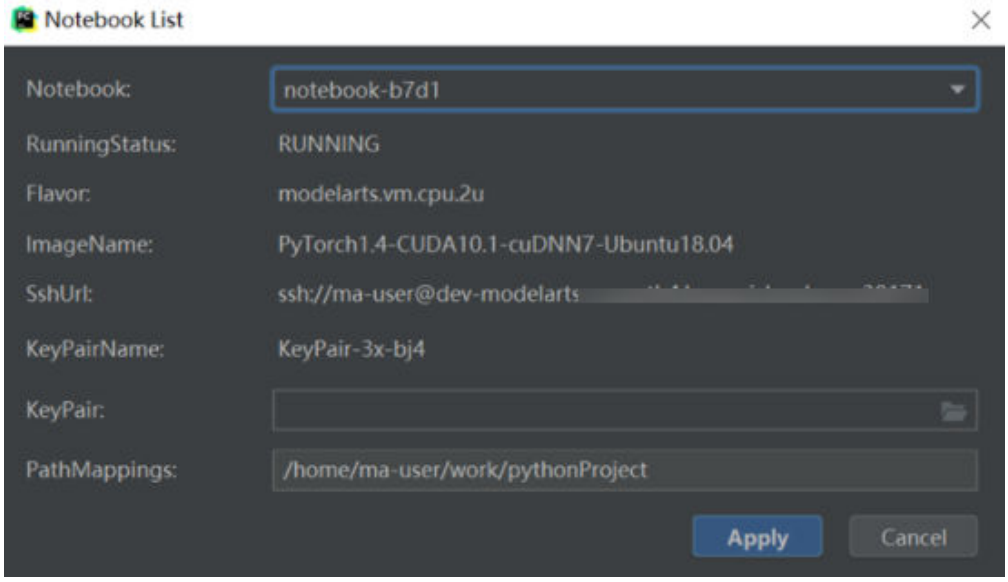
1. 在本地的PyCharm开发环境中，单击“ModelArts > Notebook > Remote Config...”，配置插件。

   图 1-162 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 1-163 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。

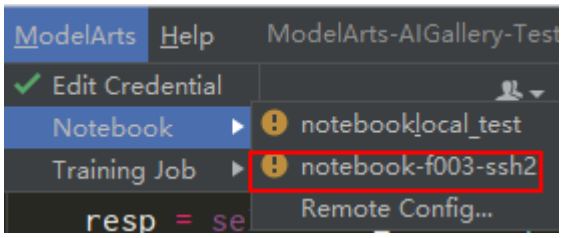
3. 单击“Apply”，配置完成后，重启IDE生效。

   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

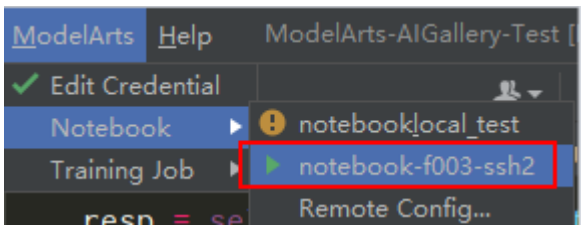
与Notebook断开连接的状态下，单击Notebook名称，根据提示启动本地IDE与Notebook的连接(默认启动时间4小时)。

图 1-164 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 1-165 停止连接 Notebook



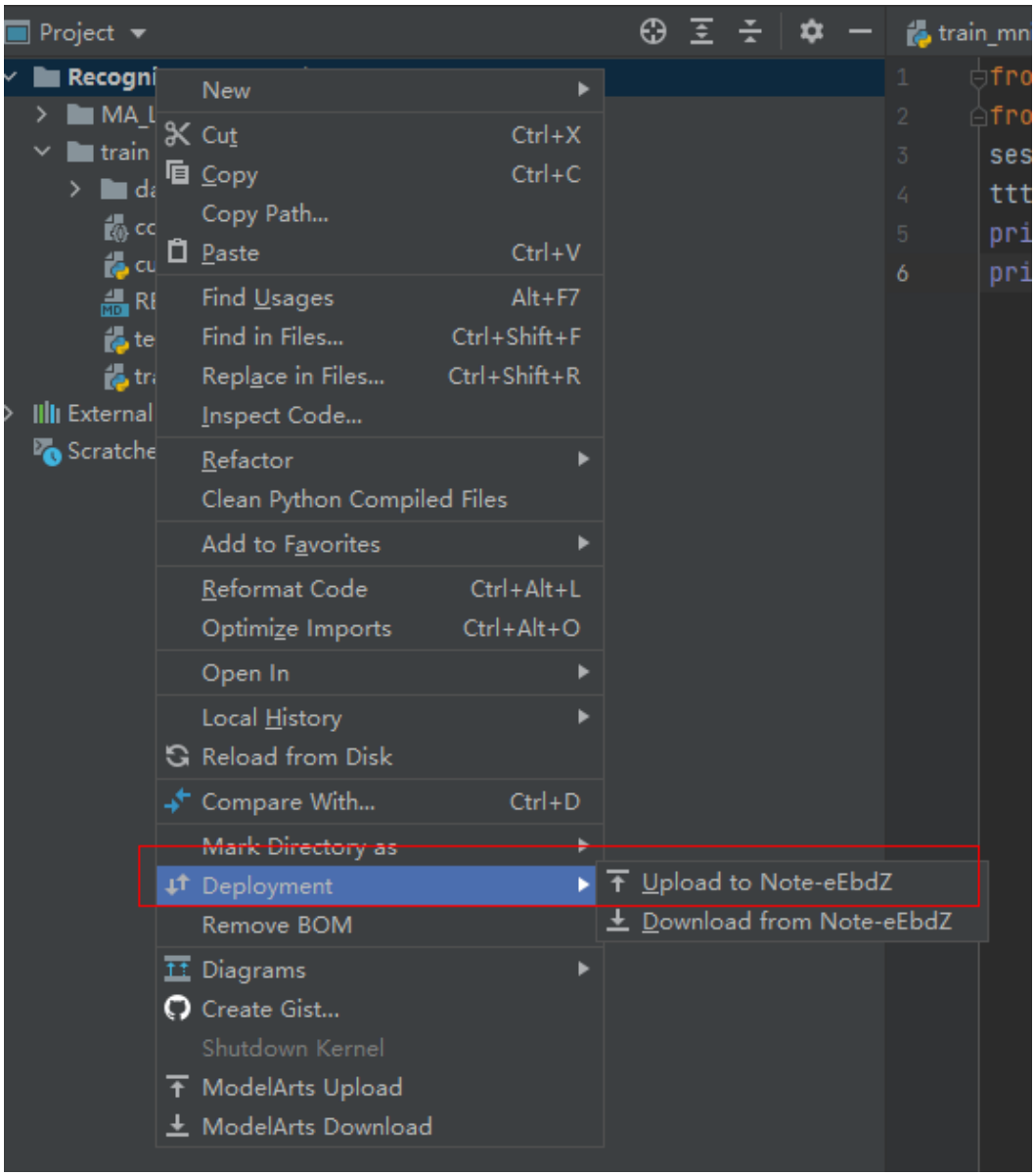
Step6 同步上传本地文件至 Notebook

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 1-166 同步本地文件至 Notebook

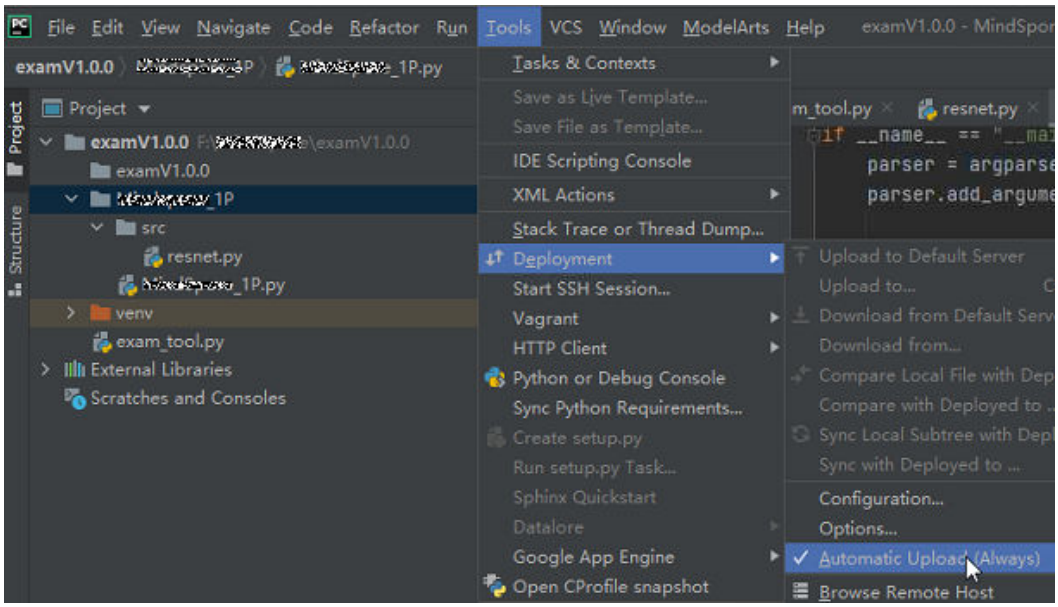


后续同步:

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。

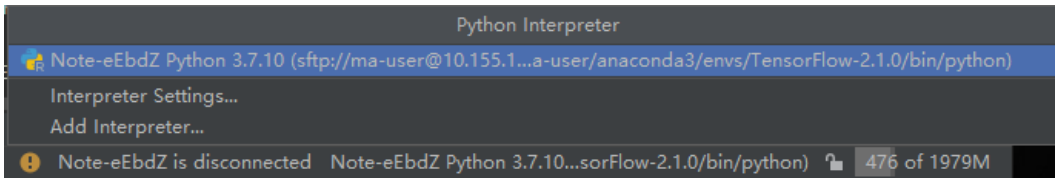
图 1-167 开启自动上传



Step7 远程调试

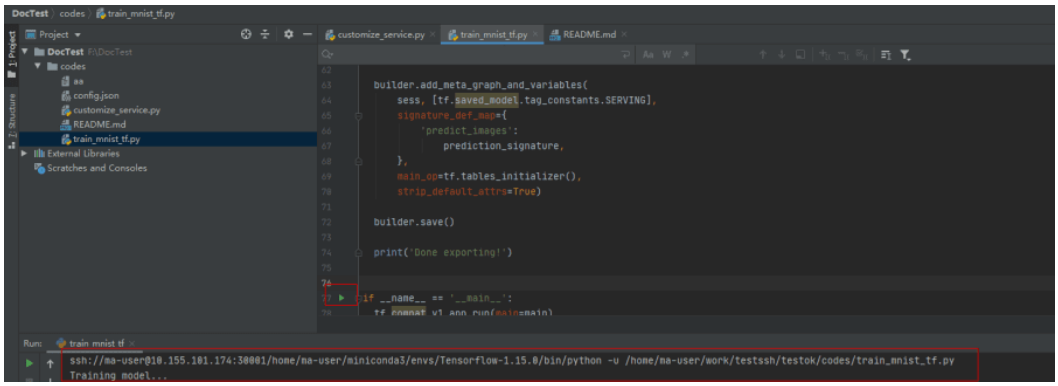
单击本地IDE右下角interpreter，选择Notebook的python解释器。

图 1-168 选择 Python 解释器



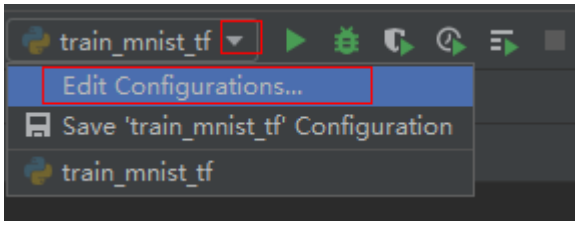
像本地运行代码一样，直接单击运行按钮运行代码即可，此时虽然是在本地IDE点的运行按钮，实际上运行的是云端Notebook里的代码，日志可以回显在本地的日志窗口。

图 1-169 查看运行日志



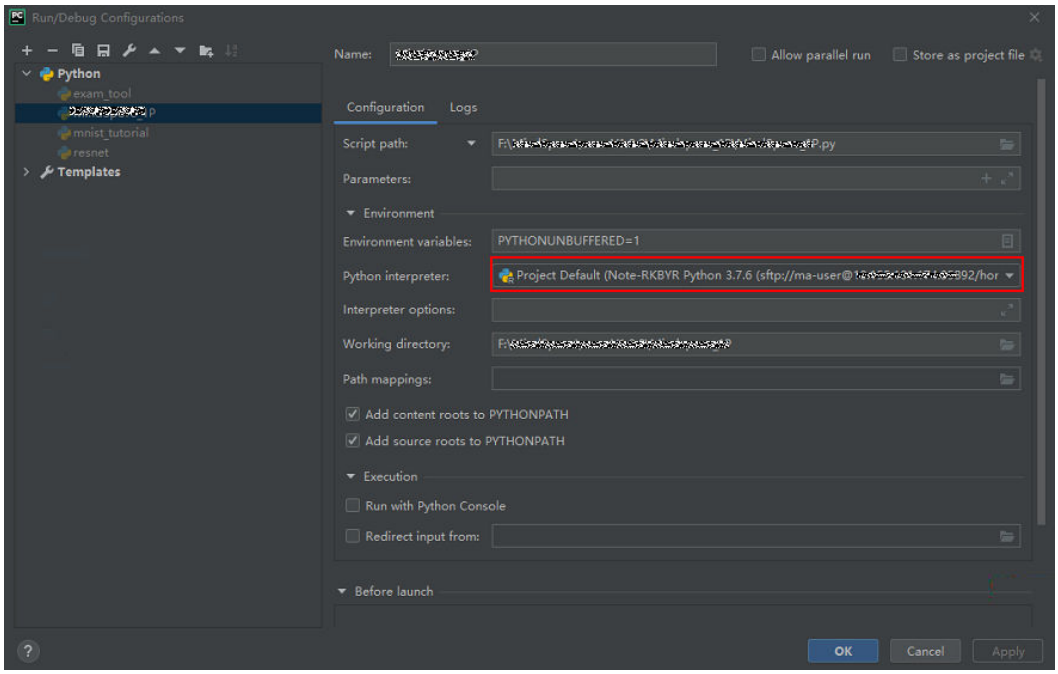
也可以单击本地IDE右上角的Run/Debug Configuration按钮来设置运行参数。

图 1-170 设置运行参数（1）



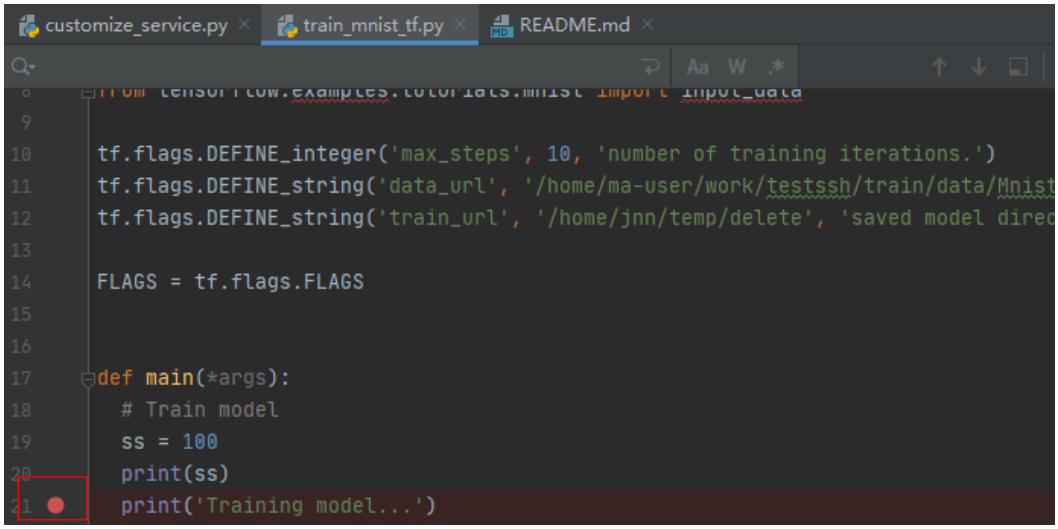
选择远程连接到云上开发环境实例对应的Python解释器。

图 1-171 设置运行参数（2）



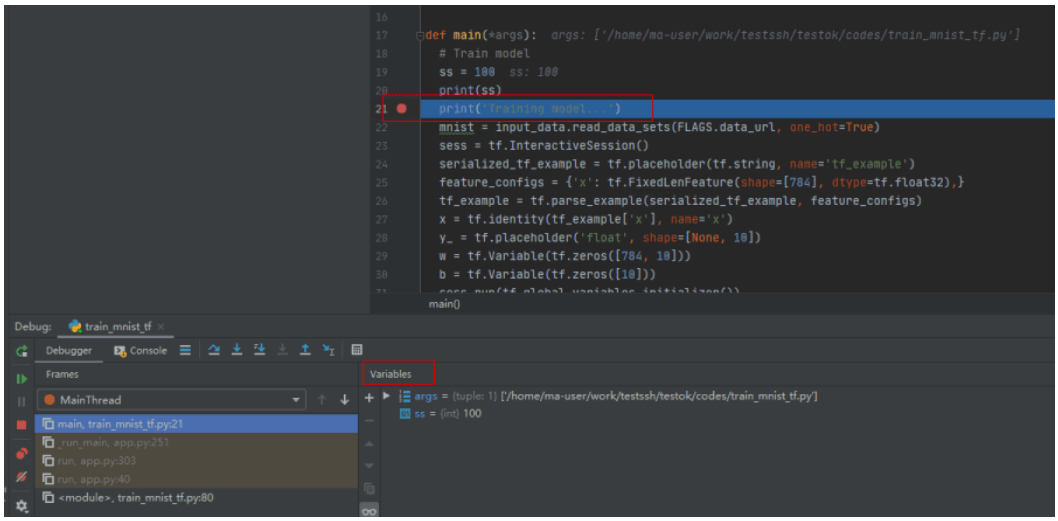
当需要调试代码时，可以直接打断点，然后使用debug方式运行程序。

图 1-172 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 1-173 Debug 模式下查看变量值



2 使用 Workflow 实现低代码 AI 开发

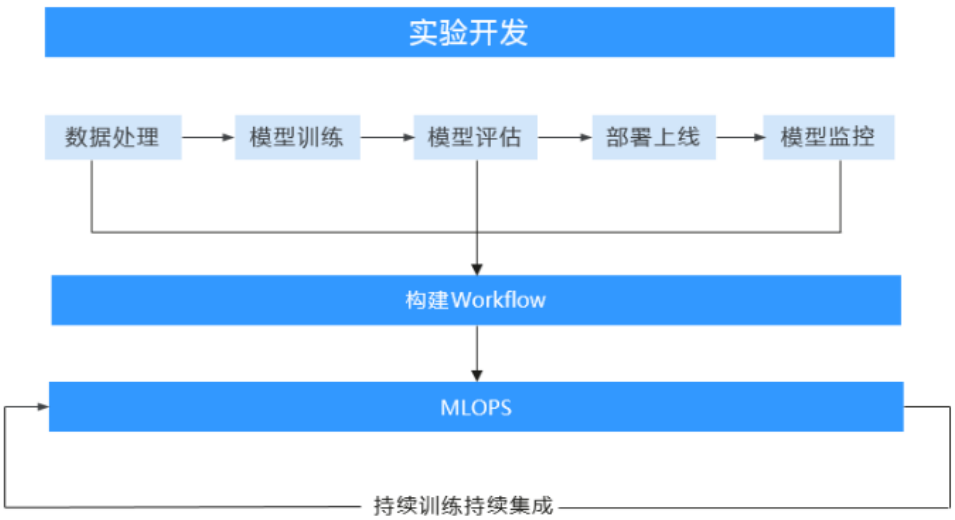
2.1 什么是 Workflow

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 2-1 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。

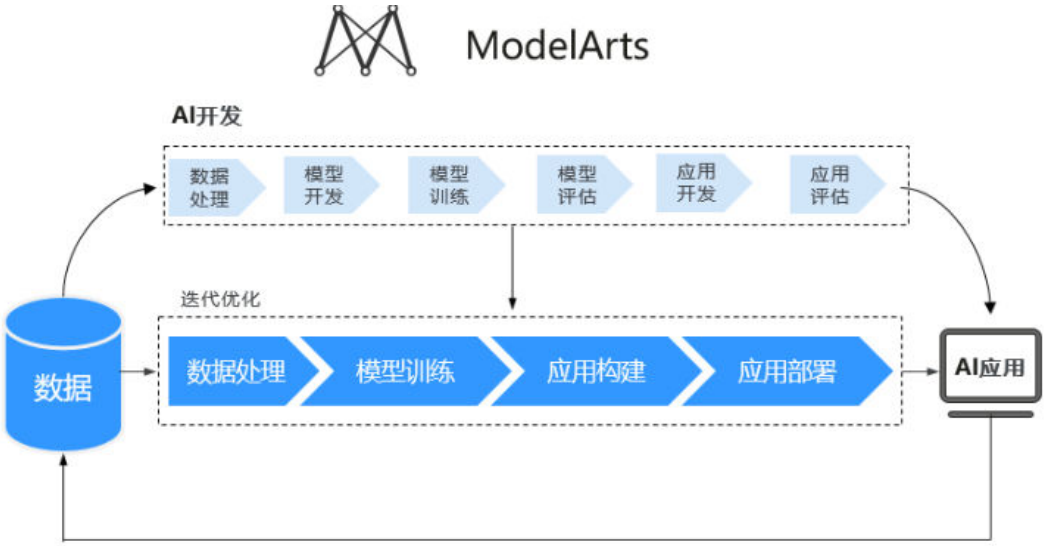
这个工具需要具备以下的能力：

- 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 2-2 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AI 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。

2.2 管理 Workflow

2.2.1 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 2-3 属性类型

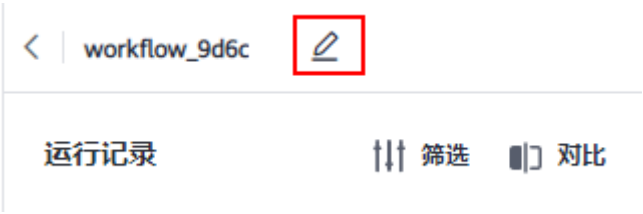
3. 单击搜索框右侧的按钮，可设置Workflow列表页需要展示的内容和展示效果。
   - 表格内容折行：默认为关闭状态。启用此功能可以让Workflow列表页中的内容在显示时自动换行。禁用此功能可截断文本，Workflow列表页中仅显示部分内容。
   - 操作列：默认为开启状态，启用此能力可让操作列固定在最后一列永久可见。
   - 自定义显示列：默认所有显示项全部勾选，您可以根据实际需要定义您的显示列。
4. 设置完成后，单击“确定”即可。
5. 同时可支持对Workflow显示列进行排序，单击表头中的箭头，就可对该列进行排序。

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 2-4 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 2-5 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。

2.2.2 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 2-6 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 2-7 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 2-8 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）。

2.2.3 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。

有以下三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，运行过程中将产生费用。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，有以下两种操作方式：

- 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。

删除工作流有以下两种方式：

- 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. 在“删除Workflow”对话框，确认删除的信息无误后，输入“DELETE”，单击“确定”，删除Workflow。
- 工作流运行页面
  a. 单击界面右上角的“删除Workflow”，出现删除Workflow询问弹窗。
  b. 在“删除Workflow”对话框，确认删除的信息无误后，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。
  - 删除操作相关的实例和生成的文件不会被删除，运行中的Workflow会停止运行中实例。
  - 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。

2.2.4 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。

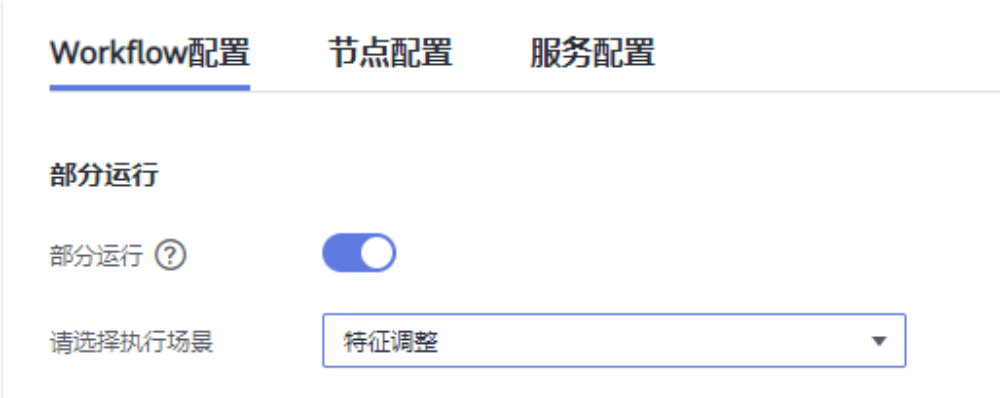
部分运行 Workflow 节点

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 2-9 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。

2.3 开发 Workflow 命令参考

2.3.1 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 2-1 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 2-2 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
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

属性	描述	是否必填	数据类型
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 2-3 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AI 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 2-4 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 2-5 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 2-6 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 2-7 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 2-8 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 2-9 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 2-10 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

属性	描述	是否必填	数据类型
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 2-11 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 2-12 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 2-13 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 2-14 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 2-15 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 2-16 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 2-17 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）
```

2.3.2 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

属性	描述	是否必填	数据类型
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
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")
```

2.3.3 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=False)
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

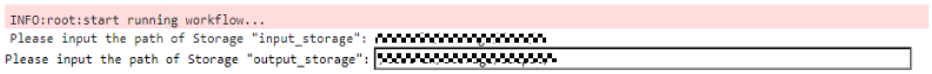
统一存储主要用于JobStep中，下面代码示例全部以单训练节点为例。

```
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 2-10 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 2-11 根目录配置

2.3.4 创建 Workflow 节点

2.3.4.1 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 2-18 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

属性	描述	是否必填	数据类型
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 2-19 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
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 2-20 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

属性	描述	是否必填	数据类型
config	数据集创建节点的输出相关配置。	是	当前仅支持OBSOutputConfig

表 2-21 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
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 2-22 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

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	DAnnotationFormaTypeEtConumfig的列表

表 2-23 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
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**从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]
)
```

2.3.4.2 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 2-24 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

属性	描述	是否必填	数据类型
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 2-25 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 2-26 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 2-27 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

属性	描述	是否必填	数据类型
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 2-28 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
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]
)
```

2.3.4.3 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 2-29 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 2-30 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
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 2-31 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 2-32 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

属性	描述	是否必填	数据类型
excluded_labels	不导入包含指定标签的样本。	否	Label的列表
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 2-33 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 2-34 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]
)
```

2.3.4.4 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 2-35 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 2-36 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 2-37 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]
)
```

2.3.4.5 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 2-38 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 2-39 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 2-40 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 2-41 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 2-42 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 2-43 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 2-44 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 2-45 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 2-46 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 2-47 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 2-48 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 2-49 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 2-50 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 2-51 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 2-52 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AI 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 2-53 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。

2.3.4.6 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 2-54 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 2-55 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 2-56 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 2-57 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 2-58 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 2-59 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 workflow",
    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AI 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")

2.3.4.7 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 2-60 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 2-61 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 2-62 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**VPC管理控制台**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。

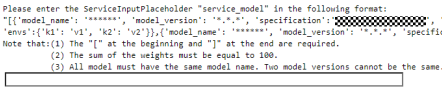
- 针对部署在公共资源池的服务，可以通过**access_address**属性从输出中获取注册在公网的推理地址。
- 针对部署在专属资源池的服务，除了可以获取注册在公网的推理地址，还能通过**cluster_inner_access_address**属性从输出中获取内部使用的推理地址，并且该地址只能在其他推理服务中进行访问。

  ```
  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]
)
```

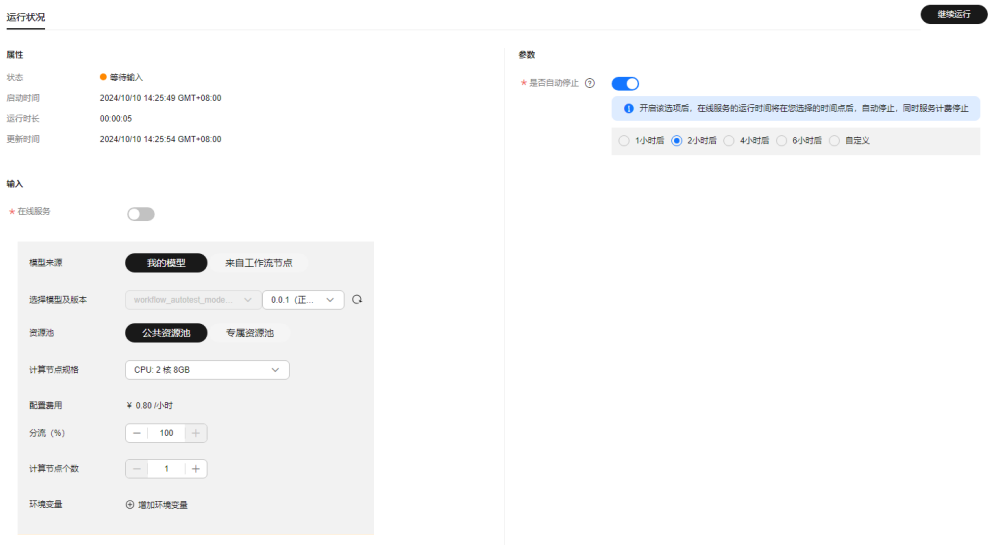
同步推理服务部署相关信息配置操作

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，系统会自动匹配相应的**服务启动参数**。

2.3.5 构建 Workflow 多分支运行场景

2.3.5.1 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，可以简单地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。

2.3.5.2 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，可以简单地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 2-63 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 2-64 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 2-65 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False，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", # 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。

2.3.5.3 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）。

2.3.5.4 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。

2.3.6 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
```

2.3.7 发布 Workflow

2.3.7.1 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**ModelArts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**ModelArts管理控制台**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 an 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。

2.3.7.2 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AI 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AI 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AI 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。

2.3.8 Workflow 高阶能力

2.3.8.1 在 Workflow 中使用大数据能力（ 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ModelArts管理控制台上如何使用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。

2.3.8.2 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。