ModelArts

最佳实践

文档版本 01

发布日期 2025-10-15

华为云计算技术有限公司

商标声明


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

注意

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

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

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 ModelArts 最佳实践案例列表

在最佳实践文档中，提供了针对多种场景、多种AI引擎的ModelArts案例，方便您通过如下案例快速了解使用ModelArts完成AI开发的流程和操作。

图像生成模型训练推理场景

样例	场景	说明
SD系列模型对应Diffusers/ComfyUI框架基于Lite Server适配NPU推理指导（6.5.907）	SD1.5、SDXL、SD3.5、HUNYUAN模型推理	介绍常见的图像生成模型基于ModelArts Lite Server的推理过程，推理使用昇腾NPU计算资源。 启动推理服务后，可应用于图像生成场景。
Stable Diffusion XL基于ModelArts Notebook的推理指导（6.5.907）	SDXL模型推理	介绍常见的图像生成模型基于ModelArts Standard Notebook的推理过程，推理使用昇腾NPU计算资源。 启动推理服务后，可应用于图像生成场景。

视频生成模型训练推理场景

样例	场景	说明
• Wan系列视频生成模型基于Lite Server适配NPU推理指导	Wan系列模型推理	介绍Wan系列模型基于ModelArts Lite Server的推理过程，推理使用PyTorch框架和昇腾NPU计算资源。

ModelArts Standard 权限配置

样例	对应功能	场景	说明
ModelArts Standard 权限管理	IAM权限配置、权限管理	为子账号配置权限	当一个华为云账号下需创建多个IAM子账号时，可参考此样例，为IAM子账号赋予使用ModelArts所需的权限。避免IAM子账号因权限问题导致使用时出现异常。

ModelArts Standard 模型训练案例

表 1-1 自定义算法样例列表

样例	镜像	对应功能	场景	说明
使用ModelArts Standard自定义算法实现手写数字识别	PyTorch	自定义算法	手写数字识别	使用用户自己的算法，训练得到手写数字识别模型，并部署后进行预测。

ModelArts Standard 推理部署

表 1-2 推理部署列表

样例	对应功能	场景	说明
第三方推理框架迁移到ModelArts Standard推理自定义引擎	第三方框架推理部署	-	ModelArts支持第三方的推理框架在ModelArts上部署，本文以TFServing框架、Triton框架为例，介绍如何迁移到推理自定义引擎。

2 DeepSeek 基于 MaaS 推理和应用

2.1 基于 ModelArts Studio（MaaS） DeepSeek API 和 Cherry Studio 快速构建个人 AI 智能助手

操作场景

在人工智能飞速发展的当下，大语言模型（LLMs）已成为推动自然语言处理应用革新的关键力量。DeepSeek作为一系列具有高效计算架构和强大推理能力的大规模语言模型，备受瞩目。

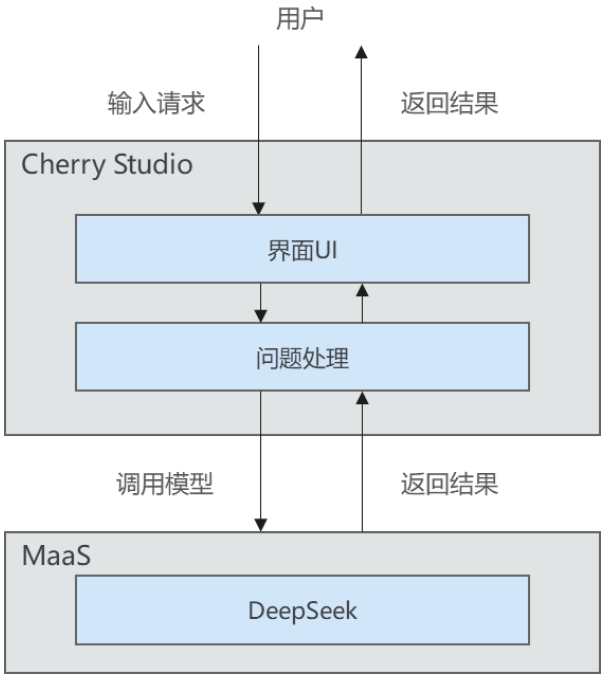
与此同时，为了让用户更便捷地使用这些大语言模型，各种模型调用工具和平台应运而生。Cherry Studio便是其中一款功能强大的开源多模型桌面客户端，支持Windows、macOS和Linux系统。它集成了多种主流大语言模型（例如OpenAI、DeepSeek、Gemini等），并支持本地模型运行。此外，它还具备丰富的功能，例如对话知识库、AI绘画、翻译、多模型切换等。

ModelArts Studio（简称MaaS）服务将DeepSeek系列模型部署到平台，基于AI云服务进行全面适配和优化，使得精度和性能显著提升。MaaS DeepSeek支持广大开发者进行API调用。

本文介绍如何使用MaaS（大模型即服务平台）DeepSeek API接入Cherry Studio，快速构建个人AI智能助手。

方案架构

图 2-1 方案架构



- 用户通过Cherry Studio提交问题请求。
- Cherry Studio将请求发送给MaaS DeepSeek。
- MaaS DeepSeek大预言模型处理请求后将结果返回给Cherry Studio。
- Cherry Studio将结果优化处理后通过界面返回给用户。

计费影响

在MaaS进行模型推理时，会产生计算资源和存储资源的累计值计费。计算资源为运行模型服务的费用。存储资源包括数据存储到OBS的计费。在调用MaaS预置服务时，将根据实际使用的Tokens数量进行计费。更多信息，请参见**ModelArts Studio（MaaS）模型推理计费项**。

前提条件

- 已注册华为云账号。具体操作，请参见**注册华为账号并开通华为云**。
- 已完成ModelArts委托授权。具体操作，请参见**配置ModelArts Studio（MaaS）访问授权**。

步骤一：下载并安装 Cherry Studio

您可以通过**官方网站**或**开源地址**下载并安装Cherry Studio。

步骤二：获取 MaaS DeepSeek 对接信息

获取MaaS DeepSeek模型服务的关键信息，用于后续Cherry Studio对接MaaS DeepSeek。

1. 创建API Key，用于调用MaaS DeepSeek模型服务时的鉴权认证。

   最多可创建30个密钥。每个密钥仅在创建时显示一次，请确保妥善保存。如果密钥丢失，无法找回，需要重新创建API Key以获取新的访问密钥。

   a. 登录**ModelArts Studio（MaaS）控制台**，在顶部导航栏选择“中国-香港”区域。
   b. 在左侧导航栏，单击“API Key管理”。
   c. 在“API Key管理”页面，单击“创建API Key”，填写标签和描述信息后，单击“确定”。

      标签和描述信息在创建完成后，不支持修改。

      表 2-1 创建 API Key 参数说明

参数	说明
标签	自定义API Key的标签。标签具有唯一性，不可重复。仅支持大小写英文字母、数字、下划线、中划线，长度范围为1~100个字符。
描述	自定义API Key的描述，长度范围为1~100个字符。

   d. 在“您的密钥”对话框，复制密钥并保存至安全位置。
   e. 保存完毕后，单击“关闭”。

      单击“关闭”后将无法再次查看密钥。

2. 获取MaaS DeepSeek模型服务的API地址和模型名称。

   下文以部署模型服务为例进行说明。您也可以使用商用服务进行API调用，详情请参见**在ModelArts Studio（MaaS）预置服务中开通商用服务**。

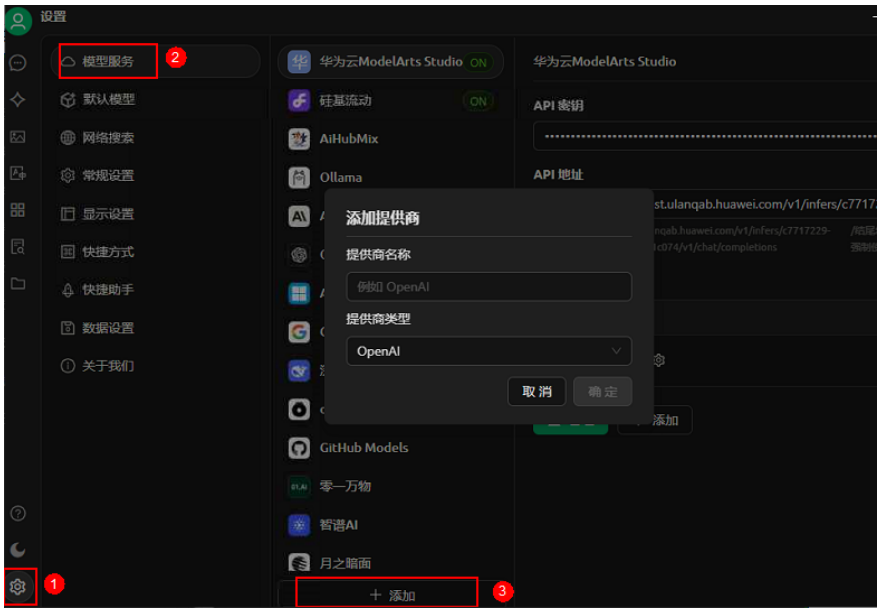
   a. 在**ModelArts Studio（MaaS）控制台**左侧导航栏，单击“在线推理”。
   b. 在“在线推理”页面，单击“我的服务”页签，在右上角单击“部署模型服务”，创建模型部署服务。具体操作，请参见**使用MaaS部署模型服务**。
   c. 在“状态”为“运行中”的模型服务右侧，单击操作列下的“更多 > 调用说明”。
   d. 在“调用说明”页面，可以查看调用该服务需要的基础API地址和模型名称信息，在后续Cherry Studio配置中使用。

步骤三：在 Cherry Studio 中配置 MaaS DeepSeek

1. 在Cherry Studio添加MaaS提供商。

   a. 在Cherry Studio客户端左下角，单击设置图标，在“模型服务”中单击“添加”。

图 2-2 添加提供商



b. 在“添加提供商”对话框，配置提供商名称和提供商类型，然后单击“确定”。

表 2-2 添加提供商参数说明

参数	说明
提供商名称	配置为“华为云ModelArts Studio”，您可以按需修改。
提供商类型	配置为“OpenAI”。

2. 在Cherry Studio添加MaaS DeepSeekAPI密钥和API地址。
   a. 在Cherry Studio客户端左下角，单击设置图标。
   b. 在“设置”页面，找到“华为云ModelArts Studio”选项，配置API密钥和API地址。

参数	说明
API密钥	**步骤二：获取MaaS DeepSeek对接信息**中创建的API Key。
API地址	**步骤二：获取MaaS DeepSeek对接信息**获取的MaaS服务的基础API地址，需要去掉地址尾部的“/v1/chat/completions”后填入。

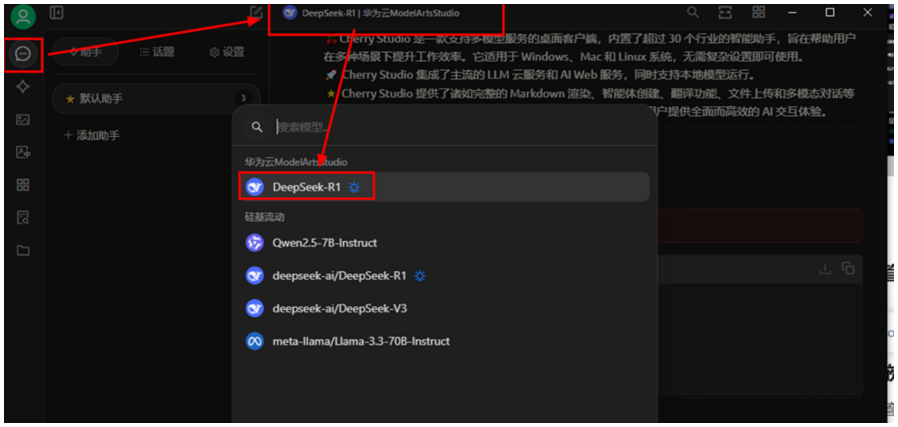
3. 在Cherry Studio添加模型。
   a. 在“模型”区域，单击“添加”。
   b. 在“添加模型”对话框，配置模型ID、模型名称和分组名称，单击“添加模型”。

参数	说明
模型 ID	**步骤二：获取MaaS DeepSeek对接信息**获取的模型名称。
模型名称	自定义模型名称。
分组名称	自定义分组名称。

步骤四：在 Cherry Studio 中使用 MaaS DeepSeek

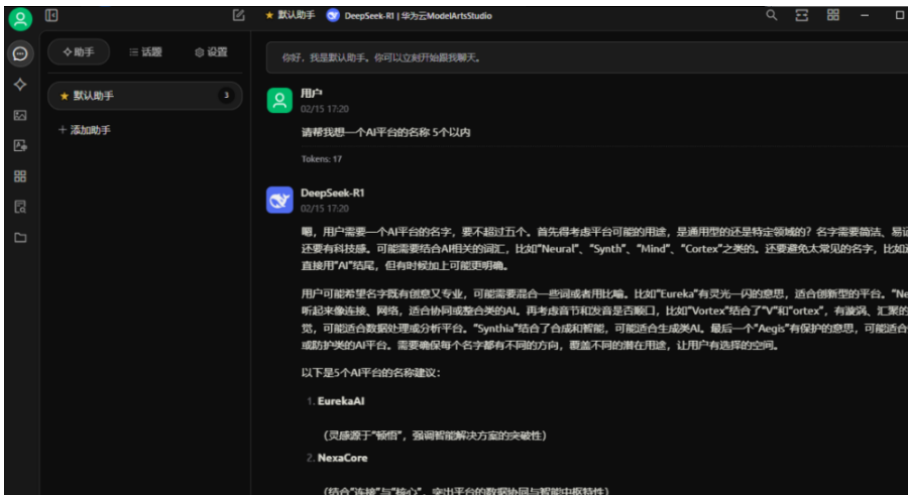
1. 在Cherry Studio左侧导航栏，单击  图标，选择已配置好的模型。

图 2-3 选择模型



2. 在文本框中输入文字，开始对话。
   您也可以选择顶部菜单中的模型名字切换模型。

图 2-4 模型问答示例



常见问题

- **在ModelArts Studio（MaaS）创建API Key后需要等待多久才能生效?**

MaaS API Key在创建后不会立即生效，通常需要等待几分钟才能生效。

- **MaaS如何接入Cherry Studio、Cursor、Cline?**

  MaaS集成了DeepSeek大模型，支持对接多个平台进行AI开发，详情请参见**使用ModelArts Studio（MaaS）DeepSeek API搭建AI应用**。

2.2 基于 ModelArts Studio（MaaS）DeepSeek API 和 Cursor 快速构建代码编辑器

本文介绍如何使用Cursor调用部署在ModelArts Studio上的DeepSeek模型，构建代码编辑器。

操作场景

Cursor是一款基于人工智能技术的现代化代码编辑器，专为开发者设计。它结合了传统编辑器（例如VS Code）的强大功能与AI驱动的智能编程能力，能够提供智能代码补全、自然语言编程、代码库理解等核心功能，极大地提升了开发效率。此外，Cursor支持多种主流AI模型（例如OpenAI的GPT-4、DeepSeek等），并提供灵活的自定义配置选项，适合从初学者到专业开发者的广泛用户群体。

ModelArts Studio（简称MaaS）服务将DeepSeek系列模型部署到平台，支持广大开发者进行API调用。

前提条件

- 已注册华为云账号。具体操作，请参见**注册华为账号并开通华为云**。
- 已完成ModelArts委托授权。具体操作，请参见**配置ModelArts Studio（MaaS）访问授权**。

步骤一：下载并安装 Cursor

通过**Cursor官网**下载并安装Cursor。

步骤二：MaaS 模型 API 接入准备

1. 创建API Key。

   最多可创建30个密钥。每个密钥仅在创建时显示一次，请确保妥善保存。如果密钥丢失，无法找回，需要重新创建API Key以获取新的访问密钥。

   a. 登录**ModelArts Studio（MaaS）控制台**，在顶部导航栏选择“中国-香港”区域。

   b. 在左侧导航栏，单击“API Key管理”。

   c. 在“API Key管理”页面，单击“创建API Key”，填写标签和描述信息后，单击“确定”。

      标签和描述信息在创建完成后，不支持修改。

表 2-3 创建 API Key 参数说明

参数	说明
标签	自定义API Key的标签。标签具有唯一性，不可重复。仅支持大小写英文字母、数字、下划线、中划线，长度范围为1~100个字符。
描述	自定义API Key的描述，长度范围为1~100个字符。

d. 在“您的密钥”对话框，复制密钥并保存至安全位置。

e. 保存完毕后，单击“关闭”。

单击“关闭”后将无法再次查看密钥。

2. 使用我的服务接入。

下文以我的服务为例进行说明。您也可以使用商用服务进行API调用，详情请参见**在ModelArts Studio（MaaS）预置服务中开通商用服务**。

a. 在**ModelArts Studio（MaaS）控制台**左侧导航栏，单击“在线推理”。

b. 在“在线推理”页面，单击“我的服务”页签，在右上角单击“部署模型服务”，创建模型部署服务。具体操作，请参见**使用MaaS部署模型服务**。

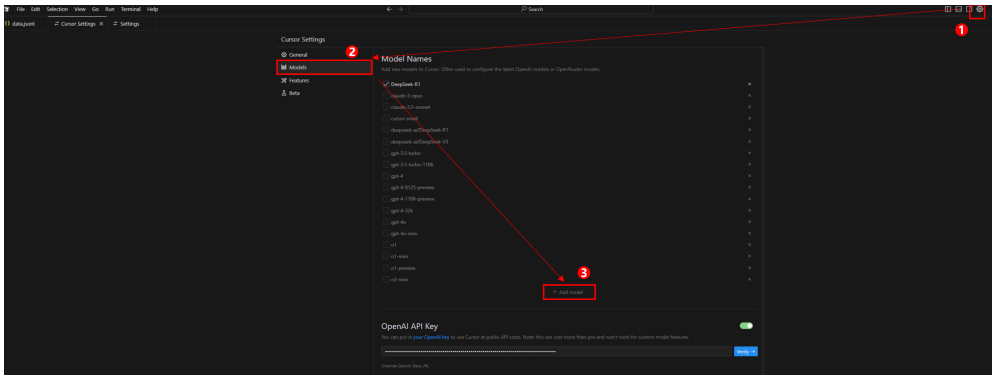
c. 在“状态”为“运行中”的模型服务右侧，单击操作列下的“更多 > 调用说明”。

d. 在“调用说明”页面，可以查看调用该服务需要的基础API地址和模型名称信息，在后续Cursor配置中使用。

步骤三：在 Cursor 中配置 MaaS API

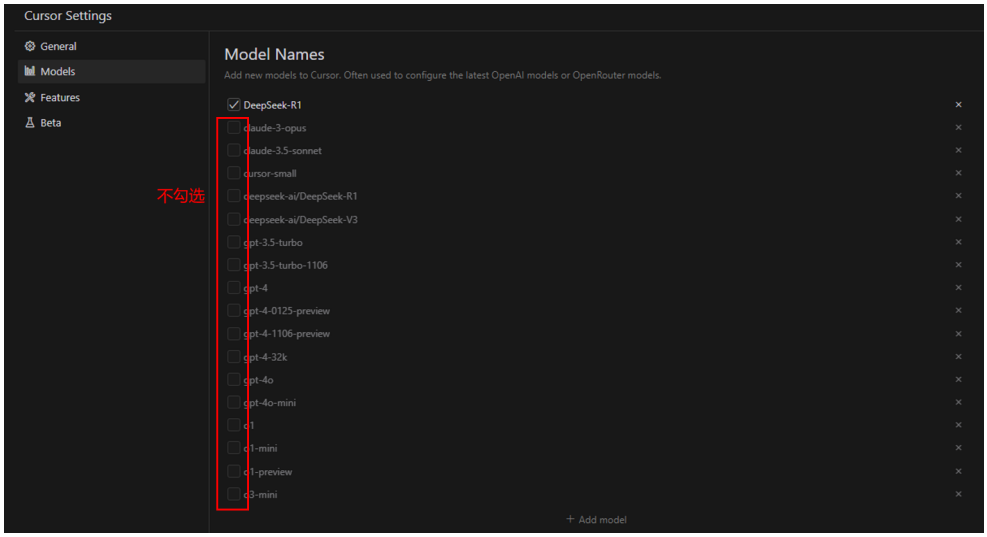
1. 在Cursor平台右上角单击设置图标。
2. 在“Cursor Settings”页面左侧导航栏，单击“Models”，然后单击“Add model”。

图 2-5 添加模型



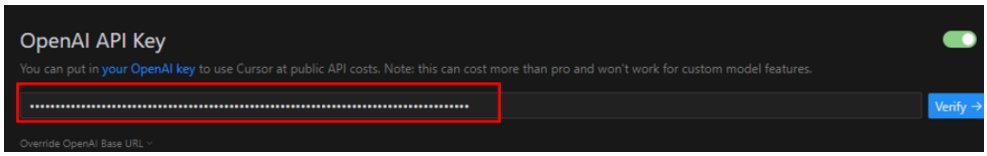
3. 在文本框中输入**步骤二.2**获取的模型名称，然后单击右侧的“Add model”。
4. 仅勾选刚添加的MaaS模型，其余模型去勾选（否则验证时可能会出现调不通的问题）。

图 2-6 勾选 MaaS 模型



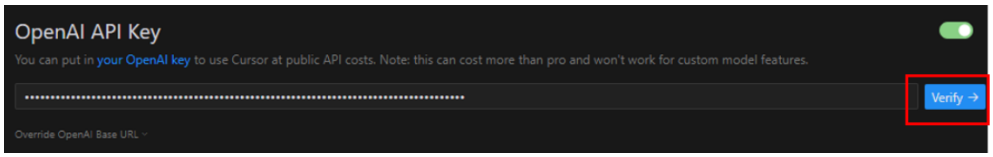
5. 在“OpenAI Key”区域填写**步骤二.1**创建的API Key。

图 2-7 填写 API Key



6. 单击“Override Openai Base URL”，修改基础接口地址，填入**步骤二.2**获取的接口地址（需去掉尾部的/chat/completions），单击“Save”。
7. 单击“Verify”验证接口连通性。如果无报错信息则配置成功，可以开始使用。

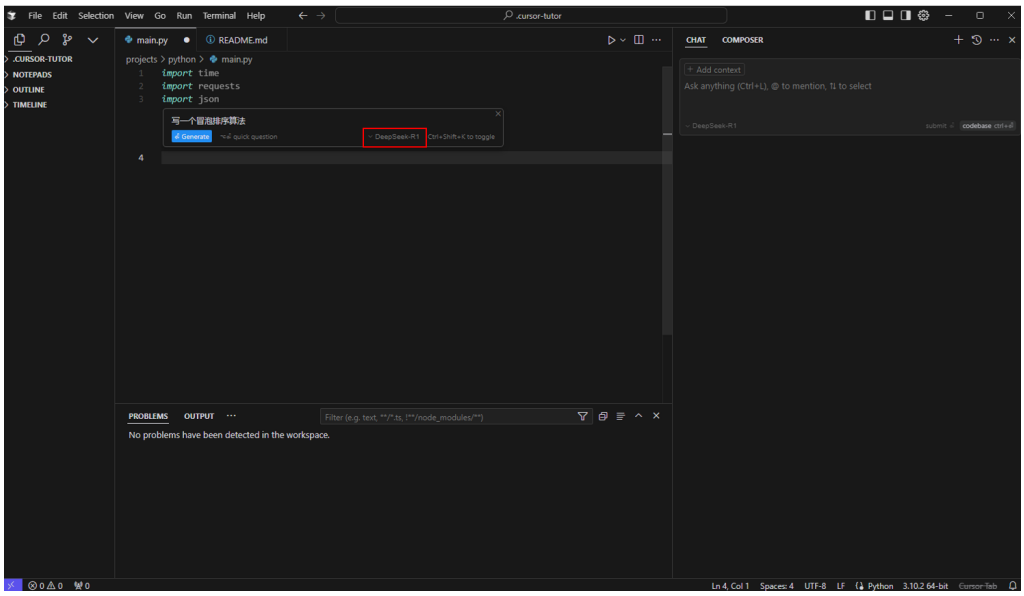
图 2-8 验证接口连通性



步骤四：在 Cursor 中使用 MaaS API 进行代码生成

在代码编辑页面，红框处选择刚配置好的模型即可进行对话、代码生成、代码解析等操作。

图 2-9 使用 MaaS API



2.3 基于 ModelArts Studio（MaaS） DeepSeek API 和 Cline 快速构建 AI 编程助手

本文介绍如何使用Cline调用部署在ModelArts Studio上的DeepSeek模型，构建AI编程助手。

操作场景

Cline是一款基于大语言模型的VS Code插件，能够有效处理复杂的软件开发任务。借助VS Code开发平台，Cline为开发者带来了便捷高效的编程体验。Cline的优势如下：

- 深度融合ModelArts Studio（简称MaaS）平台：Cline支持接入MaaS平台的DeepSeek系列模型服务。
- 文件管理与代码纠错：支持便捷地创建和编辑文件，实时监控Linter和编译器的错误信息。一旦发现代码中存在缺少导入、语法错误等问题，能迅速进行分析诊断，并给出对应的修复建议，极大地提升代码编写的流畅性和准备性，保障开发工作顺利进行。
- 终端交互与即时响应：集成便捷的终端交互界面，允许开发者在其中执行各类命令，并实时查看命令输出。当文件编辑完成后，Cline可帮助开发者快速定位并解决服务器出现的问题，使开发流程紧密衔接，有效提高开发效率。
- Web一站式解决方案：对于Web开发任务，Cline可以在无头浏览器中启动网站，自动模拟用户的单击、输入、滚动等操作，并实时捕获截图和控制台日志。通过对这些数据的深入分析，精准定位并修复运行时的错误和视觉错误，确保Web应用的高质量交付。

前提条件

- 已注册华为云账号。具体操作，请参见**注册华为账号并开通华为云**。
- 已完成ModelArts委托授权。具体操作，请参见**配置ModelArts Studio（MaaS）访问授权**。

支持的模型

支持文本生成类的模型，且模型的上下文长度（序列长度）≥32K。

您可以登录**ModelArts Studio（MaaS）控制台**，在“模型广场”页面的“筛选”区域，“模型类型”选择“文本生成”，“上下文长度”选择“32K”和“64K”，查看支持的模型。

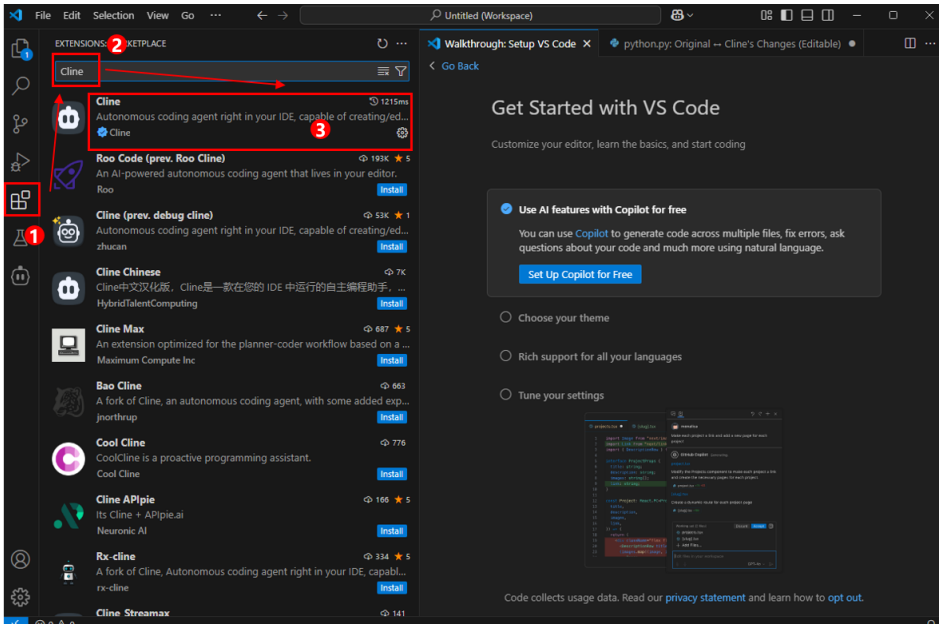
图 2-10 在模型广场查看支持的模型

步骤一：在 VS Code 中安装 Cline

1. 打开VS Code，在左侧导航栏单击图标，在搜索框中输入“Cline”，然后单击“Install”。

   在左侧导航栏出现小机器人图标，表示Cline安装完成。

图 2-11 安装 Cline



步骤二：MaaS 模型 API 接入准备

1. 创建API Key。

   每个密钥仅在创建时显示一次，请确保妥善保存。如果密钥丢失，无法找回，需要重新创建API Key以获取新的访问密钥。

   a. 登录**ModelArts Studio（MaaS）控制台**，在顶部导航栏选择“中国-香港”区域。

   b. 在左侧导航栏，单击“API Key管理”。

   c. 在“API Key管理”页面，单击“创建API Key”，填写标签和描述信息后，单击“确定”。

      标签和描述信息在创建完成后，不支持修改。

      表 2-4 创建 API Key 参数说明

参数	说明
标签	自定义API Key的标签。标签具有唯一性，不可重复。仅支持大小写英文字母、数字、下划线、中划线，长度范围为1~100个字符。
描述	自定义API Key的描述，长度范围为1~100个字符。

   d. 在“您的密钥”对话框，复制密钥并保存至安全位置。

   e. 保存完毕后，单击“关闭”。

      单击“关闭”后将无法再次查看密钥。

2. 使用我的服务接入。

   下文以我的服务为例进行说明。您也可以使用商用服务进行API调用，详情请参见**在ModelArts Studio（MaaS）预置服务中开通商用服务**。

   a. 在**ModelArts Studio（MaaS）控制台**左侧导航栏，单击“在线推理”。

   b. 在“在线推理”页面，单击“我的服务”页签，在右上角单击“部署模型服务”，创建模型部署服务。具体操作，请参见**使用MaaS部署模型服务**。

   c. 在“状态”为“运行中”的模型服务右侧，单击操作列下的“更多 > 调用说明”。

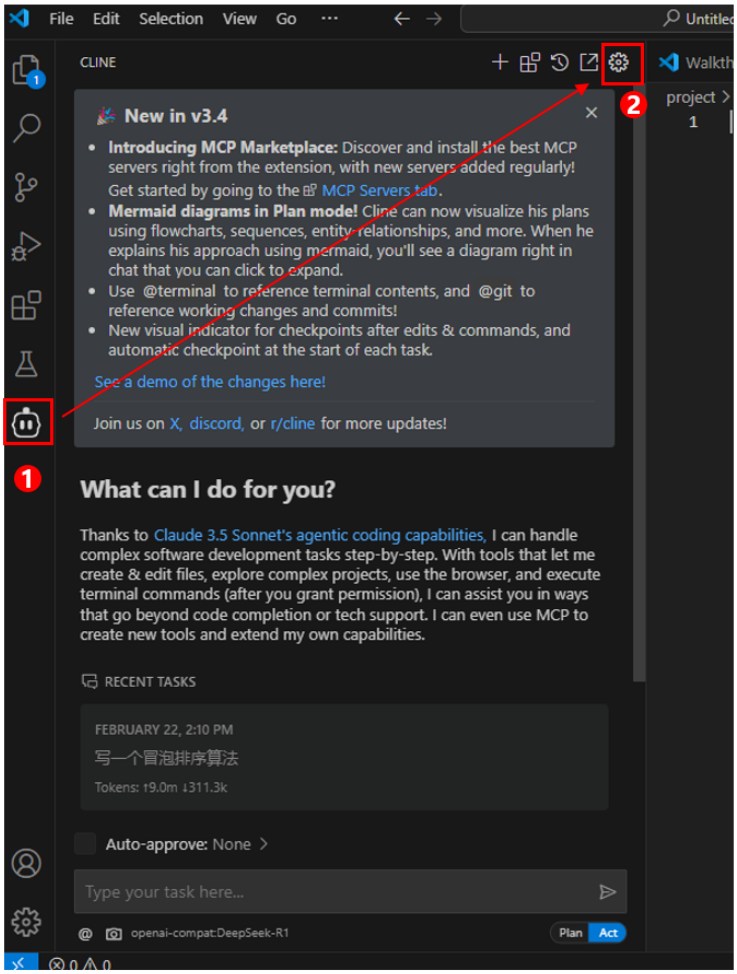
   d. 在“调用说明”页面，可以查看调用该服务需要的基础API地址和模型名称信息，在后续Cline配置中使用。

步骤三：在 Cline 中配置 MaaS API

1. 配置MaaS模型服务。

   a. 打开VS Code，在左侧导航栏单击  图标，打开Cline插件，在右上角单击  图标。

图 2-12 打开 Cline 插件



b. 在“Settings”页面，配置相关信息，然后单击“Done”。

表 2-5 Cline 配置说明

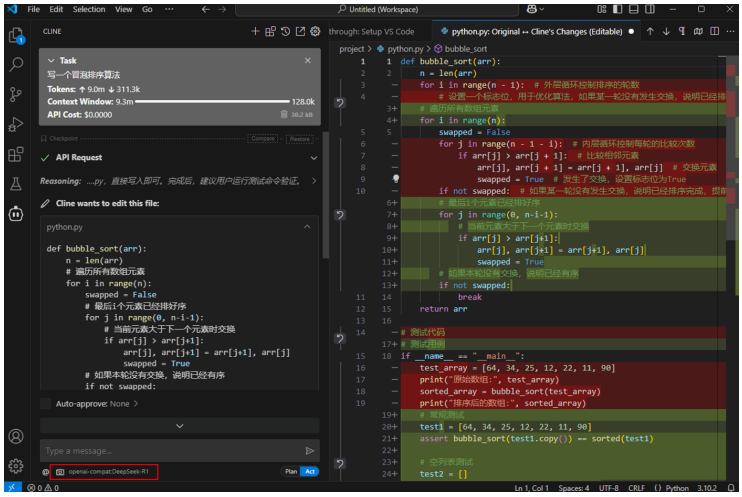
参数	说明
API Provider	选择“OpenAI Compatible”。
Base URL	步骤二.2获取的API地址，需要去掉尾部的“/chat/completions”后填入。
API Key	步骤二.1创建的API Key。
Model ID	步骤二.2获取的模型名称。

2. 通过VS Code的Cline插件调用MaaS API实现自动化代码生成。

a. 在VS Code左侧导航栏，单击  图标。

b. 在左下角红框处选择已配置的MaaS服务，进行对话和代码辅助生成。Cline功能强大，可以进行代码生成，对写好的代码进行纠错、优化等操作。

图 2-13 代码生成示例



3 图像生成模型训练推理

3.1 SD 系列模型对应 Diffusers/ComfyUI 框架基于 Lite Server 适配 NPU 推理指导（6.5.907）

本文档主要介绍如何在ModelArts Lite Server环境中部署文生图模型Stable Diffusion系列、HUNYUAN模型，/对应Diffusers、ComfyUI框架，使用NPU卡进行推理。

方案概览

本方案介绍了在Server上使用NPU计算资源部署Diffusers、ComfyUI框架用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推理部署推荐使用ModelArts Lite Server的Snt9B和Snt9B23资源。

表 3-1 环境要求

名称	版本
driver	25.2.1
PyTorch	pytorch_2.5.1

获取软件和镜像

表 3-2 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.5.907软件包中的AscendCloud-AIGC-6.5.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts **6.5.907.2**版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
Snt9b基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129 中国-香港： swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	从SWR拉取。
Snt9b23基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129 中国-香港： swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129	从SWR拉取。

表 3-3 支持特性

套件类型	模型名称
Diffusers	SD1.5 SDXL SD3.5 HUNYUAN
ComfyUI	SD1.5 SDXL SD3.5

步骤一：准备环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. 检查环境。

   a. SSH登录机器后，检查NPU设备状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   b. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

   c. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

3. 获取基础镜像。建议使用官方提供的镜像部署推理服务。镜像地址{image_url}参见**表3-2**。

   ```
   docker pull {image_url}
   ```

   如需登录，请**登录SWR控制台**，参考以下图示获取登录指令。

步骤二：启动容器镜像

启动snt9b容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

```
docker run -itd \
--name ${container_name} \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-p 8183:8183 \
-v /etc/localtime:/etc/localtime \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
--shm-size 60g \
--device=/dev/davinci_manager \
--device=/dev/hisi_hdc \
--device=/dev/devmm_svm \
--device=/dev/davinci3 \
--network=host \
${image_name} bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如comfyui。
- --device=/dev/davinci3：挂载主机的/dev/davinci3到容器的/dev/davinci3。可以使用npu-smi info查看空闲卡号，修改davinci后数字可以更改挂载卡。
- 若需要启动多卡推理服务，则需要挂载多卡，例如再添加一个--device=/dev/davinci2
- ${image_name} 代表镜像名称。
- -p 8183:8183 开启一个端口，可以通过 http://宿主机IP:8183 访问容器服务（如冲突，可自行更换其他端口）。

1. 进入**snt9b**容器。需要将${container_name}替换为实际的容器名称，例如：comfyui。
   ```
   docker exec -it ${container_name} bash
   ```
2. 启动**snt9b23**容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。
   ```
   docker run -itd \
   --privileged \
   --name ${container_name} \
   -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
   -p 8183:8183 \
   -v /etc/localtime:/etc/localtime \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
   ```

```
--shm-size 60g \
--device=/dev/davinci_manager \
--device=/dev/hisi_hdc \
--device=/dev/devmm_svm \
--device=/dev/davinci3 \
--network=host \
${image_name} bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如comfyui。
- --device=/dev/davinci3：挂载主机的/dev/davinci3到容器的/dev/davinci3。可以使用npu-smi info查看空闲卡号，修改davinci后数字可以更改挂载卡。
- 若需要启动多卡推理服务，则需要挂载多卡，例如再添加一个--device=/dev/davinci2
- ${image_name} 代表镜像名称。
- -p 8183:8183 开启一个端口，可以通过 http://宿主机IP:8183 访问容器服务（如冲突，可自行更换其他端口）。

3. 进入**snt9b23**容器。需要将${container_name}替换为实际的容器名称，例如：comfyui。

```
docker exec -itu root ${container_name} bash
```

步骤三：Diffusers 部署

安装依赖和模型包

1. 使用如下命令登录huggingface，并输入个人账号的token，用于自动下载模型权重。

   登录成功后，直接启动Diffusers推理脚本即可实现自动下载。

   ```
   huggingface-cli login
   ```

   也可以手动下载模型权重，上传到容器的/home/ma-user目录下，官网下载地址（需登录）。

   - SD1.5下载链接：**https://huggingface.co/stable-diffusion-v1-5/stable-diffusion-v1-5**
   - SDXL下载链接：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
   - SD3.5-medium下载链接：**https://huggingface.co/stabilityai/stable-diffusion-3.5-medium/tree/main**
   - SD3.5-large下载链接：**https://huggingface.co/stabilityai/stable-diffusion-3.5-large/tree/main**
   - HUNYUAN下载链接：**https://huggingface.co/Tencent-Hunyuan/HunyuanDiT-Diffusers/tree/main**

2. 安装插件代码包。

   a. 将获取到的插件代码包AscendCloud-AIGC-xxx.zip文件上传到容器的/home/ma-user/temp目录下，并解压。插件代码包获取地址参见**表3-2**。

   ```
   mkdir -p /home/ma-user/temp
   cd /home/ma-user/temp
   unzip AscendCloud-AIGC-*.zip #解压
   ```

   b. 将AIGC包解压后，进入到获取的/home/ma-user/temp/aigc_inference/torch_npu/utils/ascend_diffusers目录下，安装ascend_diffusers包。

   ```
   cd /home/ma-user/temp/aigc_inference/torch_npu/utils/ascend_diffusers
   pip install -e .
   ```


c. 将AIGC包解压后，进入到获取的/home/ma-user/temp/aigc_inference/torch_npu/utils/AscendX-MM目录下，安装AscendX-MM包。

```
cd /home/ma-user/temp/aigc_inference/torch_npu/utils/AscendX-MM
pip install -e .
```

启动服务

```
export MODEL_PATH='下载好的huggingface模型路径'，例如/home/ma-user/stable-diffusion-3.5-medium。如果未手动下载，想要自动下载则不加model_id参数即可。
cd /home/ma-user/temp/aigc_inference/torch_npu/diffusers/0.31.0/examples
```

单卡模型推理启动服务命令如下，详细参数请参考/home/ma-user/temp/aigc_inference/torch_npu/diffusers目录下的Readme文件。

- SD1.5模型推理启动命令：

```
pip install diffusers==0.30.2
python sd_inference_example.py --model_name sd15 --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 20 --width 512 768 1024 --height 512 768 1024
```

- SDXL模型推理启动命令：

```
pip install diffusers==0.30.2
python sd_inference_example.py --model_name sdxl --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 20 --width 768 1024 --height 768 1024
```

- SD3.5模型推理启动命令：

```
pip install diffusers==0.31.0
python sd_inference_example.py --model_name sd35 --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 28 --width 512 768 1024 --height 512 768 1024
```

- HUNYUAN模型推理启动命令：

```
pip install diffusers==0.30.2
export INF_NAN_MODE_FORCE_DISABLE=1
python sd_inference_example.py --model_name hunyuan --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 20 --width 512 768 1024 --height 512 768 1024
```

步骤四：ComfyUI 部署

安装依赖和模型包

1. 下载ComfyUI软件包。

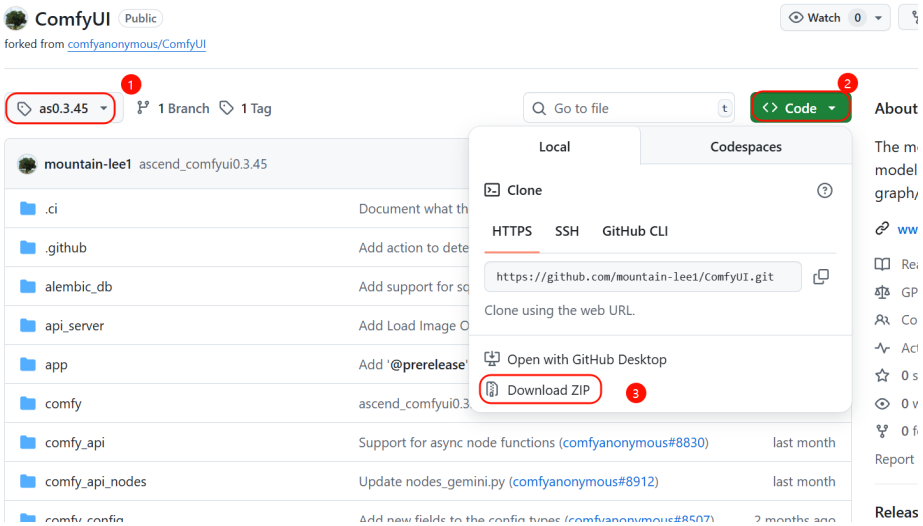
   下载ComfyUI源码。

   ```
   git clone -b as0.3.45 https://github.com/mountain-lee1/ComfyUI.git
   cd ComfyUI
   ```

   如果上述方法无法下载ComfyUI源码，可参考如下操作，手动下载到本地再上传到容器中，如**图3-1**所示。

   a. 登录https://github.com/mountain-lee1/ComfyUI页面，切换Tag为as0.3.45，单击Code按钮，通过Download ZIP下载ComfyUI源码到本地。

图 3-1 下载 ComfyUI 源码



须知

访问Github下载开源软件需要连通外网，请用户自行配置网络代理。

b. 将下载好的ComfyUI-as0.3.45.zip文件上传到容器的/home/ma-user/目录下，并解压。

```
cd /home/ma-user/
unzip ComfyUI-as0.3.45.zip
cd ComfyUI-as0.3.45
```

2. 安装依赖，将requirements.txt中的torch修改为torch==2.5.1。

```
pip install -r requirements.txt
```

3. 下载模型权重。

sd1.5：将v1-5-pruned-emaonly.safetensors复制到ComfyUI/models/checkpoints目录下。

https://huggingface.co/runwayml/stable-diffusion-v1-5/resolve/main/v1-5-pruned-emaonly.safetensors

sdxl：将sd_xl_base_1.0.safetensors复制到ComfyUI/models/checkpoints目录下。

https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/resolve/main/sd_xl_base_1.0.safetensors

sd3.5：将sd3.5_medium.safetensors复制到ComfyUI/models/checkpoints目录下。

https://www.modelscope.cn/models/cutemodel/comfyui-sd3.5-medium/file/view/master/sd3.5_medium.safetensors?status=2

将diffusion_pytorch_model.safetensors复制到ComfyUI/models/vae目录下。

https://www.modelscope.cn/models/cutemodel/comfyui-sd3.5-medium/file/view/master/sd3.5vae.safetensors?status=2

此外需要额外下载三个text_encoder相关模型：复制到ComfyUI/models/clip目录下。

https://huggingface.co/Comfy-Org/stable-diffusion-3.5-fp8/blob/main/text_encoders/clip_l.safetensors

https://huggingface.co/Comfy-Org/stable-diffusion-3.5-fp8/blob/main/text_encoders/clip_g.safetensors

https://huggingface.co/Comfy-Org/stable-diffusion-3.5-fp8/blob/main/text_encoders/t5xxl_fp16.safetensors

ComfyUI框架还需要下载推理所需的workflow：

https://openart.ai/workflows/sneakyrobot/sd35-basic/CX6pkiT9lzJPlTpF9Cgu

4. 安装插件代码包。

   a. 将获取到的插件代码包AscendCloud-AIGC-xxx.zip文件上传到容器的/home/ma-user/目录下，并解压。插件代码包获取请参见表3-2。

   ```
   cd /home/ma-user/
   unzip AscendCloud-AIGC-*.zip
   ```

   b. 进入ComfyUI/custom_nodes目录下，将解压AIGC包后获取的aigc_inference/torch_npu/comfyui/0.3.7/comfyui_ascend_node文件夹复制到该目录下。

   ```
   cd ComfyUI/custom_nodes
   cp -r /home/ma-user/aigc_inference/torch_npu/comfyui/0.3.45/comfyui_ascend_node /home/ma-user/ComfyUI/custom_nodes
   ```

   c. 进入到获取的aigc_inference/torch_npu/utils/ascend_diffusers目录下，安装ascend_diffusers包。

   ```
   cd /home/ma-user/aigc_inference/torch_npu/utils/ascend_diffusers
   pip install -e .
   ```

   d. 进入到获取的aigc_inference/torch_npu/utils/AscendX-MM目录下，安装AscendX-MM包。

   ```
   cd /home/ma-user/aigc_inference/torch_npu/utils/AscendX-MM
   pip install -e .
   ```

开启高性能模式

SD模型开启高性能模式，按启动服务执行。

```
export CACHE_MODE=1
```

启动服务

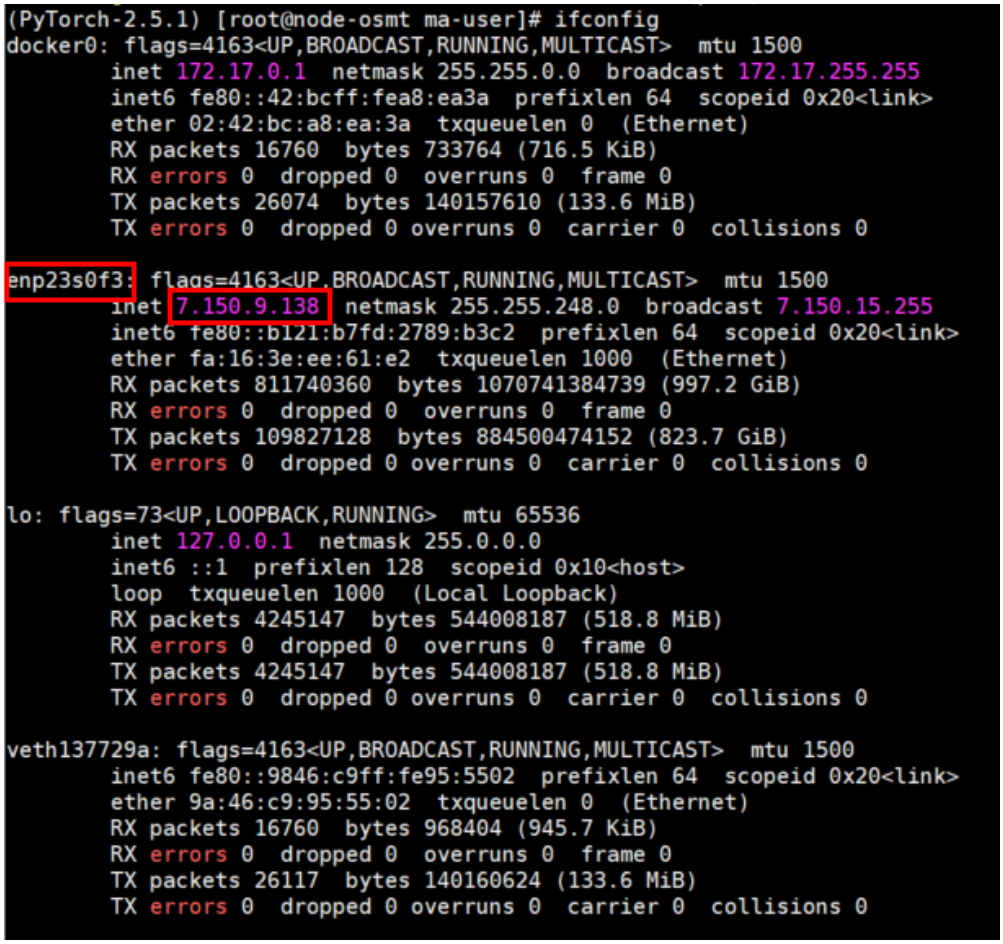
1. 用ifconfig命令获取容器IP（若无效可使用ip addr，或者自行寻找其他方式获取到容器IP）。

图 3-2 snt9b 获取容器 IP

```
(PyTorch-2.1.0) [ma-user@57a5b008b312 custom_nodes]$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.7  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:07  txqueuelen 0  (Ethernet)
        RX packets 6581  bytes 5071762 (4.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2651  bytes 148084 (144.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 3-3 snt9b23 获取容器 IP



2. 进入目录。

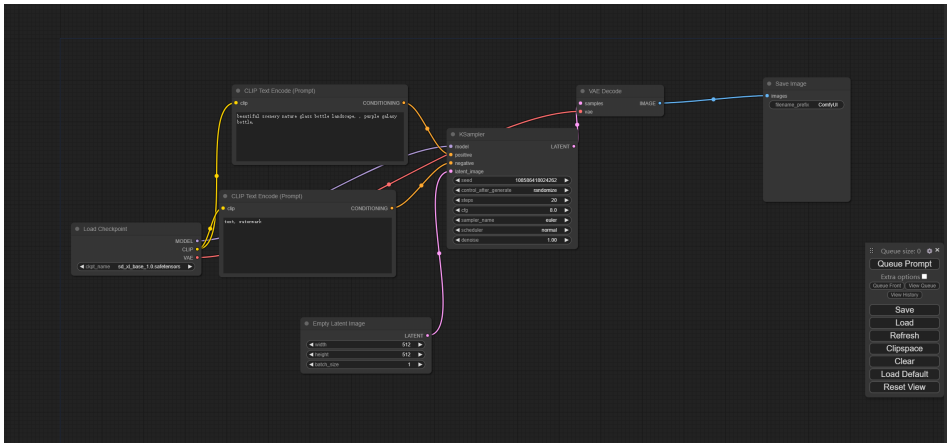
```
cd /home/ma-user/ComfyUI/custom_nodes
git config --global http.sslVerify false #根据不同workflow，可能需要下载新的节点
git clone https://github.com/ltdrdata/ComfyUI-Manager ComfyUI/custom_nodes/ComfyUI-Manager  #下载comfyUI管理器，便于后面下载节点
cd /home/ma-user/ComfyUI
```

3. 启动服务命令如下。

```
python main.py --port 8183 --listen 172.17.0.7 --force-fp16 --bf16-unet
```

4. 使用http://{宿主机ip}:8183可以访问前端页面。
   a. 如下运行文生图。

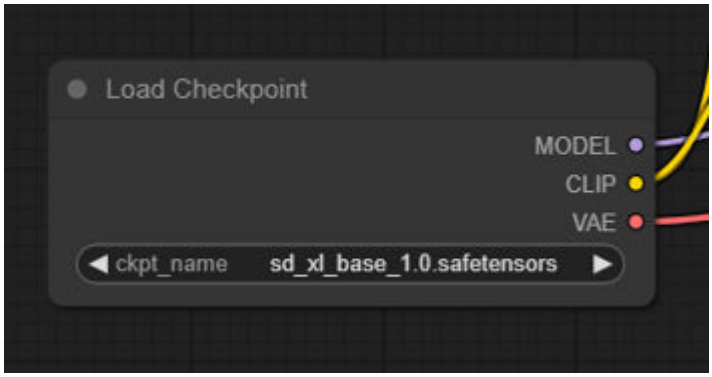
图 3-4 1 访问前端页面



除使用默认workflow外，也可加载其他workflow（如SD3.5提供的workflow），可能因为workflow内含有没有安装的节点报错，此时可以使用刚安装的comfyUI管理器对缺失节点进行下载安装，安装完需重启comfyUI服务（建议直接在终端重新输入启动服务命令），按照各节点需加载模型选择相应模型，再进行推理服务。

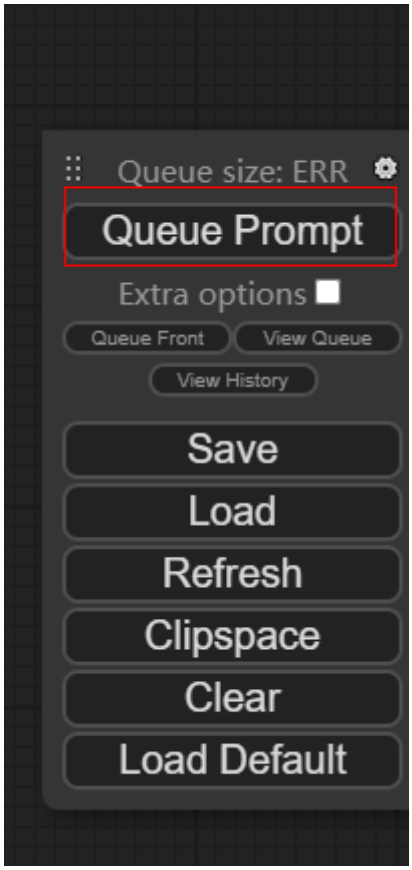
根据上面checkpoint的箭头，对新的npu的checkpoint进行规划，如下图。

图 3-5 规划 checkpoint



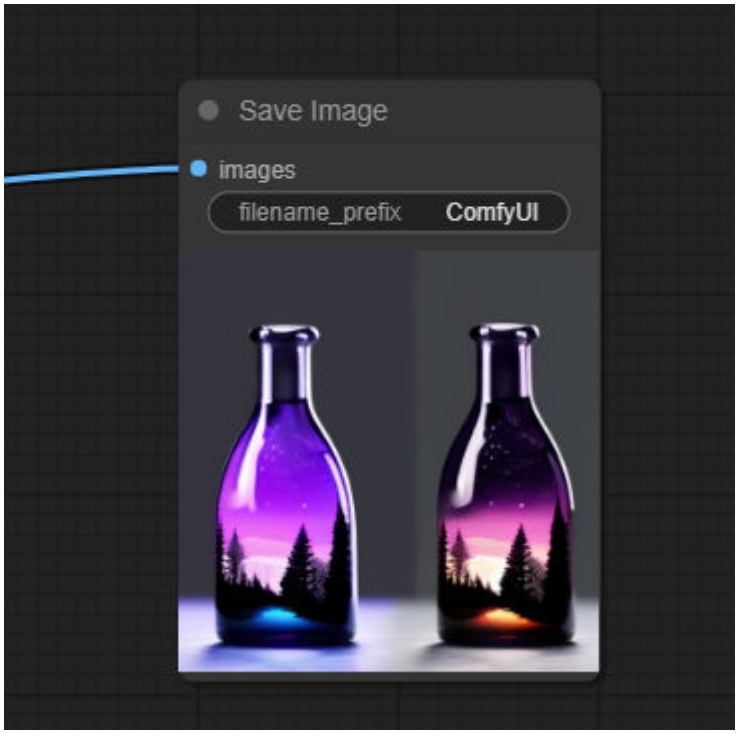
在ckpt_name中选择要使用的权重文件，单击Queue Prompt加入推理队列进行推理，如下图。

图 3-6 进入推理队列



成功之后结果如下图。

图 3-7 推理成功



3.2 Stable Diffusion XL 基于 ModelArts Notebook 的推理指导（6.5.907）

方案概览

本方案主要介绍如何在ModelArts Notebook环境中部署文生图模型Stable Diffusion XL对应的Diffusers框架，并使用NPU计算资源进行推理。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推理部署推荐使用中国-香港Region上的Notebook和Snt9B资源。

获取软件

表 3-4 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.5.907软件包中的AscendCloud-AIGC-6.5.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts **6.5.907.2**版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
Notebook使用镜像	Snt9B：西南-贵阳一 swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129 Snt9B：中国-香港 swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	需要将镜像注册到ModelArts中，供创建Notebook实例时选择。

支持特性

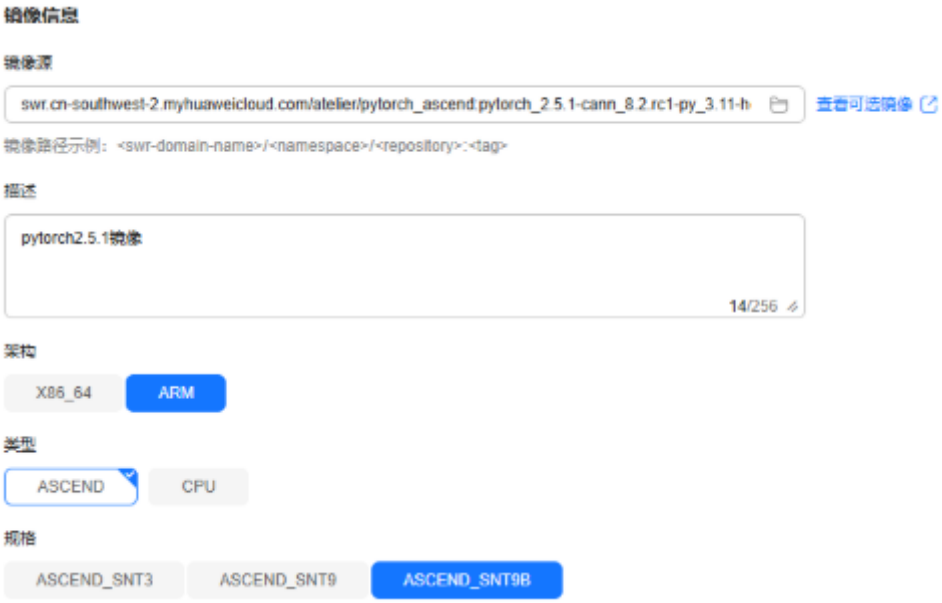
表 3-5 支持特性

套件类型	模型名称
Diffusers	SDXL

步骤一：准备环境

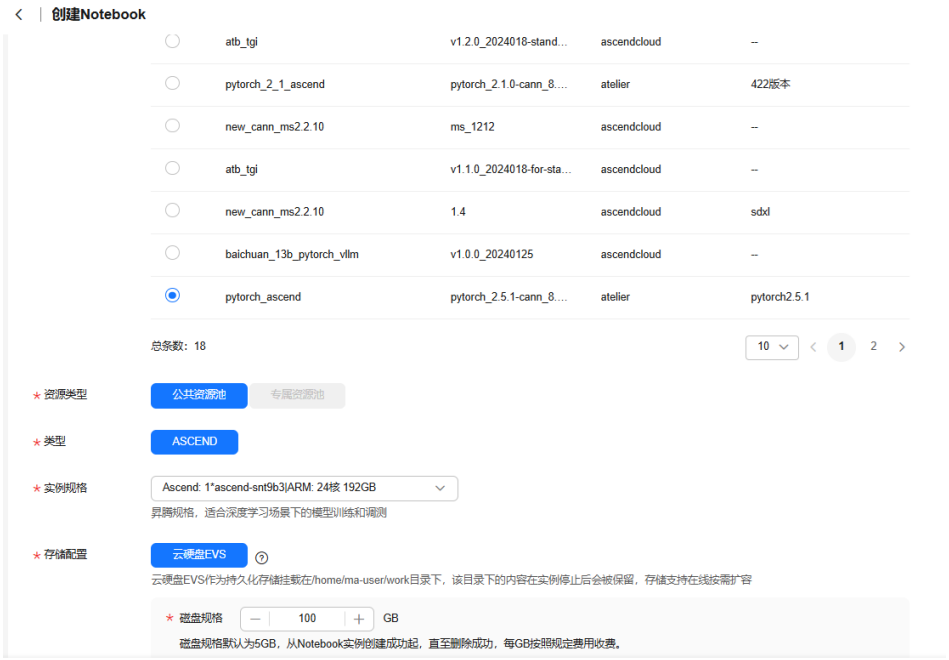
1. 登录**ModelArts控制台**，在左侧导航选择“资产管理 > 镜像管理”，在右上角选择“注册镜像”"，填写**表1 获取软件和镜像**中Notebook使用镜像为镜像源，其余选项参考下图进行注册。

图 3-8 注册镜像



2. 请参考**Notebook资源开通**，创建Notebook实例，在镜像选择时选择自定义镜像，可参考如下配置。资源规格默认5GB，根据需要自行扩容。

图 3-9 创建 Notebook 实例



说明

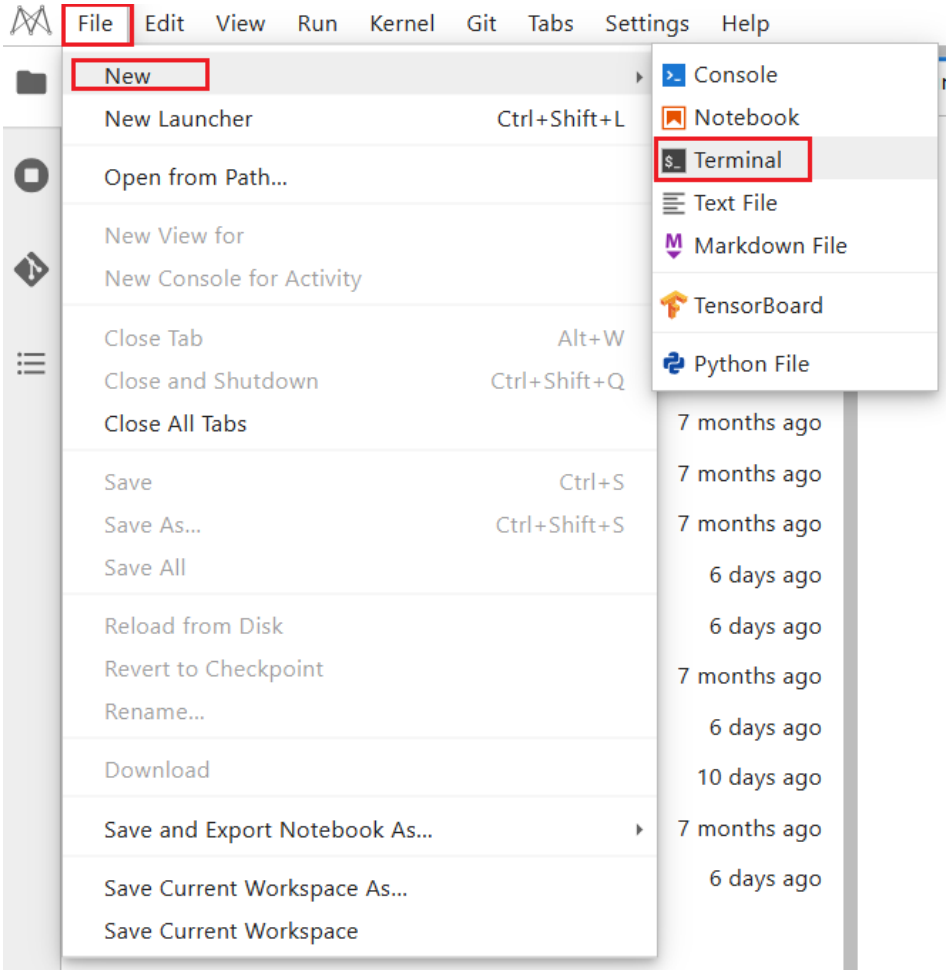
购买Notebook资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

3. 检查环境。

   打开创建好的Notebook实例（即状态为“运行中”的实例），可在开发环境中启动编码。

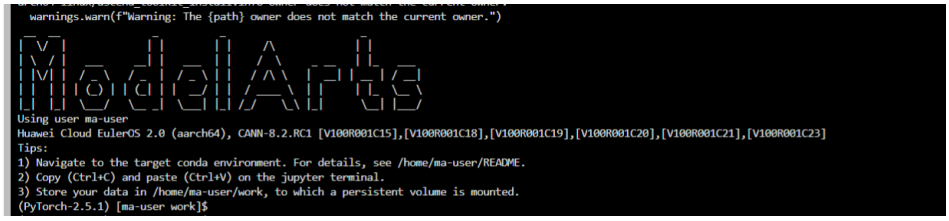
ModelArts提供的Notebook实例默认是以ma-user启动。用户进入实例后，工作目录默认是/home/ma-user/work。

图 3-10 进入 Notebook 实例的 Terminal



工作目录示例如下图所示：

图 3-11 工作目录



步骤二：Diffusers 部署

安装依赖和模型包

1. 手动下载模型权重到/home/ma-user/work目录下：
   官网下载地址（需登录）。

- SDXL下载链接：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**

modelscope下载地址：

- SDXL下载链接：**https://modelscope.cn/models/MusePublic/47_ckpt_SD_XL/summary?version=master**

2. 安装插件代码包。
   a. 将获取到的插件代码包AscendCloud-AIGC-xxx.zip文件上传到容器的/home/ma-user/work目录下，并解压。插件代码包获取地址参见**表3-4**。上传文件操作请参见**上传文件至Notebook Jupyterlab**。
   ```
   cd /home/ma-user/work
   unzip AscendCloud-AIGC-*.zip #解压
   ```
   b. 将AIGC包解压后，进入到获取的/home/ma-user/work/aigc_inference/torch_npu/utils/ascend_diffusers目录下，安装ascend_diffusers包。
   ```
   cd /home/ma-user/work/aigc_inference/torch_npu/utils/ascend_diffusers
   pip install -e .
   ```
   c. 将AIGC包解压后，进入到获取的/home/ma-user/work/aigc_inference/torch_npu/utils/AscendX-MM目录下，安装AscendX-MM包。
   ```
   cd /home/ma-user/work/aigc_inference/torch_npu/utils/AscendX-MM
   pip install -e .
   ```

启动服务

```
export MODEL_PATH='下载好的huggingface模型路径'，例如/home/ma-user/work/stable-diffusion-xl-base-1.0。如果未手动下载，想要自动下载则不加model_id参数即可。
export TASK_QUEUE_ENABLE=2
cd /home/ma-user/work/aigc_inference/torch_npu/diffusers/0.31.0/examples
```

单卡模型推理启动服务命令如下，生成图片在当前目录下。

- SDXL模型推理启动命令：
  ```
  pip install diffusers==0.30.2
  python sd_inference_example.py --model_name sdxl --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 20 --width 768 1024 --height 768 1024
  ```

4 视频生成模型训练推理

4.1 Wan 系列视频生成模型基于 Lite Server 适配 NPU 推理指导

方案概览

本文主要介绍如何在ModelArts的Lite Server环境中，使用NPU卡进行Wan2.1、Wan2.2频生成模型进行文生视频推理、图生视频推理和文生图推理。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Server资源。

本案例支持Wan系列（包含Wan2.1-T2V-14B-Diffusers, Wan2.1-T2V-1.3B-Diffusers, Wan2.1-I2V-14B-480P-Diffusers, Wan2.1-I2V-14B-720P-Diffusers， Wan2.2-T2V-A14B-Diffusers， Wan2.2-I2V-A14B-Diffusers）生成模型。

资源规格要求

建议使用Lite Server环境中的Snt9B单机或Snt9B23单机资源。

表 4-1 Snt9B23 环境要求

名称	版本
driver	25.2.1
PyTorch	pytorch_2.5.1

表 4-2 Snt9B 环境要求

名称	版本
driver	25.2.1
PyTorch	pytorch_2.5.1

获取软件和镜像

表 4-3 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.5.907-xxx.zip软件包中的AscendCloud-AIGC-6.5.907-xxx.zip包。 **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts **6.5.907.2**版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	Snt9B23：乌兰一、华东二、西南-贵阳一 swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129 Snt9B：华东二、西南-贵阳一 swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	从SWR拉取。
基础镜像	Snt9B23：中国-香港 swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129 Snt9B：中国-香港 swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	从SWR拉取。

约束限制

- 本文档适配ModelArts 6.5.907.2版本，请参考**表4-3**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。

- 确保容器可以访问公网。

步骤一：准备环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二：获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表4-3**。

```
docker pull {image_url}
```

如需登录，请**登录SWR控制台**，参考以下图示获取登录指令。

步骤三：启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

   Snt9B23容器启动命令：


```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=host \
   --privileged \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=256g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```

Snt9B容器启动命令：

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=bridge \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=256g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下可存放项目所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_name}：对应机型的基础镜像的名称，具体参见**表4-3**。
- --device=/dev/davinci0 ： 挂载对应卡到容器，当需要挂载多卡，请依次添加多项该配置。

2. 通过容器名称进入容器中。

   Snt9B23使用root用户登录

   ```
   docker exec -it -u root ${container_name} bash
   ```

   Snt9B默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

   ```
   docker exec -it ${container_name} bash
   ```

步骤四：安装依赖和软件包

1. git clone和git lfs下载大模型可以参考如下操作。

   a. 在浏览器中输入如下地址下载git-lfs压缩包并上传到容器的/home/ma-user目录下。

   ```
   https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
   ```

   或直接下载到容器，这样在容器中可以直接使用。

   ```
   cd /home/ma-user
   wget https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
   ```

   b. 进入容器，执行安装git lfs命令。

   ```
   cd /home/ma-user
   tar -zxvf git-lfs-linux-arm64-v3.2.0.tar.gz
   cd git-lfs-3.2.0
   sudo sh install.sh
   ```

   c. 设置git配置去掉ssl校验。

   ```
   git config --global http.sslVerify false
   ```

2. 安装AscendX_Video软件包。

   a. 将获取到的AscendX_Video软件包AscendCloud-AIGC-*.zip文件上传到容器的/home/ma-user目录下。获取路径参见**获取软件和镜像**。

   b. 解压AscendCloud-AIGC-*.zip文件，解压后按照步骤安装Python依赖，执行以下命令即可。

   ```
   cd /home/ma-user
   unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
   cp -r /home/ma-user/AscendCloud/aigc_inference/torch_npu/ascendx_video ./
   cd /home/ma-user/ascendx_video
   pip install seal-*-linux_aarch64.whl
   pip install check_device-*-linux_aarch64.whl
   pip install ascendx_video-*-none-any.whl
   ```

   c. 安装算子环境。

   如果使用的是Snt9B23机器，执行：

   ```
   cd /home/ma-user/AscendCloud/opp/A3
   ```

   如果使用的是Snt9B机器，执行：

   ```
   cd /home/ma-user/AscendCloud/opp/A2
   ```

   安装算子:

   ```
   unzip AscendCloud-OPP-*.zip
   unzip AscendCloud-OPP-*-torch-2.5.1-py311-*.zip -d ./AscendCloud_OPP
   cd AscendCloud_OPP
   pip install *.whl
   mkdir -p /home/ma-user/operate
   bash ./ascend_cloud_ops_ascend_turbo-*_linux_aarch64.run --install-path=/home/ma-user/operate
   bash ./ascend_cloud_ops_custom_opp-*_linux_aarch64_ascend910b_ascend910_93.run --install-path=/home/ma-user/operate
   cd ..
   unzip AscendCloud-OPS-ADV-*.zip -d ./AscendCloud_OPS-ADV
   cd AscendCloud_OPS-ADV
   bash ./CANN-custom_ops-*-linux.aarch64.run --install-path=/home/ma-user/operate
   ```

3. 初始化环境变量

注意每次进入容器需要重新初始化环境。

```
source /home/ma-user/operate/AscendTurbo/set_env.bash
source /home/ma-user/operate/vendors/customize/bin/set_env.bash
source /home/ma-user/operate/vendors/customize_cloud/bin/set_env.bash
```

步骤五：下载模型权重

下载权重文件至容器目录，需要用到的模型地址如下。

- Wan-AI/Wan2.1-T2V-14B-Diffusers，地址：**https://huggingface.co/Wan-AI/Wan2.1-T2V-14B-Diffusers**
- Wan-AI/Wan2.1-T2V-1.3B-Diffusers，地址：**https://huggingface.co/Wan-AI/Wan2.1-T2V-1.3B-Diffusers**
- Wan-AI/Wan2.1-I2V-14B-480P-Diffusers，地址: **https://huggingface.co/Wan-AI/Wan2.1-I2V-14B-480P-Diffusers**
- Wan-AI/Wan2.1-I2V-14B-720P-Diffusers，地址: **https://huggingface.co/Wan-AI/Wan2.1-I2V-14B-720P-Diffusers**
- Wan-AI/Wan2.2-T2V-A14B-Diffusers，地址：**https://huggingface.co/Wan-AI/Wan2.2-T2V-A14B-Diffusers**
- Wan-AI/Wan2.2-I2V-A14B-Diffusers，地址：**https://huggingface.co/Wan-AI/Wan2.2-I2V-A14B-Diffusers**

将权重放到 /home/ma-user/ascendx_video/weights目录下，例如：

```
weights
└──Wan-AI
    ├──Wan2.1-I2V-14B-480P-Diffusers
    ├──Wan2.1-I2V-14B-720P-Diffusers
    ├──Wan2.1-T2V-14B-Diffusers
    ├──Wan2.1-T2V-1.3B-Diffusers
    ├──Wan2.2-I2V-A14B-Diffusers
    └──Wan2.2-T2V-A14B-Diffusers
```

步骤六：Wan2.1 文生视频模型推理

在/home/ma-user/ascendx_video/scripts/目录中有如下脚本：

- infer_wan2.1_14b_t2v_480p.sh：表示Wan文生视频模型Wan2.1-T2V-14B的480P推理脚本。
- infer_wan2.1_14b_t2v_720p.sh：表示Wan文生视频模型Wan2.1-T2V-14B的720P推理脚本。
- infer_wan2.1_1.3b_t2v.sh：表示Wan文生视频模型Wan2.1-T2V-1.3B的推理脚本。

执行以下命令开始推理任务，以infer_wan2.1_14b_t2v_480p.sh为例。

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.1_14b_t2v_480p.sh
```

文生视频推理脚本infer_wan2.1_14b_t2v_480p.sh参数介绍如下。
infer_wan2.1_1.3b_t2v.sh和infer_wan2.1_14b_t2v_720p.sh脚本参数和infer_wan2.1_14b_t2v_480p.sh类似。

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
```

```
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false

export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
N_NPUS=8
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../
infer.py \
    --model Wan2.1-T2V-14B \
    --pretrained_model_name_or_path "../weights/Wan-AI/Wan2.1-T2V-14B-Diffusers" \
    --save_path ./output.mp4 \
    --num_inference_steps 50 \
    --width 832 \
    --height 480 \
    --frames 81 \
    --sp $N_NPUS \
    --fsdp \
    --vae_lightning \
    --turbo_mode faiz \
    --atten_a8w8 \
    --matmul_a8w8 \
    --rope_fused \
    --seed 42 \
    --prompt "A young boy with short brown hair, dressed in a dark blue t-shirt and red pants, is seen
playing a KAWAI upright piano with skill and concentration. The piano's glossy black surface reflects the
room's lighting, and its white and black keys are arranged in a standard layout, indicating a scene of
musical practice or learning. The boy's hands move over the keys, suggesting he is engaged in playing or
practicing a piece." \
    --negative_prompt "色调艳丽，过曝，静态，细节模糊不清，字幕，风格，作品，画作，画面，静止，整
体发灰，最差质量，低质量，JPEG压缩残留，丑陋的，残缺的，多余的手指，画得不好的手部，画得不好的脸
部，畸形的，毁容的，形态畸形的肢体，手指融合，静止不动的画面，杂乱的背景，三条腿，背景人很多，倒着
走"
```

- ASCEND_RT_VISIBLE_DEVICES：使用的卡号。
- N_NPUS： 使用卡数量。建议使用8卡运行。
- model：支持的推理模型，目前支持: **Wan2.1-T2V-14B**、**Wan2.1-I2V-14B**、**Wan2.1-T2V-1.3B**、**Wan2.2-T2V-A14B**、**Wan2.2-I2V-A14B**。
- pretrained_model_name_or_path：对应模型的权重地址。
- save_path：推理生成的视频保存路径。
- num_inference_steps：推理步数。
- frames,height,width：生成视频的尺寸，分别是帧数，高，宽。目前支持81x480x832、121x480x832、81x720x1280、121x720x1280。
- prompt,negative_prompt：生成视频的正向提示词和反向提示词。
- sp：序列并行参数，推荐和推理卡数保持一致。
- fsdp: 数据并行。支持 **None**, "**all**","**text_encoder**","**transformer**"。默认为 **None** 不启用。启动不填写默认为 **"all**"，对 "**text_encoder**" 和 "**transformer**" 启用并行，若填写 "**text_encoder**" 或 "**transformer**" 则仅对该模块启动并行。
- vae_lightning：VAE加速，该参数仅在多卡场景下支持。不设置此参数时，不启用VAE加速。VAE加速启用能够加速VAE性能。
- turbo_mode：加速模式。支持 "**default**","**faiz**" 模式，默认为"**default**"不启用。推荐使用 **faiz** 模式，达到最高性能。不设置此参数时，表示不启用加速模式。turbo_mode加速模式能加速视频推理，但会对精度造成轻微影响。
- atten_a8w8: atten量化加速。推荐打开，达到最高性能。不设置此参数时，表示不启用atten量化加速。atten量化加速能加速视频推理，但会对精度造成轻微影响。
- matmul_a8w8: matmul量化加速。推荐打开，达到最高性能。不设置此参数时，表示不启用matmul量化加速。matmul量化加速能加速视频推理，但会对精度造成轻微影响。

- rope_fused: 旋转位置编码融合算子。推荐打开，达到最高性能。不设置此参数时，表示不启用融合算子。融合算子加速能加速视频推理，但会对精度造成轻微影响。
- seed: 随机数种子。默认为42，会影响生成图片的效果。

推理任务运行结束后，生成的视频文件output.mp4存放在设置的save_path目录下，脚本中默认放在/home/ma-user/ascendx_video/scripts目录，请查看推理结果。

步骤七：Wan2.1 图生视频模型推理

开始图生视频模型推理前，先下载示例图片，并将示例图片放在 /home/ma-user/ascendx_video/scripts 目录下。

图 4-1 示例图片

在/home/ma-user/ascendx_video/scripts/目录中有如下脚本：

- infer_wan2.1_14b_i2v_480p.sh：表示Wan图生视频模型Wan2.1-I2V-14B的480P推理脚本。
- infer_wan2.1_14b_i2v_720p.sh：表示Wan图生视频模型Wan2.1-I2V-14B的720P推理脚本。

执行以下命令开始推理任务，以infer_wan2.1_14b_i2v_480p.sh为例。

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.1_14b_i2v_480p.sh
```

图生视频推理脚本infer_wan2.1_14b_i2v_480p.sh参数介绍如下。和infer_wan2.1_14b_i2v_720p.sh脚本参数和infer_wan2.1_14b_i2v_480p.sh类似。

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
```



```
N_NPUS=8
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../
infer.py \
    --model Wan2.1-I2V-14B \
    --pretrained_model_name_or_path "../weights/Wan-AI/Wan2.1-I2V-14B-480P-Diffusers" \
    --task_type i2v \
    --i2v_image_path ./astronaut.jpg \
    --save_path ./output.mp4 \
    --num_inference_steps 40 \
    --width 832 \
    --height 480 \
    --frames 81 \
    --sp $N_NPUS \
    --fsdp \
    --vae_lightning \
    --turbo_mode faiz \
    --atten_a8w8 \
    --matmul_a8w8 \
    --rope_fused \
    --seed 42 \
    --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth
of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera
shot." \
    --negative_prompt "色调艳丽，过曝，静态，细节模糊不清，字幕，风格，作品，画作，画面，静止，整
体发灰，最差质量，低质量，JPEG压缩残留，丑陋的，残缺的，多余的手指，画得不好的手部，画得不好的脸
部，畸形的，毁容的，形态畸形的肢体，手指融合，静止不动的画面，杂乱的背景，三条腿，背景人很多，倒着
走"
```

- task_type：推理任务分为**t2v, i2v，t2i**。文生视频、图生视频、文生图。默认为图生视频。
- i2v_image_path：图生视频的图路径。
- 其他参数和infer_wan_14b_t2v_480p.sh参数一致，具体请参见**步骤六：Wan2.1 文生视频模型推理**中的参数解释。

推理任务运行结束后，生成的视频文件output.mp4存放在设置的save_path目录下，脚本中默认放在/home/ma-user/ascendx_video/scripts目录，请查看推理结果。

步骤八：Wan2.1 文生图模型推理

在/home/ma-user/ascendx_video/scripts/目录中有如下脚本：

- infer_wan2.1_14b_t2i_480p.sh：表示Wan文生图模型Wan2.1-T2V-14B的480P推理脚本。
- infer_wan2.1_14b_t2i_720p.sh：表示Wan文生图模型Wan2.1-T2V-14B的720P推理脚本。

执行以下命令开始推理任务，以infer_wan2.1_14b_t2i_480p.sh为例。

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.1_14b_t2i_480p.sh
```

文生图推理脚本infer_wan2.1_14b_t2i_480p.sh参数介绍如下。
infer_wan2.1_14b_t2i_720p.sh脚本参数和infer_wan2.1_14b_t2i_480p.sh类似。

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0
N_NPUS=1
```

```
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../
infer.py \
    --model Wan2.1-T2V-14B \
    --pretrained_model_name_or_path "../weights/Wan-AI/Wan2.1-T2V-14B-Diffusers" \
    --task_type t2i \
    --save_path ./output.png \
    --num_inference_steps 40 \
    --width 832 \
    --height 480 \
    --frames 1 \
    --atten_a8w8 \
    --matmul_a8w8 \
    --rope_fused \
    --seed 42 \
    --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth
of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera
shot." \
    --negative_prompt "色调艳丽，过曝，静态，细节模糊不清，字幕，风格，作品，画作，画面，静止，整
体发灰，最差质量，低质量，JPEG压缩残留，丑陋的，残缺的，多余的手指，画得不好的手部，画得不好的脸
部，畸形的，毁容的，形态畸形的肢体，手指融合，静止不动的画面，杂乱的背景，三条腿，背景人很多，倒着
走"
```

参数和infer_wan_14b_t2v.sh参数一致，具体请参见**步骤六：Wan2.1文生视频模型推理**中的参数解释。

推理任务运行结束后，生成的图片output.png存放在设置的save_path目录下，脚本中默认放在/home/ma-user/ascendx_video/scripts目录，请查看推理结果。

步骤九：Wan2.2 文生视频模型推理

在/home/ma-user/ascendx_video/scripts/目录中有如下脚本：

- infer_wan2.2_14b_t2v_480p.sh：表示Wan文生视频模型Wan2.2-T2V-A14B-Diffusers的480P推理脚本。
- infer_wan2.2_14b_t2v_720p.sh：表示Wan文生视频模型Wan2.2-T2V-A14B-Diffusers的720P推理脚本。

执行以下命令开始推理任务，以infer_wan2.2_14b_t2v_480p.sh为例。

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.2_14b_t2v_480p.sh
```

文生视频推理脚本infer_wan2.2_14b_t2v_480p.sh参数介绍如下。
infer_wan2.2_14b_t2v_720p.sh脚本参数和infer_wan2.2_14b_t2v_480p.sh类似。

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
N_NPUS=8
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../
infer.py \
    --model Wan2.2-T2V-A14B \
    --pretrained_model_name_or_path ../weights/Wan-AI/Wan2.2-T2V-A14B-Diffusers \
    --task_type t2v \
    --save_path ./output.mp4 \
    --num_inference_steps 40 \
    --width 832 \
    --height 480 \
    --frames 81 \
    --sp $N_NPUS \
    --fsdp text_encoder \
```

```
    --vae_lightning \
    --inf_vram_blocks_num 1 \
    --vae_lightning \
    --atten_a8w8 \
    --matmul_a8w8 \
    --rope_fused \
    --guidance_scale 3.0 \
    --guidance_scale_2 4.0 \
    --seed 42 \
    --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera shot." \
    --negative_prompt "色调艳丽，过曝，静态，细节模糊不清，字幕，风格，作品，画作，画面，静止，整体发灰，最差质量，低质量，JPEG压缩残留，丑陋的，残缺的，多余的手指，画得不好的手部，画得不好的脸部，畸形的，毁容的，形态畸形的肢体，手指融合，静止不动的画面，杂乱的背景，三条腿，背景人很多，倒着走"
```

- inf_vram_blocks_num: 显存优化，目前只支持1。开启时，要求参数fsdp text_encoder。
- guidance_scale: transformer 无分类器引导,按对应模型填写对应值。
- guidance_scale_2: wan2.2 的 transformer_2 无分类器引导,按对应模型填写对应值。

参数和infer_wan_14b_t2v.sh参数一致，具体请参见**步骤六：Wan2.1文生视频模型推理**中的参数解释。

推理任务运行结束后，生成的视频文件output.mp4存放在设置的save_path目录下，脚本中默认放在/home/ma-user/ascendx_video/scripts目录，请查看推理结果。

步骤十：Wan2.2 图生视频模型推理

开始图生视频模型推理前，先下载示例图片，并将示例图片放在 /home/ma-user/ascendx_video/scripts 目录下。

图 4-2 示例图片

在/home/ma-user/ascendx_video/scripts/目录中有如下脚本：

- infer_wan2.2_14b_i2v_480p.sh：表示Wan图生视频模型Wan2.2-I2V-A14B-Diffusers的480P推理脚本。

- infer_wan2.2_14b_i2v_720p.sh：表示Wan图生视频模型Wan2.2-I2V-A14B-Diffusers的720P推理脚本。

执行以下命令开始推理任务，以infer_wan2.2_14b_i2v_480p.sh为例。

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.2_14b_i2v_480p.sh
```

文生视频推理脚本infer_wan2.2_14b_i2v_480p.sh参数介绍如下。
infer_wan2.2_14b_i2v_720p.sh脚本参数和infer_wan2.2_14b_i2v_480p.sh类似。

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
N_NPUS=8
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../infer.py \
        --model Wan2.2-I2V-A14B \
        --pretrained_model_name_or_path ../weights/Wan-AI/Wan2.2-I2V-A14B-Diffusers \
        --task_type i2v \
        --i2v_image_path ./astronaut.jpg \
        --save_path ./output.mp4 \
        --num_inference_steps 40 \
        --width 832 \
        --height 480 \
        --frames 81 \
        --sp $N_NPUS \
        --fsdp \
        --vae_lightning \
        --atten_a8w8 \
        --matmul_a8w8 \
        --rope_fused \
        --guidance_scale 3.5 \
        --guidance_scale_2 3.5 \
        --seed 42 \
        --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera shot." \
        --negative_prompt "色调艳丽，过曝，静态，细节模糊不清，字幕，风格，作品，画作，画面，静止，整体发灰，最差质量，低质量，JPEG压缩残留，丑陋的，残缺的，多余的手指，画得不好的手部，画得不好的脸部，畸形的，毁容的，形态畸形的肢体，手指融合，静止不动的画面，杂乱的背景，三条腿，背景人很多，倒着走"
```

参数和infer_wan_14b_t2v.sh参数一致，具体请参见**步骤六：Wan2.1文生视频模型推理**中的参数解释。

推理任务运行结束后，生成的视频文件output.mp4存放在设置的save_path目录下，脚本中默认放在/home/ma-user/ascendx_video/scripts目录，请查看推理结果。

步骤十一：Wan2.2 文生图模型推理

在/home/ma-user/ascendx_video/scripts/目录中有如下脚本：

- infer_wan2.2_14b_t2i_480p.sh：表示Wan文生图模型Wan2.2-T2V-A14B-Diffusers的480P推理脚本。
- infer_wan2.2_14b_t2i_720p.sh：表示Wan文生图模型Wan2.2-T2V-A14B-Diffusers的720P推理脚本。

执行以下命令开始推理任务，以infer_wan2.2_14b_t2i_480p.sh为例。


```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.2_14b_t2i_480p.sh
```

文生图推理脚本infer_wan2.2_14b_t2i_480p.sh参数介绍如下。
infer_wan2.2_14b_t2i_720p.sh脚本参数和infer_wan2.2_14b_t2i_480p.sh类似。

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0,1
N_NPUS=2
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../
infer.py \
        --model Wan2.2-T2V-A14B \
        --pretrained_model_name_or_path ../weights/Wan-AI/Wan2.2-T2V-A14B-Diffusers \
        --task_type t2i \
        --save_path ./output.png \
        --num_inference_steps 40 \
        --width 832 \
        --height 480 \
        --frames 1 \
        --atten_a8w8 \
        --matmul_a8w8 \
        --rope_fused \
        --guidance_scale 3.0 \
        --guidance_scale_2 4.0 \
        --seed 42 \
        --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth
of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera
shot." \
        --negative_prompt "色调艳丽，过曝，静态，细节模糊不清，字幕，风格，作品，画作，画面，静止，整
体发灰，最差质量，低质量，JPEG压缩残留，丑陋的，残缺的，多余的手指，画得不好的手部，画得不好的脸
部，畸形的，毁容的，形态畸形的肢体，手指融合，静止不动的画面，杂乱的背景，三条腿，背景人很多，倒着
走"
```

参数和infer_wan_14b_t2v.sh参数一致，具体请参见**步骤六：Wan2.1文生视频模型推理**中的参数解释。

推理任务运行结束后，生成的图片output.png存放在设置的save_path目录下，脚本中默认放在/home/ma-user/ascendx_video/scripts目录，请查看推理结果。

5 Standard 权限管理

5.1 ModelArts 权限管理基本概念

ModelArts作为一个完备的AI开发平台，支持用户对其进行细粒度的权限配置，以达到精细化资源、权限管理之目的。这类特性在大型企业用户的使用场景下很常见，但对个人用户则显得复杂而意义不足，所以建议个人用户在使用ModelArts时，参照**个人用户快速配置ModelArts访问权限**来进行初始权限设置。

说明

您是否需要阅读本文档?

如果下述问题您的任何一个回答为“是”，则需要阅读此文档。

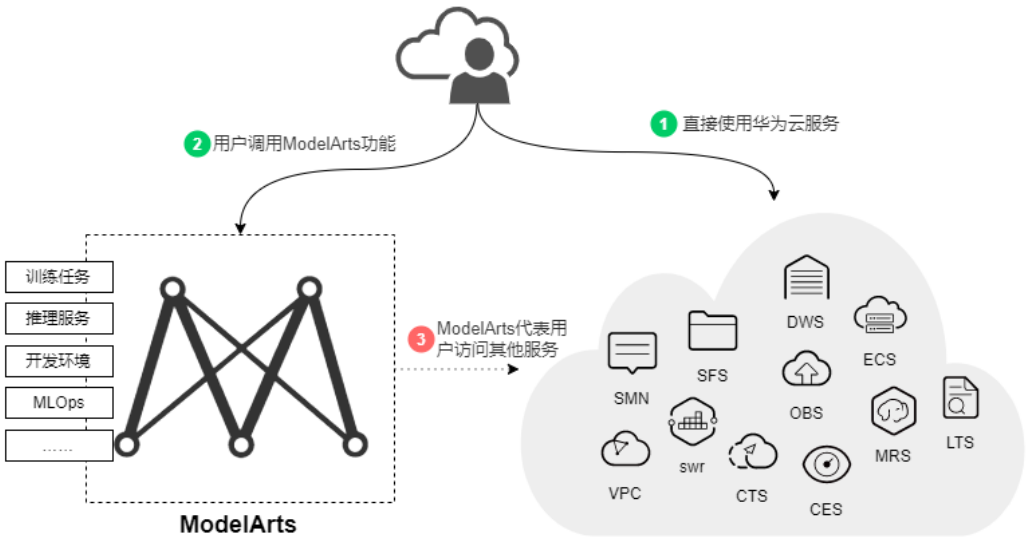
- 您是企业用户，且存在多个部门，且需要限定不同部门的用户只能访问其专属资源、功能存在多种角色（如管理员、算法开发者、应用运维）。
- 您是企业用户，希望限制不同角色只能使用特定功能，逻辑上存在多套“环境”且相互隔离（如开发环境、预生产环境、生产环境），并限定不同用户在不同环境上的操作权限。或其他任何需要对特定子账号（组）做出特定权限限制的情况。
- 您是个人用户，但已经在IAM创建多个子账号，且期望限定不同子账号所能使用的ModelArts功能、资源不同。
- 希望了解ModelArts的权限控制能力细节，期望理解其概念和实操方法。

ModelArts的大部分权限管理能力均基于统一身份认证服务（Identity and Access Management，简称IAM）来实现，在您继续往下阅读之前，强烈建议您先行熟悉**IAM基本概念**，如果能完整理解IAM的所有概念，将更加有助于您理解本文档。

为了支持用户对ModelArts的权限做精细化控制，提供了3个方面的能力来支撑，分别是：权限、委托和工作空间。下面分别讲解。

理解 ModelArts 的权限与委托

图 5-1 权限管理抽象



ModelArts与其他服务类似，功能都通过IAM的权限来进行控制。比如，用户（此处指IAM子账号，而非租户）希望在ModelArts创建训练作业，则该用户必须拥有"modelarts:trainJob:create" 的权限才可以完成操作（无论界面操作还是API调用）。关于如何给一个用户赋权（准确讲是需要先将用户加入用户组，再面向用户组赋权），可以参考IAM的文档《**权限管理**》。

而ModelArts还有一个特殊的地方在于，为了完成AI计算的各种操作，AI平台在任务执行过程中需要访问用户的其他服务，典型的就是训练过程中，需要访问OBS读取用户的训练数据。在这个过程中，就出现了ModelArts“代表”用户去访问其他云服务的情形。从安全角度出发，ModelArts代表用户访问任何云服务之前，均需要先获得用户的授权，而这个动作就是一个“委托”的过程。用户授权ModelArts再代表自己访问特定的云服务，以完成其在ModelArts平台上执行的AI计算任务。

综上，对于**图1 权限管理抽象**可以做如下解读：

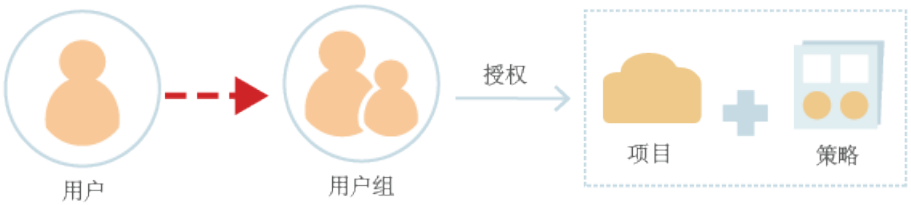
- 用户访问任何云服务，均是通过标准的IAM权限体系进行访问控制。用户首先需要具备相关云服务的权限（根据您具体使用的功能不同，所需的相关服务权限亦有差异）。
- **权限**：用户使用ModelArts的任何功能，亦需要通过IAM权限体系进行正确权限授权。
- **委托**：ModelArts上的AI计算任务执行过程中需要访问其他云服务，此动作需要获得用户的委托授权。

ModelArts 权限管理

默认情况下，管理员创建的IAM用户没有任何权限，需要将其加入用户组，并给用户组授予策略，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于授予的权限对云服务进行操作。

⚠ 注意

- ModelArts部署时通过物理区域划分，为项目级服务，授权时“选择授权范围方案”可以选择“指定区域项目资源”，如果授权时指定了区域对应的项目，则该权限仅对此项目生效；简单的做法是直接选择“所有资源”。
- ModelArts也支持企业项目，所以选择授权范围方案时，也可以指定企业项目。具体操作参见《**创建用户组并授权**》。



IAM在对用户组授权的时候，并不是直接将具体的某个权限进行赋权，而是需要先将权限加入到“策略”当中，再把策略赋给用户组。为了方便用户的权限管理，各个云服务都提供了一些预置的“系统策略”供用户直接使用。如果预置的策略不能满足您的细粒度权限控制要求，则可以通过“自定义策略”来进行精细控制。

表5-1列出了ModelArts的所有预置系统策略。

表 5-1 ModelArts 系统策略

策略名称	描述	类型
ModelArts FullAccess	ModelArts管理员用户，拥有所有ModelArts服务的权限	系统策略
ModelArts CommonOperations	ModelArts操作用户，拥有所有ModelArts服务操作权限除了管理专属资源池的权限	系统策略
ModelArts Dependency Access	ModelArts服务的常用依赖服务的权限	系统策略

通常来讲，只给管理员开通“ModelArts FullAccess”，如果不需要太精细的控制，直接给所有用户开通“ModelArts CommonOperations”即可满足大多数小团队的开发场景诉求。如果您希望通过自定义策略做深入细致的权限控制，请阅读**ModelArts的IAM权限控制详解**。

说明

ModelArts的权限不会凌驾于其他服务的权限之上，当您给用户进行ModelArts赋权时，系统不会自动对其他相关服务的相关权限进行赋权。这样做的好处是更加安全，不会出现预期外的“越权”，但缺点是，您必须同时给用户赋予不同服务的权限，才能确保用户可以顺利完成某些ModelArts操作。

举例，如果用户需要用OBS中的数据进行训练，当已经为IAM用户配置ModelArts训练权限时，仍需同时为其配置对应的OBS权限（读、写、列表），才可以正常使用。其中OBS的列表权限用于支持用户从ModelArts界面上选择要进行训练的数据路径；读权限主要用于数据的预览以及训练任务执行时的数据读取；写权限则是为了保存训练结果和日志。

- 对于个人用户或小型组织，一个简单做法是为IAM用户配置“作用范围”为“全局级服务”的“Tenant Administrator”策略，这会使用户获得除了IAM以外的所有用户权限。在获得便利的同时，由于用户的权限较大，会存在相对较大的安全风险，需谨慎使用。（对于个人用户，其默认IAM账号就已经属于admin用户组，且具备Tenant Administrator权限，无需额外操作）
- 当您需要限制用户操作，仅为ModelArts用户配置OBS相关的最小化权限项，具体操作请参见**OBS权限管理**。对于其他云服务，也可以进行精细化权限控制，具体请参考对应的云服务文档。

ModelArts 委托授权

前文已经介绍，ModelArts在执行AI计算任务过程中，需要“代表”用户去访问其他云服务，而此动作需要提前获得用户的授权。在IAM权限体系下，此类授权动作是通过“委托”来完成。

关于委托的基本概念及操作可以参考对应的IAM文档《**委托其他云服务管理资源**》。

为了简化用户的委托授权操作，ModelArts增加了自动配置委托授权的支持，用户仅需在**ModelArts控制台**的“权限管理”页面中，为自己或特定用户配置委托即可。

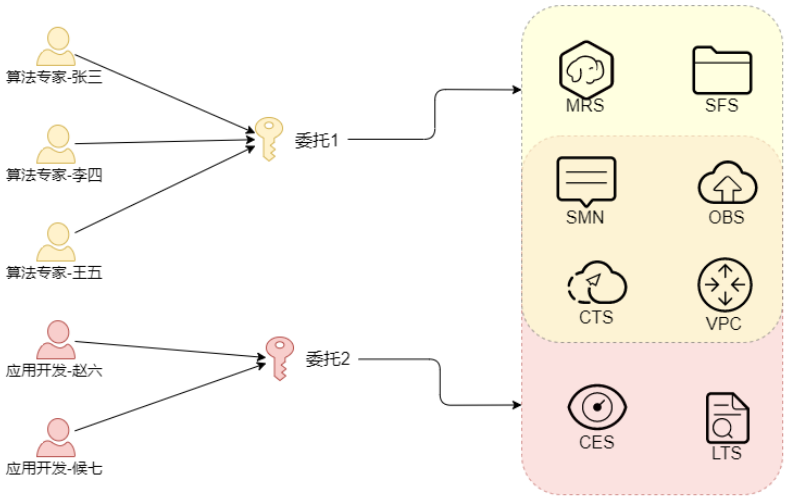
说明

- 只有具备IAM委托管理权限的用户才可以进行此项操作，通常是IAM admin用户组的成员才具备此权限。
- 目前ModelArts的委托授权操作是分区域操作的，这意味着您需要在每个您所用到的区域均执行委托授权操作。

在**ModelArts控制台**的“权限管理”页面，单击“添加授权”后，系统会引导您为特定用户或所有用户进行委托配置，通常默认会创建一个名为“modelarts_agency_<用户名>_随机ID”的委托条目。在权限配置的区域，您可以选择ModelArts提供的预置配置，也可以自定义选择您所授权的策略。如果这两种形态对于您的诉求均过于粗犷，您也可以直接在IAM管理页面里创建完全由您进行精细化配置的委托（需要委托给ModelArts服务），然后在此页面的委托选择里使用“已有委托”“”（而非“新增委托”）。

至此，您应该已经发现了一个细节，ModelArts在使用委托时，是将其与用户进行关联的，用户与委托的关系是多对1的关系。这意味着，如果两个用户需要配置的委托一致，那么不需要为每个用户都创建一个独立的委托项，只需要将两个用户都“指向”同一个委托项即可。

图 5-2 用户与委托对应关系



说明

每个用户必须关联委托才可以使用ModelArts，但即使委托所赋之权限不足，在API调用之初也不会报错，只有到系统具体使用到该功能时，才会发生问题。例如，用户在创建训练任务时打开了“消息通知”，该功能依赖SMN委托授权，但只有训练任务运行过程中，真正需要发送消息时，系统才会“出错”，而有些错误系统会选择“忽略”，另一些错误则可能导致任务直接失败。当您做深入的“权限最小化”限制时，请确保您在ModelArts上将要执行的操作仍旧有足够的权限。

严格授权模式

严格授权模式是指在IAM中创建的子账号必须由账号管理员显式在IAM中授权，才能访问ModelArts服务，管理员用户可以通过授权策略为普通用户精确添加所需使用的ModelArts功能的权限。

相对的，在非严格授权模式下，子账号不需要显式授权就可以使用ModelArts，管理员需要在IAM上为子账号配置Deny策略来禁止子账号使用ModelArts的某些功能。

账号的管理员用户可以在“权限管理”页面修改授权模式。

须知

如无特殊情况，建议优先使用严格授权模式。在严格授权模式下，子账号要使用ModelArts的功能都需经过授权，可以更精确的控制子账号的权限范围，达成权限最小化的安全策略。

用工作空间限制资源访问

工作空间是ModelArts面向企业用户提供的一个高阶功能，用于进一步将用户的资源划分在多个**逻辑隔离**的空间中，并支持以空间维度进行访问的权限限定。

在开通工作空间后，系统会默认为您创建一个“default”空间，您之前所创建的所有资源，均在该空间下。当您创建新的工作空间之后，相当于您拥有了一个新的“ModelArts分身”，您可以通过菜单栏的左上角进行工作空间的切换，不同工作空间中的工作互不影响。

创建工作空间时，必须绑定一个企业项目。多个工作空间可以绑定到同一个企业项目，但一个工作空间**不可以**绑定多个企业项目。借助工作空间，您可以对不同用户的资源访问和权限做更加细致的约束，具体为如下两种约束：

- 只有被授权的用户才能访问特定的工作空间（在创建、管理工作空间的页面进行配置），这意味着，像数据集、算法等AI资产，均可以借助工作空间做访问的限制。
- 在前文提到的权限授权操作中，如果“选择授权范围方案”时设定为“指定企业项目资源”，那么该授权仅对绑定至该企业项目的工作空间生效。

说明

- 工作空间的约束与权限授权的约束是叠加生效的，意味着对于一个用户，必须同时拥有工作空间的访问权和训练任务的创建权限（且该权限覆盖至当前的工作空间），他才可以在这个空间里提交训练任务。
- 对于已经开通企业项目但没有开通工作空间的用户，其所有操作均相当于在“default”企业项目里进行，请确保对应权限已覆盖了名为default的企业项目。
- 对于未开通企业项目的用户，不受上述约束限制。

本章小结

对于ModelArts的权限管理，总结了如下几条关键点：

- 如果您是个人用户，则不需要考虑细粒度权限问题，您的账户默认具备使用ModelArts的所有权限。
- ModelArts平台的所有功能均通过IAM体系进行了权限管控，您可以通过标准的IAM**授权**动作，来对特定用户进行精细化的权限管控。
- 对于所有用户（包括个人用户），需要完成对ModelArts的**委托**授权（ModelArts > 权限管理 > 添加授权），才能使用特定的功能，否则会造成您的操作出现不可预期的错误。
- 对于开通了企业项目的用户，可以进一步申请开通ModelArts的**工作空间**，通过组合使用基础授权和工作空间，来达成更加复杂的权限控制目的。

5.2 权限控制方式

5.2.1 IAM

介绍ModelArts所有功能涉及到的IAM权限配置。

IAM 权限简介

如果您需要为企业中的员工设置不同的权限访问ModelArts资源，以达到不同员工之间的权限隔离，您可以使用统一身份认证服务（Identity and Access Management，简称IAM）进行精细的权限管理。该服务提供用户身份认证、权限分配、访问控制等功能，可以帮助您安全的控制云服务资源的访问。如果华为账号已经能满足您的要求，不需要通过IAM对用户进行权限管理，您可以跳过本章节，不影响您使用ModelArts服务的其他功能。

IAM是提供权限管理的基础服务，无需付费即可使用，您只需要为您账号中的资源进行付费。

通过IAM，您可以通过授权控制用户对服务资源的访问范围。例如您的员工中有负责软件开发的人员，您希望这些用户拥有ModelArts的使用权限，但是不希望这些用户拥

有删除ModelArts等高危操作的权限，那么您可以使用IAM进行权限分配，通过授予用户仅能使用ModelArts，但是不允许删除ModelArts的权限，控制用户对ModelArts资源的使用范围。

关于IAM的详细介绍，请参见**IAM产品介绍**。

角色与策略权限管理

ModelArts服务支持角色与策略授权。默认情况下，管理员创建的IAM用户没有任何权限，需要将其加入用户组，并给用户组授予策略或角色，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于被授予的权限对云服务进行操作。

ModelArts部署时通过物理区域划分，为项目级服务。授权时，“授权范围”需要选择“指定区域项目资源”，然后在指定区域（如亚太-曼谷）对应的项目（ap-southeast-2）中设置相关权限，并且该权限仅对此项目生效；如果“授权范围”选择“所有资源”，则该权限在所有区域项目中都生效。访问ModelArts时，需要先切换至授权区域。

如**表5-2**所示，包括了ModelArts的所有系统策略权限。如果系统预置的ModelArts权限，不满足您的授权要求，可以创建自定义策略，可参考**策略JSON格式字段介绍**。

表 5-2 ModelArts 系统策略

策略名称	描述	类型
ModelArts FullAccess	ModelArts管理员用户，拥有所有ModelArts服务的权限。	系统策略
ModelArts CommonOperations	ModelArts操作用户，拥有所有ModelArts服务操作权限除了管理专属资源池的权限。	系统策略
ModelArts Dependency Access	ModelArts服务的常用依赖服务的权限。	系统策略

ModelArts对其他云服务有依赖关系，因此在ModelArts控制台的各项功能需要配置相应的服务权限后才能正常查看或使用，依赖服务及其预置的权限如下。

表 5-3 ModelArts 控制台依赖服务的角色或策略

控制台功能	依赖服务	需配置角色/策略
数据管理（数据集/ 数据标注/数据处理）	对象存储服务OBS	OBS Administrator
	数据湖探索DLI	DLI FullAccess
	MapReduce服务MRS	MRS Administrator
	数据仓库服务GaussDB(DWS)	DWS Administrator
	云审计服务CTS	CTS Administrator

控制台功能	依赖服务	需配置角色/策略
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
开发环境Notebook/镜像管理/弹性节点Server	对象存储服务OBS	OBS Administrator
	凭据管理服务CSMS	CSMS ReadOnlyAccess
	云审计服务CTS	CTS Administrator
	弹性云服务器ECS	ECS FullAccess
	容器镜像服务SWR	SWR Admin
	弹性文件服务SFS	SFS Turbo FullAccess
	应用运维管理服务AOM	AOM FullAccess
	密钥管理服务KMS	KMS CMKFullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
算法管理/训练管理/Workflow	对象存储服务OBS	OBS Administrator
	消息通知服务SMN	SMN Administrator
	云审计服务CTS	CTS Administrator
	企业项目管理服务EPS	EPS FullAccess
	弹性文件服务SFS	SFS ReadOnlyAccess
	容器镜像服务SWR	SWR Admin
	应用运维管理服务AOM	AOM FullAccess
	密钥管理服务KMS	KMS CMKFullAccess
	虚拟私有云服务VPC	VPC FullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
模型管理/在线服务/批量服务/边缘服务/边缘部署专属资源池	对象存储服务OBS	OBS Administrator
	云监控服务CES	CES ReadOnlyAccess
	消息通知服务SMN	SMN Administrator
	企业项目管理服务EPS	EPS FullAccess
	云审计服务CTS	CTS Administrator
	云日志服务LTS	LTS FullAccess

控制台功能	依赖服务	需配置角色/策略
	虚拟私有云VPC	VPC FullAccess
	容器镜像服务SWR	SWR Admin
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
AI Gallery	对象存储服务OBS	OBS Administrator
	云审计服务CTS	CTS Administrator
	容器镜像服务SWR	SWR Admin
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
标准算力集群（Standard Cluster）和轻量算力集群（Lite Cluster）	云审计服务CTS	CTS Administrator
	云容器引擎CCE	CCE Administrator
	裸金属服务器BMS	BMS FullAccess
	镜像服务IMS	IMS FullAccess
	密码安全中心DEW	DEW KeypairReadOnlyAccess
	虚拟私有云VPC	VPC FullAccess
	弹性云服务器ECS	ECS FullAccess
	弹性文件服务SFS	SFS Turbo FullAccess
	对象存储服务OBS	OBS Administrator
	应用运维管理服务AOM	AOM FullAccess
	标签管理服务TMS	TMS FullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
	费用中心	BSS Administrator
	云硬盘EVS	EVS FullAccess

如果系统预置的权限，不满足您的授权要求，可以创建自定义策略。自定义策略中可以添加的授权项（Action）请参考**ModelArts资源权限项**。

目前支持以下两种方式创建自定义策略：

- 可视化视图创建自定义策略：无需了解策略语法，按可视化视图导航栏选择云服务、操作、资源、条件等策略内容，可自动生成策略。
- JSON视图创建自定义策略：可以在选择策略模板后，根据具体需求编辑策略内容；也可以直接在编辑框内编写JSON格式的策略内容。

具体创建步骤请参见：**创建自定义策略**。下面为您介绍常用的ModelArts自定义策略样例。

- 示例1：授权镜像管理的权限。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:image:register",
                "modelarts:image:listGroup"
            ]
        }
    ]
}
```

- 示例2：拒绝用户创建、更新、删除专属资源池。

  拒绝策略需要同时配合其他策略使用，否则没有实际作用。用户被授予的策略中，一个授权项的作用如果同时存在Allow和Deny，则遵循**Deny优先原则**。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "modelarts:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "swr:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "smn:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "modelarts:pool:create",
                "modelarts:pool:update",
                "modelarts:pool:delete"
            ],
            "Effect": "Deny"
        }
    ]
}
```

- 示例3：多个授权项策略。

  一个自定义策略中可以包含多个授权项，且除了可以包含本服务的授权项外，还可以包含其他服务的授权项，可以包含的其他服务必须跟本服务同属性，即都是项目级服务或都是全局级服务。多个授权语句策略描述如下：

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:service:*"
            ]
        },
        {
```



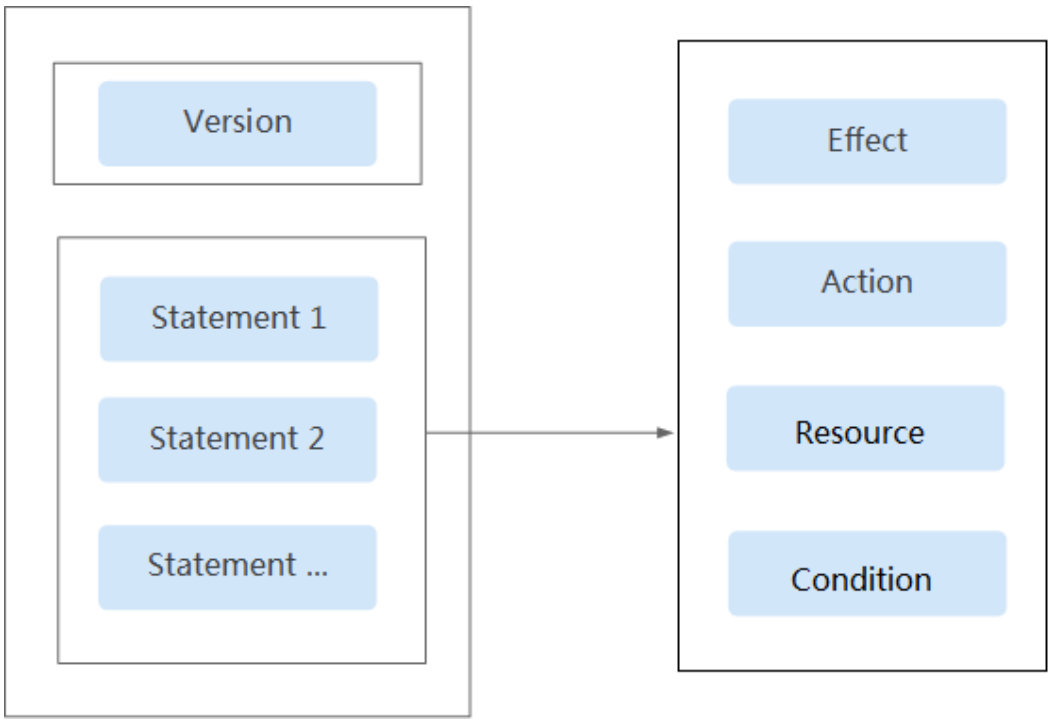
```
            "Effect": "Allow",
            "Action": [
                "lts:logs:list"
            ]
        }
    ]
}
```

策略 JSON 格式字段介绍

策略结构

策略结构包括Version（策略版本号）和Statement（策略权限语句）两部分，其中Statement可以有多个，表示不同的授权项。

图 5-3 策略结构



策略参数

下面介绍策略参数详细说明。了解策略参数后，您可以根据场景自定义策略。具体可以参考文档**自定义策略使用样例**。

表 5-4 策略参数说明

参数		含义	值
Version		策略的版本。	1.1：代表基于策略的访问控制。
Statement：策略的授权语句	Effect：作用	定义Action中的操作权限是否允许执行。	• Allow：允许执行。 • Deny：不允许执行。 **说明** 当同一个Action的Effect既有Allow又有Deny时，遵循Deny优先的原则。

参数		含义	值
	Action：授权项	操作权限。	格式为“服务名:资源类型:操作”。授权项支持通配符号*，通配符号*表示所有。 示例： "modelarts:notebook:list"：表示查看Notebook实例列表权限，其中modelarts为服务名，notebook为资源类型，list为操作。 您可以在对应服务“API参考”资料中查看该服务所有授权项。
	Condition：条件	使策略生效的特定条件，包括**条件键**和**运算符**。	格式为“条件运算符:{条件键：[条件值1,条件值2]}”。 如果您设置多个条件，同时满足所有条件时，该策略才生效。 示例： "StringEndWithIfExists":{"g:UserName":["specialCharacter"]}：表示当用户输入的用户名以"specialCharacter"结尾时该条statement生效。
	Resource: 资源类型	策略所作用的资源。	格式为“服务名:<region>:<account-id>:资源类型:资源路径”，资源类型支持通配符号*，通配符号*表示所有。 **说明** ModelArts的授权不支持指定具体资源路径。

ModelArts 资源类型

管理员可以按ModelArts的资源类型选择授权范围。ModelArts支持的资源类型如下表：

表 5-5 ModelArts 资源类型（角色与策略授权）

资源类型	说明
notebook	开发环境的Notebook实例
workflow	Workflow项目
pool	专属资源池
network	专属资源池网络连接
trainJob	训练作业
trainJobLog	训练作业的运行日志
trainJobInnerModel	系统预置模型

资源类型	说明
model	模型
service	在线服务
nodeservice	边缘服务
workspace	工作空间
dataset	数据集
dataAnnotation	数据集的标注信息
aiAlgorithm	训练算法
image	镜像
devserver	弹性裸金属

ModelArts 资源权限项

参考《 ModelArts API参考 》中的权限策略和授权项。

- **数据管理权限**
- **开发环境权限**
- **训练作业权限**
- **模型管理权限**
- **服务管理权限**

5.2.2 依赖和委托

功能依赖

功能依赖策略项

您在使用ModelArts的过程中，需要和其他云服务交互，比如需要在提交训练作业时选择指定数据集OBS路径和日志存储OBS路径。因此管理员在为用户配置细粒度授权策略时，需要同时配置依赖的权限项，用户才能使用完整的功能。

说明

- 如果您使用根用户（与账户同名的缺省子用户）使用ModelArts，根用户默认拥有所有权限，不再需要单独授权。
- 请用户确保当前用户具备委托授权中包含的依赖策略项权限。例如，用户给ModelArts的委托需要授权SWR Admin权限，需要保证用户本身具备SWR Admin权限。

表 5-6 基本配置

业务场景	依赖的服务	依赖策略项	支持的功能
全局配置	IAM	iam:users:listUsers	查询用户列表（仅管理员需要）
基本功能	IAM	iam:tokens:assume	使用委托获取用户临时认证凭据（必需）
基本功能	BSS	bss:balance:view	在ModelArts控制台创建资源后，页面展示账号当前余额

表 5-7 管理工作空间

业务场景	依赖的服务	依赖策略项	支持的功能
工作空间	IAM	iam:users:listUsers	按用户进行工作空间授权
	ModelArts	modelarts:*:delete*	删除工作空间时，同时清理空间内的资源

表 5-8 管理开发环境 Notebook

业务场景	依赖的服务	依赖策略项	支持的功能
开发环境实例生命周期管理	ModelArts	modelarts:notebook:create modelarts:notebook:list modelarts:notebook:get modelarts:notebook:update modelarts:notebook:delete modelarts:notebook:start modelarts:notebook:stop modelarts:notebook:updateStopPolicy modelarts:image:delete modelarts:image:list modelarts:image:create modelarts:image:get modelarts:pool:list modelarts:tag:list modelarts:network:get	实例的启动、停止、创建、删除、更新等依赖的权限。
	AOM	aom:metric:get aom:metric:list aom:alarm:list	
	VPC	vpc:securityGroups:get vpc:vpcs:list vpc:securityGroups:get vpc:vpcs:list	

业务场景	依赖的服务	依赖策略项	支持的功能
动态挂载存储配置	ModelArts	modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage modelarts:notebook:getMountedStorage modelarts:notebook:umountStorage	动态挂载存储配置。
	OBS	obs:bucket:ListAllMyBuckets obs:bucket:ListBucket	
镜像管理	ModelArts	modelarts:image:register modelarts:image:listGroup	在镜像管理中注册和查看镜像。
保存镜像	SWR	SWR Admin	SWR Admin为SWR最大权限，用于： • 开发环境运行的实例，保存成镜像。 • 使用自定义镜像创建开发环境Notebook实例。
使用SSH功能	ECS	ecs:serverKeypairs:list ecs:serverKeypairs:get ecs:serverKeypairs:delete ecs:serverKeypairs:create	为开发环境Notebook实例配置登录密钥。
	DEW	kps:domainKeypairs:get kps:domainKeypairs:list kps:domainKeypairs:createkmskey	
	KMS	kms:cmk:list	
挂载SFS Turbo盘	SFS Turbo	SFS Turbo FullAccess	子用户对SFS目录的读写操作权限。专属池Notebook实例挂载SFS（公共池不支持），且挂载的SFS不是当前子用户创建的。

业务场景	依赖的服务	依赖策略项	支持的功能
查看所有实例	ModelArts	modelarts:notebook:listAllNotebooks	ModelArts开发环境界面上，查询所有用户的实例列表，适用于给开发环境的实例管理员配置该权限。
	IAM	iam:users:listUsers	
VSCode插件（本地）/ PyCharm Toolkit（本地）	ModelArts	modelarts:notebook:listAllNotebooks modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJob:update modelarts:trainJobVersion:delete modelarts:trainJob:get modelarts:trainJob:logExport modelarts:workspace:getQuotas（如果开通了**工作空间**功能，则需要配置此权限。）	从本地VSCode连接云上的Notebook实例、提交训练作业等。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	
	IAM	iam:projects:listProjects	从本地PyCharm查询IAM项目列表，完成连接配置。

表 5-9 弹性节点 Server

业务场景	依赖的服务	依赖策略项	支持的功能
弹性节点Server实例生命周期管理	ModelArts	modelarts:devserver:create modelarts:devserver:listByUser modelarts:devserver:list modelarts:devserver:get modelarts:devserver:delete modelarts:devserver:start modelarts:devserver:stop modelarts:devserver:sync	创建实例、查询实例列表、查询租户所有实例列表、查询实例详情、删除实例、启动实例、停止实例、同步实例状态。
	ECS	ecs:serverKeypairs:createecs:*:get	
	IAM	iam:users:getUser iam:users:listUsers iam:projects:listProjects	
	VPC	vpc.*.list	
	EPS	eps.*.list	
	EVS	evs.*.list	
	IMS	ims.*.list ims.*.get	

表 5-10 管理训练作业

业务场景	依赖的服务	依赖策略项	支持的功能
训练管理	ModelArts	modelarts:trainJob:* modelarts:trainJobLog:* modelarts:aiAlgorithm:* modelarts:image:list modelarts:network:get modelarts:workspace:get	创建训练作业和查看训练日志。

业务场景	依赖的服务	依赖策略项	支持的功能
		modelarts:workspace:getQuota	查询工作空间配额。如果开通了**工作空间**功能，则需要配置此权限。
		modelarts:tag:list	在训练作业中使用标签管理服务TMS。
	IAM	iam:credentials:listCredentials iam:agencies:listAgencies	使用配置的委托授权项。
	SFS Turbo	sfsturbo:shares:getShare sfsturbo:shares:getAllShares	在训练作业中使用SFS Turbo。
	SWR	swr:repo:getRepo swr:system:createLoginSecret swr:instance:createTempCredential	使用自定义镜像运行训练作业。
	SMN	smn:topic:publish smn:topic:list	通过SMN通知训练作业状态变化事件。
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	使用OBS桶中的数据集运行训练作业。

表 5-11 使用 Workflow

业务场景	依赖的服务	依赖策略项	支持的功能
使用数据集	ModelArts	modelarts:dataset:getDataset modelarts:dataset:createDataset modelarts:dataset:createDatasetVersion modelarts:dataset:createImportTask modelarts:dataset:updateDataset modelarts:processTask:createProcessTask modelarts:processTask:getProcessTask modelarts:dataset:listDatasets	在工作流中使用ModelArts数据集
管理模型	ModelArts	modelarts:model:list modelarts:model:get modelarts:model:create modelarts:model:delete modelarts:model:update	在工作流中管理ModelArts模型
部署上线	ModelArts	modelarts:service:get modelarts:service:create modelarts:service:update modelarts:service:delete modelarts:service:getLogs	在工作流中管理ModelArts在线服务
训练作业	ModelArts	modelarts:trainJob:get modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJobVersion:list modelarts:trainJobVersion:create modelarts:trainJob:delete modelarts:trainJobVersion:delete modelarts:trainJobVersion:stop	在工作流中管理ModelArts训练作业
工作空间	ModelArts	modelarts:workspace:get modelarts:workspace:getQuotas	在工作流中使用ModelArts工作空间

业务场景	依赖的服务	依赖策略项	支持的功能
管理数据	OBS	obs:bucket:ListAllMybuckets（获取桶列表） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:ListBucket（列举桶内对象） obs:bucket:GetBucketLocation（获取桶区域位置） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:GetObjectVersion（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:object:DeleteObjectVersion（删除对象、批量删除对象） obs:object:ListMultipartUploadParts（列举已上传的段） obs:object:AbortMultipartUpload（取消多段上传任务） obs:object:GetObjectAcl（获取对象ACL） obs:object:GetObjectVersionAcl（获取对象ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:object:PutObjectAcl（设置对象ACL）	在工作流中使用OBS数据
工作流运行	IAM	iam:users:listUsers（查询用户列表） iam:agencies:getAgency（查询指定委托详情） iam:tokens:assume（获取委托Token）	在工作流运行时，调用ModelArts其他服务
集成DLI	DLI	dli:jobs:get（查询作业详情） dli:jobs:listAll（查询作业列表） dli:jobs:create（创建新作业）	在工作流中集成DLI

业务场景	依赖的服务	依赖策略项	支持的功能
集成MRS	MRS	mrs:job:get（查询作业详情） mrs:job:submit（创建并执行作业） mrs:job:list（查询作业列表） mrs:job:stop（停止作业） mrs:job:batchDelete（批量删除作业） mrs:file:list（查询文件列表）	在工作流中集成MRS

表 5-12 管理模型

业务场景	依赖的服务	依赖策略项	支持的功能
管理模型	SWR	SWR Admin	从自定义镜像导入、从OBS导入时使用自定义引擎。 SWR共享版不支持细粒度权限项，因此需要配置Admin权限。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets（获取桶列表） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:ListBucket（列举桶内对象） obs:bucket:GetBucketLocation（获取桶区域位置） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:GetObjectVersion（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:object:DeleteObjectVersion（删除对象、批量删除对象） obs:object:ListMultipartUploadParts（列举已上传的段） obs:object:AbortMultipartUpload（取消多段上传任务） obs:object:GetObjectAcl（获取对象ACL） obs:object:GetObjectVersionAcl（获取对象ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:object:PutObjectAcl（设置对象ACL）	从OBS导入模型 模型转换指定OBS路径

表 5-13 管理部署上线

业务场景	依赖的服务	依赖策略项	支持的功能
在线服务	LTS	lts:logs:list（查询日志列表）	查询和展示LTS日志。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:GetBucketPolicy （获取桶策略） obs:bucket:HeadBucket （获取桶元数据） obs:bucket:ListAllMyBuckets（获取桶列表） obs:bucket:PutBucketPolicy （设置桶策略） obs:bucket:DeleteBucketPolicy（删除桶策略）	服务运行时容器挂载外部存储卷。
批量服务	OBS	obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:bucket:CreateBucket（创建桶） obs:bucket:ListBucket（列举桶内对象） obs:bucket:ListAllMyBuckets（获取桶列表）	创建批量服务，批量推理。
边缘服务	CES	ces:metricData:list（查询指标数据）	查看服务的监控指标
	IEF	ief:deployment:delete（删除应用部署）	管理边缘服务
AOM指标告警事件	AOM	aom:alarm:list	查看AOM监控相关信息。

表 5-14 管理数据集

业务场景	依赖的服务	依赖策略项	支持的功能
管理数据集和标注	OBS	obs:bucket:GetBucketLocation obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:GetObjectVersion obs:object:GetObject obs:object:GetObjectVersionAcl obs:object:DeleteObject obs:object:ListMultipartUploadParts obs:bucket:HeadBucket obs:object:AbortMultipartUpload obs:object:DeleteObjectVersion obs:object:GetObjectAcl obs:bucket:ListAllMyBuckets obs:bucket:ListBucket obs:object:PutObject	管理OBS中的数据集 标注OBS数据 创建数据管理作业
管理表格数据集	DLI	dli:database:displayAllDatabases dli:database:displayAllTables dli:table:describeTable	在数据集中管理DLI数据
管理表格数据集	DWS	dws:openAPICluster:list dws:openAPICluster:getDetail dws:cluster:list	在数据集中管理DWS数据
管理表格数据集	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:cluster:get	在数据集中管理MRS数据
智能标注	ModelArts	modelarts:service:list modelarts:model:list modelarts:model:get modelarts:model:create modelarts:trainJobInnerModel:list modelarts:workspace:get modelarts:workspace:list	使用智能标注

业务场景	依赖的服务	依赖策略项	支持的功能
团队标注	IAM	iam:projects:listProjects（查询租户项目） iam:users:listUsers（查询用户列表） iam:agencies:createAgency（创建委托） iam:quotas:listQuotasForProject（查询指定项目的配额）	管理标注团队

表 5-15 资源管理

业务场景	依赖的服务	依赖策略项	支持的功能
资源池管理	BSS	bss:coupon:view bss:order:view bss:balance:view bss:discount:view bss:renewal:view bss:bill:view bss:contract:update bss:order:pay bss:unsubscribe:update bss:renewal:update bss:order:update	资源池的创建、续费、退订等与计费相关的功能。
	CCE	cce:cluster:list cce:cluster:get	获取CCE集群列表、集群详情、集群证书等信息。
	KMS	kms:cmk:list kms:cmk:getMaterial	获取用户创建的密钥对列表信息。
	AOM	aom:metric:get	获取资源池的监控数据。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion	获取AI诊断日志。
	ECS	ecs:availabilityZones:list ecs:cloudServerFlavors:get ecs:cloudServerQuotas:get ecs:quotas:get ecs:serverKeypairs:list	查询可用区列表、规格、配额，配置密钥匙。
	EVS	evs:types:get evs:quotas:get	查询云硬盘类型列表、配额。
	BMS	bms:serverFlavors:get	查询裸金属规格。依赖权限需要配置在IAM项目视图中。
	DEW	kps:domainKeypairs:list	配置密钥对。依赖权限需要配置在IAM项目视图中。

业务场景	依赖的服务	依赖策略项	支持的功能
网络管理	VPC	vpc:routes:create vpc:routes:list vpc:routes:get vpc:routes:delete vpc:peerings:create vpc:peerings:accept vpc:peerings:get vpc:peerings:delete vpc:routeTables:update vpc:routeTables:get vpc:routeTables:list vpc:vpcs:create vpc:vpcs:list vpc:vpcs:get vpc:vpcs:delete vpc:subnets:create vpc:subnets:get vpc:subnets:delete vpcep:endpoints:list vpcep:endpoints:create vpcep:endpoints:delete vpcep:endpoints:get vpc:ports:create vpc:ports:get vpc:ports:update vpc:ports:delete vpc:networks:create vpc:networks:get vpc:networks:update vpc:networks:delete vpc:securityGroups:get	ModelArts网络资源创建和删除、VPC网络打通。
	SFS Turbo	sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics	用户的网络和SFS Turbo资源打通。

业务场景	依赖的服务	依赖策略项	支持的功能
边缘资源池	IEF	ief:node:list ief:group:get ief:application:list ief:application:get ief:node:listNodeCert ief:node:get ief:IEFInstance:get ief:deployment:list ief:group:listGroupInstanceState ief:IEFInstance:list ief:deployment:get ief:group:list	边缘池增删改查管理。
Lite Cluster	ECS	ecs:cloudServers:get ecs:cloudServers:showServer ecs:cloudServers:changeVpc ecs:cloudServers:start ecs:cloudServers:listServerInterfaces ecs:cloudServers:delete ecs:cloudServers:redeploy ecs:cloudServers:batchSetServerTags ecs:cloudServers:reboot ecs:cloudServerFlavors:get ecs:cloudServers:list ecs:quotas:get ecs:cloudServers:create ecs:cloudServers:stop	创建云服务器、查询云服务器详情、删除云服务器等。
	APM	apm:icmgr:create	安装ICAgent。
	EVS	evs:volumes:list evs:types:get evs:volumes:get evs:quotas:get	查询云硬盘类型、列表、配额、详情。
	DEW	kps:domainKeypairs:get''	配置密钥对。依赖权限需要配置在IAM项目视图中。

业务场景	依赖的服务	依赖策略项	支持的功能
	BMS	bms:servers:create bms:serverFlavors:get	查询裸金属规格。依赖权限需要配置在IAM项目视图中。
	IMS	ims:images:get ims:images:share	查询镜像详情和镜像共享。
	CCE	cce:node:delete cce:nodepool:delete cce:addonInstance:update cce:cluster:revokeClientCredential cce:node:get cce:addonInstance:list cce:node:create cce:node:list cce:addonInstance:get cce:accessPolicy:* cce:node:remove cce:nodepool:create cce:addonInstance:delete cce:cluster:get cce:addonInstance:create cce:job:get	获取CCE集群列表、集群详情、集群证书等信息。
	VPC	vpc:routeTables:get vpc:routeTables:list vpc:routes:create vpc:vpcs:get vpc:routes:get vpc:subnets:get vpc:routes:delete vpc:routes:list vpc:peerings:accept vpc:routeTables:update	ModelArts网络资源创建和删除、VPC网络打通。
	SFSTurbo	sfsturbo:shares:listShareNics sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:addShareNic	用户的网络和SFS Turbo资源打通。

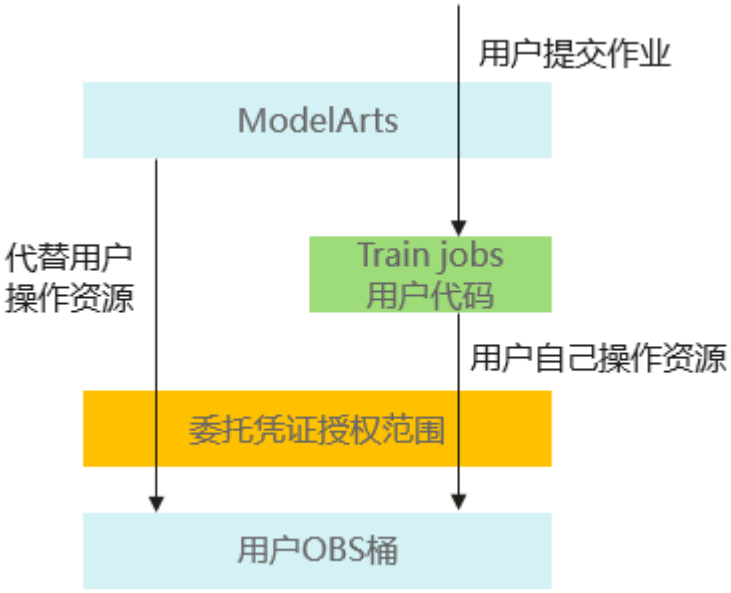
业务场景	依赖的服务	依赖策略项	支持的功能
	SFSTurboReadOnly	SFS Turbo ReadOnlyAccess	用户的网络和SFS Turbo资源打通。

委托授权

用户在使用ModelArts服务的过程中，为了简化用户的操作，ModelArts后台可以代替用户完成一些工作，如训练作业启动前自动下载用户OBS桶中的数据集到作业空间、自动转储训练作业日志到用户OBS桶中。

ModelArts服务不会保存用户的Token认证凭据，在后台异步作业中操作用户的资源（如OBS桶）前，需要用户通过IAM委托向ModelArts显式授权，ModelArts在需要时使用用户的委托获取临时认证凭据用于操作用户资源，见“**添加授权**”。

图 5-4 委托授权



如图5-4所示，用户向ModelArts授权后，ModelArts使用委托授权的临时凭证访问和操作用户资源，协助用户自动化一些繁琐和耗时的操作。同时，委托凭证会同步到用户的作业中（Notebook实例和训练作业），用户在作业中可以使用委托凭证自行访问自己的资源。

在ModelArts服务中委托授权有两种方式：

1、一键式委托授权

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

这种方式为保证使用业务过程中有足够的权限，基于依赖服务的预置系统策略指定授权范围，创建的委托的权限比较大，基本覆盖了依赖服务的全部权限。如果您需要对委托授权的权限范围进行精确控制，请使用第二种方式。

2、定制化委托授权

管理员在IAM中为不同用户创建不同的委托授权策略，再到ModelArts中为用户配置已创建好的委托。管理员在IAM中为用户创建委托时，根据用户的实际权限范围为委托指定最小权限范围，控制用户在使用ModelArts过程中可访问的资源内容。具体参考**配置ModelArts基本使用权限**。

委托授权的越权风险

可以看到用户的委托授权是独立的，理论上用户的委托授权范围是可以超出用户自身用户组的授权策略的授权范围，如果配置不当就会出现用户越权的问题。

为了控制委托授权的越权风险，ModelArts服务的权限管理功能要求只有租户管理员才能为用户配置委托，由管理员保证委托授权的安全性。

委托授权的最小化

管理员在配置委托授权时，应严格控制授权的范围。

ModelArts为用户异步自动化完成作业的准备、清理等操作，所需的委托授权内容是基础授权范围。如果用户只使用ModelArts的部分功能，管理员可以依据委托授权表格的说明屏蔽不使用的基础权限项。相反地，如果用户需要在作业中使用基础授权范围外的资源权限，管理员也可以为用户在委托授权中增加新的权限项。总之，委托授权的范围应该基于实际业务场景所需权限范围来进行定制，保持委托授权范围的最小化。

委托基础授权范围

当您需要定制委托授权的权限列表时，请参考下面表格，根据实际业务选择授权项。

表 5-16 开发环境基础委托授权

业务场景	依赖的服务	委托授权项	说明
Notebook实例中操作OBS数据。	OBS	obs:object:DeleteObject obs:object:GetObject obs:object:GetObjectVersion obs:bucket:CreateBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	您可通过以下方式在Notebook实例中操作OBS中的数据： • 通过ModelArts SDK操作OBS数据。 • 通过Notebook文件上传功能操作OBS数据。 • 通过在Console页面添加OBS桶到Notebook实例的/data目录下，以文件方式操作OBS数据。

业务场景	依赖的服务	委托授权项	说明
Notebook实例事件上报。	AOM	aom:alarm:put	在Notebook实例的生命周期中，部分事件会上报到用户AOM账号下，事件列表详见**Notebook实例事件**。
VPC与Notebook实例网络互联。	VPC	vpc:ports:create vpc:ports:get vpc:ports:delete vpc:subnets:get	Notebook实例中新增一个可以与用户指定VPC的子网的网卡，用于与用户VPC下的服务进行网络互联。
VS Code一键连接Notebook。	ModelArts	modelarts:notebook:get	用于管理Notebook实例信息，单击VS Code插件时，获取实例详情信息，以方便将SSH配置信息写入本地VS Code。
停止Notebook实例。	ModelArts	modelarts:notebook:stop	用于停止运行中的Notebook实例。
更新Notebook实例自动停止时间。	ModelArts	modelarts:notebook:updateStopPolicy	用于更新Notebook实例的自动停止时间。
OBS并行文件系统场景下使用MindInsight/TensorBoard可视化工具。	ModelArts	modelarts:notebook:umountStorage modelarts:notebook:getMountedStorage modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage	在开发环境Notebook实例中开启MindInsight/TensorBoard可视化工具，且需要访问的是OBS并行文件系统时，需要配置左侧的权限。

表 5-17 训练作业基础委托授权

业务场景	依赖的服务	委托授权项	说明
训练作业访问OBS文件。	OBS	obs:bucket:HeadBucket obs:bucket:GetBucketLocation obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:GetObject obs:object:GetObjectVersion obs:object:GetObjectAcl obs:object:GetObjectVersionAcl	训练作业配置代码目录、输入、输出和日志的OBS桶路径时，需要OBS服务相关操作权限，用于OBS对象路径的合法性校验。
训练作业以自定义容器镜像方式启动。	SWR	swr:repo:getRepo swr:system:createLoginSecret swr:instance:createTempCredential	训练作业以自定义容器镜像方式启动时，需要获取用户SWR容器镜像的临时登录指令，用于下载容器镜像。
训练作业状态变化通知。	SMN	smn:template:list smn:template:create smn:topic:list smn:topic:publish	若要配置训练作业状态变化通知，需要SMN服务相关操作权限，用于发送模板化的消息通知。
训练作业配置挂载SFS Turbo。	SFS Turbo	SFS Turbo ReadOnlyAccess	训练作业配置挂载SFS Turbo时，需要SFS Turbo读权限，以通过SFS Turbo ID获取其详情。

表 5-18 推理部署基础委托授权

业务场景	依赖的服务	委托授权项	说明
在线服务	LTS	lts:groups:create lts:groups:list lts:topics:create lts:topics:delete lts:topics:list	建议配置，在线服务配置LTS日志上报。
批量服务	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	使用批量服务时必须配置。

业务场景	依赖的服务	委托授权项	说明
边缘服务	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	使用边缘服务时必须配置，通过IEF部署边缘服务。
从OBS导入模型。	OBS	obs:object:DeleteObject obs:object:GetObject obs:bucket:CreateBucket obs:bucket:ListBucket obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	必须配置。若有使用并行文件系统，则需额外配置obs:bucket:HeadBucket。
从容器镜像中导入模型。	SWR	swr:instance:get swr:repository:getArtifact swr:repo:getRepoTag swr:repo:getRepo swr:repo:getRepoDomain swr:instance:createTempCredential swr:system:createLoginSecret swr:repo:deleteRepoTag swr:repo:deleteRepo swr:repo:createRepoDomain swr:repo:deleteRepoDomain	必须配置。
使用ModelArts Edge功能。	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	可选配置，如果使用ModelArts Edge功能需要配置。

业务场景	依赖的服务	委托授权项	说明
AOM指标告警事件	AOM	aom:log:get aom:alarm:get aom:metric:put aom:alarm:put aom:event:put aom:event:list aom:event:get	建议配置，若需要AOM查看告警事件则需要配置。
监控指标上报CES	CES	ces:metricMeta:create	建议配置，监控指标上报CES。
消息订阅推送	SMN	smn:topic:list smn:topic:publish smn:application:publish	可选配置，如果使用消息订阅推送功能需要配置。

表 5-19 数据管理基础委托授权

业务场景	依赖的服务	委托授权项	说明
使用数据标注、数据处理功能。	ModelArts	modelarts:trainJob:create modelarts:trainJob:update modelarts:trainJob:delete modelarts:trainJob:get modelarts:trainJob:list modelarts:trainJob:logExport modelarts:aiAlgorithm:get modelarts:model:get modelarts:service:list modelarts:model:create modelarts:workspace:list modelarts:workspace:get modelarts:trainJobInnerModel:list	必须配置，使用数据标注、数据处理功能时会进行创建训练作业、查询训练作业、算法查询等操作。

业务场景	依赖的服务	委托授权项	说明
访问OBS数据	OBS	obs:bucket:GetBucketLocation obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:GetObjectVersion obs:object:GetObject obs:object:GetObjectVersionAcl obs:object:DeleteObject obs:object:ListMultipartUploadParts obs:bucket:HeadBucket obs:object:AbortMultipartUpload obs:object:DeleteObjectVersion obs:object:GetObjectAcl obs:bucket:ListAllMyBuckets obs:bucket:ListBucket obs:object:PutObject	必须配置，在OBS中存储、查询、删除数据。
访问DLI数据。	DLI	dli:queue:createQueue dli:queue:dropQueue dli:queue:scaleQueue dli:queue:submitJob dli:database:displayDatabase dli:database:displayAllTables dli:table:describeTable dli:table:showPrivileges dli:table:dropTable	可选配置，如果访问DLI数据需要配置。
访问MRS数据。	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:file:list	可选配置，如果访问MRS数据需要配置。
访问DWS数据。	DWS	dws:openAPICluster:list dws:openAPICluster:getDetail dws:cluster:list	可选配置，如果访问DWS数据需要配置。

表 5-20 专属资源池管理基础委托授权

业务场景	依赖的服务	委托授权项	说明
通过关联sfsturbo功能实现专属资源池和SFS Turbo资源打通。	SFS Turbo	sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic	使用该特性时需要配置。
用户的ModelArts网络和VPC进行打通，同时添加相关路由。	VPC	vpc:vpcs:get vpc:subnets:get vpc:peerings:accept vpc:routes:create vpc:routes:delete vpc:routes:get vpc:routeTables:update vpc:routeTables:get vpc:routeTables:list vpc:routes:list	使用该特性时需要配置。
使用ModelArts Lite Cluster资源池。	CCE APM	cce:cluster:get cce:node:get cce:node:list cce:job:get cce:node:create cce:node:delete cce:node:remove cce:addonInstance:get cce:addonInstance:list cce:addonInstance:create cce:addonInstance:update cce:addonInstance:delete cce:accessPolicy:* cce:cluster:revokeClientCredential cce:nodepool:create cce:nodepool:delete apm:icmgr:create	使用ModelArts Lite Cluster资源池时必须配置。 ModelArts通过委托的方式管理用户的CCE集群，同步集群信息、纳管节点等。

业务场景	依赖的服务	委托授权项	说明
	ECS BMS EVS DEW	ecs:cloudServers:create ecs:cloudServers:delete ecs:cloudServers:get ecs:cloudServers:start ecs:cloudServers:stop ecs:cloudServers:reboot ecs:cloudServers:redeploy ecs:cloudServers:listServerInterfaces ecs:cloudServers:changeVpc ecs:cloudServerFlavors:get ecs:quotas:get ecs:cloudServers:batchSetServerTags ecs:cloudServers:list bms:servers:create bms:serverFlavors:get evs:types:get evs:volumes:list evs:quotas:get evs:volumes:get kps:domainKeypairs:get	使用ModelArts Lite Cluster资源池时必须配置。 ModelArts通过委托的方式对用户的BMS/ECS节点进行生命周期的管理。
	IMS	ims:images:get ims:images:share	使用ModelArts Lite Cluster资源池时必须配置。 ModelArts Lite Cluster专属资源池节点创建在用户账号下，创建前需要将节点系统镜像共享给用户账号。

业务场景	依赖的服务	委托授权项	说明
	CloudMatrix	cloudmatrix:hyperinstanceCluster:create cloudmatrix:hyperinstanceCluster:list cloudmatrix:hyperinstanceCluster:get cloudmatrix:hyperinstanceCluster:delete cloudmatrix:hyperinstanceCluster:getClusterSpec cloudmatrix:hyperinstance:create cloudmatrix:hyperinstance:delete cloudmatrix:hyperinstance:get cloudmatrix:hyperinstance:list cloudmatrix:hyperinstance:listNodes cloudmatrix:hyperinstance:getNode cloudmatrix:hyperinstance:stopNode cloudmatrix:hyperinstance:rebootNode cloudmatrix:hyperinstance:startNode cloudmatrix:hyperinstance:changeOs cloudmatrix:hyperinstance:reinstallNodeOs cloudmatrix:hyperinstance:liveScaleUp cloudmatrix:hyperinstance:liveScaleDown cloudmatrix:operation:get cloudmatrix:operation:list cloudmatrix:scheduledEvent:create cloudmatrix:scheduledEvent:list cloudmatrix:scheduledEvent:accept cloudmatrix:scheduledEvent:cancel cloudmatrix:hyperinstance:detachNodeInterface	使用ModelArts Lite Cluster资源池使用CloudMatrix超节点时必须配置。 ModelArts通过委托的方式对用户的CloudMatrix超节点进行生命周期的管理。

5.2.3 工作空间

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

工作空间支持3种访问控制：

- PUBLIC：租户（主账号和所有子账号）内部公开访问。
- PRIVATE：仅创建者和主账号可访问。
- INTERNAL：创建者、主账号、指定IAM子账号可访问当授权类型为INTERNAL时需要指定可访问的子账号的账号名，可选择多个。

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

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

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

目前工作空间功能是“受邀开通”状态，作为企业用户您可以通过您对口的技术支持申请开通。

5.3 典型场景配置实践

5.3.1 个人用户快速配置 ModelArts 访问权限

ModelArts使用过程中涉及到OBS、SWR等服务交互，需要用户配置委托授权，允许ModelArts访问这些依赖服务。如果没有授权，ModelArts的部分功能将不能正常使用。

约束与限制

- 只有主账号可以使用委托授权，可以为当前账号授权，也可以为当前账号下的所有IAM用户授权。
- 多个IAM用户或账号，可使用同一个委托。
- 一个账号下，最多可创建50个委托。
- 对于首次使用ModelArts新用户，请直接新增委托即可。一般用户新增普通用户权限即可满足使用要求。如果有精细化权限管理的需求，可以自定义权限按需设置。
- 如果未获得委托授权，当打开“访问授权”页面时，ModelArts会提醒您当前用户未配置授权，需联系此IAM用户的管理员账号进行委托授权。

添加授权

1. 登录**ModelArts控制台**，在左侧导航栏选择“系统管理 > 权限管理”，进入“权限管理”页面。

   说明

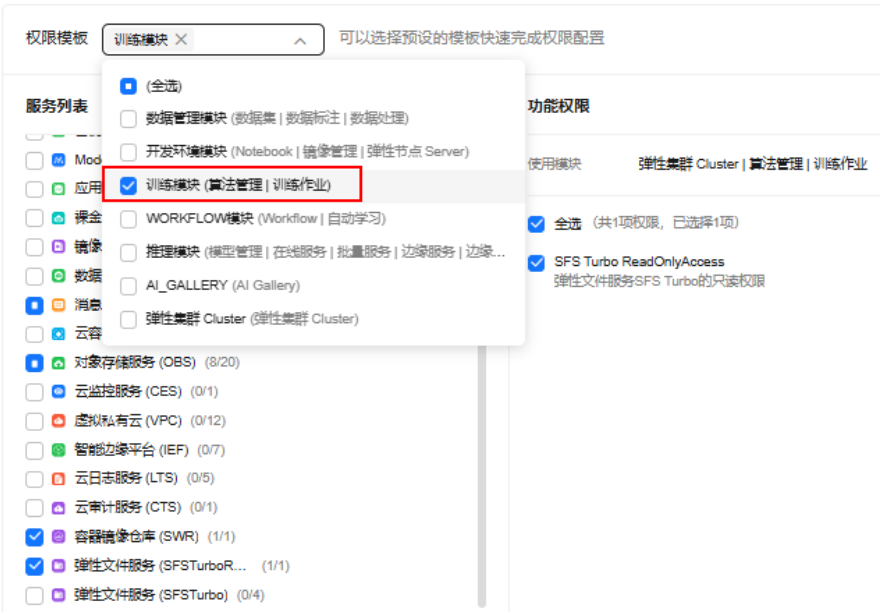
   ModelArts旧版导航页面中，在左侧导航栏选择“全局配置”，进入权限配置页面。

2. 单击“添加授权”，进入“添加授权”配置页面，根据参数说明进行配置。

表 5-21 参数说明

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

参数	说明
“授权对象”	“授权对象类型”选择“所有用户”时不涉及此参数。 • IAM子用户：选择指定的IAM子用户，给指定的IAM子用户配置委托授权。 **图 5-5** 选择 IAM 子用户  • 联邦用户：输入联邦用户的用户名或用户ID。 **图 5-6** 选择联邦用户  • 委托用户：选择委托名称。使用账号A创建一个权限委托，在此处将该委托授权给账号B拥有的委托。在使用账号B登录控制台时，可以在控制台右上角的个人账号切换角色到账号A，使用账号A的委托权限。 **图 5-7** 委托用户切换角色 
“委托选择”	• 已有委托：列表中如果已有委托选项，则直接选择一个可用的委托为上述选择的用户授权。单击委托名称查看该委托的权限详情。 • 新增委托：如果没有委托可选，可以在新增委托中创建委托权限。对于首次使用ModelArts的用户，需要新增委托。
“新增委托 > 委托名称”	系统自动创建委托名称，用户可以手动修改。

参数	说明
“新增委托 > 权限配置 > 普通模式”	普通模式下，用户可以基于使用的具体业务场景，自由定制授权范围，确保最小范围授权，授权方式灵活安全。适用于对授权范围有精细化管理要求的场景。 可以根据实际需要在权限列表中勾选要配置的权限。 普通模式下提供了预设的权限模板，可以通过权限模板快速配置权限。如下图所示，选择“训练模块”的权限模板，系统会自动勾选训练模块所需的所有服务和对应的功能权限。 **图 5-8** 基于预设的权限模板配置权限 

参数	说明
“新增委托 > 权限配置 > 高权限模式”	高权限模式下，配置的权限范围较大，适用于有管理员权限需求的用户。可以在搜索栏中搜索关键字快速筛选权限。 **图 5-9** 高权限模式 

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

5.3.2 配置 ModelArts 基本使用权限

5.3.2.1 场景描述

ModelArts作为顶层服务，其部分功能依赖于其他服务的访问权限。本章节主要介绍对于IAM子账号使用ModelArts时，如何根据需要开通的功能配置子账号相应权限。

权限列表

子账号的权限，由主用户来控制，主用户通过IAM的权限配置功能设置用户组的权限，从而控制用户组内的子账号的权限。此处的授权列表均按照ModelArts和其他服务的系统预置策略来举例。

表 5-22 服务授权列表

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

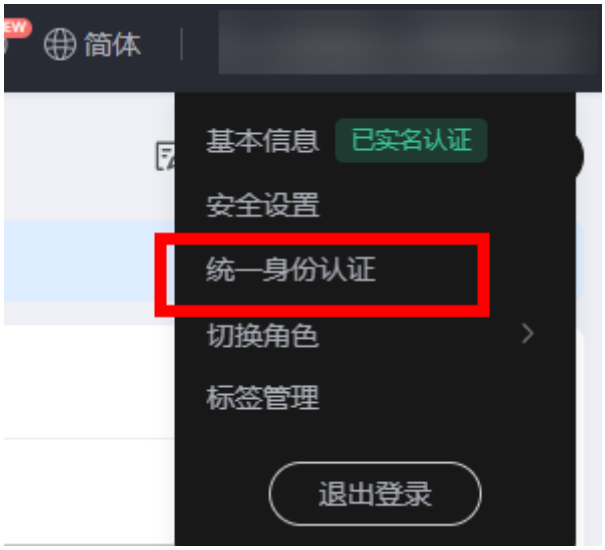
待授权的服务	授权说明	IAM权限设置	是否必选
SMN消息服务	授予子账号使用SMN消息服务的权限。SMN消息通知服务配合CES监控告警功能一起使用。	SMN FullAccess	可选
VPC虚拟私有云	子账号在创建ModelArts的专属资源池过程中，如果需要开启自定义网络配置，需要配置VPC权限。	VPC FullAccess	可选
SFS弹性文件服务	授予子账号使用SFS服务的权限，ModelArts的专属资源池中可以挂载SFS系统作为开发环境或训练的存储。	SFS Turbo FullAccess SFS FullAccess	可选

5.3.2.2 Step1 创建用户组并加入用户

主用户账号下面可以创建多个子账号，并对子账号的权限进行分组管理。此步骤介绍如何创建用户组、子账号、并将子账号加入用户组中。

1. 主用户登录**统一身份认证服务管理控制台**。

   图 5-10 统一身份认证

2. 创建用户组。在左侧菜单栏中，选择“用户组”。单击右上角“创建用户组”，在“用户组名称”中填入“用户组02”，然后单击“确定”完成用户组创建。

   创建完成后，返回用户组列表。通过用户组管理，将已有子账号加入到用户组中。如果没有子用户账号，可以创建子账号并加入用户组。

3. 创建子用户账号并加入用户组。在IAM左侧菜单栏中，选择“用户”，单击右上角“创建用户”，在“创建用户”页面中，添加多个用户。

   请根据界面提示，填写必选参数，然后单击“下一步”。

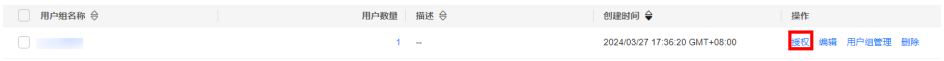
4. 在“加入用户组”步骤中，选择“用户组02”，然后单击“创建用户”。
   系统将逐步创建好前面设置的2个用户。

5.3.2.3 Step2 为用户配置云服务使用权限

主用户为子账号授予ModelArts、OBS等云服务的使用权限后，子账号才可以使用这些云服务。此步骤介绍如何为用户组中的所有子账号授予使用ModelArts、OBS、SWR等各类云服务的权限。

1. 主用户在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子账号配置权限。

   图 5-11 为用户组授权

2. 配置授权前，请先了解ModelArts各模块使用到的最小权限要求，如表5-22所示。
3. 配置ModelArts使用权限。在搜索框搜索ModelArts。ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。
   选择说明如下：
   - ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子账号配置此权限。
   - 如果需要给子账号开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。
4. 配置OBS使用权限。搜索OBS，勾选“OBS Administrator”。ModelArts训练作业中需要依赖OBS作为数据中转站，需要配置OBS的使用权限。
5. 配置SWR使用权限。搜索SWR，勾选“SWR FullAccess”。ModelArts的自定义镜像功能依赖镜像服务SWR FullAccess权限。
6. （可选）配置密钥管理权限。如果需要使用ModelArts Notebook的SSH访问功能，依赖密钥管理权限。搜索DEW，勾选“DEW KeypairFullAccess”。
   此处需要注意以下Region配置的是DEW密钥管理权限：华北-北京一、华北-北京四、华东-上海一、华东-上海二、华南-广州、西南-贵阳一、中国-香港、亚太-新加坡。其他Region配置的是KMS密钥管理权限。本示例中使用“华南-广州”Region举例，所以需要配置DEW密钥管理权限。
7. （可选）配置智能边缘平台使用权限。ModelArts的边缘服务依赖智能边缘平台，要求配置Tenant Administrator权限。
   注意：Tenant Administrator权限比较大，包含全部云服务的管理权限，而不仅是使用ModelArts服务。请谨慎配置。
8. （可选）配置CES云监控和SMN消息通知使用权限。ModelArts推理部署的在线服务详情页面内有调用次数详情，单击可查看该在线服务的调用次数随时间详细分布的情况。如果想进一步通过CES云监控查看ModelArts的在线服务和对应模型负载运行状态的整体情况，需要给子账号授予CES权限。
   如果只是查看监控，给子账号授予CES ReadOnlyAccess权限即可。
   如果还需要在CES上设置监控告警，则需要再加上CES FullAccess权限，以及SMN消息通知权限。
9. （可选）配置VPC权限。如果用户在创建专属资源池过程中，需要开启自定义网络配置，此处需要授予用户VPC权限。

10. （可选）配置SFS和SFS Turbo权限。如果用户在专属资源池中挂载SFS系统作为开发环境或训练的存储时，需要授予使用权限。
11. 单击左上角的“查看已选”，确认已勾选的权限。
12. 再单击“下一步”，设置最小授权范围。单击“指定区域项目资源”，勾选待授权使用的区域，单击“确定”。
13. 提示授权成功，查看授权信息，单击“完成”。此处的授权生效需要15-30分钟。

5.3.2.4 Step3 为用户配置 ModelArts 的委托访问授权

配置完IAM权限之后，需要在ModelArts页面为子用户设置ModelArts访问授权，允许ModelArts访问OBS、SWR、IEF等依赖服务。

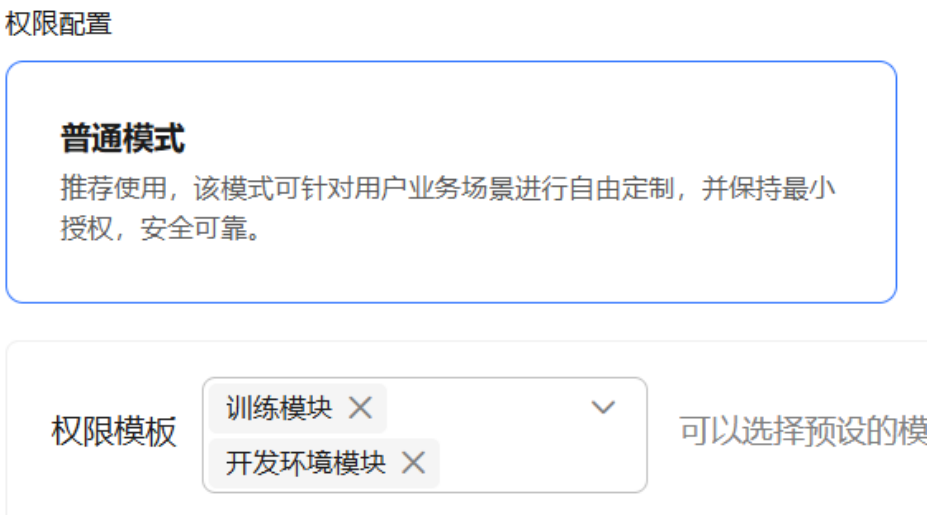
此方式只允许主用户为子用户进行配置。因此，本示例中，管理员账号需为子用户完成访问授权的配置。

1. 使用主用户的账号登录**ModelArts控制台**。请注意选择左上角的区域。
2. 在左侧导航栏单击“权限管理”，进入“权限管理”页面。
3. 单击“添加授权”。在“添加授权”页面，在“授权对象类型”下面选择“IAM子用户”，“授权对象”选择要授权的子账号，选择“新增委托 ”。

图 5-12 授权对象类型

4. 在权限配置区域，选择“普通模式 ”，权限模板区域勾选子用户需要使用的功能场景，可以基于预设权限模板为该子用户配置所需的最小授权。

图 5-13 配置权限



5. 勾选“我已经详细阅读并同意《ModelArts服务声明》”，单击“创建”，完成委托授权配置。

5.3.2.5 Step4 测试用户权限

由于4中的权限需要等待15-30分钟生效，建议在配置完成后，等待30分钟，再执行如下验证操作。

1. 使用用户组02中任意一个子账号登录**ModelArts控制台**。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。

2. 验证ModelArts权限。
   a. 在左上角选择区域，区域需与授权配置中的区域相同。
   b. 在ModelArts左侧菜单栏中，选择“开发环境>Notebook”，界面未提示权限不足，表明ModelArts的使用权限和委托授权配置成功。

      如果提示“需获取依赖服务的授权”，说明未配置ModelArts委托访问授权，请参考**Step3 为用户配置ModelArts的委托访问授权**，使用主用户为子账号配置ModelArts委托访问授权。
   c. 在ModelArts左侧菜单栏中，选择“开发环境>Notebook”，单击“创建”，如果可以正常打开创建页面，说明具备ModelArts的操作权限。

      您也可以尝试其他功能，例如“训练管理>训练作业”等，如能正常打开创建页面，即可正常使用ModelArts。
3. 验证OBS权限。
   a. 在左上角的服务列表中，选择OBS服务，进入OBS管理控制台。
   b. 在OBS管理控制台，单击右上角的“创建桶”，如果能正常打开页面，表示当前用户具备OBS的操作权限。
4. 验证SWR权限。
   a. 在左上角的服务列表中，选择SWR服务，进入SWR管理控制台。
   b. 在SWR管理控制台，如果能正常打开页面，表示当前用户具备SWR的操作权限。
5. 依次验证其他可选权限。

6. 验证结束，当前用户同时具备ModelArts部分功能的操作权限，可正常开始使用ModelArts服务。

5.3.3 管理员和开发者权限分离

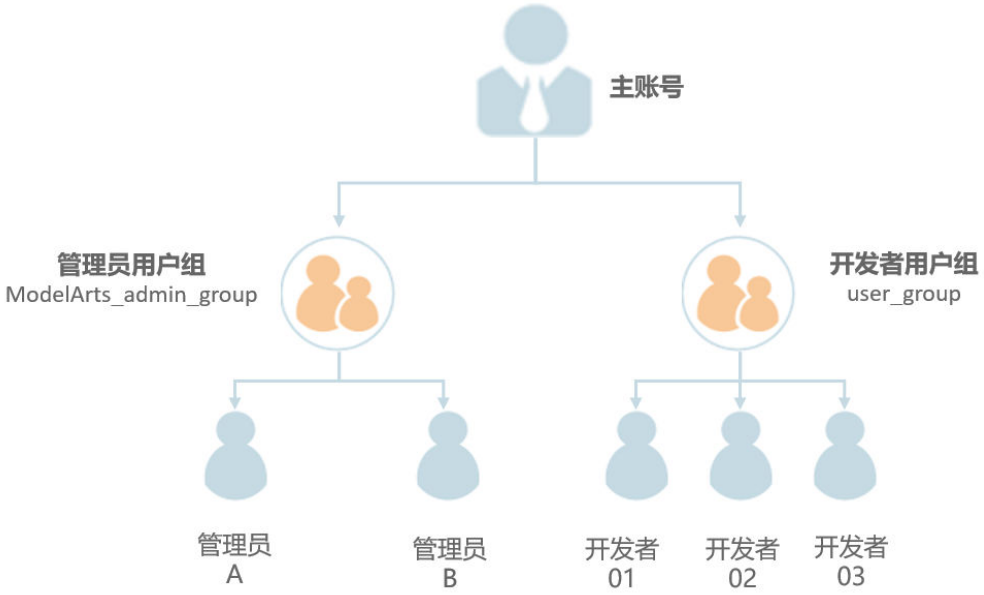
对于中小规模团队，管理员希望对ModelArts资源进行主导分配，全局控制，而对于普通开发者只需关注自己实例的生命周期控制。对于开发者账号，一般不会具有te_admin的权限，相应的权限也需要主账号进行统一配置。本章节以使用Notebook进行项目开发为例，通过自定义策略配置实现管理员和开发者分离。

场景描述

以使用Notebook进行项目开发为例，管理员账号需要拥有ModelArts专属资源池的完全控制权限，以及Notebook所有实例的访问和操作权限。

普通开发者使用开发环境，只需关注对自己Notebook实例的操作权限，包括对自己实例的创建、启动、停止、删除等权限以及周边依赖服务的权限。普通开发者不需要ModelArts专属资源池的操作权限，也不需要查看其他用户的Notebook实例。

图 5-14 账号关系示意图



配置管理员权限

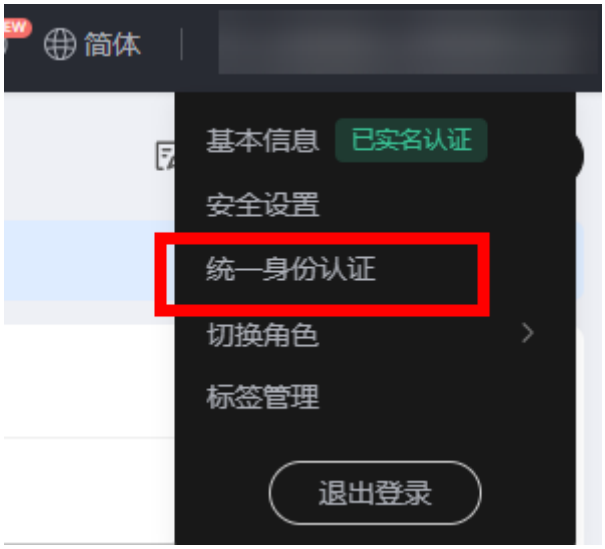
管理员账号需要拥有ModelArts专属资源池的完全控制权限，以及Notebook所有实例的访问和操作权限。可以通过以下配置流程实现管理员权限配置。

步骤1 使用主账号创建一个管理员用户组ModelArts_admin_group，将管理员账号加入用户组ModelArts_admin_group中。具体操作请参见**Step1 创建用户组并加入用户**。

步骤2 创建自定义策略。

1. 使用管理员账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

图 5-15 登录控制台



2. 创建自定义策略1，赋予用户IAM和OBS服务权限。在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，在“策略名称”中填入“Policy1_IAM_OBS”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy1_IAM_OBS”的具体内容如下，赋予用户IAM和OBS操作权限。可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:users:listUsers",
                "iam:projects:listProjects",
                "obs:object:PutObject",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:bucket:HeadBucket",
                "obs:object:DeleteObject",
                "obs:bucket:CreateBucket",
                "obs:bucket:ListBucket"
                ]
        }
    ]
}
```

3. 重复**步骤2.2**创建自定义策略2，赋予用户依赖服务ECS、SWR、MRS和SMN的操作权限，ModelArts的操作权限。“策略名称”为“Policy2_AllowOperation”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy2_AllowOperation”的具体内容如下，赋予用户依赖服务ECS、SWR、MRS和SMN的操作权限，ModelArts的操作权限。可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:serverKeypairs:list",
                "ecs:serverKeypairs:get",
                "ecs:serverKeypairs:delete",
                "ecs:serverKeypairs:create",
```

```
                "swr:repository:getNamespace",
                "swr:repository:listNamespaces",
                "swr:repository:deleteTag",
                "swr:repository:getRepository",
                "swr:repository:listTags",
                "swr:instance:createTempCredential",
                "mrs:cluster:get",
                "modelarts:*:*"
            ]
        }
    ]
}
```

步骤3 将**步骤2**创建的自定义策略授权给管理员用户组ModelArts_admin_group。

1. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称ModelArts_admin_group操作列的“授权”，勾选策略“Policy1_IAM_OBS”和“Policy2_AllowOperation”。单击“下一步”。
2. 选择授权范围方案为所有资源，单击“确定”。

步骤4 给管理员用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用主账号登录**ModelArts控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”。在“访问授权”页面，在“授权对象类型”下面选择“IAM子用户”，“授权对象”选择管理员的账号，选择“新增委托”，“权限配置”选择“普通用户”。管理员不做权限控制，此处默认使用普通用户委托即可。
3. 勾选“我已经详细阅读并同意《 ModelArts服务声明 》”，单击“创建”。

步骤5 测试管理员用户权限。

1. 使用管理员用户登录**ModelArts控制台**。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 在ModelArts控制台的左侧导航栏中，选择“专属资源池”，单击创建，未提示权限不足，表明管理员用户的权限配置成功。

----结束

配置开发者权限

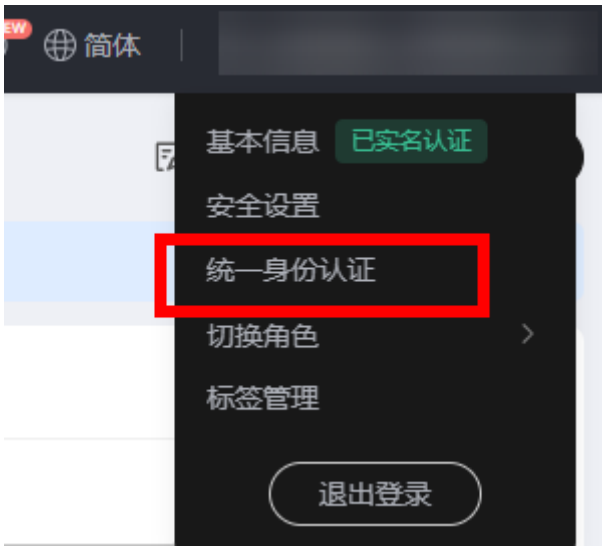
开发者权限需要通过IAM的细粒度授权控制实现，可以通过以下配置流程实现开发者权限配置。

步骤1 使用主账号创建一个开发者用户组user_group，将开发者账号加入用户组user_group中。具体操作请参见**Step1 创建用户组并加入用户**。

步骤2 创建自定义策略。

1. 使用主账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

图 5-16 登录控制台



2. 创建自定义策略3，拒绝用户操作ModelArts专属资源池并拒用户查看其他用户的Notebook。

   在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，“策略名称”为“Policy3_DenyOperation”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy3_DenyOperation”的具体内容如下，可以直接复制粘贴。

```
{
   "Version": "1.1",
   "Statement": [
         {
          "Effect": "deny",
          "Action": [
              "modelarts:pool:create",
              "modelarts:pool:update",
              "modelarts:pool:delete",
       "modelarts:notebook:listAllNotebooks"
            ]

         }
   ]
}
```

步骤3 将自定义策略授权给开发者用户组user_group。

1. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称user_group操作列的“授权”，勾选策略“Policy1_IAM_OBS”、“Policy2_AllowOperation”和“Policy3_DenyOperation”。单击“下一步”。
2. 选择授权范围方案为所有资源，单击“确定”。

步骤4 给开发者用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用主账号登录**ModelArts控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”。在“访问授权”页面，在“授权对象类型”下面选择“IAM子用户”，“授权对象”选择开发者的账号，“委托选择”选择“新增委托”，“委托名称”设置为“ma_agency_develop_user”，“权限配置”选择“自定义”，“权限名称”勾选“OBS Administrator”。开发者用户只需要配置OBS的委托授权即可，允许开发者用户在使用Notebook时，与OBS服务交互。

3. 单击“创建”。
4. 在“权限管理”页面，再次单击“添加授权”，进入“访问授权”页面，为其他开发者用户配置委托。

   “授权对象类型”选择“IAM子用户”，“授权对象”选择开发者的账号，“委托选择”选择“已有委托”，“委托名称”勾选上一步创建的“ma_agency_develop_user”，

步骤5 测试开发者用户权限。

1. 使用user_group用户组中任意一个子账号登录**ModelArts控制台**。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 在ModelArts左侧菜单栏中，选择“专属资源池”，单击创建，界面未提示权限不足，表明开发者用户的权限配置成功。

----结束

5.3.4 给子账号配置查看所有 Notebook 实例的权限

查找实例

Notebook页面展示了所有创建的实例。如果需要查找特定的实例，可根据筛选条件快速查找。

- 参考**给子账号配置查看所有Notebook实例的权限**后，进入“开发空间>Notebook”页面，打开“查看所有”开关，可以看到IAM项目下所有子账号创建的Notebook实例。
- 按实例名称、实例ID、实例状态、使用的镜像、实例规格、实例描述、创建时间等单个筛选或组合筛选。

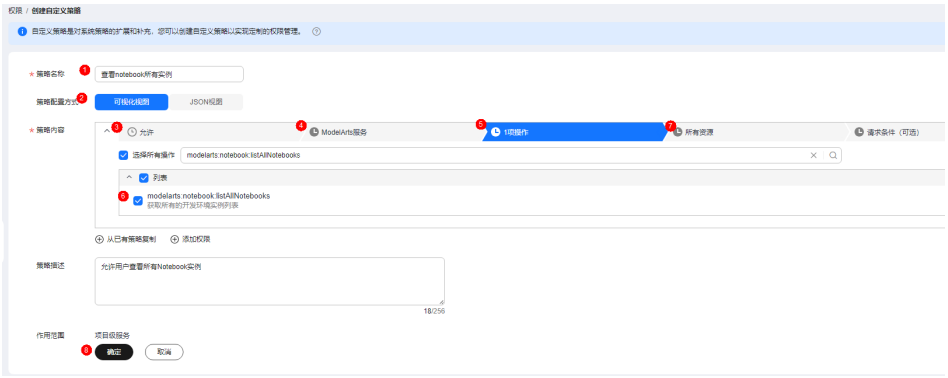
给子账号配置查看所有 Notebook 实例的权限

当子账号被授予“listAllNotebooks”和“listUsers”权限时，在Notebook页面上，单击“查看所有”，可以看到IAM项目下所有子账号创建的Notebook实例 。配置该权限后，也可以在Notebook中访问子账号的OBS、SWR等。

1. 使用主用户账号登录ModelArts管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，需要设置两条策略。

   策略1：设置查看Notebook所有实例，如**图5-17**所示，单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：查看Notebook所有实例。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:notebook:listAllNotebooks并选中，所有资源选择默认值。

图 5-17 创建自定义策略



策略2：设置查看Notebook实例创建者信息的策略。

- “策略名称”：设置自定义策略名称，例如：查看所有子账号信息。
- “策略配置方式”：选择可视化视图。
- “策略内容”：允许，云服务中搜索IAM服务并选中，操作列中搜索关键词iam:users:listUsers并选中，所有资源选择默认值。

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的两条自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限查看该用户组内成员创建的所有Notebook实例。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子账号没有在用户组中，也可以通过“用户组管理”功能增加用户。

子账号启动其他用户的 SSH 实例

子账号可以看到所有用户的Notebook实例后，如果要通过SSH方式远程连接其他用户的Notebook实例，需要将SSH密钥对更新成自己的，否则会报错ModelArts.6786。更新密钥对具体操作请参见**修改Notebook SSH远程连接配置**。具体的错误信息提示：ModelArts.6789: 在ECS密钥对管理中找不到指定的ssh密钥对xxx，请更新密钥对并重试。

5.3.5 使用 Cloud Shell 登录训练容器

使用场景

允许用户使用ModelArts控制台提供的Cloud Shell登录运行中的训练容器。

约束限制

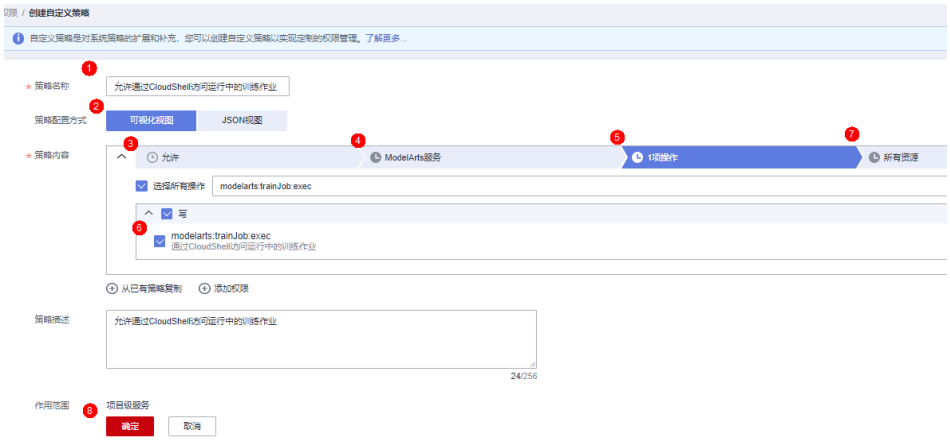
仅专属资源池支持使用Cloud Shell，且训练作业必须处于“运行中”状态。

前提条件：给子账号配置允许使用 Cloud Shell 的权限

1. 使用主用户账号登录华为云的管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”按如下要求设置完成后单击“确定”。

- “策略名称”：设置自定义策略名称，例如：允许通过Cloud Shell访问运行中的训练作业。
- “策略配置方式”：选择可视化视图。
- “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:trainJob:exec并选中，所有资源选择默认值。

图 5-18 创建自定义策略



3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限通过Cloud Shell登录运行中的训练作业容器。

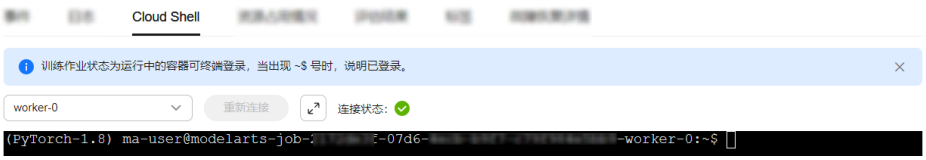
   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

如何使用 Cloud Shell

1. 参考**前提条件：给子账号配置允许使用Cloud Shell的权限**，完成配置。
2. 在ModelArts管理控制台的左侧导航栏中选择“模型训练 > 训练作业”。
3. 在训练作业列表中，单击作业名称进入训练作业详情页面。
4. 在训练作业详情页面，单击“Cloud Shell”页签，登录训练容器。

   连接成功后，Cloud Shell界面提示如下。

图 5-19 Cloud Shell 界面



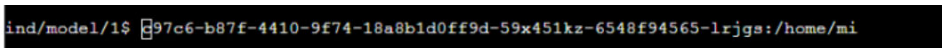
当作业处于非运行状态或权限不足时会导致无法使用Cloud Shell，请根据提示定位原因即可。

图 5-20 报错提示

说明

部分用户登录Cloud Shell界面时，可能会出现路径显示异常情况，此时在Cloud Shell中单击回车键即可恢复正常。

图 5-21 路径异常



5.3.6 不允许子账号使用公共资源池创建作业

本章节介绍如何控制ModelArts用户权限，限制用户使用ModelArts公共资源池的资源创建训练作业、创建开发环境实例，部署推理服务等。

场景介绍

对于ModelArts专属资源池的用户，不允许使用公共资源池创建训练作业、创建Notebook实例或者部署推理服务时，可以通过权限控制限制用户使用公共资源池。

涉及配置的自定义权限策略项如下;

- modelarts:notebook:create：此策略项表示创建Notebook实例。
- modelarts:trainJob:create：此策略项表示创建训练作业。
- modelarts:service:create：此策略项表示创建推理服务。

给子账号配置权限：限制使用公共资源池

1. 使用主用户账号登录管理控制台，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”，设置策略，单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：不允许用户使用公共资源池创建。
   - “策略配置方式”：选择可视化视图或者JSON视图均可。
   - “策略内容”：拒绝，云服务中搜索“ModelArts”服务并选中，“操作”中查找写操作“modelarts:trainJob:create”、“modelarts:notebook:create”和“modelarts:service:create”并选中。“所有资源”选择“默认值”。“请求条件”中单击“添加条件”，设置“条件键”为

“modelarts:poolType”，“运算符”为“StringEquals”，“值”为“public”。

JSON视图的策略内容如下：

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "modelarts:trainJob:create",
                "modelarts:notebook:create",
                "modelarts:service:create"
            ],
            "Condition": {
                "StringEquals": {
                    "modelarts:poolType": [
                        "public"
                    ]
                }
            }
        }
    ]
}
```

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

4. 在用户的委托授权中同步增加此策略，避免在租户面通过委托token突破限制。

   在统一身份认证服务页面的左侧导航中选择委托，找到该用户组在ModelArts上使用的委托名称，单击右侧的“修改”操作，选择“授权记录”页签，单击“授权”，选中上一步创建的自定义策略“不允许用户使用公共资源池”，单击“下一步”，选择允许使用的资源区域，单击“确定”。

验证

使用子账号用户登录**ModelArts控制台**，选择“模型训练 > 训练作业”，单击“创建训练作业”，在创建训练页面，资源池规格只能选择专属资源池。

使用子账号用户登录**ModelArts控制台**，选择“开发空间 > Notebook”，单击“创建”，在创建Notebook页面，资源池规格只能选择专属资源池。

使用子账号用户登录**ModelArts控制台**，选择“模型部署 > 在线服务”，单击“部署”，在部署服务页面，资源池规格只能选择专属资源池。

5.3.7 委托授权 ModelArts 云服务使用 SFS Turbo

本章节介绍如何配置ModelArts委托权限，允许用户使用专属资源池的网络中的“关联sfsturbo”和“解除关联”功能。

- 当用户新增委托并授权操作SFS Turbo时，请参考**新增委托授权操作SFS Turbo**。
- 当用户为已有的委托新增权限，授权操作SFS Turbo，请参考**已有委托新增授权操作SFS Turbo**。

场景介绍

对于使用ModelArts专属资源池的用户，在控制台创建完网络后，在网络列表页“操作 > 更多”下拉框中可见“关联sfsturbo”和“解除关联”。其中，“关联sfsturbo”用于将此网络与某个选定的SFS Turbo资源做关联操作，关联完成后，表示SFS Turbo与网络已进行打通，可在训练和开发环境等功能时使用此SFS Turbo。

关联与解除关联操作需要用户委托授权ModelArts云服务操作SFS Turbo的部分权限。

涉及配置的自定义权限策略项如下：

- sfsturbo:shares:addShareNic：此策略项表示sfsturbo创建网卡的权限。
- sfsturbo:shares:deleteShareNic：此策略项表示sfsturbo删除网卡的权限。
- sfsturbo:shares:showShareNic：此策略项表示sfsturbo显示网卡详情的权限。
- sfsturbo:shares:listShareNics：此策略项表示sfsturbo显示网卡列表的权限。

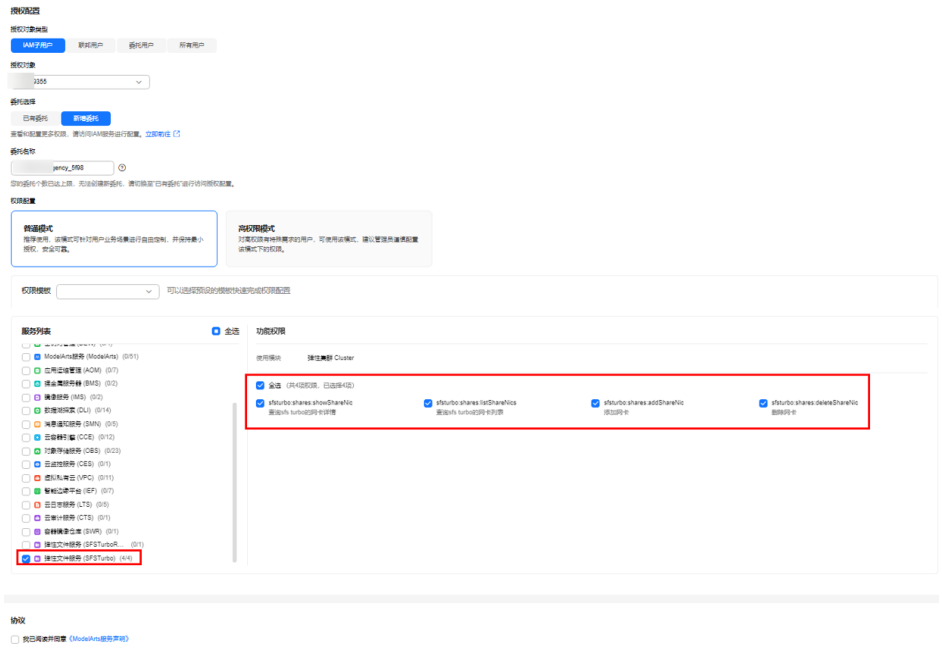
约束限制

相应region区域开放此功能。

新增委托授权操作 SFS Turbo

1. 登录**ModelArts控制台**，在左侧导航栏选择“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”，进入“访问授权”配置页面，根据参数说明进行配置。
   - “授权对象类型”：根据需要选择"IAM子用户"、"联邦用户"、"委托用户"、"所有用户"
   - “授权对象”：选择授权对象
   - “委托选择”：新增委托
   - “权限配置”：普通模式，选中弹性文件服务(SFSTurbo)下的"sfsturbo:shares:addShareNic"、"sfsturbo:shares:deleteShareNic"、"sfsturbo:shares:showShareNic"、"sfsturbo:shares:listShareNics"
3. 单击“创建”。此时，拥有该委托的所有用户均有权限进行关联与解除关联操作。

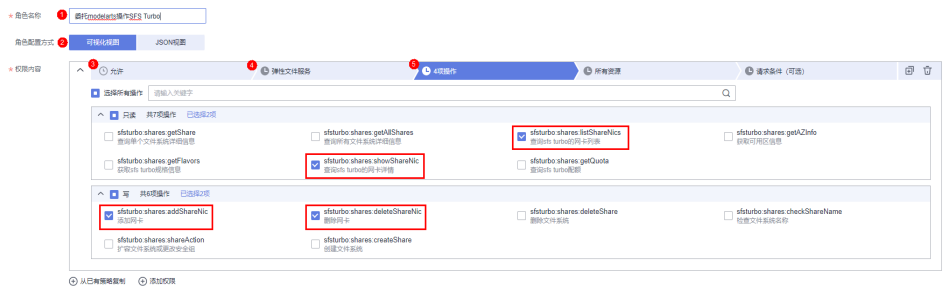
图 5-22 创建授权 ModelArts 云服务操作 SFS Turbo 的部分权限



已有委托新增授权操作 SFS Turbo

1. 使用主用户账号登录管理控制台，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”，设置策略，单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：委托modelarts操作SFS Turbo。
   - “策略配置方式”：选择可视化视图或者JSON视图均可。
   - “策略内容”：允许，云服务中搜索“SFSTurbo”服务并选中，“操作”中查找**只读**操作“sfsturbo:shares:showShareNic”、“sfsturbo:shares:listShareNics”并选中，查找**写**操作“sfsturbo:shares:addShareNic”、“sfsturbo:shares:deleteShareNic”并选中。“所有资源”选择“所有资源”。

     图 5-23 创建自定义策略（可视化视图）

JSON视图的策略内容如下：

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sfsturbo:shares:addShareNic",
                "sfsturbo:shares:listShareNics",
                "sfsturbo:shares:deleteShareNic",
                "sfsturbo:shares:showShareNic"
            ]
        }
    ]
}
```

3. 以委托“modelarts_agency”为例，在统一身份认证服务页面的左侧导航选择“委托”，在委托页面查找“modelarts_agency”，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略“委托modelarts操作SFS Turbo”，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，拥有该委托的所有用户均有权限进行关联与解除关联操作。

验证

登录**ModelArts控制台**，选择“专属资源池 > 网络”，单击“更多”，选择“关联sfsturbo”，关联成功。

登录**ModelArts控制台**，选择“专属资源池 > 网络”，单击“更多”，选择“解除关联”，解除成功。

5.3.8 给子账号配置文件夹级的 SFS Turbo 访问权限

场景描述

本文介绍如何配置文件夹级的SFS Turbo访问权限，实现在ModelArts中访问挂载的SFS Turbo时，只允许子账号访问特定的SFS Turbo文件夹内容。

说明

给子账号配置文件夹级的SFS Turbo访问权限为白名单功能，如果有试用需求，请提工单申请权限。

前提条件

- 需要在ModelArts控制台打开严格授权模式，单击“权限管理 > 启用严格模式”，在输入框中输入“YES”，单击“确定”。
- 如果打开严格模式前没有为子账号配置过ModelArts权限，开启严格授权模式后可能会导致子账号无法使用ModelArts功能，请根据您的业务需求配置需要的ModelArts服务的权限（参见**依赖和委托**中ModelArts服务对应的依赖策略项）。

操作步骤

步骤1 使用主用户账号登录管理控制台，鼠标放在右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤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”，设置策略。

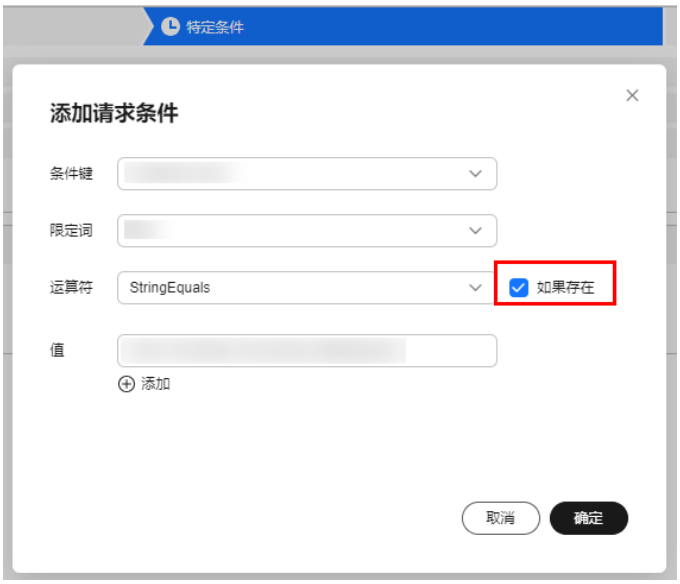
- “策略名称”：设置自定义策略名称，例如：ma_sfs_turbo。
- “策略配置方式”：JSON视图。
- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "<modelarts_action>"
      ],
      "Condition": {
        "StringEqualsIfExists": {
          "modelarts:sfsId": [
            "<your_ssf_id>"
          ],
          "modelarts:sfsPath": [
            "<sfs_path>"
          ],
          "modelarts:sfsOption": [
            "<sfs_option>"
          ]
        }
      }
    }
  ]
}
```

说明

- 未创建以上权限策略前，所有子账号默认可以挂载SFS Turbo。当您创建了以上SFS权限管控策略后，没有被授予以上权限的子账号，默认在ModelArts Console上创建训练作业时无法挂载SFS Turbo（具有Tenant Administrator权限的子账号除外）。
- 当前仅支持配置允许策略的权限（即以上“策略内容”中的“Effect”只能配置为“Allow”），请勿配置拒绝策略的权限。
- Condition参数必须使用“StringEqualsIfExists”字段，对应可视化视图为勾选“如果存在”的开关。

图 5-24 “如果存在”的开关



以上代码中的"*<modelarts_action>*"、"*<your_ssf_id>*"、"*<sfs_path>*"、"*<sfs_option>*"，需要根据您的业务需求替换为实际的参数，各参数含义如下。

表 5-23 参数解释

参数	参数解释
Action	表示在何种场景下授予SFS Turbo文件夹访问权限。 ● 创建开发环境实例：modelarts:notebook:create ● 创建训练作业：modelarts:trainJob:create 支持填写多种Action，例如： "Action": ["modelarts:trainJob:create", "modelarts:notebook:create"],
modelarts:sfsId	SFS Turbo的ID，在SFS Turbo详情页查看。支持填写多个ID，例如： "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"],
modelarts:sfsPath	需要进行权限配置的SFS Turbo文件夹路径。支持填写多个路径，例如： "modelarts:sfsPath": ["/path1", "/path2/path2-1"], 如果sfsId中填写了多个ID，则sfsPath会应用于所有sfsId。例如以下代码含义为：为"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"的"/path1"和"/path2/path2-1"配置访问权限，同时也为"2a70da1e-ea87-4ee4-ae1e-55df846e7f41"的"/path1"和"/path2/path2-1"配置访问权限。 "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"], "modelarts:sfsPath": ["/path1", "/path2/path2-1"],

参数	参数解释
modelarts:sfsOption	设置用户对于SFS Turbo文件夹的权限类型，支持填写以下参数： • 仅读权限：readonly • 读写权限：readwrite（创建开发环境实例modelarts:notebook:create仅支持配置readwrite）

如果需要在一个自定义策略中添加多个不同的sfsOption，需要“Statement”中新增JSON结构体，例如：

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:trainJob:create"
            ],
            "Condition": {
                "StringEqualsIfExists": {
                    "modelarts:sfsId": [
                        "0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
                    ],
                    "modelarts:sfsPath": [
                        "/path1"
                    ],
                    "modelarts:sfsOption": [
                        "readonly"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:trainJob:create"
            ],
            "Condition": {
                "StringEqualsIfExists": {
                    "modelarts:sfsId": [
                        "0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
                    ],
                    "modelarts:sfsPath": [
                        "/path2"
                    ],
                    "modelarts:sfsOption": [
                        "readwrite"
                    ]
                }
            }
        }
    ]
}
```

步骤3 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤4 给用户组授权策略。在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子账号配置权限。勾选**步骤2**中创建的“ma_sfs_turbo”策略。单击“下一步”和“确定”。

步骤5 在已有的ModelArts委托权限中，追加IAM ReadOnlyAccess权限。

1. 在ModelArts管理控制台，单击“权限管理 ”，在对应委托的操作列，单击“查看权限 > 去IAM修改委托权限”。

2. 在新页面中，单击“授权记录 > 授权”，搜索“IAM ReadOnlyAccess”，勾选后单击“下一步”并单击“确认”。

步骤6 验证权限是否配置成功。

登录子用户账号，在创建训练作业/创建Notebook时，仅能看到配置的SFS Turbo文件夹，则表示权限配置成功。

----结束

5.4 FAQ

5.4.1 使用 ModelArts 时提示“权限不足”，如何解决?

当您使用ModelArts时如果提示权限不足，请您按照如下指导对相关服务和用户进行授权，并对用户权限进行检查操作。

本案例中以OBS权限不足为例，介绍如何为用户授予OBS服务权限。其它权限不足的场景也可以参考本案例操作，只是授权范围不同。不同业务场景下的授权范围请参考**权限依赖和委托**章节。

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

- 如果您需要授予用户关于OBS的所有权限和ModelArts的基础操作权限，请参见**配置基础操作权限**。
- 如果您需要对用户使用OBS和ModelArts的权限进行精细化管理，进行自定义策略配置，请参见**创建ModelArts自定义策略**。

配置基础操作权限

使用ModelArts的基本功能，您需要为用户配置“作用范围”为“项目级服务”的“ModelArts CommonOperations”权限，由于ModelArts依赖OBS权限，您还需要登录IAM管理控制台为用户授予“作用范围”为“全局级服务”的“OBS Administrator”策略。

具体操作步骤如下：

步骤1 创建用户组。

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

步骤2 配置用户组权限。

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

1. 配置“作用范围”为“项目级服务”的“ModelArts CommonOperations”权限，如下图所示，然后单击“确定”完成授权。

   说明

   区域级项目授权后只在授权区域生效，如果需要所有区域都生效，则所有区域都需要进行授权操作。

2. 配置“作用范围”为“全局级服务”的“OBS Administrator”权限，然后单击“确定”完成授权。

步骤3 创建用户并加入用户组。

在IAM控制台创建用户，并将其加入步骤1中创建的用户组。

步骤4 用户登录并验证权限。

新创建的用户登录控制台，切换至授权区域，验证权限：

- 在“服务列表”中选择ModelArts，进入ModelArts主界面，选择不同类型的专属资源池，在页面单击“创建”，如果无法进行创建（当前权限仅包含ModelArts CommonOperations），表“ModelArts CommonOperations”已生效。
- 在“服务列表”中选择除ModelArts外（假设当前策略仅包含ModelArts CommonOperations）的任一服务，如果提示权限不足，表示“ModelArts CommonOperations”已生效。
- 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“数据管理>数据集>创建数据 > 集”，如果可以成功访问对应的OBS路径，表示全局级服务的“OBS Administrator”已生效。

----结束

创建 ModelArts 自定义策略

如果系统预置的ModelArts权限不满足您的授权要求，或者您需要管理用户操作OBS的操作权限，可以创建自定义策略。更多关于创建自定义策略操作和参数说明请参见**创建自定义策略**。

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

说明

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

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

1. 创建ModelArts相关OBS的最小化权限的自定义策略。

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

   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts依赖的OBS权限自定义策略样例**，如果您需要了解更多关于OBS的系统权限，请参见**OBS权限管理**。

2. 创建ModelArts开发环境的使用权限的自定义策略。参数配置说明如下：

   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts开发环境使用权限的自定义策略样例**，ModelArts自定义策略中可以添加的授权项（Action）请参见**《ModelArts API参考》>权限策略和授权项**。
   - 如果您需要对除ModelArts和OBS之外的其它服务授权，IAM支持服务的所有策略请参见**权限策略**。

3. 在IAM控制台**创建用户组并授权**。

   在IAM控制台创建用户组之后，将步骤**1**中创建的自定义策略授权给该用户组。

4. **创建用户并加入用户组**。

   在IAM控制台创建用户，并将其加入**3**中创建的用户组。

5. **用户登录**并验证权限。

   新创建的用户登录控制台，切换至授权区域，验证权限：

   - 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“数据管理 > 数据集”，如果无法进行创建（当前仅包含开发环境的使用权限），表示仅为ModelArts用户授予开发环境的使用权限已生效。
   - 在“服务列表”中选择除ModelArts，进入ModelArts主界面，单击“开发环境>Notebook>创建”，如果可以成功访问“存储配置”项对应的OBS路径，表示为用户配置的OBS相关权限已生效。

ModelArts 依赖的 OBS 权限自定义策略样例

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

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "obs:bucket:ListAllMybuckets",
        "obs:bucket:HeadBucket",
        "obs:bucket:ListBucket",
        "obs:bucket:GetBucketLocation",
        "obs:object:GetObject",
        "obs:object:GetObjectVersion",
        "obs:object:PutObject",
        "obs:object:DeleteObject",
        "obs:object:DeleteObjectVersion",
        "obs:object:ListMultipartUploadParts",
        "obs:object:AbortMultipartUpload",
        "obs:object:GetObjectAcl",
        "obs:object:GetObjectVersionAcl",
        "obs:bucket:PutBucketAcl",
        "obs:object:PutObjectAcl"
      ],
      "Effect": "Allow"
    }
  ]
}
```

ModelArts 开发环境使用权限的自定义策略样例

```
{
  "Version": "1.1",
  "Statement": [

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



```
    }
  ]
}
```

6 Standard 开发环境

6.1 将 Notebook 的 Conda 环境迁移到 SFS 磁盘

本文介绍了如何将Notebook的Conda环境迁移到SFS磁盘上。这样重启Notebook实例后，Conda环境不会丢失。

步骤如下:

1. **创建新的虚拟环境并保存到SFS目录**
2. **克隆原有的虚拟环境到SFS盘**
3. **重新启动镜像激活SFS盘中的虚拟环境**
4. **保存并共享虚拟环境**

前提条件

创建一个Notebook，“资源类型”选择“专属资源池”，“存储配置”选择“SFS弹性文件服务器”，打开terminal。

创建新的虚拟环境并保存到 SFS 目录

创建新的conda虚拟环境。

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-new-env python=3.7.10 -y
```

查看现有的conda虚拟环境，此时可能出现新创建的虚拟环境的名称为空的情况。

```
# shell
conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         *  /home/ma-user/anaconda3/envs/python-3.7.10
                      /home/ma-user/work/envs/user_conda/sfs-new-env
```

添加新创建的虚拟环境到conda env。

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
```

查看现有的conda虚拟环境，此时新的虚拟环境已经能够正常显示，可以直接通过名称进行虚拟环境的切换。

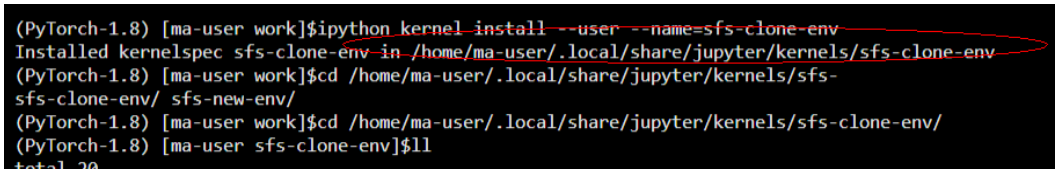
```
# shell
conda env list
conda activate sfs-new-env
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         *  /home/ma-user/anaconda3/envs/python-3.7.10
sfs-new-env              /home/ma-user/work/envs/user_conda/sfs-new-env
```

（可选）将新建的虚拟环境注册到JupyterLab kernel（可以在JupyterLab中直接使用虚拟环境）。

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-new-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-new-env/logo-*
```

说明：此处“.local/share/jupyter/kernels/sfs-new-env”为举例，请以用户实际的安装路径为准。

图 6-1 安装路径回显



刷新JupyterLab页面，可以看到新的kernel。

说明

重启Notebook后kernel需要重新注册。

克隆原有的虚拟环境到 SFS 盘

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-clone-env --clone PyTorch-1.8 -
y
Source:      /home/ma-user/anaconda3/envs/PyTorch-1.8
Destination: /home/ma-user/work/envs/user_conda/sfs-clone-env
Packages: 20
Files: 39687
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate /home/ma-user/work/envs/user_conda/sfs-clone-env
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

查看新创建的clone虚拟环境，如果出现新创建的虚拟环境的名称为空的情况，可以参考**添加新创建到虚拟环境到conda env**。

```
# shell
conda env list
# conda environments:
#
```

```
base                  /home/ma-user/anaconda3
PyTorch-1.8           /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         /home/ma-user/anaconda3/envs/python-3.7.10
sfs-clone-env         /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env         *  /home/ma-user/work/envs/user_conda/sfs-new-env
```

（可选）将新建的虚拟环境注册到JupyterLab kernel（可以在JupyterLab中直接使用虚拟环境）

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-clone-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-clone-env/logo-*
```

说明：此处“.local/share/jupyter/kernels/sfs-clone-env”为举例，请以用户实际的安装路径为准。

刷新JupyterLab页面，可以看到新的kernel。

重新启动镜像激活 SFS 盘中的虚拟环境

方法一，直接使用完整conda env路径。

```
# shell
conda activate /home/ma-user/work/envs/user_conda/sfs-new-env
```

方法二，先添加虚拟环境到conda env，然后使用名称激活。

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
conda activate sfs-new-env
```

方法三，直接使用完成虚拟环境中的python或者pip。

```
# shell
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/pip list
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/python -V
```

保存并共享虚拟环境

将要迁移的虚拟环境打包。

```
# shell
pip install conda-pack
conda pack -n sfs-clone-env -o sfs-clone-env.tar.gz --ignore-editable-packages
Collecting packages...
Packing environment at '/home/ma-user/work/envs/user_conda/sfs-clone-env' to 'sfs-clone-env.tar.gz'
[########################################] | 100% Completed |  3min 33.9s
```

解压到SFS目录。

```
# shell

mkdir /home/ma-user/work/envs/user_conda/sfs-tar-env
tar -zxvf sfs-clone-env.tar.gz -C /home/ma-user/work/envs/user_conda/sfs-tar-env
```

查看现有的conda虚拟环境。

```
# shell
conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
```

```
PyTorch-1.8           *  /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10            /home/ma-user/anaconda3/envs/python-3.7.10
sfs-clone-env            /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env              /home/ma-user/work/envs/user_conda/sfs-new-env
sfs-tar-env              /home/ma-user/work/envs/user_conda/sfs-tar-env
test-env                 /home/ma-user/work/envs/user_conda/test-env
```

7 Standard 模型训练

7.1 使用 ModelArts Standard 自定义算法实现手写数字识别

本文为用户提供如何将本地的自定义算法通过简单的代码适配，实现在ModelArts上进行模型训练与部署的全流程指导。

场景描述

本案例用于指导用户使用PyTorch1.8实现手写数字图像识别，示例采用的数据集为MNIST官方数据集。

通过学习本案例，您可以了解如何在ModelArts平台上训练作业、部署推理模型并预测的完整流程。

操作流程

开始使用如下样例前，请务必按**准备工作**指导完成必要操作。

1. **Step1 准备训练数据**：下载MNIST数据集。
2. **Step2 准备训练文件和推理文件**：编写训练与推理代码。
3. **Step3 创建OBS桶并上传文件**：创建OBS桶和文件夹，并将数据集和训练脚本，推理脚本，推理配置文件上传到OBS中。
4. **Step4 创建训练作业**：进行模型训练。
5. **Step5 推理部署**：训练结束后，将生成的模型导入ModelArts用于创建模型，并将模型部署为在线服务。
6. **Step6 预测结果**：上传一张手写数字图片，发起预测请求获取预测结果。
7. **Step7 清除资源**：运行完成后，停止服务并删除OBS中的数据，避免不必要的扣费。

准备工作

- 已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。
- 配置委托访问授权

ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

a. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

b. 在弹出的“添加授权”窗口中，选择：

- 授权对象类型：IAM子用户
- 委托选择：新增委托
- 权限配置：普通用户
- 权限模板：勾选训练模块和推理模块
- 服务列表：会根据“权限模板”所勾选的内容自动勾选所需的最小权限。

选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

图 7-1 配置委托访问授权

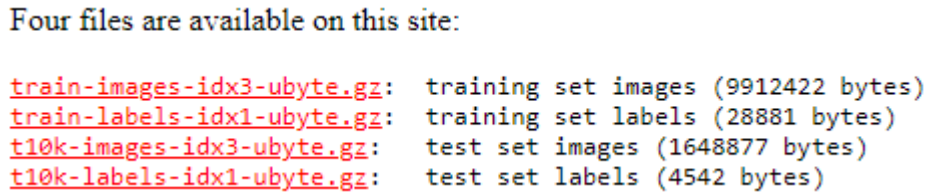
c. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

图 7-2 查看委托配置信息

Step1 准备训练数据

本案例使用的数据是MNIST数据集，您可以在浏览器中搜索“MNIST数据集”下载如图7-3所示的4个文件。

图 7-3 MNIST 数据集



- “train-images-idx3-ubyte.gz”：训练集的压缩包文件，共包含60000个样本。
- “train-labels-idx1-ubyte.gz”：训练集标签的压缩包文件，共包含60000个样本的类别标签。
- “t10k-images-idx3-ubyte.gz”：验证集的压缩包文件，共包含10000个样本。
- “t10k-labels-idx1-ubyte.gz”：验证集标签的压缩包文件，共包含10000个样本的类别标签。

Step2 准备训练文件和推理文件

针对此案例，ModelArts提供了需使用的训练脚本、推理脚本和推理配置文件。请参考如下文件内容。

说明

粘贴“.py”文件代码时，请直接新建“.py”文件，否则会可能出现“SyntaxError: 'gbk' codec can't decode byte 0xa4 in position 324: illegal multibyte sequence”报错。

在本地电脑中创建训练脚本“train.py”，内容如下：

```
# base on https://github.com/pytorch/examples/blob/main/mnist/main.py

from __future__ import print_function

import os
import gzip
import codecs
import argparse
from typing import IO, Union

import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR

import shutil


# 定义网络模型
class Net(nn.Module):
```



```
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


# 模型训练，设置模型为训练模式，加载训练数据，计算损失函数，执行梯度下降
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break


# 模型验证，设置模型为验证模式，加载验证数据，计算损失函数和准确率
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


# 以下为pytorch mnist
# https://github.com/pytorch/vision/blob/v0.9.0/torchvision/datasets/mnist.py
def get_int(b: bytes) -> int:
    return int(codecs.encode(b, 'hex'), 16)
```

```
def open_maybe_compressed_file(path: Union[str, IO]) -> Union[IO, gzip.GzipFile]:
    """Return a file object that possibly decompresses 'path' on the fly.
       Decompression occurs when argument `path` is a string and ends with '.gz' or '.xz'.
    """
    if not isinstance(path, torch._six.string_classes):
        return path
    if path.endswith('.gz'):
        return gzip.open(path, 'rb')
    if path.endswith('.xz'):
        return lzma.open(path, 'rb')
    return open(path, 'rb')


SN3_PASCALVINCENT_TYPEMAP = {
    8: (torch.uint8, np.uint8, np.uint8),
    9: (torch.int8, np.int8, np.int8),
    11: (torch.int16, np.dtype('>i2'), 'i2'),
    12: (torch.int32, np.dtype('>i4'), 'i4'),
    13: (torch.float32, np.dtype('>f4'), 'f4'),
    14: (torch.float64, np.dtype('>f8'), 'f8')
}


def read_sn3_pascalvincent_tensor(path: Union[str, IO], strict: bool = True) -> torch.Tensor:
    """Read an SN3 file in "Pascal Vincent" format (Lush file 'libidx/idx-io.lsh').
       Argument may be a filename, compressed filename, or file object.
    """
    # read
    with open_maybe_compressed_file(path) as f:
        data = f.read()
    # parse
    magic = get_int(data[0:4])
    nd = magic % 256
    ty = magic // 256
    assert 1 <= nd <= 3
    assert 8 <= ty <= 14
    m = SN3_PASCALVINCENT_TYPEMAP[ty]
    s = [get_int(data[4 * (i + 1): 4 * (i + 2)]) for i in range(nd)]
    parsed = np.frombuffer(data, dtype=m[1], offset=(4 * (nd + 1)))
    assert parsed.shape[0] == np.prod(s) or not strict
    return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)


def read_label_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 1)
    return x.long()


def read_image_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 3)
    return x


def extract_archive(from_path, to_path):
    to_path = os.path.join(to_path, os.path.splitext(os.path.basename(from_path))[0])
    with open(to_path, "wb") as out_f, gzip.GzipFile(from_path) as zip_f:
        out_f.write(zip_f.read())
# --- 以上为pytorch mnist
# --- end


# raw mnist 数据处理
def convert_raw_mnist_dataset_to_pytorch_mnist_dataset(data_url):
```

```
    """
    raw

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz

    processed

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz
        MNIST/raw
            train-images-idx3-ubyte
            train-labels-idx1-ubyte
            t10k-images-idx3-ubyte
            t10k-labels-idx1-ubyte
        MNIST/processed
            training.pt
            test.pt
    """
    resources = [
        "train-images-idx3-ubyte.gz",
        "train-labels-idx1-ubyte.gz",
        "t10k-images-idx3-ubyte.gz",
        "t10k-labels-idx1-ubyte.gz"
    ]

    pytorch_mnist_dataset = os.path.join(data_url, 'MNIST')

    raw_folder = os.path.join(pytorch_mnist_dataset, 'raw')
    processed_folder = os.path.join(pytorch_mnist_dataset, 'processed')

    os.makedirs(raw_folder, exist_ok=True)
    os.makedirs(processed_folder, exist_ok=True)

    print('Processing...')

    for f in resources:
        extract_archive(os.path.join(data_url, f), raw_folder)

    training_set = (
        read_image_file(os.path.join(raw_folder, 'train-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 'train-labels-idx1-ubyte'))
    )
    test_set = (
        read_image_file(os.path.join(raw_folder, 't10k-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 't10k-labels-idx1-ubyte'))
    )
    with open(os.path.join(processed_folder, 'training.pt'), 'wb') as f:
        torch.save(training_set, f)
    with open(os.path.join(processed_folder, 'test.pt'), 'wb') as f:
        torch.save(test_set, f)

    print('Done!')


def main():
    # 定义可以接收的训练作业运行参数
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')

    parser.add_argument('--data_url', type=str, default=False,
                        help='mnist dataset path')
    parser.add_argument('--train_url', type=str, default=False,
                        help='mnist model path')
```

```
parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                    help='input batch size for training (default: 64)')
parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                    help='input batch size for testing (default: 1000)')
parser.add_argument('--epochs', type=int, default=14, metavar='N',
                    help='number of epochs to train (default: 14)')
parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                    help='learning rate (default: 1.0)')
parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                    help='Learning rate step gamma (default: 0.7)')
parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')
parser.add_argument('--dry-run', action='store_true', default=False,
                    help='quickly check a single pass')
parser.add_argument('--seed', type=int, default=1, metavar='S',
                    help='random seed (default: 1)')
parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                    help='how many batches to wait before logging training status')
parser.add_argument('--save-model', action='store_true', default=True,
                    help='For Saving the current Model')
args = parser.parse_args()

use_cuda = not args.no_cuda and torch.cuda.is_available()

torch.manual_seed(args.seed)

# 设置使用 GPU 还是 CPU 来运行算法
device = torch.device("cuda" if use_cuda else "cpu")

train_kwargs = {'batch_size': args.batch_size}
test_kwargs = {'batch_size': args.test_batch_size}
if use_cuda:
    cuda_kwargs = {'num_workers': 1,
                   'pin_memory': True,
                   'shuffle': True}
    train_kwargs.update(cuda_kwargs)
    test_kwargs.update(cuda_kwargs)

# 定义数据预处理方法
transform=transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
    ])

# 将 raw mnist 数据集转换为 pytorch mnist 数据集
convert_raw_mnist_dataset_to_pytorch_mnist_dataset(args.data_url)

# 分别创建训练和验证数据集
dataset1 = datasets.MNIST(args.data_url, train=True, download=False,
                    transform=transform)
dataset2 = datasets.MNIST(args.data_url, train=False, download=False,
                    transform=transform)

# 分别构建训练和验证数据迭代器
train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

# 初始化神经网络模型并复制模型到计算设备上
model = Net().to(device)
# 定义训练优化器和学习率策略，用于梯度下降计算
optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

# 训练神经网络，每一轮进行一次验证
for epoch in range(1, args.epochs + 1):
    train(args, model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
    scheduler.step()
```

```
    # 保存模型与适配 ModelArts 推理模型包规范
    if args.save_model:

        # 在 train_url 训练参数对应的路径内创建 model 目录
        model_path = os.path.join(args.train_url, 'model')
        os.makedirs(model_path, exist_ok = True)

        # 按 ModelArts 推理模型包规范，保存模型到 model 目录内
        torch.save(model.state_dict(), os.path.join(model_path, 'mnist_cnn.pt'))

        # 复制推理代码与配置文件到 model 目录内
        the_path_of_current_file = os.path.dirname(__file__)
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/customize_service.py'),
os.path.join(model_path, 'customize_service.py'))
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/config.json'), os.path.join(model_path,
'config.json'))

if __name__ == '__main__':
    main()
```

在本地电脑中创建推理脚本“customize_service.py”，内容如下：

```
import os
import log
import json

import torch.nn.functional as F
import torch.nn as nn
import torch
import torchvision.transforms as transforms

import numpy as np
from PIL import Image

from model_service.pytorch_model_service import PTServingBaseService

logger = log.getLogger(__name__)

# 定义模型预处理
infer_transformation = transforms.Compose([
    transforms.Resize(28),
    transforms.CenterCrop(28),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# 模型推理服务
class PTVisionService(PTServingBaseService):

    def __init__(self, model_name, model_path):
        # 调用父类构造方法
        super(PTVisionService, self).__init__(model_name, model_path)

        # 调用自定义函数加载模型
        self.model = Mnist(model_path)

        # 加载标签
        self.label = [0,1,2,3,4,5,6,7,8,9]

    # 接收request数据，并转换为模型可以接受的输入格式
    def _preprocess(self, data):
        preprocessed_data = {}
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # 灰度处理
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
```

```
                    input_batch.append(infer_transformation(image1).cuda())
                else:
                    input_batch.append(infer_transformation(image1))
            input_batch_var = torch.autograd.Variable(torch.stack(input_batch, dim=0), volatile=True)
            print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var

        return preprocessed_data

    # 将推理的结果进行后处理，得到预期的输出格式，该结果就是最终的返回值
    def _postprocess(self, data):
        results = []
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

    # 对于输入数据进行前向推理，得到推理结果
    def _inference(self, data):

        result = {}
        for k, v in data.items():
            result[k] = self.model(v)

        return result

# 定义网络
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


def Mnist(model_path, **kwargs):
    # 生成网络
    model = Net()

    # 加载模型
    if torch.cuda.is_available():
        device = torch.device('cuda')
        model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
    else:
        device = torch.device('cpu')
        model.load_state_dict(torch.load(model_path, map_location=device))

    # CPU 或者 GPU 映射
    model.to(device)
```

```
    # 声明为推理模式
    model.eval()

    return model
```

在本地电脑中推理配置文件“config.json”，内容如下：

```
{
    "model_algorithm": "image_classification",
    "model_type": "PyTorch",
    "runtime": "pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64"
}
```

Step3 创建 OBS 桶并上传文件

将上一步中的数据和代码文件、推理代码文件与推理配置文件，从本地上传到OBS桶中。在ModelArts上运行训练作业时，需要从OBS桶中读取数据和代码文件。

1. 登录**OBS管理控制台**，按照如下示例创建OBS桶和文件夹。

```
{OBS桶}                  # OBS对象桶，用户可以自定义名称，例如：test-modelarts-xx
    -{OBS文件夹}          # OBS文件夹，自定义名称，此处举例为pytorch
        - mnist-data     # OBS文件夹，用于存放训练数据集，可以自定义名称，此处举例为mnist-data
        - mnist-code     # OBS文件夹，用于存放训练脚本train.py，可以自定义名称，此处举例为mnist-code
            - infer      # OBS文件夹，用于存放推理脚本customize_service.py和配置文件config.json
        - mnist-output   # OBS文件夹，用于存放训练输出模型，可以自定义名称，此处举例为mnist-output
```

注意

- 创建的OBS桶所在区域和后续使用ModelArts必须在同一个区域Region，否则会导致训练时找不到OBS桶。具体操作可参见**查看OBS桶与ModelArts是否在同一区域**。
- 创建OBS桶时，桶的存储类别请勿选择“归档存储”，归档存储的OBS桶会导致模型训练失败。

2. 上传**Step1 准备训练数据**中下载的MNIST数据集压缩包文件到OBS的“mnist-data”文件夹中。

注意

- 上传数据到OBS中时，请不要加密，否则会导致训练失败。
- 文件无需解压，直接上传压缩包至OBS中即可。

3. 上传训练脚本“train.py”到“mnist-code”文件夹中。
4. 上传推理脚本“customize_service.py”和推理配置文件“config.json”到“mnist-code”的“infer”文件中。

Step4 创建训练作业

1. 登录**ModelArts管理控制台**，选择和OBS桶相同的区域。
2. 在“权限管理”中检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。

3. 在左侧导航栏选择“模型训练 > 训练作业”进入训练作业页面，单击“创建训练作业”。
4. 填写创建训练作业相关信息。
   - “创建方式”：选择“自定义算法”。
   - “启动方式”：选择“预置框架”，引擎及版本下拉框中选择PyTorch，pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64。
   - “代码目录”：选择已创建的OBS代码目录路径，例如“/test-modelarts-xx/pytorch/mnist-code/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “启动文件”：选择代码目录下上传的训练脚本“train.py”。
   - “输入”：单击“增加训练输入”，设置训练输入的“参数名称”为“data_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-xx/pytorch/mnist-data/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “输出”：单击“增加训练输出”，设置训练输出的“参数名称”为“train_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-xx/pytorch/mnist-output/”（test-modelarts-xx需替换为您的OBS桶名称）。预下载至本地目录选择“否”。
   - “资源类型”：选择GPU单卡的规格。
   - 其他参数保持默认即可。

     说明

     本样例代码为单机单卡场景，选择多卡规格会导致训练失败。
5. 单击“提交”，确认训练作业的参数信息，确认无误后单击“确定”。

   页面自动返回“训练作业”列表页，当训练作业状态变为“已完成”时，即完成了模型训练过程。

   说明

   本案例的训练作业预计运行十分钟。
6. 单击训练作业名称，进入作业详情界面查看训练作业日志信息，观察日志是否有明显的Error信息，如果有则表示训练失败，请根据日志提示定位原因并解决。
7. 在训练详情页左下方单击训练输出路径，如图7-4所示，跳转到OBS目录，查看是否存在model文件夹，且model文件夹中是否有生成训练模型。如果未生成model文件夹或者训练模型，可能是训练输入数据不完整导致，请检查训练数据上传是否完整，并重新训练。

图 7-4 训练输出路径

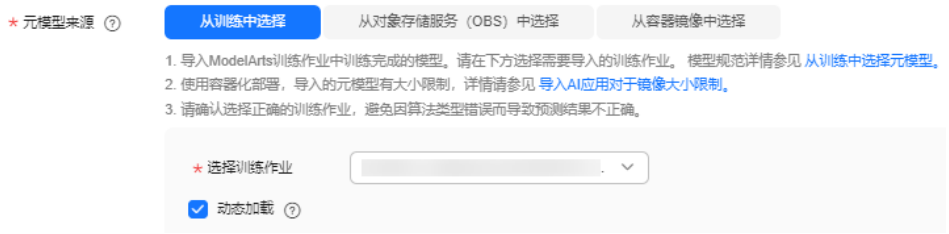
Step5 推理部署

模型训练完成后，可以创建模型，将模型部署为在线服务。

1. 登录**ModelArts管理控制台**，单击左侧导航栏中的“模型管理”，进入“自定义模型”页面，单击“创建模型”。
2. 在“创建模型”页面，填写相关参数，然后单击“立即创建”。

   在“元模型来源”中，选择“从训练中选择”页签，选择**Step4 创建训练作业**中完成的训练作业，勾选“动态加载”。AI引擎的值是系统自动写入的，无需设置。

   图 7-5 设置元模型来源

3. 在模型列表页面，当模型状态变为“正常”时，表示模型创建成功。单击模型操作列的“部署”，弹出“版本列表”，单击操作列“部署>在线服务”，将模型部署为在线服务。

   图 7-6 部署在线服务

4. 在“部署”页面，参考下图填写参数，然后根据界面提示完成在线服务创建。本案例适用于CPU规格，节点规格需选择CPU。如果有免费CPU规格，可选择免费规格进行部署（每名用户限部署一个免费的在线服务，如果您已经部署了一个免费在线服务，需要先将其删除才能部署新的免费在线服务）。其他参数保持默认即可。确认后单击“提交”。

图 7-7 部署模型

完成服务部署后，返回在线服务页面列表页，等待服务部署完成，当服务状态显示为“运行中”，表示服务已部署成功。

Step6 预测结果

1. 在“在线服务”页面，单击在线服务名称，进入服务详情页面。
2. 单击“预测”页签，请求类型选择“multipart/form-data”，请求参数填写“image”，单击“上传”按钮上传示例图片，然后单击“预测”。

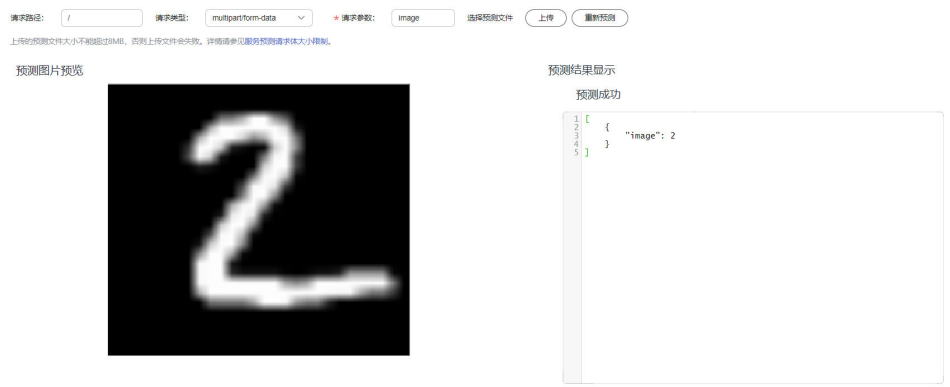
预测完成后，预测结果显示区域将展示预测结果，根据预测结果内容，可识别出此图片的数字是“2”。

说明

本案例中使用的MNIST是比较简单的用做demo的数据集，配套算法也是比较简单的用于教学的神经网络算法。这样的数据和算法生成的模型仅适用于教学模式，并不能应对复杂的预测场景。即生成的模型对预测图片有一定范围和要求，预测图片必须和训练集中的图片相似（黑底白字）才可能预测准确。

图 7-8 示例图片

图 7-9 预测结果展示



Step7 清除资源

如果不再需要使用此模型及在线服务，建议清除相关资源，避免产生不必要的费用。

- 在“在线服务”页面，“停止”或“删除”刚创建的在线服务。
- 在“自定义模型”页面，“删除”刚创建的模型。
- 在“训练作业”页面，“删除”运行结束的训练作业。
- 进入OBS，删除本示例使用的OBS桶及文件夹，以及文件夹的文件。

常见问题

- 训练作业一直在等待中（排队）？

  训练作业状态一直在等待中状态表示当前所选的资源池规格资源紧张，作业需要进行排队，请耐心等待。请参考**训练作业一直在等待中（排队）？** 。
- 在ModelArts中选择OBS路径时，找不到已创建的OBS桶?

  请确保创建的桶和ModelArts服务在同一区域，详细操作请参考**查看OBS桶与ModelArts是否在同一个区域**。

7.2 基于 ModelArts Standard 运行训练作业

7.2.1 在 ModelArts Standard 上运行训练作业的场景介绍

不同AI模型训练所需要的数据量和算力不同，在训练时选择合适的存储及训练方案可提升模型训练效率与资源性价比。ModelArts Standard支持单机单卡、单机多卡和多机多卡的训练场景，满足不同AI模型训练的要求。

ModelArts Standard提供了公共资源池和专属资源池，专属资源池不与其他用户共享资源，更加高效。针对企业多用户场景，推荐使用专属资源池开展AI模型训练。

本文提供了端到端案例指导，帮助您快速了解如何在ModelArts Standard上选择合适的训练方案并进行模型训练。

针对不同的数据量和算法情况，推荐以下训练方案：

- 单机单卡：小数据量（1G训练数据）、低算力场景（1卡Vnt1），存储方案推荐使用“OBS的并行文件系统（存放数据和代码）”。
- 单机多卡：中等数据量（50G左右训练数据）、中等算力场景（8卡Vnt1），存储方案推荐使用“SFS（存放数据和代码）”。
- 多机多卡：大数据量（1T训练数据）、高算力场景（4台8卡Vnt1），存储方案推荐使用“SFS（存放数据）+普通OBS桶（存放代码）”，采用分布式训练。

表 7-1 不同场景所需服务及购买推荐

场景	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
单机单卡	按需购买（并行文件系统）	×	免费	免费	包月购买	免费	×	按需购买

场景	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
单机多卡	×	包月购买（HPC型 500G）	免费	免费	包月购买	免费	包月购买（Ubuntu 18.04，建议不小于2U8G，本地存储空间100G，带EIP全动态BGP，按流量10M带宽）	×
多机多卡	按需购买（普通OBS桶）	包月购买（HPC型 500G）	免费	免费	包月购买	免费	包月购买（Ubuntu 18.04，建议不小于2U8G，本地存储空间100G，带EIP全动态BGP，按流量10M带宽）	×

表 7-2 开源数据集训练效率参考

算法及数据	资源规格	Epoch数	预计运行时长（hh:mm:ss）
算法：PyTorch官方针对ImageNet的样例 数据：ImageNet分类数据子集	1机1卡Vnt1	10	0:05:03
算法：YOLOX 数据：COCO2017	1机1卡Vnt1	10	03:33:13
	1机8卡Vnt1	10	01:11:48
	4机8卡Vnt1	10	0:36:17

算法及数据	资源规格	Epoch数	预计运行时长（hh:mm:ss）
算法：Swin-Transformer 数据：ImageNet21K	1机1卡Vnt1	10	197:25:03
	1机8卡Vnt1	10	26:10:25
	4机8卡Vnt1	10	07:08:44

表 7-3 训练各步骤性能参考

步骤	说明	预计时长
镜像下载	首次下载镜像的时间（25G）。	8分钟
资源调度	点创建训练作业开始到变成运行中的时间（资源充足、镜像已缓存）。	20秒
训练列表页打开	已有50条训练作业，单击训练模块后的时间。	6秒
日志加载	作业运行中，已经输出1兆的日志文本，单击训练详情页面需要多久加载出日志。	2.5秒
训练详情页	作业运行中，没有用户日志情况下，在ModelArts控制台主页面单击训练详情页面后加载页面内容。	2.5秒
JupyterLab页面	进入JupyterLab页面后加载页面内容。	0.5秒
Notebook列表页	已有50个Notebook实例，在ModelArts控制台主页面单击开发环境后的时间。	4.5秒

说明

镜像下载时间受节点规格、节点硬盘类型（高IO/普通IO）、是否SSD等因素影响，以上数据仅供参考。

7.2.2 在 ModelArts Standard 运行训练作业的准备工作

使用ModelArts Standard的专属资源池训练时，需要完成以下准备工作。

购买服务资源

表 7-4 购买服务资源

服务	使用说明	参考文档
弹性文件服务 SFS	弹性文件服务默认为按需计费，即按购买的存储容量和时长收费。您也可以购买包年包月套餐，提前规划资源的使用额度和时长。在欠费时，您需要及时（15天之内）续费以避免您的文件系统资源被清空。 购买的SFS可以用于存储数据和代码。	如何购买弹性文件服务?
容器镜像服务 SWR	容器镜像服务分为企业版和共享版。共享版计费项包括存储空间和流量费用，目前均免费提供给您。企业版支持按需计费模式。 购买的SWR可以用于上传自定义镜像。	上传镜像
对象存储服务 OBS	对象存储服务提供按需计费和包年包月两种计费模式，用户可以根据实际需求购买OBS服务。 OBS服务支持以下两种存储方式，单机单卡场景使用文件系统，多机多卡场景使用普通OBS桶。	• 创建普通OBS桶 • 创建并行文件系统
虚拟私有云 VPC	虚拟私有云可以为您构建隔离的、用户自主配置和管理的虚拟网络环境。 通过打通专属资源池的VPC，可以方便用户跨VPC使用资源，提升资源利用率。	创建虚拟私有云和子网
弹性云服务器 ECS	如果您需要在服务器上部署相关业务，较之物理服务器，弹性云服务器的创建成本较低，并且可以在几分钟之内快速获得基于云服务平台的弹性云服务器设施，并且这些基础设施是弹性的，可以根据需求伸缩。 购买的ECS服务可以用于挂载SFS Turbo存储。 **说明** 购买时需注意，ECS需要和SFS买到同一个VPC才能挂载SFS存储。	自定义购买ECS
密码安全中心 DEW	在使用Notebook进行代码调试时，如果要开启“SSH远程开发”功能，需要选择密钥对，便于用户登录弹性云服务器时使用密钥对方式进行身份认证，提升通信安全。密钥对可免费创建。	如何创建密钥对?

配置权限

步骤1 配置IAM权限。

1. 使用华为云主账号创建一个开发者用户组user_group，将开发者账号加入用户组user_group中。具体操作请参见**Step1 创建用户组并加入用户**。
2. 创建自定义策略。

a. 使用华为云主账号登录**华为云管理控制台**，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

b. 在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，“策略名称”为“Policy1”或“Policy2”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

说明

创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。**了解更多**。

- 项目级云服务的自定义策略“Policy1”的具体内容如下，可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "modelarts:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "modelarts:pool:create",
                "modelarts:pool:update",
                "modelarts:pool:delete"
            ],
            "Effect": "Deny"
        },
        {
            "Action": [
                "sfsturbo:*:*",
                "vpc:*:*",
                "dss:*:get",
                "dss:*:list"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "ecs:*:*",
                "evs:*:get",
                "evs:*:list",
                "evs:volumes:create",
                "evs:volumes:delete",
                "evs:volumes:attach",
                "evs:volumes:detach",
                "evs:volumes:manage",
                "evs:volumes:update",
                "evs:volumes:use",
                "evs:volumes:uploadImage",
                "evs:snapshots:create",
                "vpc:*:get",
                "vpc:*:list",
                "vpc:networks:create",
                "vpc:networks:update",
                "vpc:subnets:update",
                "vpc:subnets:create",
                "vpc:ports:*",
                "vpc:routers:get",
                "vpc:routers:update",
                "vpc:securityGroups:*",
                "vpc:securityGroupRules:*",
                "vpc:floatingIps:*",
                "vpc:publicIps:*",
                "ims:images:create",
```

```
            "ims:images:delete",
            "ims:images:get",
            "ims:images:list",
            "ims:images:update",
            "ims:images:upload"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "vpc:*:*",
            "ecs:*:get*",
            "ecs:*:list*"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "kms:cmk:*",
            "kms:dek:*",
            "kms:grant:*",
            "kms:cmkTag:*",
            "kms:partition:*"
        ],
        "Effect": "Allow"
    }
  ]
}
```

- 全局级云服务的自定义策略“Policy2”的具体内容如下，可以直接复制粘贴。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "obs:bucket:ListAllMybuckets",
        "obs:bucket:HeadBucket",
        "obs:bucket:ListBucket",
        "obs:bucket:GetBucketLocation",
        "obs:object:GetObject",
        "obs:object:GetObjectVersion",
        "obs:object:PutObject",
        "obs:object:DeleteObject",
        "obs:object:DeleteObjectVersion",
        "obs:object:ListMultipartUploadParts",
        "obs:object:AbortMultipartUpload",
        "obs:object:GetObjectAcl",
        "obs:object:GetObjectVersionAcl"
      ],
      "Effect": "Allow"
    }
  ]
}
```

3. 将自定义策略授权给开发者用户组user_group。
   a. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称user_group操作列的“授权”，勾选策略“Policy1”、“Policy2”、“SWR Admin”。单击“下一步”。

      说明

      SWR的权限有SWR FullAccess、SWR OperateAccess、SWR ReadOnlyAccess。但SWR FullAccess、SWR OperateAccess、SWR ReadOnlyAccess仅限容器镜像服务企业版使用，目前企业版已暂停公测。非企业版用户暂不支持使用此权限。因此需要在此勾选“SWR Admin” 策略。

b. 设置最小授权范围，选择授权范围方案为“所有资源”，单击“确定”。

更多权限管理的信息请参见**ModelArts权限管理基本概念**。

步骤2 配置ModelArts委托权限。

给用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。
2. 在弹出的“添加授权”窗口中，选择：
   - 授权对象类型：IAM子用户
   - 委托选择：新增委托
   - 权限配置：普通用户
   - 权限模板：勾选训练模块和推理模块
   - 服务列表：会根据“权限模板”所勾选的内容自动勾选所需的最小权限。

   选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

图 7-10 配置委托访问授权

3. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

图 7-11 查看委托配置信息

步骤3 配置SWR组织权限。

IAM用户创建后，需要管理员在组织中为用户添加授权，使IAM用户对组织内所有镜像享有读取/编辑/管理的权限。

只有具备“管理”权限的账号和IAM用户才能添加授权。

1. 登录**容器镜像服务控制台**。
2. 在左侧菜单栏选择“组织管理”，单击组织名称。
3. 在“用户”页签下单击“添加授权”，在弹出的窗口中为IAM用户选择权限，然后单击“确定”。

SWR授权管理详情可参考**授权管理**。

说明

如果给子账号的SWR授权不是SWR Admin权限，则需要继续配置SWR组织权限。

步骤4 测试用户权限。

由于权限配置需要等待15-30分钟生效，建议在配置完成后，等待30分钟，再执行如下验证操作。

1. 使用用户组02中任意一个子账号登录**ModelArts管理控制台**。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 验证ModelArts权限。
   a. 在左上角的服务列表中，选择ModelArts服务，进入**ModelArts管理控制台**。
   b. 在ModelArts管理控制台，可正常创建Notebook、训练作业、注册镜像。
3. 验证SFS权限。
   a. 在左上角的服务列表中，选择SFS服务，进入**SFS管理控制台**。
   b. 在SFS管理控制台的SFS Turbo中单击右上角的“创建文件系统”，如果能正常打开页面，表示当前用户具备SFS的操作权限。
4. 验证ECS权限。
   a. 在左上角的服务列表中，选择ECS服务，进入**ECS管理控制台**。
   b. 在ECS管理控制台，单击右上角的“购买弹性云服务器”，如果能正常打开页面，表示当前用户具备ECS的操作权限。
5. 验证VPC权限。
   a. 在左上角的服务列表中，选择VPC服务，进入**VPC管理控制台**。
   b. 在VPC管理控制台，单击右上角的“创建虚拟私有云”，如果能正常打开页面，表示当前用户具备VPC的操作权限。
6. 验证DEW权限。
   a. 在左上角的服务列表中，选择DEW服务，进入**DEW管理控制台**。
   b. 在DEW管理控制台，选择“密钥对管理 > 私有密钥对”，单击“创建密钥对”，如果能正常打开页面，表示当前用户具备DEW的操作权限。
7. 验证OBS权限。
   a. 在左上角的服务列表中，选择OBS服务，进入**OBS管理控制台**。
   b. 在OBS管理控制台，单击右上角的“创建桶”，如果能正常打开页面，表示当前用户具备OBS的操作权限。

8. 验证SWR权限。
   a. 在左上角的服务列表中，选择SWR服务，进入**容器镜像服务控制台**。
   b. 在SWR管理控制台，如果能正常打开页面，表示当前用户具备SWR的操作权限。
   c. 单击右上角的“上传镜像”，如果能看到授权的组织，表示当前用户具备SWR组织权限。

----结束

创建专属资源池

ModelArts提供独享的计算资源，可用于Notebook、训练作业、部署模型。专属资源池不与其他用户共享，更加高效。在使用专属资源池之前，您需要先创建一个专属资源池，操作指导请参考**创建Standard专属资源池**。

- 配置“网络”时需要选择已打通VPC的网络。如果需要新建网络和打通VPC可以参考**配置Standard专属资源池可访问公网**。
- “规格类型”和“节点数量”根据训练计划使用的资源选择。

在 ECS 服务器挂载 SFS Turbo 存储

在ECS服务器挂载SFS Turbo存储后，支持将训练所需的数据通过ECS上传至SFS Turbo。

1. 检查云服务环境。
   - ECS服务器和SFS的共享硬盘在相同的VPC或者对应VPC能够互联。
   - ECS服务器基础镜像用的是Ubuntu 18.04。
   - ECS服务器和SFS Turbo在同一子网中。
2. 在ECS服务器中设置华为云镜像源。
   ```
   sudo sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
   sudo sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
   ```
3. 安装NFS客户端，挂载对应盘。
   ```
   sudo apt-get update
   sudo apt-get install nfs-common
   ```
4. 获取SFS Turbo的挂载命令。
   a. 进入弹性文件服务**SFS管理控制台**。
   b. 选择“SFS Turbo”进入文件系统列表，单击文件系统名称，进入详情页面。
   c. 在“基本信息”页签获取并记录“Linux挂载命令”。
5. 在ECS服务器中挂载NFS存储。
   确认对应目录存在后，输入对应指令，命令如下。
   ```
   mkdir -p /mnt/sfs_turbo
   mount -t nfs -o vers=3,nolock 192.168.0.169:/ /mnt/sfs_turbo
   ```

在 ECS 中设置 ModelArts 用户可读权限

在ModelArts训练平台使用自定义镜像时，默认用户为ma-user、默认用户组为ma-group。如果在训练时调用ECS中的文件，需要修改文件权限改为ma-user可读，否则会出现Permission denied错误。

1. 在Terminal中执行以下命令，在ECS中提前创建好ma-user和ma-group。

```
default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
    groupdel -f ${default_group}; \
    groupadd -g 100 ma-group; \
fi && \
if [ -z ${default_group} ]; then \
    groupadd -g 100 ma-group; \
fi && \
if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
    userdel -r ${default_user}; \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
fi && \
if [ -z ${default_user} ]; then \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
fi && \
# set bash as default
rm /bin/sh && ln -s /bin/bash /bin/sh
```

2. 执行以下命令，查看创建的用户信息。

```
id ma-user
```

如果出现以下信息则表示创建成功。

```
uid=1000(ma-user) gid=100(ma-group) groups=100(ma-group)
```

安装和配置 OBS 命令行工具

obsutil是用于访问、管理对象存储服务OBS的命令行工具，使用该工具可以对OBS进行常用的配置管理操作，如创建桶、上传文件/文件夹、下载文件/文件夹、删除文件/文件夹等。

obsutil安装和配置的具体操作指导请参见**obsutils快速入门**。

须知

操作命令中的AK/SK和Endpoint要替换为用户实际获取的AK/SK和Endpoint。

（可选）工作空间配置

ModelArts支持设置子账号的细粒度权限、不同工作空间之间资源隔离。ModelArts工作空间帮您实现项目资源隔离、多项目分开结算等功能。

如果您开通了企业项目管理服务的权限，可以在创建工作空间的时候绑定企业项目ID，并在企业项目下添加用户组，为不同的用户组设置细粒度权限供组里的用户使用。

如果您未开通企业项目管理服务的权限，也可以在ModelArts创建自己独立的工作空间，但是无法使用跟企业项目相关的功能。

7.2.3 在 ModelArts Standard 上运行单机单卡训练作业

操作流程

1. 准备工作
   a. **购买服务资源**（OBS和SWR）

b. **配置权限**
c. **创建专属资源池**（不需要打通VPC）
d. **安装和配置OBS命令行工具**
e. **（可选）工作空间配置**

2. 模型训练
a. **本地构建镜像及调试**
b. **上传镜像**
c. **上传数据和算法到OBS**
d. **使用Notebook进行代码调试**
e. **创建单机单卡训练作业**
f. **监控资源**

本地构建镜像及调试

本节通过打包conda env来构建环境，也可以通过pip install、conda install等方式安装conda环境依赖。

说明

- 容器镜像的大小建议小于15G，详细的自定义镜像规范要求请参见**训练作业自定义镜像规范**。
- 建议通过开源的官方镜像来构建，例如PyTorch的官方镜像。
- 建议容器分层构建，单层容量不要超过1G、文件数不大于10w个。分层时，先构建不常变化的层，例如：先OS，再cuda驱动，再Python，再pytorch，再其他依赖包。
- 如果训练数据和代码经常变动，则不建议把数据、代码放到容器镜像里，避免频繁地构建容器镜像。
- 容器已经能满足隔离需求，不建议在容器内再创建多个conda env。

1. 导出conda环境。
a. 启动线下的容器镜像：
```
# run on terminal
docker run -ti ${your_image:tag}
```
b. 在容器中输入如下命令，得到pytorch.tar.gz：
```
# run on container

# 基于想要迁移的base环境创建一个名为pytorch的conda环境
conda create --name pytorch --clone base

pip install conda-pack

#将pytorch env打包生成pytorch.tar.gz
conda pack -n pytorch -o pytorch.tar.gz
```
c. 将打包好的压缩包传到本地：
```
# run on terminal
docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
```
d. 将pytorch.tar.gz上传到OBS并**设置公共读**，并在构建时使用**wget**命令获取、解压、清理。

2. 构建新镜像。

基础镜像一般选用“ubuntu 18.04”的官方镜像，或者nvidia官方提供的带cuda驱动的镜像。相关镜像直接到dockerhub官网查找即可。

构建流程：安装所需的apt包、驱动，配置ma-user用户、导入conda环境、配置Notebook依赖。

说明

- 推荐使用Dockerfile的方式构建镜像。这样既满足dockerfile可追溯及构建归档的需求，也保证镜像内容无冗余和残留。
- 每层构建的时候都尽量把tar包等中间态文件删除，保证最终镜像更小，清理缓存的方法可参考：**conda clean**。

3. 构建参考样例

   Dockerfile样例：

```
FROM nvidia/cuda:11.3.1-cudnn8-devel-ubuntu18.04

USER root

# section1: add user ma-user whose uid is 1000 and user group ma-group whose gid is 100. If there already exists 1000:100 but not ma-user:ma-group, below code will remove it
RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
        groupadd -g 100 ma-group; \
    fi && \
    if [ -z ${default_group} ]; then \
        groupadd -g 100 ma-group; \
    fi && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    if [ -z ${default_user} ]; then \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    # set bash as default
    rm /bin/sh && ln -s /bin/bash /bin/sh

# section2: config apt source and install tools needed.
RUN sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y ca-certificates curl ffmpeg git libgl1-mesa-glx libglib2.0-0 libibverbs-dev libjpeg-dev libpng-dev libsm6 libxext6 libxrender-dev ninja-build screen sudo vim wget zip && \
    apt-get clean  && \
    rm -rf /var/lib/apt/lists/*

USER ma-user

# section3: install miniconda and rebuild conda env
RUN mkdir -p /home/ma-user/work/ && cd /home/ma-user/work/ && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    chmod 777 Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    bash Miniconda3-py37_4.12.0-Linux-x86_64.sh -bfp /home/ma-user/anaconda3 && \
    wget https://${bucketname}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz && \
    mkdir -p /home/ma-user/anaconda3/envs/pytorch && \
    tar -xzf pytorch.tar.gz -C /home/ma-user/anaconda3/envs/pytorch && \
    source /home/ma-user/anaconda3/envs/pytorch/bin/activate && conda-unpack && \
    /home/ma-user/anaconda3/bin/conda init bash && \
    rm -rf /home/ma-user/work/*

ENV PATH=/home/ma-user/anaconda3/envs/pytorch/bin:$PATH

# section4: settings of Jupyter Notebook for pytorch env
RUN source /home/ma-user/anaconda3/envs/pytorch/bin/activate && \
    pip install ipykernel==6.7.0 --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/repository/pypi/simple && \
    ipython kernel install --user --env PATH /home/ma-user/anaconda3/envs/pytorch/bin:$PATH --name=pytorch && \
    rm -rf /home/ma-user/.local/share/jupyter/kernels/pytorch/logo-* && \
```

```
    rm -rf ~/.cache/pip/* && \
    echo 'export PATH=$PATH:/home/ma-user/.local/bin' >> /home/ma-user/.bashrc && \
    echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/nvidia/lib64' >> /home/ma-user/.bashrc && \
    echo 'conda activate pytorch' >> /home/ma-user/.bashrc

ENV DEFAULT_CONDA_ENV_NAME=pytorch
```

说明

Dockerfile中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz"，需要替换为**1**中pytorch.tar.gz在OBS上的路径（需将文件设置为公共读）。

进入Dockerfile目录，通过Dockerfile构建镜像命令：

```
# cd 到Dockerfile所在目录下，输入构建命令
# docker build -t ${image_name}:${image_version} .
# 例如
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

4. 调试镜像

说明

建议把调试过程中的修改点通过Dockerfile固化到容器构建正式流程，并重新测试。

a. 确认对应的脚本、代码、流程在linux服务器上运行正常。

如果在linux服务器上运行就有问题，那么先调通以后再做容器镜像。

b. 确认打入镜像的文件是否在正确的位置、是否有正确的权限。

训练场景主要查看自研的依赖包是否正常，查看pip list是否包含所需的包，查看容器直接调用的python是否是自己所需要的那个（如果容器镜像装了多个python，需要设置python路径的环境变量）。

c. 测试训练启动脚本。

i. 优先使用手工进行数据复制的工作并验证

一般在镜像里不包含训练所用的数据和代码，所以在启动镜像以后需要手工把需要的文件复制进去。建议数据、代码和中间数据都放到"/cache"目录，防止正式运行时磁盘占满。建议linux服务器申请的时候，有足够大的内存（8G以上）以及足够大的硬盘（100G以上）。

docker和linux的文件交互命令如下：

```
docker cp data/ 39c9ceedb1f6:/cache/
```

数据准备完成后，启动训练的脚本，查看训练是否能够正常拉起。一般来说，启动脚本为：

```
cd /cache/code/
python start_train.py
```

如果训练流程不符合预期，可以在容器实例中查看日志、错误等，并进行代码、环境变量的修正。

ii. 预置脚本测试整体流程

一般使用run.sh封装训练外的文件复制工作（数据、代码：OBS-->容器，输出结果：容器-->OBS）。

如果预置脚本调用结果不符合预期，可以在容器实例中进行修改和迭代。

iii. 针对专属池场景

由于专属池支持SFS挂载，因此代码、数据的导入会更简单，甚至可以不用再关注OBS的相关操作。

可以直接把SFS的目录直接挂载到调试节点的"/mnt/sfs_turbo"目录，或者保证对应目录的内容和SFS盘匹配。

调试时建议使用接近的方式，即：启动容器实例时使用"-v"参数来指定挂载某个宿主机目录到容器环境。

```
docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
```

上述命令表示把宿主机的"/mnt/sfs_turbo"目录挂载到容器的"/sfs"目录，在宿主机和容器对应目录的所有改动都是实时同步的。

d. 分析错误时：训练镜像先看日志，推理镜像先看API的返回。

可以通过命令查看容器输出到stdout的所有日志：

```
docker logs -f 39c9ceedb1f6
```

一般在做推理镜像时，部分日志是直接存储在容器内部的，所以需要进入容器看日志。注意：重点对应日志中是否有ERROR（包括，容器启动时、API执行时）。

e. 牵扯部分文件用户组不一致的情况，可以在宿主机用root权限执行命令进行修改

```
docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
```

f. 针对调试中遇到的错误，可以直接在容器实例里修改，修改结果可以通过commit命令持久化。

上传镜像

客户端上传镜像，是指在安装了容器引擎客户端的机器上使用docker命令将镜像上传到容器镜像服务的镜像仓库。

如果容器引擎客户端机器为云上的ECS或CCE节点，根据机器所在区域有两种网络链路可以选择：

- 如果机器与容器镜像仓库在同一区域，则上传镜像走内网链路。
- 如果机器与容器镜像仓库不在同一区域，则上传镜像走公网链路，机器需要绑定弹性公网IP。

说明

- 使用客户端上传镜像，镜像的每个layer大小不能大于10G。
- 上传镜像的容器引擎客户端版本必须为1.11.2及以上。

1. 连接容器镜像服务。
   a. 登录容器镜像服务控制台。
   b. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
   c. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

      说明

      - 此处生成的登录指令有效期为24小时，如果需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
      - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。

   d. 在安装容器引擎的机器中执行上一步复制的登录指令。

登录成功会显示“Login Succeeded”。

2. 在安装容器引擎的机器上执行如下命令，为镜像打标签。

   docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   - [镜像名称1:版本名称1]：${image_name}:${image_version}请替换为您所要上传的实际镜像的名称和版本名称。
   - [镜像仓库地址]：可在SWR控制台上查询，即**1.c**中登录指令末尾的域名。
   - [组织名称]：/${organization_name}请替换为您创建的组织。
   - [镜像名称2:版本名称2]：${image_name}:${image_version}请替换为您期待的镜像名称和镜像版本。

   示例：

   ```
   docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

3. 上传镜像至镜像仓库。

   docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   示例：

   ```
   docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

   上传镜像完成后，返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

上传数据和算法到 OBS

- 已经在OBS上创建好并行文件系统，请参见**创建并行文件系统**。
- 已经安装和配置obsutil，请参见**安装和配置OBS命令行工具**。

步骤1 准备数据

1. 单击下载**动物数据集**至本地，并解压。
2. 通过obsutil将数据集上传至OBS桶中。

   ```
   ./obsutil cp ./dog_cat_1w obs://${your_obs_buck}/demo/ -f -r
   ```

   说明

   OBS支持多种文件上传方式，当文件少于100个时，可以在OBS Console中上传，当文件大于100个时，推荐使用工具，推荐**OBS Browser+**（win）、**obsutil**（linux）。上述例子为obsutil使用方法。

步骤2 准备算法

main.py文件内容如下，并将其上传至OBS桶的demo文件夹中：

```
import argparse
import os
import random
import shutil
import time
import warnings
from enum import Enum
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.distributed as dist
import torch.optim
from torch.optim.lr_scheduler import StepLR
```



```
import torch.multiprocessing as mp
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
model_names = sorted(name for name in models.__dict__
                     if name.islower() and not name.startswith("__")
                     and callable(models.__dict__[name]))
parser = argparse.ArgumentParser(description='PyTorch ImageNet Training')
parser.add_argument('data', metavar='DIR', default='imagenet',
                    help='path to dataset (default: imagenet)')
parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet18',
                    choices=model_names,
                    help='model architecture: ' +
                        ' | '.join(model_names) +
                        ' (default: resnet18)')
parser.add_argument('-j', '--workers', default=4, type=int, metavar='N',
                    help='number of data loading workers (default: 4)')
parser.add_argument('--epochs', default=90, type=int, metavar='N',
                    help='number of total epochs to run')
parser.add_argument('--start-epoch', default=0, type=int, metavar='N',
                    help='manual epoch number (useful on restarts)')
parser.add_argument('-b', '--batch-size', default=256, type=int,
                    metavar='N',
                    help='mini-batch size (default: 256), this is the total '
                        'batch size of all GPUs on the current node when '
                        'using Data Parallel or Distributed Data Parallel')
parser.add_argument('--lr', '--learning-rate', default=0.1, type=float,
                    metavar='LR', help='initial learning rate', dest='lr')
parser.add_argument('--momentum', default=0.9, type=float, metavar='M',
                    help='momentum')
parser.add_argument('--wd', '--weight-decay', default=1e-4, type=float,
                    metavar='W', help='weight decay (default: 1e-4)',
                    dest='weight_decay')
parser.add_argument('-p', '--print-freq', default=10, type=int,
                    metavar='N', help='print frequency (default: 10)')
parser.add_argument('--resume', default='', type=str, metavar='PATH',
                    help='path to latest checkpoint (default: none)')
parser.add_argument('-e', '--evaluate', dest='evaluate', action='store_true',
                    help='evaluate model on validation set')
parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                    help='use pre-trained model')
parser.add_argument('--world-size', default=-1, type=int,
                    help='number of nodes for distributed training')
parser.add_argument('--rank', default=-1, type=int,
                    help='node rank for distributed training')
parser.add_argument('--dist-url', default='tcp://224.66.41.62:23456', type=str,
                    help='url used to set up distributed training')
parser.add_argument('--dist-backend', default='nccl', type=str,
                    help='distributed backend')
parser.add_argument('--seed', default=None, type=int,
                    help='seed for initializing training. ')
parser.add_argument('--gpu', default=None, type=int,
                    help='GPU id to use.')
parser.add_argument('--multiprocessing-distributed', action='store_true',
                    help='Use multi-processing distributed training to launch '
                        'N processes per node, which has N GPUs. This is the '
                        'fastest way to use PyTorch for either single node or '
                        'multi node data parallel training')
best_acc1 = 0


def main():
    args = parser.parse_args()
    if args.seed is not None:
        random.seed(args.seed)
        torch.manual_seed(args.seed)
        cudnn.deterministic = True
```

```
        warnings.warn('You have chosen to seed training. '
                      'This will turn on the CUDNN deterministic setting, '
                      'which can slow down your training considerably! '
                      'You may see unexpected behavior when restarting '
                      'from checkpoints.')
    if args.gpu is not None:
        warnings.warn('You have chosen a specific GPU. This will completely '
                      'disable data parallelism.')
    if args.dist_url == "env://" and args.world_size == -1:
        args.world_size = int(os.environ["WORLD_SIZE"])
    args.distributed = args.world_size > 1 or args.multiprocessing_distributed
    ngpus_per_node = torch.cuda.device_count()
    if args.multiprocessing_distributed:
        # Since we have ngpus_per_node processes per node, the total world_size
        # needs to be adjusted accordingly
        args.world_size = ngpus_per_node * args.world_size
        # Use torch.multiprocessing.spawn to launch distributed processes: the
        # main_worker process function
        mp.spawn(main_worker, nprocs=ngpus_per_node, args=(ngpus_per_node, args))
    else:
        # Simply call main_worker function
        main_worker(args.gpu, ngpus_per_node, args)
def main_worker(gpu, ngpus_per_node, args):
    global best_acc1
    args.gpu = gpu
    if args.gpu is not None:
        print("Use GPU: {} for training".format(args.gpu))
    if args.distributed:
        if args.dist_url == "env://" and args.rank == -1:
            args.rank = int(os.environ["RANK"])
        if args.multiprocessing_distributed:
            # For multiprocessing distributed training, rank needs to be the
            # global rank among all the processes
            args.rank = args.rank * ngpus_per_node + gpu
        dist.init_process_group(backend=args.dist_backend, init_method=args.dist_url,
                                world_size=args.world_size, rank=args.rank)
    # create model
    if args.pretrained:
        print("=> using pre-trained model '{}'".format(args.arch))
        model = models.__dict__[args.arch](pretrained=True)
    else:
        print("=> creating model '{}'".format(args.arch))
        model = models.__dict__[args.arch]()
    if not torch.cuda.is_available():
        print('using CPU, this will be slow')
    elif args.distributed:
        # For multiprocessing distributed, DistributedDataParallel constructor
        # should always set the single device scope, otherwise,
        # DistributedDataParallel will use all available devices.
        if args.gpu is not None:
            torch.cuda.set_device(args.gpu)
            model.cuda(args.gpu)
            # When using a single GPU per process and per
            # DistributedDataParallel, we need to divide the batch size
            # ourselves based on the total number of GPUs of the current node.
            args.batch_size = int(args.batch_size / ngpus_per_node)
            args.workers = int((args.workers + ngpus_per_node - 1) / ngpus_per_node)
            model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[args.gpu])
        else:
            model.cuda()
            # DistributedDataParallel will divide and allocate batch_size to all
            # available GPUs if device_ids are not set
            model = torch.nn.parallel.DistributedDataParallel(model)
    elif args.gpu is not None:
        torch.cuda.set_device(args.gpu)
        model = model.cuda(args.gpu)
    else:
        # DataParallel will divide and allocate batch_size to all available GPUs
        if args.arch.startswith('alexnet') or args.arch.startswith('vgg'):
```

```
        model.features = torch.nn.DataParallel(model.features)
        model.cuda()
    else:
        model = torch.nn.DataParallel(model).cuda()
# define loss function (criterion), optimizer, and learning rate scheduler
criterion = nn.CrossEntropyLoss().cuda(args.gpu)
optimizer = torch.optim.SGD(model.parameters(), args.lr,
                            momentum=args.momentum,
                            weight_decay=args.weight_decay)
"""Sets the learning rate to the initial LR decayed by 10 every 30 epochs"""
scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
# optionally resume from a checkpoint
if args.resume:
    if os.path.isfile(args.resume):
        print("=> loading checkpoint '{}'".format(args.resume))
        if args.gpu is None:
            checkpoint = torch.load(args.resume)
        else:
            # Map model to be loaded to specified single gpu.
            loc = 'cuda:{}'.format(args.gpu)
            checkpoint = torch.load(args.resume, map_location=loc)
        args.start_epoch = checkpoint['epoch']
        best_acc1 = checkpoint['best_acc1']
        if args.gpu is not None:
            # best_acc1 may be from a checkpoint from a different GPU
            best_acc1 = best_acc1.to(args.gpu)
        model.load_state_dict(checkpoint['state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer'])
        scheduler.load_state_dict(checkpoint['scheduler'])
        print("=> loaded checkpoint '{}' (epoch {})"
              .format(args.resume, checkpoint['epoch']))
    else:
        print("=> no checkpoint found at '{}'".format(args.resume))
cudnn.benchmark = True

# Data loading code
traindir = os.path.join(args.data, 'train')
valdir = os.path.join(args.data, 'val')
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
train_dataset = datasets.ImageFolder(
    traindir,
    transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize,
    ]))
if args.distributed:
    train_sampler = torch.utils.data.distributed.DistributedSampler(train_dataset)
else:
    train_sampler = None

train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, shuffle=(train_sampler is None),
    num_workers=args.workers, pin_memory=True, sampler=train_sampler)
val_loader = torch.utils.data.DataLoader(
    datasets.ImageFolder(valdir, transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        normalize,
    ])),
    batch_size=args.batch_size, shuffle=False,
    num_workers=args.workers, pin_memory=True)
if args.evaluate:
    validate(val_loader, model, criterion, args)
    return
```

```
    for epoch in range(args.start_epoch, args.epochs):
        if args.distributed:
            train_sampler.set_epoch(epoch)
        # train for one epoch
        train(train_loader, model, criterion, optimizer, epoch, args)
        # evaluate on validation set
        acc1 = validate(val_loader, model, criterion, args)
        scheduler.step()
        # remember best acc@1 and save checkpoint
        is_best = acc1 > best_acc1
        best_acc1 = max(acc1, best_acc1)
        if not args.multiprocessing_distributed or (args.multiprocessing_distributed
                                                    and args.rank % ngpus_per_node == 0):
            save_checkpoint({
                'epoch': epoch + 1,
                'arch': args.arch,
                'state_dict': model.state_dict(),
                'best_acc1': best_acc1,
                'optimizer': optimizer.state_dict(),
                'scheduler': scheduler.state_dict()
            }, is_best)
def train(train_loader, model, criterion, optimizer, epoch, args):
    batch_time = AverageMeter('Time', ':6.3f')
    data_time = AverageMeter('Data', ':6.3f')
    losses = AverageMeter('Loss', ':.4e')
    top1 = AverageMeter('Acc@1', ':6.2f')
    top5 = AverageMeter('Acc@5', ':6.2f')
    progress = ProgressMeter(
        len(train_loader),
        [batch_time, data_time, losses, top1, top5],
        prefix="Epoch: [{}]".format(epoch))
    # switch to train mode
    model.train()
    end = time.time()
    for i, (images, target) in enumerate(train_loader):
        # measure data loading time
        data_time.update(time.time() - end)
        if args.gpu is not None:
            images = images.cuda(args.gpu, non_blocking=True)
        if torch.cuda.is_available():
            target = target.cuda(args.gpu, non_blocking=True)
        # compute output
        output = model(images)
        loss = criterion(output, target)
        # measure accuracy and record loss
        acc1, acc5 = accuracy(output, target, topk=(1, 5))
        losses.update(loss.item(), images.size(0))
        top1.update(acc1[0], images.size(0))
        top5.update(acc5[0], images.size(0))
        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()
        if i % args.print_freq == 0:
            progress.display(i)
def validate(val_loader, model, criterion, args):
    batch_time = AverageMeter('Time', ':6.3f', Summary.NONE)
    losses = AverageMeter('Loss', ':.4e', Summary.NONE)
    top1 = AverageMeter('Acc@1', ':6.2f', Summary.AVERAGE)
    top5 = AverageMeter('Acc@5', ':6.2f', Summary.AVERAGE)
    progress = ProgressMeter(
        len(val_loader),
        [batch_time, losses, top1, top5],
        prefix='Test: ')
    # switch to evaluate mode
    model.eval()
```

```
    with torch.no_grad():
        end = time.time()
        for i, (images, target) in enumerate(val_loader):
            if args.gpu is not None:
                images = images.cuda(args.gpu, non_blocking=True)
            if torch.cuda.is_available():
                target = target.cuda(args.gpu, non_blocking=True)
            # compute output
            output = model(images)
            loss = criterion(output, target)
            # measure accuracy and record loss
            acc1, acc5 = accuracy(output, target, topk=(1, 5))
            losses.update(loss.item(), images.size(0))
            top1.update(acc1[0], images.size(0))
            top5.update(acc5[0], images.size(0))
            # measure elapsed time
            batch_time.update(time.time() - end)
            end = time.time()
            if i % args.print_freq == 0:
                progress.display(i)
        progress.display_summary()
    return top1.avg
def save_checkpoint(state, is_best, filename='checkpoint.pth.tar'):
    torch.save(state, filename)
    if is_best:
        shutil.copyfile(filename, 'model_best.pth.tar')
class Summary(Enum):
    NONE = 0
    AVERAGE = 1
    SUM = 2
    COUNT = 3


class AverageMeter(object):
    """Computes and stores the average and current value"""

    def __init__(self, name, fmt=':f', summary_type=Summary.AVERAGE):
        self.name = name
        self.fmt = fmt
        self.summary_type = summary_type
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
    def __str__(self):
        fmtstr = '{name} {val' + self.fmt + '} ({avg' + self.fmt + '})'
        return fmtstr.format(**self.__dict__)
    def summary(self):
        fmtstr = ''
        if self.summary_type is Summary.NONE:
            fmtstr = ''
        elif self.summary_type is Summary.AVERAGE:
            fmtstr = '{name} {avg:.3f}'
        elif self.summary_type is Summary.SUM:
            fmtstr = '{name} {sum:.3f}'
        elif self.summary_type is Summary.COUNT:
            fmtstr = '{name} {count:.3f}'
        else:
            raise ValueError('invalid summary type %r' % self.summary_type)
        return fmtstr.format(**self.__dict__)
```

```
class ProgressMeter(object):
    def __init__(self, num_batches, meters, prefix=""):
        self.batch_fmtstr = self._get_batch_fmtstr(num_batches)
        self.meters = meters
        self.prefix = prefix
    def display(self, batch):
        entries = [self.prefix + self.batch_fmtstr.format(batch)]
        entries += [str(meter) for meter in self.meters]
        print('\t'.join(entries))
    def display_summary(self):
        entries = [" *"]
        entries += [meter.summary() for meter in self.meters]
        print(' '.join(entries))
    def _get_batch_fmtstr(self, num_batches):
        num_digits = len(str(num_batches // 1))
        fmt = '{:' + str(num_digits) + 'd}'
        return '[' + fmt + '/' + fmt.format(num_batches) + ']'


def accuracy(output, target, topk=(1,)):
    """Computes the accuracy over the k top predictions for the specified values of k"""
    with torch.no_grad():
        maxk = max(topk)
        batch_size = target.size(0)
        _, pred = output.topk(maxk, 1, True, True)
        pred = pred.t()
        correct = pred.eq(target.view(1, -1).expand_as(pred))
        res = []
        for k in topk:
            correct_k = correct[:k].reshape(-1).float().sum(0, keepdim=True)
            res.append(correct_k.mul_(100.0 / batch_size))
        return res
if __name__ == '__main__':
    main()
```

----结束

使用 Notebook 进行代码调试

- Notebook使用涉及到计费，具体收费项如下：
  - 处于“运行中”状态的Notebook，会消耗资源，产生费用。根据您选择的资源不同，收费标准不同，价格详情请参见**产品价格详情**。当您不需要使用Notebook时，建议停止Notebook，避免产生不必要的费用。
  - 创建Notebook时，如果选择使用云硬盘EVS存储配置，云硬盘EVS会一直收费，建议及时停止并删除Notebook，避免产生不必要的费用。
- 在创建Notebook时，默认会开启自动停止功能，在指定时间内停止运行Notebook，避免资源浪费。
- 只有处于“运行中”状态的Notebook，才可以执行打开、停止操作。
- 一个账户最多创建10个Notebook。

操作步骤如下：

1. 注册镜像。登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。单击“注册镜像”，镜像源即为推送到SWR中的镜像。请将完整的SWR地址复制到这里即可，或单击  可直接从SWR选择自有镜像进行注册，类型加上“GPU”。
2. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”列表页面。
3. 单击“创建Notebook”，进入“创建Notebook”页面，请参见如下说明填写参数。

a. 填写Notebook基本信息，包含名称、描述、是否自动停止，详细参数请参见**表7-5**。

表 7-5 基本信息的参数描述

参数名称	说明
“名称”	Notebook的名称。只能包含数字、大小写字母、下划线和中划线，长度不能大于64位且不能为空。
“描述”	对Notebook的简要描述。
“自动停止”	默认开启，且默认值为“1小时”，表示该Notebook实例将在运行1小时之后自动停止，即1小时后停止规格资源计费。 开启自动停止功能后，可选择“1小时”、“2小时”、“4小时”、“6小时”或“自定义”几种模式。选择“自定义”模式时，可指定1~24小时范围内任意整数。

b. 填写Notebook详细参数，如镜像、资源规格等。

- 镜像：在“自定义镜像”页签选择已上传的自定义镜像。
- 资源类型：按实际情况选择已创建的专属资源池。
- 规格：选择1 GPU规格。
- 存储配置：选择“云硬盘EVS”作为存储位置。

说明

如果需要通过VS Code连接Notebook方式进行代码调试，则需开启“SSH远程开发”并选择密钥对，请参考**VS Code连接Notebook方式介绍**。

4. 参数填写完成后，单击“立即创建”进行规格确认。
5. 参数确认无误后，单击“提交”，完成Notebook的创建操作。

   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。

   如果创建Notebook启动失败，建议参考**调试要点**进行检查。
6. 在Notebook列表，单击实例名称，进入实例详情页，查看Notebook实例配置信息。
7. 挂载OBS并行文件系统：在Notebook实例详情页面，选择“存储配置”页签，单击“添加数据存储”，设置挂载参数。

   a. 设置本地挂载目录，在“/data/”目录下输入一个文件夹名称，例如：demo。挂载时，后台自动会在Notebook容器“的/data/”目录下创建该文件夹，用来挂载OBS文件系统。

   b. 选择存放OBS并行文件系统下的文件夹，单击“确定”。
8. 挂载成功后，可以在Notebook实例详情页查看到挂载结果。
9. 代码调试。

   打开Notebook，打开Terminal，进入步骤7中挂载的目录。

   ```
   cd /data/demo
   ```

   执行训练命令：

```
/home/ma-user/anaconda3/envs/pytorch/bin/python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
```

告警"RequestsDependencyWarning: urllib3 (1.26.8) or chardet (5.0.0)/ charset_normalizer (2.0.12) doesn't match a supported version!"不影响训练，可忽略。

说明

Notebook中调试完后，如果镜像有修改，可以保存镜像用于后续训练，具体操作请参见**保存Notebook镜像环境**。

创建单机单卡训练作业

说明

针对专属池场景，应注意挂载的目录设置和调试时一致。

1. 登录ModelArts管理控制台，检查当前帐号是否已完成访问授权的配置。如果未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。单击“创建训练作业”进入创建训练作业页面。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     cd ${MA_JOB_DIR}/demo && python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
     ```
     此处的“demo”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择单GPU规格。
4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

监控资源

用户可以通过资源占用情况窗口查看计算节点的资源使用情况，最多可显示最近三天的数据。在资源占用情况窗口打开时，会定期向后台获取最新的资源使用率数据并刷新。

操作一：如果训练作业使用多个计算节点，可以通过实例名称的下拉框切换节点。

操作二：单击图例“cpuUsage”、“memUsage”“npuMemUsage”、“npuUtil”等，可以添加或取消对应参数的使用情况图。

操作三：鼠标悬浮在图片上的时间节点，可查看对应时间节点的占用率情况。

表 7-6 参数说明

参数	说明
cpuUsage	cpu使用率。
gpuMemUsage	gpu内存使用率。
gpuUtil	gpu使用情况。
memUsage	内存使用率。
npuMemUsage	npu内存使用率。
npuUtil	npu使用情况。

7.2.4 在 ModelArts Standard 上运行单机多卡训练作业

操作流程

1. 准备工作：
   a. **购买服务资源**（VPC、SFS、SWR和ECS）
   b. **配置权限**
   c. **创建专属资源池**（打通VPC）
   d. **在ECS服务器挂载SFS Turbo存储**
   e. **在ECS中设置ModelArts用户可读权限**
   f. **安装和配置OBS命令行工具**
   g. **（可选）工作空间配置**
2. 模型训练：
   a. **本地构建镜像及调试**
   b. **上传镜像**
   c. **上传数据和算法至SFS（首次使用时需要）**
   d. **使用Notebook进行代码调试**
   e. **创建单机多卡训练作业**

本地构建镜像及调试

本节通过打包conda env来构建环境，也可以通过pip install、conda install等方式安装conda环境依赖。

说明

- 容器镜像的大小建议小于15G，详细的自定义镜像规范要求请参见**训练作业自定义镜像规范**。
- 建议通过开源的官方镜像来构建，例如PyTorch的官方镜像。
- 建议容器分层构建，单层容量不要超过1G、文件数不大于10w个。分层时，先构建不常变化的层，例如：先OS，再cuda驱动，再Python，再pytorch，再其他依赖包。
- 如果训练数据和代码经常变动，则不建议把数据、代码放到容器镜像里，避免频繁地构建容器镜像。
- 容器已经能满足隔离需求，不建议在容器内再创建多个conda env。

1. 导出conda环境。

   a. 启动线下的容器镜像：

   ```
   # run on terminal
   docker run -ti ${your_image:tag}
   ```

   b. 在容器中输入如下命令，得到pytorch.tar.gz：

   ```
   # run on container

   # 基于想要迁移的base环境创建一个名为pytorch的conda环境
   conda create --name pytorch --clone base

   pip install conda-pack

   #将pytorch env打包生成pytorch.tar.gz
   conda pack -n pytorch -o pytorch.tar.gz
   ```

   c. 将打包好的压缩包传到本地：

   ```
   # run on terminal
   docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
   ```

   d. 将pytorch.tar.gz上传到OBS并**设置公共读**，并在构建时使用**wget**命令获取、解压、清理。

2. 构建新镜像。

   基础镜像一般选用“ubuntu 18.04”的官方镜像，或者nvidia官方提供的带cuda驱动的镜像。相关镜像直接到dockerhub官网查找即可。

   构建流程：安装所需的apt包、驱动，配置ma-user用户、导入conda环境、配置Notebook依赖。

   说明

   - 推荐使用Dockerfile的方式构建镜像。这样既满足dockerfile可追溯及构建归档的需求，也保证镜像内容无冗余和残留。
   - 每层构建的时候都尽量把tar包等中间态文件删除，保证最终镜像更小，清理缓存的方法可参考：**conda clean**。

3. 构建参考样例

   Dockerfile样例：

   ```
   FROM nvidia/cuda:11.3.1-cudnn8-devel-ubuntu18.04

   USER root

   # section1: add user ma-user whose uid is 1000 and user group ma-group whose gid is 100. If there already exists 1000:100 but not ma-user:ma-group, below code will remove it
   RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
       default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
       if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
           groupdel -f ${default_group}; \
           groupadd -g 100 ma-group; \
       fi && \
       if [ -z ${default_group} ]; then \
   ```

```
        groupadd -g 100 ma-group; \
    fi && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    if [ -z ${default_user} ]; then \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    # set bash as default
    rm /bin/sh && ln -s /bin/bash /bin/sh

# section2: config apt source and install tools needed.
RUN sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y ca-certificates curl ffmpeg git libgl1-mesa-glx libglib2.0-0 libibverbs-dev libjpeg-
dev libpng-dev libsm6 libxext6 libxrender-dev ninja-build screen sudo vim wget zip && \
    apt-get clean  && \
    rm -rf /var/lib/apt/lists/*

USER ma-user

# section3: install miniconda and rebuild conda env
RUN mkdir -p /home/ma-user/work/ && cd /home/ma-user/work/ && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    chmod 777 Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    bash Miniconda3-py37_4.12.0-Linux-x86_64.sh -bfp /home/ma-user/anaconda3 && \
    wget https://${bucketname}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz && \
    mkdir -p /home/ma-user/anaconda3/envs/pytorch && \
    tar -xzf pytorch.tar.gz -C /home/ma-user/anaconda3/envs/pytorch && \
    source /home/ma-user/anaconda3/envs/pytorch/bin/activate && conda-unpack && \
    /home/ma-user/anaconda3/bin/conda init bash && \
    rm -rf /home/ma-user/work/*

ENV PATH=/home/ma-user/anaconda3/envs/pytorch/bin:$PATH

# section4: settings of Jupyter Notebook for pytorch env
RUN source /home/ma-user/anaconda3/envs/pytorch/bin/activate && \
    pip install ipykernel==6.7.0 --trusted-host https://repo.huaweicloud.com -i https://
repo.huaweicloud.com/repository/pypi/simple && \
    ipython kernel install --user --env PATH /home/ma-user/anaconda3/envs/pytorch/bin:$PATH --
name=pytorch && \
    rm -rf /home/ma-user/.local/share/jupyter/kernels/pytorch/logo-* && \
    rm -rf ~/.cache/pip/* && \
    echo 'export PATH=$PATH:/home/ma-user/.local/bin' >> /home/ma-user/.bashrc && \
    echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/nvidia/lib64' >> /home/ma-
user/.bashrc && \
    echo 'conda activate pytorch' >> /home/ma-user/.bashrc

ENV DEFAULT_CONDA_ENV_NAME=pytorch
```

说明

Dockerfile中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz"，需要替换为**1**中pytorch.tar.gz在OBS上的路径（需将文件设置为公共读）。

进入Dockerfile目录，通过Dockerfile构建镜像命令：

```
# cd 到Dockerfile所在目录下，输入构建命令
# docker build -t ${image_name}:${image_version} .
# 例如
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

4. 调试镜像

说明

建议把调试过程中的修改点通过Dockerfile固化到容器构建正式流程，并重新测试。

a. 确认对应的脚本、代码、流程在linux服务器上运行正常。

如果在linux服务器上运行就有问题，那么先调通以后再做容器镜像。

b. 确认打入镜像的文件是否在正确的位置、是否有正确的权限。

训练场景主要查看自研的依赖包是否正常，查看pip list是否包含所需的包，查看容器直接调用的python是否是自己所需要的那个（如果容器镜像装了多个python，需要设置python路径的环境变量）。

c. 测试训练启动脚本。

i. 优先使用手工进行数据复制的工作并验证

一般在镜像里不包含训练所用的数据和代码，所以在启动镜像以后需要手工把需要的文件复制进去。建议数据、代码和中间数据都放到"/cache"目录，防止正式运行时磁盘占满。建议linux服务器申请的时候，有足够大的内存（8G以上）以及足够大的硬盘（100G以上）。

docker和linux的文件交互命令如下：

```
docker cp data/ 39c9ceedb1f6:/cache/
```

数据准备完成后，启动训练的脚本，查看训练是否能够正常拉起。一般来说，启动脚本为：

```
cd /cache/code/
python start_train.py
```

如果训练流程不符合预期，可以在容器实例中查看日志、错误等，并进行代码、环境变量的修正。

ii. 预置脚本测试整体流程

一般使用run.sh封装训练外的文件复制工作（数据、代码：OBS-->容器，输出结果：容器-->OBS）。

如果预置脚本调用结果不符合预期，可以在容器实例中进行修改和迭代。

iii. 针对专属池场景

由于专属池支持SFS挂载，因此代码、数据的导入会更简单，甚至可以不用再关注OBS的相关操作。

可以直接把SFS的目录直接挂载到调试节点的"/mnt/sfs_turbo"目录，或者保证对应目录的内容和SFS盘匹配。

调试时建议使用接近的方式，即：启动容器实例时使用"-v"参数来指定挂载某个宿主机目录到容器环境。

```
docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
```

上述命令表示把宿主机的"/mnt/sfs_turbo"目录挂载到容器的"/sfs"目录，在宿主机和容器对应目录的所有改动都是实时同步的。

d. 分析错误时：训练镜像先看日志，推理镜像先看API的返回。

可以通过命令查看容器输出到stdout的所有日志：

```
docker logs -f 39c9ceedb1f6
```

一般在做推理镜像时，部分日志是直接存储在容器内部的，所以需要进入容器看日志。注意：重点对应日志中是否有ERROR（包括，容器启动时、API执行时）。

e. 牵扯部分文件用户组不一致的情况，可以在宿主机用root权限执行命令进行修改

```
docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
```

f. 针对调试中遇到的错误，可以直接在容器实例里修改，修改结果可以通过commit命令持久化。

上传镜像

客户端上传镜像，是指在安装了容器引擎客户端的机器上使用docker命令将镜像上传到容器镜像服务的镜像仓库。

如果容器引擎客户端机器为云上的ECS或CCE节点，根据机器所在区域有两种网络链路可以选择：

- 如果机器与容器镜像仓库在同一区域，则上传镜像走内网链路。
- 如果机器与容器镜像仓库不在同一区域，则上传镜像走公网链路，机器需要绑定弹性公网IP。

说明

- 使用客户端上传镜像，镜像的每个layer大小不能大于10G。
- 上传镜像的容器引擎客户端版本必须为1.11.2及以上。

1. 连接容器镜像服务。
   a. 登录**容器镜像服务控制台**。
   b. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
   c. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

      说明

      - 此处生成的登录指令有效期为24小时，如果需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
      - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。

   d. 在安装容器引擎的机器中执行上一步复制的登录指令。

      登录成功会显示“Login Succeeded”。
2. 在安装容器引擎的机器上执行如下命令，为镜像打标签。

   docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   - [镜像名称1:版本名称1]：${image_name}:${image_version}请替换为您所要上传的实际镜像的名称和版本名称。
   - [镜像仓库地址]：可在SWR控制台上查询，即**1.c**中登录指令末尾的域名。
   - [组织名称]：/${organization_name}请替换为您创建的组织。
   - [镜像名称2:版本名称2]：${image_name}:${image_version}请替换为您期待的镜像名称和镜像版本。

   示例：

   ```
   docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```
3. 上传镜像至镜像仓库。

   docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

示例：

```
docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
```

上传镜像完成后，返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

上传数据和算法至 SFS

- ECS服务器已挂载SFS，请参考**在ECS服务器挂载SFS Turbo存储**。
- 已经在ECS中设置权限，请参考**在ECS中设置ModelArts用户可读权限**。
- 已经安装和配置obsutil，请参见**安装和配置OBS命令行工具**。

步骤1 准备数据

1. 登录coco数据集下载官网地址：**https://cocodataset.org/#download**
2. 下载coco2017数据集的Train（18GB）、Val images（1GB）、Train/Val annotations（241MB），分别解压后并放入coco文件夹中。
3. 下载完成后，将数据上传至SFS相应目录中。由于数据集过大，推荐先通过obsutil工具将数据集传到OBS桶后，再将数据集迁移至SFS。

   a. 在本机机器上运行，通过obsutil工具将本地数据集传到OBS桶。

   ```
   # 将本地数据传至OBS中
   # ./obsutil cp ${数据集所在的本地文件夹路径} ${存放数据集的obs文件夹路径} -f -r
   # 例如
   ./obsutil cp ./coco obs://your_bucket/ -f -r
   ```

   b. 登录ECS服务器，通过obsutil工具将数据集迁移至SFS，样例代码如下：

   ```
   # 将OBS数据传至SFS中
   # ./obsutil cp ${数据集所在的obs文件夹路径} ${SFS文件夹路径} -f -r
   # 例如
   ./obsutil cp obs://your_bucket/coco/ /mnt/sfs_turbo/ -f -r
   ```

   /mnt/sfs_turbo/coco文件夹内目录结构如下：

   ```
   coco
   |---annotations
   |---train2017
   |---val2017
   ```

   更多obsutil的操作，可参考**obsutil简介**。

   c. 将文件设置归属为ma-user：

   ```
   chown -R ma-user:ma-group coco
   ```

步骤2 准备算法

1. 下载YOLOX代码。代码仓地址：**https://github.com/Megvii-BaseDetection/YOLOX.git**。

   ```
   git clone https://github.com/Megvii-BaseDetection/YOLOX.git
   cd YOLOX
   git checkout 4f8f1d79c8b8e530495b5f183280bab99869e845
   ```

2. 修改“requirements.txt”中的onnx版本，改为“onnx>=1.12.0”。
3. 将“yolox/data/datasets/coco.py”第59行的“data_dir = os.path.join(get_yolox_datadir(), "COCO")”改为“data_dir = '/home/ma-user/coco'”。

   ```
   # data_dir = os.path.join(get_yolox_datadir(), "COCO")
   data_dir = '/home/ma-user/coco'
   ```

4. 在“tools/train.py”的第13行前加两句代码。

   ```
   # 加上这两句代码，防止运行时找不到yolox module
   import sys
   sys.path.append(os.getcwd())
   ```

```
# line13
from yolox.core import launch
from yolox.exp import Exp, get_exp
```

5. 将“yolox/layers/jit_ops.py”第122行的“fast_cocoeval”改为“fast_coco_eval_api”。

```
# def __init__(self, name="fast_cocoeval"):
def __init__(self, name="fast_coco_eval_api"):
```

6. 将“yolox\evaluators\coco_evaluator.py”第294行的“from yolox.layers import COCOeval_opt as COCOeval”改为“from pycocotools.cocoeval import COCOeval”。

```
try:
  # from yolox.layers import COCOeval_opt as COCOeval
  from pycocotools.cocoeval import COCOeval
except ImportError:
  from pycocotools.cocoeval import COCOeval

  logger.warning("Use standard COCOeval.")
```

7. 在tools目录下新建一个“run.sh”作为启动脚本，“run.sh”内容可参考：

```
#!/usr/bin/env sh
set -x
set -o pipefail

export NCCL_DEBUG=INFO

DEFAULT_ONE_GPU_BATCH_SIZE=32
BATCH_SIZE=$((${MA_NUM_GPUS:-8} * ${VC_WORKER_NUM:-1} * $
{DEFAULT_ONE_GPU_BATCH_SIZE}))
if [ ${VC_WORKER_HOSTS} ];then
    YOLOX_DIST_URL=tcp://$(echo ${VC_WORKER_HOSTS} | cut -d "," -f 1):6666
    /home/ma-user/anaconda3/envs/pytorch/bin/python -u tools/train.py \
                            -n yolox-s \
                            --devices ${MA_NUM_GPUS:-8} \
                            --batch-size ${BATCH_SIZE} \
                            --fp16 \
                            --occupy \
                            --num_machines ${VC_WORKER_NUM:-1} \
                            --machine_rank ${VC_TASK_INDEX:-0} \
                            --dist-url ${YOLOX_DIST_URL}
else
    /home/ma-user/anaconda3/envs/pytorch/bin/python -u tools/train.py \
                            -n yolox-s \
                            --devices ${MA_NUM_GPUS:-8} \
                            --batch-size ${BATCH_SIZE} \
                            --fp16 \
                            --occupy \
                            --num_machines ${VC_WORKER_NUM:-1} \
                            --machine_rank ${VC_TASK_INDEX:-0}
fi
```

说明

部分环境变量在Notebook环境中不存在，因此需要提供默认值。

8. 将代码放到OBS上，然后通过OBS将代码传至SFS相应目录中。

a. 在本机机器上运行，通过obsutil工具将本地数据集传到OBS桶。

```
# 将本地代码传至OBS中
./obsutil cp ./YOLOX obs://your_bucket/ -f -r
```

b. 登录ECS服务器，通过obsutil工具将数据集迁移至SFS，样例代码如下：

```
# 将OBS的代码传到SFS中
./obsutil cp obs://your_bucket/YOLOX/ /mnt/sfs_turbo/code/ -f -r
```

说明

本案例中以obsutils方式上传文件，除此之外也可通过SCP方式上传文件，具体操作步骤可参考**本地Linux主机使用SCP上传文件到Linux云服务器**。

9. 在SFS中将文件设置归属为ma-user。
```
chown -R ma-user:ma-group YOLOX
```
10. 执行以下命令，去除Shell脚本的\r字符。
```
cd YOLOX
sed -i 's/\r//' run.sh
```

说明

Shell脚本在Windows系统编写时，每行结尾是\r\n，而在Linux系统中行每行结尾是\n，所以在Linux系统中运行脚本时，会认为\r是一个字符，导致运行报错“$'\r': command not found”，因此需要去除Shell脚本的\r字符。

----结束

使用 Notebook 进行代码调试

- Notebook使用涉及到计费，具体收费项如下：
  - 处于“运行中”状态的Notebook，会消耗资源，产生费用。根据您选择的资源不同，收费标准不同，价格详情请参见**产品价格详情**。当您不需要使用Notebook时，建议停止Notebook，避免产生不必要的费用。
  - 创建Notebook时，如果选择使用云硬盘EVS存储配置，云硬盘EVS会一直收费，建议及时停止并删除Notebook，避免产生不必要的费用。
- 在创建Notebook时，默认会开启自动停止功能，在指定时间内停止运行Notebook，避免资源浪费。
- 只有处于“运行中”状态的Notebook，才可以执行打开、停止操作。
- 一个账户最多创建10个Notebook。

操作步骤如下：

1. 注册镜像。登录**ModelArts管理控制台**，在左侧导航栏选择“镜像管理”，进入镜像管理页面。单击“注册镜像”，镜像源即为推送到SWR中的镜像。请将完整的SWR地址复制到这里即可，或单击 可直接从SWR选择自有镜像进行注册，类型加上“GPU”。
2. 登录**ModelArts管理控制台**，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”列表页面。
3. 单击“创建Notebook”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止，详细参数请参见**表7-7**。

表 7-7 基本信息的参数描述

参数名称	说明
“名称”	Notebook的名称。只能包含数字、大小写字母、下划线和中划线，长度不能大于64位且不能为空。
“描述”	对Notebook的简要描述。

参数名称	说明
“自动停止”	默认开启，且默认值为“1小时”，表示该Notebook实例将在运行1小时之后自动停止，即1小时后停止规格资源计费。 开启自动停止功能后，可选择“1小时”、“2小时”、“4小时”、“6小时”或“自定义”几种模式。选择“自定义”模式时，可指定1~24小时范围内任意整数。

b. 填写Notebook详细参数，如镜像、资源规格等。

- 镜像：在“自定义镜像”页签选择已上传的自定义镜像。
- 资源类型：按实际情况选择已创建的专属资源池。
- 规格：选择8卡GPU规格，“run.sh”文件中默认MA_NUM_GPUS为8卡，因此选择notebook规格时需要与MA_NUM_GPUS默认值相同。
- 存储配置：选择“弹性文件服务SFS”作为存储位置。子目录挂载可不填写，如果需挂载SFS指定目录，则在子目录挂载处填写具体路径。

说明

如果需要通过VS Code连接Notebook方式进行代码调试，则需开启“SSH远程开发”并选择密钥对，请参考**VS Code连接Notebook方式介绍**。

4. 参数填写完成后，单击“立即创建”进行规格确认。
5. 参数确认无误后，单击“提交”，完成Notebook的创建操作。

   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。
6. 在Notebook列表，单击实例名称，进入实例详情页，查看Notebook实例配置信息。
7. 在Notebook中打开Terminal，输入启动命令调试代码。

```
# 建立数据集软链接
# ln -s /home/ma-user/work/${coco数据集在SFS上的路径} /home/ma-user/coco
# 进入到对应目录
# cd /home/ma-user/work/${YOLOX在SFS上的路径}
# 安装环境并执行脚本
# /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh

# 例如
ln -s /home/ma-user/work/coco /home/ma-user/coco
cd /home/ma-user/work/code/YOLOX/
/home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh
```

说明

Notebook中调试完后，如果镜像有修改，可以保存镜像用于后续训练，具体操作请参见**保存Notebook镜像环境**。

创建单机多卡训练作业

1. 登录**ModelArts管理控制台**，检查当前帐号是否已完成访问授权的配置。如果未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。单击“创建训练作业”进入创建训练作业页面。

3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     ln -s /home/ma-user/work/coco /home/ma-user/coco && cd /home/ma-user/work/code/YOLOX/ && /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh
     ```
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择8卡GPU规格。
   - 计算节点：1。
   - SFS Turbo：增加挂载配置，选择SFS名称，云上挂载路径为“/home/ma-user/work”。

     说明

     为了和Notebook调试时代码路径一致，保持相同的启动命令，因此云上挂载路径需要填写为“/home/ma-user/work”。
4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

7.2.5 在 ModelArts Standard 上运行多机多卡训练作业

操作流程

1. 准备工作：
   a. **购买服务资源**（VPC/SFS/OBS/SWR/ECS）
   b. **配置权限**
   c. **创建专属资源池**（打通VPC）
   d. **ECS服务器挂载SFS Turbo存储**
   e. **在ECS中设置ModelArts用户可读权限**
   f. **安装和配置OBS命令行工具**
   g. **（可选）工作空间配置**
2. 模型训练：
   a. **线下容器镜像构建及调试**
   b. **上传镜像**
   c. **上传数据至OBS（首次使用时需要）**
   d. **上传算法至SFS**
   e. **使用Notebook进行代码调试**
   f. **创建多机多卡训练作业**

本地构建镜像及调试

本节通过打包conda env来构建环境，也可以通过pip install、conda install等方式安装conda环境依赖。

说明

- 容器镜像的大小建议小于15G，详细的自定义镜像规范要求请参见**训练作业自定义镜像规范**。
- 建议通过开源的官方镜像来构建，例如PyTorch的官方镜像。
- 建议容器分层构建，单层容量不要超过1G、文件数不大于10w个。分层时，先构建不常变化的层，例如：先OS，再cuda驱动，再Python，再pytorch，再其他依赖包。
- 如果训练数据和代码经常变动，则不建议把数据、代码放到容器镜像里，避免频繁地构建容器镜像。
- 容器已经能满足隔离需求，不建议在容器内再创建多个conda env。

1. 导出conda环境。
   a. 启动线下的容器镜像：
   ```
   # run on terminal
   docker run -ti ${your_image:tag}
   ```
   b. 在容器中输入如下命令，得到pytorch.tar.gz：
   ```
   # run on container

   # 基于想要迁移的base环境创建一个名为pytorch的conda环境
   conda create --name pytorch --clone base

   pip install conda-pack

   #将pytorch env打包生成pytorch.tar.gz
   conda pack -n pytorch -o pytorch.tar.gz
   ```
   c. 将打包好的压缩包传到本地：
   ```
   # run on terminal
   docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
   ```
   d. 将pytorch.tar.gz上传到OBS并**设置公共读**，并在构建时使用**wget**命令获取、解压、清理。
2. 构建新镜像。

   基础镜像一般选用“ubuntu 18.04”的官方镜像，或者nvidia官方提供的带cuda驱动的镜像。相关镜像直接到dockerhub官网查找即可。

   构建流程：安装所需的apt包、驱动，配置ma-user用户、导入conda环境、配置Notebook依赖。

   说明

   - 推荐使用Dockerfile的方式构建镜像。这样既满足dockerfile可追溯及构建归档的需求，也保证镜像内容无冗余和残留。
   - 每层构建的时候都尽量把tar包等中间态文件删除，保证最终镜像更小，清理缓存的方法可参考：**conda clean**。
3. 构建参考样例

   Dockerfile样例：
   ```
   FROM nvidia/cuda:11.3.1-cudnn8-devel-ubuntu18.04

   USER root

   # section1: add user ma-user whose uid is 1000 and user group ma-group whose gid is 100. If there already exists 1000:100 but not ma-user:ma-group, below code will remove it
   RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
   ```



```
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
        groupadd -g 100 ma-group; \
    fi && \
    if [ -z ${default_group} ]; then \
        groupadd -g 100 ma-group; \
    fi && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    if [ -z ${default_user} ]; then \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    # set bash as default
    rm /bin/sh && ln -s /bin/bash /bin/sh

# section2: config apt source and install tools needed.
RUN sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y ca-certificates curl ffmpeg git libgl1-mesa-glx libglib2.0-0 libibverbs-dev libjpeg-
dev libpng-dev libsm6 libxext6 libxrender-dev ninja-build screen sudo vim wget zip && \
    apt-get clean  && \
    rm -rf /var/lib/apt/lists/*

USER ma-user

# section3: install miniconda and rebuild conda env
RUN mkdir -p /home/ma-user/work/ && cd /home/ma-user/work/ && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    chmod 777 Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    bash Miniconda3-py37_4.12.0-Linux-x86_64.sh -bfp /home/ma-user/anaconda3 && \
    wget https://${bucketname}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz && \
    mkdir -p /home/ma-user/anaconda3/envs/pytorch && \
    tar -xzf pytorch.tar.gz -C /home/ma-user/anaconda3/envs/pytorch && \
    source /home/ma-user/anaconda3/envs/pytorch/bin/activate && conda-unpack && \
    /home/ma-user/anaconda3/bin/conda init bash && \
    rm -rf /home/ma-user/work/*

ENV PATH=/home/ma-user/anaconda3/envs/pytorch/bin:$PATH

# section4: settings of Jupyter Notebook for pytorch env
RUN source /home/ma-user/anaconda3/envs/pytorch/bin/activate && \
    pip install ipykernel==6.7.0 --trusted-host https://repo.huaweicloud.com -i https://
repo.huaweicloud.com/repository/pypi/simple && \
    ipython kernel install --user --env PATH /home/ma-user/anaconda3/envs/pytorch/bin:$PATH --
name=pytorch && \
    rm -rf /home/ma-user/.local/share/jupyter/kernels/pytorch/logo-* && \
    rm -rf ~/.cache/pip/* && \
    echo 'export PATH=$PATH:/home/ma-user/.local/bin' >> /home/ma-user/.bashrc && \
    echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/nvidia/lib64' >> /home/ma-
user/.bashrc && \
    echo 'conda activate pytorch' >> /home/ma-user/.bashrc

ENV DEFAULT_CONDA_ENV_NAME=pytorch
```

说明

Dockerfile中的“https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz”，需要替换为**1**中“pytorch.tar.gz”在OBS上的路径（需将文件设置为公共读）。

进入Dockerfile目录，通过Dockerfile构建镜像命令：

```
# cd 到Dockerfile所在目录下，输入构建命令
# docker build -t ${image_name}:${image_version} .
```

```
# 例如
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

4. 调试镜像

   说明

   建议把调试过程中的修改点通过Dockerfile固化到容器构建正式流程，并重新测试。

   a. 确认对应的脚本、代码、流程在linux服务器上运行正常。

      如果在linux服务器上运行就有问题，那么先调通以后再做容器镜像。

   b. 确认打入镜像的文件是否在正确的位置、是否有正确的权限。

      训练场景主要查看自研的依赖包是否正常，查看pip list是否包含所需的包，查看容器直接调用的python是否是自己所需要的那个（如果容器镜像装了多个python，需要设置python路径的环境变量）。

   c. 测试训练启动脚本。

      i. 优先使用手工进行数据复制的工作并验证

         一般在镜像里不包含训练所用的数据和代码，所以在启动镜像以后需要手工把需要的文件复制进去。建议数据、代码和中间数据都放到"/cache"目录，防止正式运行时磁盘占满。建议linux服务器申请的时候，有足够大的内存（8G以上）以及足够大的硬盘（100G以上）。

         docker和linux的文件交互命令如下：

         ```
         docker cp data/ 39c9ceedb1f6:/cache/
         ```

         数据准备完成后，启动训练的脚本，查看训练是否能够正常拉起。一般来说，启动脚本为：

         ```
         cd /cache/code/
         python start_train.py
         ```

         如果训练流程不符合预期，可以在容器实例中查看日志、错误等，并进行代码、环境变量的修正。

      ii. 预制脚本测试整体流程

         一般使用run.sh封装训练外的文件复制工作（数据、代码：OBS-->容器，输出结果：容器-->OBS），run.sh的构建方法参考**基于ModelArts Standard运行训练作业**。

         如果预置脚本调用结果不符合预期，可以在容器实例中进行修改和迭代。

      iii. 针对专属池场景

         由于专属池支持SFS挂载，因此代码、数据的导入会更简单，甚至可以不用再关注OBS的相关操作。

         可以直接把SFS的目录直接挂载到调试节点的“/mnt/sfs_turbo”目录，或者保证对应目录的内容和SFS盘匹配。

         调试时建议使用接近的方式，即：启动容器实例时使用"-v"参数来指定挂载某个宿主机目录到容器环境。

         ```
         docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
         ```

         上述命令表示把宿主机的"/mnt/sfs_turbo"目录挂载到容器的"/sfs"目录，在宿主机和容器对应目录的所有改动都是实时同步的。

   d. 分析错误时：训练镜像先看日志，推理镜像先看API的返回。

      可以通过命令查看容器输出到stdout的所有日志：

      ```
      docker logs -f 39c9ceedb1f6
      ```

一般在做推理镜像时，部分日志是直接存储在容器内部的，所以需要进入容器看日志。注意：重点对应日志中是否有ERROR（包括，容器启动时、API执行时）。

e. 牵扯部分文件用户组不一致的情况，可以在宿主机用root权限执行命令进行修改

```
docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
```

f. 针对调试中遇到的错误，可以直接在容器实例里修改，修改结果可以通过commit命令持久化。

上传镜像

客户端上传镜像，是指在安装了容器引擎客户端的机器上使用docker命令将镜像上传到容器镜像服务的镜像仓库。

如果容器引擎客户端机器为云上的ECS或CCE节点，根据机器所在区域有两种网络链路可以选择：

- 如果机器与容器镜像仓库在同一区域，则上传镜像走内网链路。
- 如果机器与容器镜像仓库不在同一区域，则上传镜像走公网链路，机器需要绑定弹性公网IP。

说明

- 使用客户端上传镜像，镜像的每个layer大小不能大于10G。
- 上传镜像的容器引擎客户端版本必须为1.11.2及以上。

1. 连接容器镜像服务。
   a. 登录**容器镜像服务控制台**。
   b. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
   c. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

      说明

      - 此处生成的登录指令有效期为24小时，如果需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
      - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。

   d. 在安装容器引擎的机器中执行上一步复制的登录指令。

      登录成功会显示“Login Succeeded”。
2. 在安装容器引擎的机器上执行如下命令，为镜像打标签。

   docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   - [镜像名称1:版本名称1]：${image_name}:${image_version}请替换为您所要上传的实际镜像的名称和版本名称。
   - [镜像仓库地址]：可在SWR控制台上查询，即**1.c**中登录指令末尾的域名。
   - [组织名称]：/${organization_name}请替换为您创建的组织。
   - [镜像名称2:版本名称2]：${image_name}:${image_version}请替换为您期待的镜像名称和镜像版本。

示例：

```
docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
```

3. 上传镜像至镜像仓库。

   docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   示例：

   ```
   docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

   上传镜像完成后，返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

上传数据至 OBS

- 已经在OBS上创建好普通OBS桶，请参见**创建普通OBS桶**。
- 已经安装obsutil，请参考**安装和配置OBS命令行工具**。
- OBS和训练容器间的数据传输原理可以参考**基于ModelArts Standard运行训练作业**。

1. 登录Imagenet数据集下载官网地址，下载Imagenet21k数据集：**http://image-net.org/**
2. 下载格式转换后的annotation文件：**ILSVRC2021winner21k_whole_map_train.txt**和**ILSVRC2021winner21k_whole_map_val.txt**。
3. 下载完成后将上述3个文件数据上传至OBS桶中的imagenet21k_whole文件夹中。文件上传OBS桶方法请参考**上传数据和算法到OBS**。

上传算法到 SFS

1. 下载Swin-Transformer代码。
   ```
   git clone --recursive https://github.com/microsoft/Swin-Transformer.git
   ```
2. 修改lr_scheduler.py文件，把第27行：t_mul=1. 注释掉。
3. 修改data文件夹下imagenet22k_dataset.py，把第28行：print("ERROR IMG LOADED: ", path) 注释掉。
4. 修改data文件夹下的build.py文件，把第112行：prefix = 'ILSVRC2011fall_whole'，改为prefix = 'ILSVRC2021winner21k_whole'。
5. 在Swin-Transformer目录下创建requirements.txt指定python依赖库：
   ```
   # requirements.txt内容如下

   timm==0.4.12
   termcolor==1.1.0
   yacs==0.1.8
   ```
6. 准备run.sh文件中所需要的obs文件路径。

   a. 准备imagenet数据集的分享链接。

      勾选要分享的imagenet21k_whole数据集文件夹，单击分享按钮，选择分享链接有效期，自定义提取码，例如123456，单击“复制链接”，记录该链接。

   b. 准备“obsutil_linux_amd64.tar.gz”的分享链接。

      参考**下载和安装obsutil**下载“obsutil_linux_amd64.tar.gz”，将其上传至OBS桶中，设置为公共读。单击属性，单击复制链接。

      链接样例如下：

```
https://${bucketname_name}.obs.cn-north-4.myhuaweicloud.com/${folders_name}/pytorch.tar.gz
```

7. 在Swin-Transformer目录下，创建运行脚本run.sh。

   说明

   - 脚本中的"SRC_DATA_PATH=${imagenet数据集在obs中分享链接}"，需要替换为上一步中的imagenet21k_whole文件夹分享链接。
   - 脚本中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz"，需要替换为上一步中obsutil_linux_amd64.tar.gz在OBS上的路径（需将文件设置为公共读）。

   单机单卡运行脚本：

```
# 在代码主目录下创建一个run.sh，内容如下

#!/bin/bash

# 从obs中下载数据到本地SSD盘
DIS_DATA_PATH=/cache
SRC_DATA_PATH=${imagenet数据集在obs中分享链接}
OBSUTIL_PATH=https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz
mkdir -p $DIS_DATA_PATH && cd $DIS_DATA_PATH && wget $OBSUTIL_PATH && tar -xzvf obsutil_linux_amd64.tar.gz && $DIS_DATA_PATH/obsutil_linux_amd64*/obsutil share-cp $SRC_DATA_PATH $DIS_DATA_PATH/ -ac=123456 -r -f -j 256 && cd -
IMAGE_DATA_PATH=$DIS_DATA_PATH/imagenet21k_whole

# 模型训练过程中模型权重、训练配置等的保存路径
OUTPUT_PATH=/cache/output

MASTER_PORT="6061"

/home/ma-user/anaconda3/envs/pytorch/bin/python -m torch.distributed.launch --nproc_per_node=1 --master_addr localhost --master_port=$MASTER_PORT main.py --data-path $IMAGE_DATA_PATH --output $OUTPUT_PATH --cfg ./configs/swin/swin_base_patch4_window7_224_22k.yaml --local_rank 0
```

   多机多卡运行脚本：

```
# 创建run.sh

#!/bin/bash

# 从obs中下载数据到本地SSD盘
DIS_DATA_PATH=/cache
SRC_DATA_PATH=${imagenet数据集在obs中分享链接}
OBSUTIL_PATH=https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz
mkdir -p $DIS_DATA_PATH && cd $DIS_DATA_PATH && wget $OBSUTIL_PATH && tar -xzvf obsutil_linux_amd64.tar.gz && $DIS_DATA_PATH/obsutil_linux_amd64*/obsutil share-cp $SRC_DATA_PATH $DIS_DATA_PATH/ -ac=123456 -r -f -j 256 && cd -
IMAGE_DATA_PATH=$DIS_DATA_PATH/imagenet21k_whole

# 模型训练过程中模型权重、训练配置等的保存路径
OUTPUT_PATH=/cache/output

MASTER_ADDR=$(echo ${VC_WORKER_HOSTS} | cut -d "," -f 1)
MASTER_PORT="6060"
NNODES="$VC_WORKER_NUM"
NODE_RANK="$VC_TASK_INDEX"
NGPUS_PER_NODE="$MA_NUM_GPUS"

/home/ma-user/anaconda3/envs/pytorch/bin/python -m torch.distributed.launch --nnodes=$NNODES --node_rank=$NODE_RANK --nproc_per_node=$NGPUS_PER_NODE --master_addr $MASTER_ADDR --master_port=$MASTER_PORT main.py --data-path $IMAGE_DATA_PATH --output=$OUTPUT_PATH --cfg ./configs/swin/swin_base_patch4_window7_224_22k.yaml
```


说明

- 推荐先使用单机单卡运行脚本，待正常运行后再改用多机多卡运行脚本。
- 多机多卡run.sh中的“VC_WORKER_HOSTS”、“VC_WORKER_NUM”、“VC_TASK_INDEX”、“MA_NUM_GPUS”为ModelArts训练容器中预置的环境变量。训练容器环境变量详细介绍可参考**查看训练容器环境变量**。
- run.sh中的OUTPUT_PATH是训练过程中保存模型权重、训练配置等中间结果的路径。如果训练脚本中“config.TRAIN.AUTO_RESUME”为“True”（默认值为True），在开始训练时会自动在OUTPUT_PATH路径中加载最新的模型权重。

8. 通过obsutils，将代码文件夹放到OBS上，然后通过OBS将代码传至SFS相应目录中。
9. 在SFS中将代码文件Swin-Transformer-main设置归属为ma-user。
   ```
   chown -R ma-user:ma-group Swin-Transformer
   ```
10. 执行以下命令，去除Shell脚本的\r字符。
    ```
    cd Swin-Transformer
    sed -i 's/\r//' run.sh
    ```

    说明

    Shell脚本在Windows系统编写时，每行结尾是\r\n，而在Linux系统中行每行结尾是\n，所以在Linux系统中运行脚本时，会认为\r是一个字符，导致运行报错“$'\r': command not found”，因此需要去除Shell脚本的\r字符。

使用 notebook 进行代码调试

由于Notebook的/cache目录只能支持500G的存储，超过后会导致实例重启，ImageNet数据集大小超过该限制，因此建议用线下资源调试、或用小批量数据集在Notebook调试（Notebook调试方法请参见**使用Notebook进行代码调试**）。

创建多机多卡训练作业

1. 登录**ModelArts管理控制台**，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     cd /home/ma-user/work/code/Swin-Transformer && /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh run.sh
     ```
   - 自动重启：打开自动重启，配置重启次数，这样当节点发生故障后，平台会自动完成作业重启和故障节点隔离。

     如果打开自动重启，需要在“run.sh”脚本中配置output输出路径，从而保障当发生故障时，能基于上一轮保存的模型权重继续训练，最大化减少资源的浪费。
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择所需GPU规格。
   - 计算节点个数：选择需要的节点个数。

- SFS Turbo：增加挂载配置，选择SFS名称，云上挂载路径为“/home/ma-user/work”。

  说明

  为了和Notebook调试时代码路径一致，保持相同的启动命令，云上挂载路径需要填写为“/home/ma-user/work”。

4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

8 Standard 推理部署

8.1 ModelArts Standard 推理服务访问公网方案

本章节提供了推理服务访问公网的方法。

应用场景

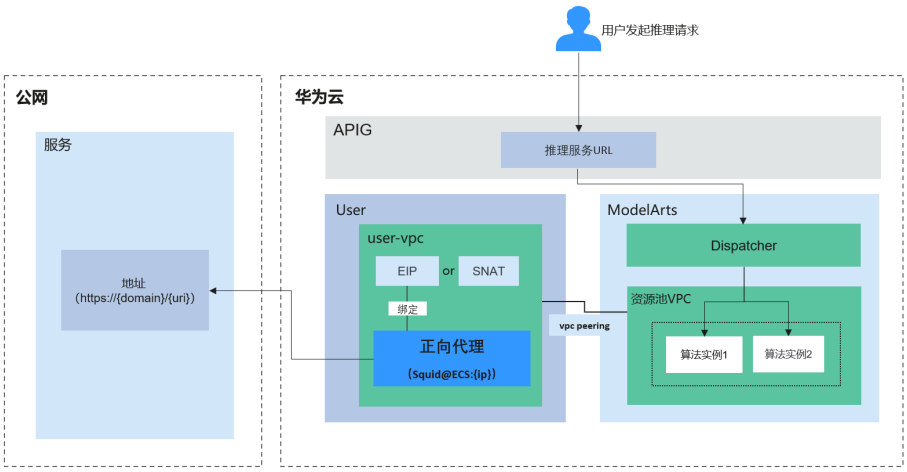
推理服务访问公网地址的场景，如：

- 输入图片，先进行公网OCR服务调用，然后进行NLP处理；
- 进行公网文件下载，然后进行分析；
- 分析结果回调给公网服务终端。

方案设计

从推理服务的算法实例内部，访问公网服务地址的方案。如下图所示：

图 8-1 推理服务访问公网



步骤一：ModelArts 专属资源池打通 VPC

1. 创建好VPC和子网，具体步骤请参考**创建虚拟私有云和子网**。

2. 创建Modelarts专属资源池网络。
   a. 登录**ModelArts管理控制台**，在左侧导航栏中选择“资源管理 > 网络管理”。
   b. 单击“创建网络”，弹出“创建网络”页面。
   c. 在“创建网络”弹窗中填写网络信息。
   d. 确认无误后，单击“确定”。
3. Modelarts专属资源池网络打通VPC。
   a. 在控制台左侧导航栏中选择“资源管理 > 网络管理”。
   b. 选择上一步骤创建的网络，单击操作列的“打通VPC”。

      图 8-2 打通 VPC

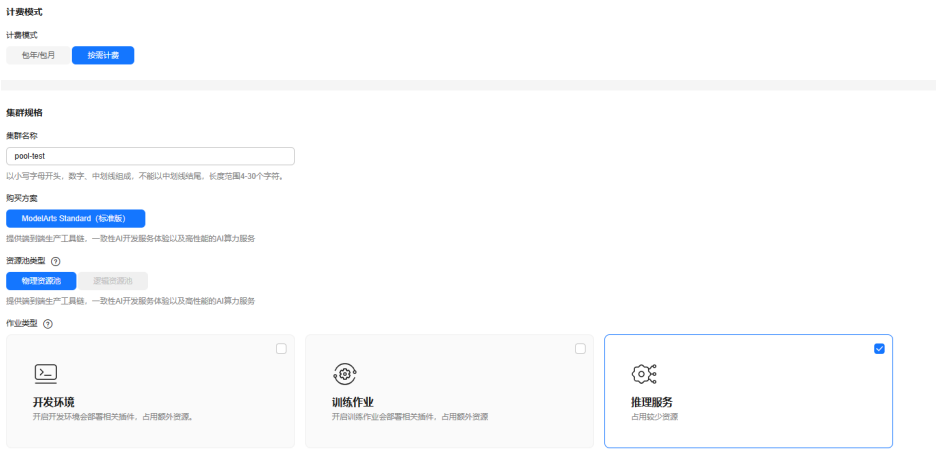
      

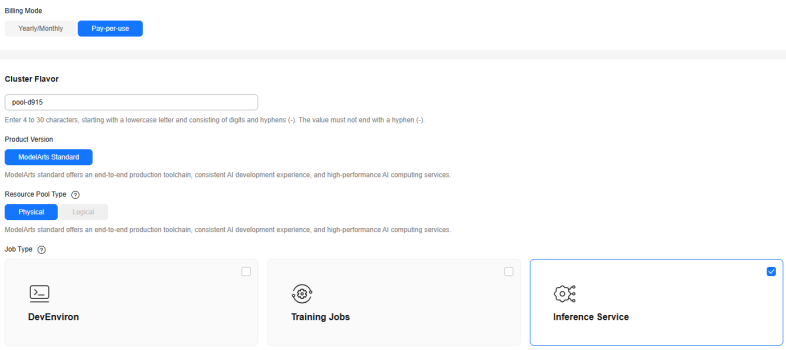
   c. 在打通VPC弹框中，打开“打通VPC”开关，在下拉框中选择提前创建好的VPC和子网。

      说明

      需要打通的对端网络不能和当前网段重叠。
4. 创建Modelarts专属资源池。
   a. 在控制台左侧导航栏中选择“资源管理 > 标准算力集群（Standard Cluster）”。
   b. 单击“购买标准算力集群”，进入购买标准算力集群界面填写参数。
      “作业类型”包括推理服务。“网络”选择上文中已打通VPC的网络。

      图 8-3 作业类型



c. 单击“立即购买”确认规格。产品规格和协议许可确认无误后，单击“提交”，即可创建专属资源池。

步骤二：使用 Docker 安装和配置正向代理

1. 购买弹性云服务器ECS，详情请见**购买ECS**。镜像可选择Ubuntu最新版本。虚拟私有云选择提前创建好的VPC。
2. 申请弹性公网IP EIP，详情请见**申请弹性公网IP**。
3. 将弹性公网IP绑定到ECS，详情请见**将弹性公网IP绑定至实例**。
4. 登录ECS，执行如下命令进行Docker安装。如已安装，请直接进入下一步。
   ```
   curl -sSL https://get.daocloud.io/docker | sh
   ```
5. 执行如下命令安装Squid容器。
   ```
   docker pull ubuntu/squid
   ```
6. 创建主机目录。
   ```
   mkdir –p /etc/squid/
   ```
7. 打开并配置whitelist.conf文件。
   ```
   vim whitelist.conf
   ```
   配置内容为安全控制可访问的地址，支持配置通配符，例如：
   ```
   .apig.cn-east-3.huaweicloudapis.com
   ```
   说明

   如果地址访问不通，请在浏览器配置访问域名。
8. 打开并配置squid.conf文件。
   ```
   vim squid.conf
   ```
   配置内容如下。
   ```
   # An ACL named 'whitelist'
   acl whitelist dstdomain '/etc/squid/whitelist.conf'

   # Allow whitelisted URLs through
   http_access allow whitelist

   # Block the rest
   http_access deny all

   # Default port
   http_port 3128
   ```
9. 设置主机目录和配置文件权限如下。
   ```
   chmod 640 -R /etc/squid
   ```
10. 执行如下命令启动Squid实例。
    ```
    docker run -d --name squid -e TZ=UTC -v /etc/squid:/etc/squid -p 3128:3128 ubuntu/squid:latest
    ```
11. 进入docker刷新Squid。
    ```
    docker exec –it squid bash
    root@{container_id}:/# squid -k reconfigure
    ```

步骤三：设置 DNS 代理和调用公网地址

1. 在自定义模型镜像时设置代理指向代理服务器私有IP和端口，如下所示。

```
proxies = {
  "http": "http://{proxy_server_private_ip}:3128",
  "https": "http://{proxy_server_private_ip}:3128"
}
```

代理服务器IP即**步骤二：使用Docker安装和配置正向代理**中创建的ECS私有IP，获取方式请见**查看弹性云服务器详细信息**。

图 8-4 ECS 私有 IP

2. 调用公网地址时，使用服务URL进行业务请求，如：

```
https://e8a048ce25136addbbac23ce6132a.apig.cn-east-3.huaweicloudapis.com
```

8.2 端到端运维 ModelArts Standard 推理服务方案

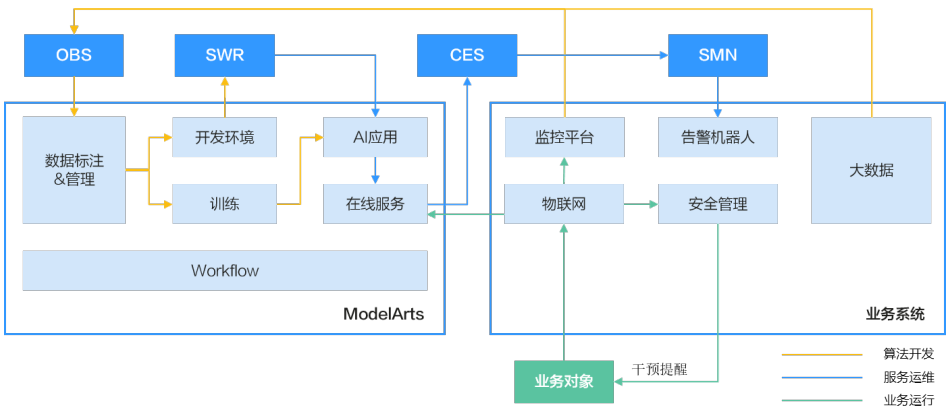
ModelArts推理服务的端到端运维覆盖了算法开发、服务运维和业务运行的整个AI流程。

方案概述

推理服务的端到端运维流程

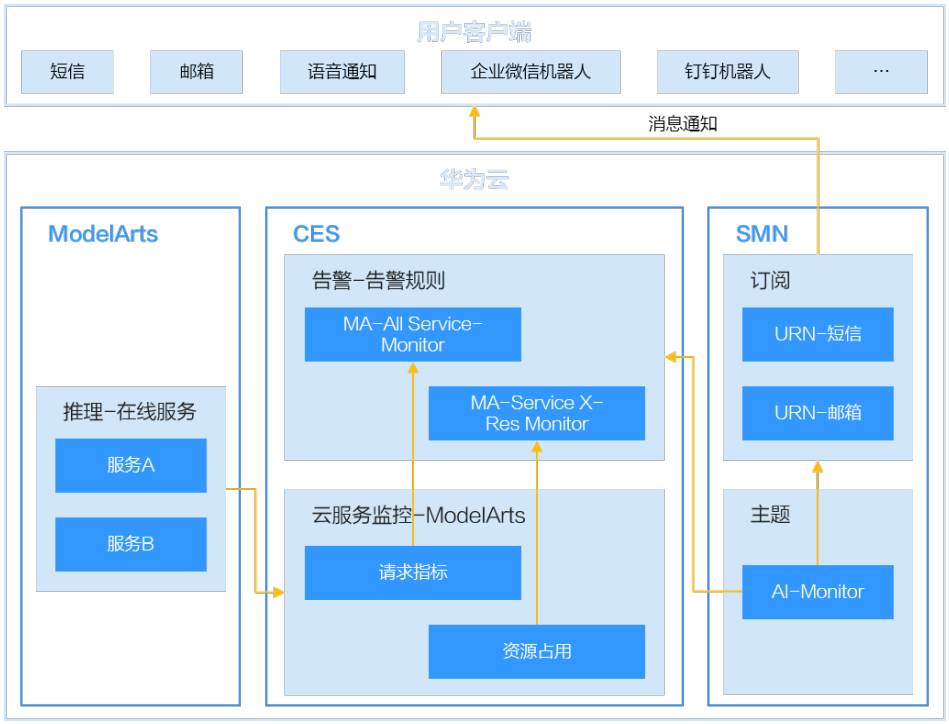
- 算法开发阶段，先将业务AI数据存放到对象存储服务（OBS）中，接着通过ModelArts数据管理进行标注和版本管理，然后通过训练获得AI模型结果，最后通过开发环境构建模型镜像。
- 服务运维阶段，先利用镜像构建模型，接着部署模型为在线服务，然后可在云监控服务（CES）中获得ModelArts推理在线服务的监控数据，最后可配置告警规则实现实时告警通知。
- 业务运行阶段，先将业务系统对接在线服务请求，然后进行业务逻辑处理和监控设置。

图 8-5 推理服务的端到端运维流程图



整个运维过程会对服务请求失败和资源占用过高的场景进行监控，当超过阈值时发送告警通知。

图 8-6 监控告警流程图



方案优势

通过端到端的服务运维配置，可方便地查看业务运行高低峰情况，并能够实时感知在线服务的健康状态。

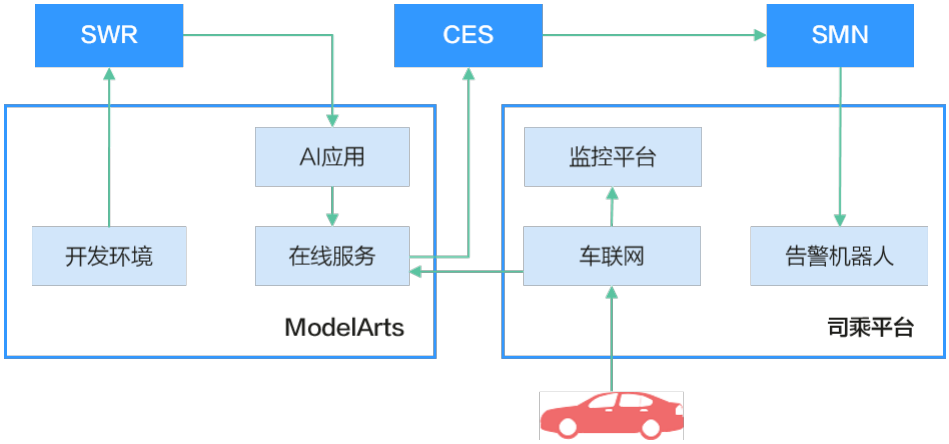
约束限制

端到端服务运维只支持在线服务，因为推理的批量服务和边缘服务无CES监控数据，不支持完整的端到端服务运维设置。

实施步骤

以出行场景的司乘安全算法为例，介绍使用ModelArts进行流程化服务部署和更新、自动化服务运维和监控的实现步骤。

图 8-7 司乘安全算法



步骤1 将用户本地开发完成的模型，使用自定义镜像构建成ModelArts Standard推理平台可以用的模型。具体操作请参考**从0-1制作自定义镜像并创建模型**。

步骤2 在**ModelArts管理控制台**，使用创建好的模型部署为在线服务。

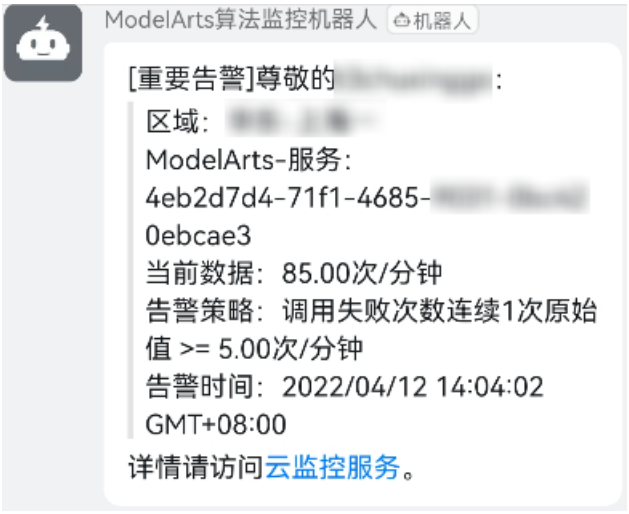
步骤3 登录**云监控服务控制台**，在云监控控制台设置ModelArts服务的告警规则并配置主题订阅方式发送通知。具体操作请参考**设置告警规则**。

当配置完成后，在左侧导航栏选择“云服务监控 > ModelArts”即可查看在线服务的请求情况和资源占用情况，如下图所示。

图 8-8 查看服务的监控指标

当监控信息触发告警时，主题订阅对象将会收到消息通知。

图 8-9 告警消息通知



----结束

8.3 使用自定义引擎在 ModelArts Standard 创建模型

使用自定义引擎创建模型，用户可以通过选择自己存储在SWR服务中的镜像作为模型的引擎，指定预先存储于OBS服务中的文件目录路径作为模型包来创建模型，轻松地应对ModelArts平台预置引擎无法满足个性化诉求的场景。

自定义引擎创建模型的规范

使用自定义引擎创建模型，用户的SWR镜像、OBS模型包和文件大小需要满足以下规范：

- SWR镜像规范：
  - 镜像必须内置一个用户名为“ma-user”，组名为“ma-group”的普通用户，且必须确保该用户的uid=1000、gid=100。内置用户的dockerfile指令如下：

    ```
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
    ```

  - 明确设置镜像的启动命令。在dockerfile文件中指定cmd，dockerfile指令示例如下：

    ```
    CMD sh /home/mind/run.sh
    ```

    启动入口文件run.sh需要自定义。示例如下：

    ```
    #!/bin/bash

    # 自定义脚本内容
    ...

    # run.sh调用app.py启动服务器，app.py请参考https示例
    python app.py
    ```

    说明

    除了按上述要求设置启动命令，您也可以在镜像中自定义启动命令，在创建模型时填写与您镜像中相同的启动命令。

  - 提供的服务可使用HTTPS/HTTP协议和监听的容器端口，端口和协议可根据镜像实际使用情况自行填写，ModelArts提供的请求协议和端口号的缺省值是HTTPS和8080。请参考**https示例**。
  - （可选）健康检查的URL路径必须为"/health"。
- OBS模型包规范

  模型包的名字必须为model。模型包规范请参见**模型包规范介绍**。
- 文件大小规范

  当使用公共资源池时，SWR的镜像大小（指下载后的镜像大小，非SWR界面显示的压缩后的镜像大小）和OBS模型包大小总和不大于30G。

https 示例

使用Flask启动https，Webserver代码示例如下：

```
from flask import Flask, request
import json

app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----------- in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----------- in goodbye func ----------")
    return '\nGoodbye!\n'


@app.route('/', methods=['POST'])
def default_func():
    print("----------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

@app.route('/health', methods=['GET'])
def healthy():
    return "{\"status\": \"OK\"}"

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080, ssl_context='adhoc')
```

在本地机器调试

自定义引擎的规范可以在安装有docker的本地机器上通过以下步骤提前验证：

1. 将自定义引擎镜像下载至本地机器，假设镜像名为custom_engine:v1。
2. 将模型包文件夹复制到本地机器，假设模型包文件夹名字为model。
3. 在模型包文件夹的同级目录下验证如下命令拉起服务：

   ```
   docker run --user 1000:100 -p 8080:8080 -v model:/home/mind/model    custom_engine:v1
   ```

   说明

   该指令无法完全模拟线上，主要是由于-v挂载进去的目录是root权限。在线上，模型文件从OBS下载到/home/mind/model目录之后，文件owner将统一修改为ma-user。

4. 在本地机器上启动另一个终端，执行以下验证指令，得到符合预期的推理结果。
```
curl https://127.0.0.1:8080/${推理服务的请求路径}
```

推理部署示例

本节将详细说明以自定义引擎方式创建模型的步骤。

1. 创建模型并查看模型详情

   登录**ModelArts管理控制台**，进入“模型管理”页面中，单击“创建模型”，进入模型创建页面，设置相关参数如下：

   - 元模型来源：选择“从对象存储服务（OBS）中选择”。
   - 选择元模型：从OBS中选择一个模型包。
   - AI引擎：选择“Custom”。
   - 引擎包：从容器镜像中选择一个镜像。
   - 容器调用接口：端口和协议可根据镜像实际使用情况自行填写。

     其他参数保持默认值。

   单击“立即创建”，跳转到模型列表页，查看模型状态，当状态变为“正常”，模型创建成功。

   图 8-10 创建模型

   单击模型名称，进入模型详情页面，查看模型详情信息。

2. 部署服务并查看详情

   在模型详情页面，单击右上角“部署>在线服务”，进入服务部署页面，模型和版本默认选中，选择合适的“实例规格”（例如CPU：2核 8GB），其他参数可保持默认值，单击“下一步”，跳转至服务列表页，当服务状态变为“运行中”，服务部署成功。

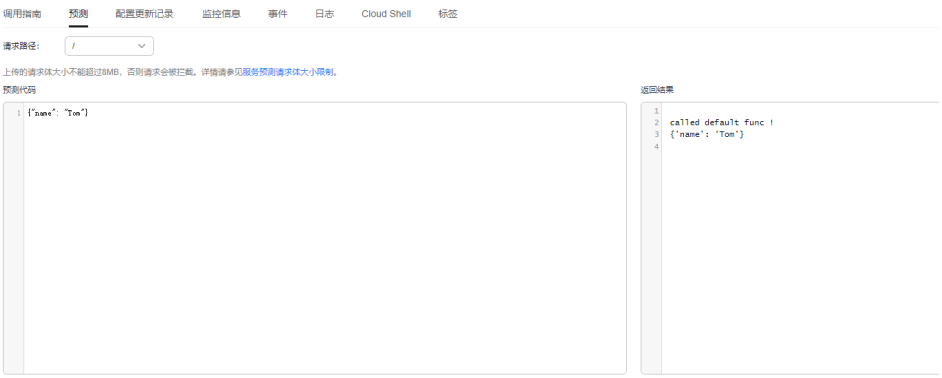
   单击服务名称，进入服务详情页面，查看服务详情信息，单击“日志”页签，查看服务日志信息。

图 8-11 查看服务日志信息

3. 服务预测

   在服务详情页面，单击“预测”页签，进行服务预测。

图 8-12 服务预测



8.4 使用大模型在 ModelArts Standard 创建模型部署在线服务

背景说明

目前大模型的参数量已经达到千亿甚至万亿，随之大模型的体积也越来越大。千亿参数大模型的体积超过200G，在版本管理、生产部署上对平台系统产生了新的要求。例如：导入模型时，需要支持动态调整租户存储配额；模型加载、启动慢，部署时需要灵活的超时配置；当负载异常重启，模型需要重新加载，服务恢复时间长的问题亟待解决。

为了应对如上诉求，ModelArts推理平台针对性给出解决方案，用于支持大模型场景下的模型管理和服务部署。

约束与限制

- 需要申请单个模型大小配额和添加使用节点本地存储缓存的白名单。
- 需要使用自定义引擎Custom，配置动态加载。
- 需要使用专属资源池部署服务。
- 专属资源池磁盘空间需大于1T。

操作事项

1. 申请扩大模型的大小配额和使用节点本地存储缓存白名单
2. 上传模型数据并校验上传对象的一致性
3. 创建专属资源池
4. 创建模型
5. 部署在线服务

申请扩大模型的大小配额和使用节点本地存储缓存白名单

服务部署时，默认情况下，动态加载的模型包位于临时磁盘空间，服务停止时已加载的文件会被删除，再次启动时需要重新加载。为了避免反复加载，平台允许使用资源池节点的本地存储空间来加载模型包，并在服务停止和重启时仍有效（通过哈希值保证数据一致性）

使用大模型要求用户采用自定义引擎，并开启动态加载的模式导入模型。基于此，需要执行以下操作：

- 如果模型超过默认配额值，需要提工单申请扩大单个模型的大小配额。单个模型大小配额默认值为20GB。
- 需要提工单申请添加使用节点本地存储缓存的白名单。

上传模型数据并校验上传对象的一致性

为了动态加载时保证数据完整性，需要在上传模型数据至OBS时，进行上传对象的一致性校验。obsutil、OBS Browser+以及OBS SDK都支持在上传对象时进行一致性校验，您可以根据自己的业务选择任意一种方式进行校验。详见**校验上传对象的一致性**。

以OBS Browser+为例，如**图8-13**。使用OBS Browser+上传数据，开启MD5校验，动态加载并使用节点本地的持久化存储时，检查数据一致性。

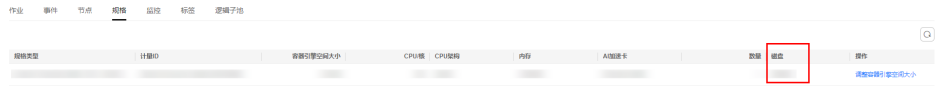
图 8-13 OBS Browser+配置 MD5 校验

创建专属资源池

使用本地的持久化存储功能，需使用专属资源池，且专属资源池磁盘空间大小必须超过1T。您可以通过专属资源池详情页面，规格页签，查看专属资源池磁盘信息。当服

务部署失败，提示磁盘空间不足时，请参考**服务部署、启动、升级和修改时，资源不足如何处理?**

图 8-14 查看专属资源池磁盘信息



创建模型

使用大模型创建模型，选择从对象存储服务（OBS）中导入，需满足以下参数配置：

1. 采用自定义引擎，开启动态加载

   使用大模型要求用户使用自定义引擎，并开启动态加载的模式导入模型。用户可以制作自定义引擎，满足大模型场景下对镜像依赖包、推理框架等的特殊需求。自定义引擎的制作请参考**使用自定义引擎在ModelArts Standard创建模型**。

   当用户使用自定义引擎时，默认开启动态加载，模型包与镜像分离，在服务部署时动态将模型加载到服务负载。

2. 配置健康检查

   大模型场景下导入的模型，要求配置健康检查，避免在部署时服务显示已启动但实际不可用。

   图 8-15 采用自定义引擎，开启动态加载并配置健康检查示例图

部署在线服务

部署服务时，需满足以下参数配置：

1. 自定义部署超时时间

   大模型加载启动的时间一般大于普通的模型创建的服务，请配置合理的“部署超时时间”，避免尚未启动完成被认为超时而导致部署失败。

2. 添加环境变量

   部署服务时，增加如下环境变量，会将负载均衡的请求亲和策略配置为集群亲和，避免未就绪的服务实例影响预测成功率。

   ```
   MODELARTS_SERVICE_TRAFFIC_POLICY: cluster
   ```

图 8-16 自定义部署超时时间和添加环境变量示例图

建议部署多实例，增加服务可靠性。

8.5 第三方推理框架迁移到 ModelArts Standard 推理自定义引擎

背景说明

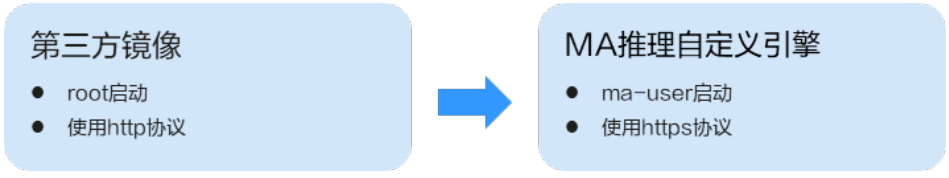
ModelArts支持第三方的推理框架在ModelArts上部署，本文以TFServing框架、Triton框架为例，介绍如何迁移到推理自定义引擎。

- TensorFlow Serving是一个灵活、高性能的机器学习模型部署系统，提供模型版本管理、服务回滚等能力。通过配置模型路径、模型端口、模型名称等参数，原生TFServing镜像可以快速启动提供服务，并支持gRPC和HTTP Restful API的访问方式。
- Triton是一个高性能推理服务框架，提供HTTP/gRPC等多种服务协议，支持TensorFlow、TensorRT、PyTorch、ONNXRuntime等多种推理引擎后端，并且支持多模型并发、动态batch等功能，能够提高芯片的使用率，改善推理服务的性能。

当从第三方推理框架迁移到使用ModelArts推理的模型管理和服务管理时，需要对原生第三方推理框架镜像的构建方式做一定的改造，以使用ModelArts推理平台的模型版本管理能力和动态加载模型的部署能力。本案例将指导用户完成原生第三方推理框架镜像到ModelArts推理自定义引擎的改造。自定义引擎的镜像制作完成后，即可以通过模型导入对模型版本进行管理，并基于模型进行部署和管理服务。

适配和改造的主要工作项如下：

图 8-17 改造工作项



针对不同框架的镜像，可能还需要做额外的适配工作，具体差异请见对应框架的操作步骤。

- **TFServing框架迁移操作步骤**
- **Triton框架迁移操作步骤**

TFServing 框架迁移操作步骤

步骤1 增加用户ma-user。

基于原生"tensorflow/serving:2.8.0"镜像构建，镜像中100的用户组默认已存在，Dockerfile中执行如下命令增加用户ma-user。

```
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

步骤2 通过增加nginx代理，支持https协议。

协议转换为https之后，对外暴露的端口从tfserving的8501变为8080。

1. Dockerfile中执行如下命令完成nginx的安装和配置。

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

2. 准备nginx目录如下：

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. 准备nginx.conf文件内容如下：

```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
```

```
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. 准备modelarts-model-server.conf配置文件内容如下：

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8501;
    }
}
```

5. 准备启动脚本。

说明

启动前先创建ssl证书，然后启动TFServing的启动脚本。

启动脚本run.sh示例代码如下：

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
```

```
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText
sh /usr/bin/tf_serving_entrypoint.sh
```

步骤3 修改模型默认路径，支持ModelArts推理模型动态加载。

Dockerfile中执行如下命令修改默认的模型路径。

```
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
```

----结束

完整的Dockerfile参考：

```
FROM tensorflow/serving:2.8.0
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

Triton 框架迁移操作步骤

本教程基于nvidia官方提供的nvcr.io/nvidia/tritonserver:23.03-py3镜像进行适配，使用开源大模型llama7b进行推理任务。

步骤1 增加用户ma-user。

Triton镜像中默认已存在id为1000的triton-server用户，需先修改triton-server用户名id后再增加用户ma-user，Dockerfile中执行如下命令。

```
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

步骤2 通过增加nginx代理，支持https协议。

1. Dockerfile中执行如下命令完成nginx的安装和配置。

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
```

2. 准备nginx目录如下：

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. 准备nginx.conf文件内容如下：

```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. 准备modelarts-model-server.conf配置文件内容如下：

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
```

```
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

5. 准备启动脚本run.sh。

   说明

   启动前先创建ssl证书，然后启动Triton的启动脚本。

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/
O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/
OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText

bash /home/mind/model/triton_serving.sh
```

步骤3 编译安装tensorrtllm_backend。

1. Dockerfile中执行如下命令获取tensorrtllm_backend源码，安装tensorrt、cmake和pytorch等相关依赖，并进行编译安装。

```
# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt

# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
```

```
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver
```

2. 准备triton serving的启动脚本triton_serving.sh，llama模型的参考样例如下：

```
MODEL_NAME=llama_7b
MODEL_DIR=/home/mind/model/${MODEL_NAME}
OUTPUT_DIR=/tmp/llama/7B/trt_engines/fp16/1-gpu/
MAX_BATCH_SIZE=1
export LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH}

# build tensorrt_llm engine
cd /opt/tritonserver/tensorrtllm_backend/tensorrt_llm/examples/llama
python build.py --model_dir ${MODEL_DIR} \
                --dtype float16 \
                --remove_input_padding \
                --use_gpt_attention_plugin float16 \
                --enable_context_fmha \
                --use_weight_only \
                --use_gemm_plugin float16 \
                --output_dir ${OUTPUT_DIR} \
                --paged_kv_cache \
                --max_batch_size ${MAX_BATCH_SIZE}

# set config parameters
cd /opt/tritonserver/tensorrtllm_backend
mkdir triton_model_repo
cp all_models/inflight_batcher_llm/* triton_model_repo/ -r

python3 tools/fill_template.py -i triton_model_repo/preprocessing/config.pbtxt tokenizer_dir:$
{MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:$
{MAX_BATCH_SIZE},preprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/postprocessing/config.pbtxt tokenizer_dir:$
{MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:$
{MAX_BATCH_SIZE},postprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/ensemble/config.pbtxt triton_max_batch_size:$
{MAX_BATCH_SIZE}
python3 tools/fill_template.py -i triton_model_repo/tensorrt_llm/config.pbtxt triton_max_batch_size:$
{MAX_BATCH_SIZE},decoupled_mode:False,max_beam_width:1,engine_dir:$
{OUTPUT_DIR},max_tokens_in_paged_kv_cache:2560,max_attention_window_size:2560,kv_cache_free_
gpu_mem_fraction:0.5,exclude_input_in_output:True,enable_kv_cache_reuse:False,batching_strategy:V1,
max_queue_delay_microseconds:600

# launch tritonserver
python3 scripts/launch_triton_server.py --world_size 1 --model_repo=triton_model_repo/
while true; do sleep 10000; done
```

部分参数说明：

- MODEL_NAME：HuggingFace格式模型权重文件所在OBS文件夹名称。
- OUTPUT_DIR：通过TensorRT-LLM转换后的模型文件在容器中的路径。

完整的Dockerfile如下：

```
FROM nvcr.io/nvidia/tritonserver:23.03-py3

# add ma-user and install nginx
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash
ma-user && \
    apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
```

```
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx

# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt

# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver

ADD nginx /etc/nginx
ADD run.sh /home/mind/
CMD /bin/bash /home/mind/run.sh
```

完成镜像构建后，将镜像注册至华为云容器镜像服务SWR中，用于后续在ModelArts上部署推理服务。

步骤4 使用适配后的镜像在ModelArts部署在线推理服务。

1. 在obs中创建model目录，并将triton_serving.sh文件和llama_7b文件夹上传至model目录下，如下图所示。

图 8-18 上传至 model 目录

2. 创建模型，源模型来源选择“从对象存储服务（OBS）中选择”，元模型选择至model目录，AI引擎选择Custom，引擎包选择步骤3构建的镜像。

图 8-19 创建模型

3. 将创建的模型部署为在线服务，大模型加载启动的时间一般大于普通的模型创建的服务，请配置合理的“部署超时时间”，避免尚未启动完成被认为超时而导致部署失败。

图 8-20 部署为在线服务

4. 调用在线服务进行大模型推理，请求路径填写/v2/models/ensemble/infer，调用样例如下：

```
{
  "inputs": [
    {
      "name": "text_input",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": ["what is machine learning"]
    },
    {
      "name": "max_tokens",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [64]
    },
    {
      "name": "bad_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "stop_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "pad_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    },
    {
      "name": "end_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    }
  ],
  "outputs": [
    {
      "name": "text_output"
    }
  ]
}
```

说明

- "inputs"中"name"为"text_input"的元素代表输入，"data"为具体输入语句，本示例中为"what is machine learning"。
- "inputs"中"name"为"max_tokens"的元素代表输出最大tokens数，"data"为具体数值，本示例中为64。

图 8-21 调用在线服务

----结束

8.6 ModelArts Standard 推理服务支持 VPC 直连的高速访问通道配置

背景说明

访问在线服务的实际业务中，用户可能会存在如下需求：

- 高吞吐量、低时延
- TCP或者RPC请求

因此，ModelArts提供了VPC直连的高速访问通道功能以满足用户的需求。

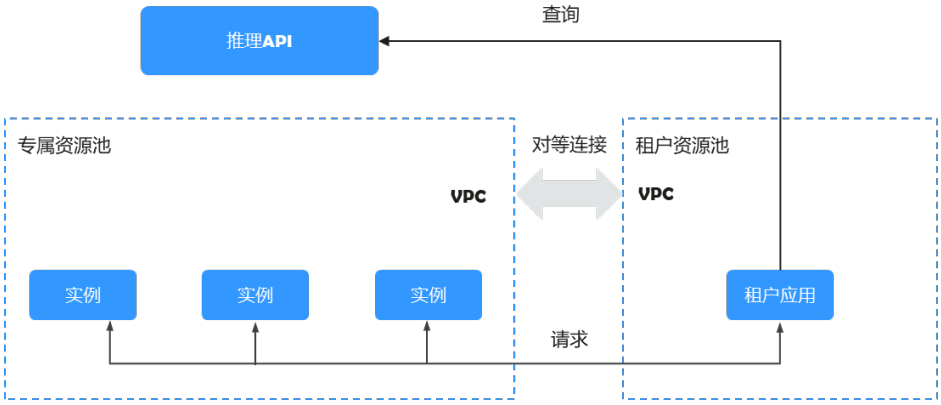
使用VPC直连的高速访问通道，用户的业务请求不需要经过推理平台，而是直接经VPC对等连接发送到实例处理，访问速度更快。此访问方式适用于需要高带宽和低延迟的场景，例如实时数据处理、视频流传输等。

警告

由于请求不经过推理平台，所以通过VPC高速访问通道的方式访问在线服务会丢失以下功能：

- 认证鉴权
- 流量按配置分发
- 负载均衡
- 告警、监控和统计

图 8-22 VPC 直连的高速访问通道示意图



约束限制

调用API访问在线服务时，对预测请求体大小和预测时间有限制：

- 请求体的大小不超过12MB，超过后请求会被拦截。
- 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。
- 只有专属资源池部署的服务才支持VPC直连的高速访问通道。
- VPC直连的高速访问通道，目前只支持访问在线服务。
- 因流量限控，获取在线服务的IP和端口号次数有限制，每个主账号租户调用次数不超过2000次/分钟，每个子账号租户不超过20次/分钟。
- 目前仅支持自定义镜像导入模型，部署的服务支持高速访问通道。

准备工作

使用专属资源池部署在线服务，服务状态为“运行中”。

操作步骤

使用VPC直连的高速访问通道访问在线服务，基本操作步骤如下：

1. **将专属资源池的网络打通VPC**
2. **VPC下创建弹性云服务器**
3. **获取在线服务的IP和端口号**
4. **通过IP和端口号直连应用**

步骤1 将专属资源池的网络打通VPC

登录ModelArts控制台，进入“资源管理 > 标准算力集群（Standard Cluster）”找到服务部署使用的专属资源池，单击“名称/ID”，进入资源池详情页面，查看网络配置信息。返回控制台，左侧导航栏选择“网络管理”，找到专属资源池关联的网络，打通VPC。打通VPC网络后，网络列表和资源池详情页面将显示VPC名称，单击后可以跳转至VPC详情页面。

图 8-23 查看网络配置

图 8-24 打通 VPC

步骤2 VPC下创建弹性云服务器

登录弹性云服务器ECS控制台，单击右上角“购买弹性云服务器”，进入购买弹性云服务器页面，完成基本配置后单击“下一步：网络配置”，进入网络配置页面，选择**步骤1**中打通的VPC，完成其他参数配置，完成高级配置并确认配置，下发购买弹性云服务器的任务。等待服务器的状态变为“运行中”时，弹性云服务器创建成功。单击“名称/ID”，进入服务器详情页面，查看虚拟私有云配置信息。

图 8-25 购买弹性云服务器时选择 VPC

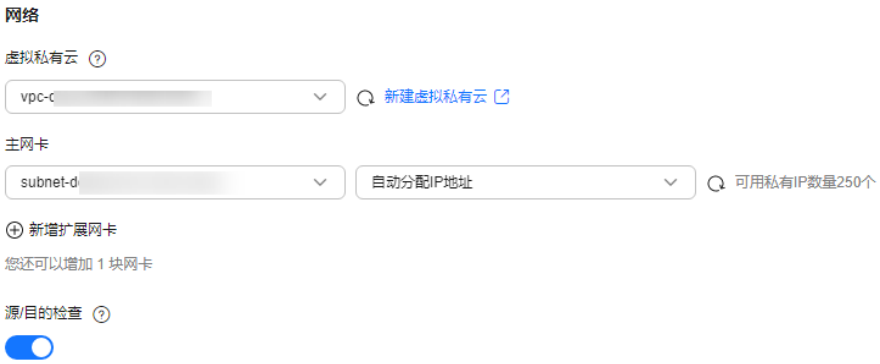


图 8-26 查看虚拟私有云配置信息

步骤3 获取在线服务的IP和端口号

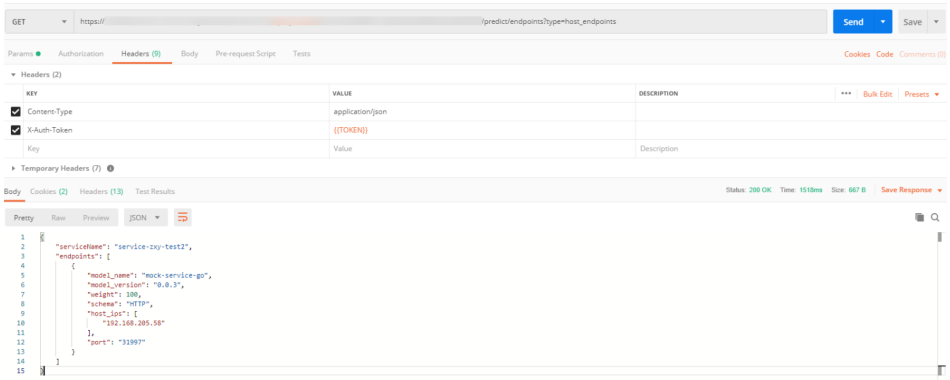
可以通过使用图形界面的软件（以Postman为例）获取服务的IP和端口号，也可以登录弹性云服务器（ECS），创建Python环境运行代码，获取服务IP和端口号。

API接口：

```
GET /v1/{project_id}/services/{service_id}/predict/endpoints?type=host_endpoints
```

- 方式一：图形界面的软件获取服务的IP和端口号

  图 8-27 接口返回示例

- 方式二：Python语言获取IP和端口号

  Python代码如下，下述代码中以下参数需要手动修改：

  - project_id：用户项目ID，获取方法请参见**获取项目ID和名称**。
  - service_id：服务ID，在服务详情页可查看。
  - REGION_ENDPOINT：服务的终端节点，查询请参见**终端节点**。

  ```
  def get_app_info(project_id, service_id):
      list_host_endpoints_url = "{}/v1/{}/services/{}/predict/endpoints?type=host_endpoints"
  ```

```
url = list_host_endpoints_url.format(REGION_ENDPOINT, project_id, service_id)
headers = {'X-Auth-Token': X_Auth_Token}
response = requests.get(url, headers=headers)
print(response.content)
```

步骤4 通过IP和端口号直连应用

登录弹性云服务器（ECS），可以通过Linux命令行访问在线服务，也可以创建Python环境运行Python代码访问在线服务。schema、 ip、port参数值从**步骤3**获取。

- 执行命令示例如下，直接访问在线服务。

```
curl --location --request POST 'http://192.168.205.58:31997' \
--header 'Content-Type:  application/json' \
--data-raw '{"a":"a"}'
```

图 8-28 访问在线服务

- 创建Python环境，运行Python代码访问在线服务。

```
def vpc_infer(schema, ip, port, body):
    infer_url = "{}://{}:{}"
    url = infer_url.format(schema, ip, port)
    response = requests.post(url, data=body)
    print(response.content)
```

说明

由于高速通道特性会缺失负载均衡的能力，因此在多实例时需要自主制定负载均衡策略。

----结束

8.7 ModelArts Standard 的 WebSocket 在线服务全流程开发

背景说明

WebSocket是一种网络传输协议，可在单个TCP连接上进行全双工通信，位于OSI模型的应用层。WebSocket协议在2011年由IETF标准化为RFC 6455，后由RFC 7936补充规范。Web IDL中的WebSocket API由W3C标准化。

WebSocket使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在WebSocket API中，浏览器和服务器只需要完成一次握手，两者之间就可以建立持久性的连接，并进行双向数据传输。

前提条件

- 用户需有一定的Java开发经验，熟悉jar打包流程。
- 用户需了解WebSocket协议的基本概念及调用方法。
- 用户需熟悉Docker制作镜像的方法。

约束与限制

- WebSocket协议只支持部署在线服务。

- 只支持自定义镜像导入模型部署的在线服务。

准备工作

ModelArts使用WebSocket完成推理需要用户自己准备自定义镜像，该自定义镜像需要在单机环境下能够提供完整的WebSocket服务，如完成WebSocket的握手，client向server发送数据，server向client发送数据等。模型的推理过程在自定义镜像中完成，如下载模型，加载模型，执行预处理，完成推理，拼装响应体等。

操作步骤

WebSocket在线服务开发操作步骤如下：

- **上传镜像至容器镜像服务**
- **使用镜像创建模型**
- **使用模型部署在线服务**
- **WebSocket在线服务调用**

上传镜像至容器镜像服务

将准备好的本地镜像上传到容器镜像服务（SWR）。

使用镜像创建模型

1. 登录**ModelArts管理控制台**，进入“ 模型管理”页面，单击“创建”，跳转至创建模型页面。
2. 完成模型配置，部分配置如下：
   - 元模型来源：选择“从容器镜像中选择”。
   - 容器镜像所在的路径：选择**上传镜像至容器镜像服务**上传的路径。
   - 容器调用接口：根据实际情况配置容器调用接口。
   - 健康检查：保持默认。如果镜像中配置了健康检查则按实际情况配置健康检查。

   图 8-29 模型配置参数

3. 单击“立即创建”，进入模型列表页，等模型状态变为“正常”，表示模型创建成功。

使用模型部署在线服务

1. 登录**ModelArts管理控制台**，进入“模型部署 >在线服务”页面，单击“部署”，跳转至在线服务部署页面。
2. 完成服务的配置，部分配置如下：
   - 选择模型及版本：选择**使用镜像创建模型**创建完成的模型及版本
   - 升级为WebSocket：打开开关

   图 8-30 升级为 WebSocket

3. 单击“下一步”，确认配置后“提交”，完成在线服务的部署。返回在线服务列表页，查看服务状态变为“运行中”，表示服务部署成功。

WebSocket 在线服务调用

WebSocket协议本身不提供额外的认证方式。不管自定义镜像里面是ws还是wss，经过ModelArts平台出去的WebSocket协议都是wss的。同时wss只支持客户端对服务端的单向认证，不支持服务端对客户端的双向认证。

可以使用ModelArts提供的以下认证方式：

- **token认证**
- **AK/SK**
- **APP认证**

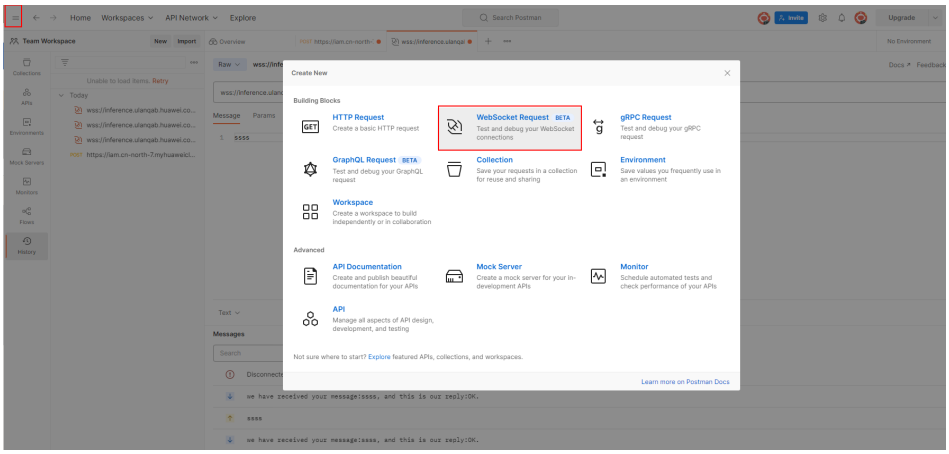
WebSocket服务调用步骤如下（以图形界面的软件Postman进行预测，token认证为例）：

1. **WebSocket连接的建立**
2. **WebSocket客户端和服务端双向传输数据**

步骤1 WebSocket连接的建立

1. 打开Postman（需选择8.5 以上版本，以10.12.0为例）工具，单击左上角  ，选择“File>New”，弹出新建对话框，选择“WebSocket Request”（当前为beta版本）功能：

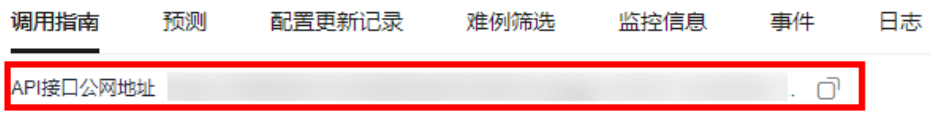
图 8-31 选择 WebSocket Request 功能



2. 在新建的窗口中填入WebSocket连接信息：

   左上角选择Raw，不要选择Socket.IO（一种WebSocket实现，要求客户端跟服务端都要基于Socket.IO），地址栏中填入从服务详情页“调用指南”页签中获取“API接口调用公网地址”后面的地址。如果自定义镜像中有更细粒度的地址，则在地址后面追加该URL。如果有queryString，那么在params栏中添加参数。在header中添加认证信息（不同认证方式有不同header，跟https的推理服务相同）。选择单击右上的connect按钮，建立WebSocket连接。

图 8-32 获取 API 接口调用公网地址

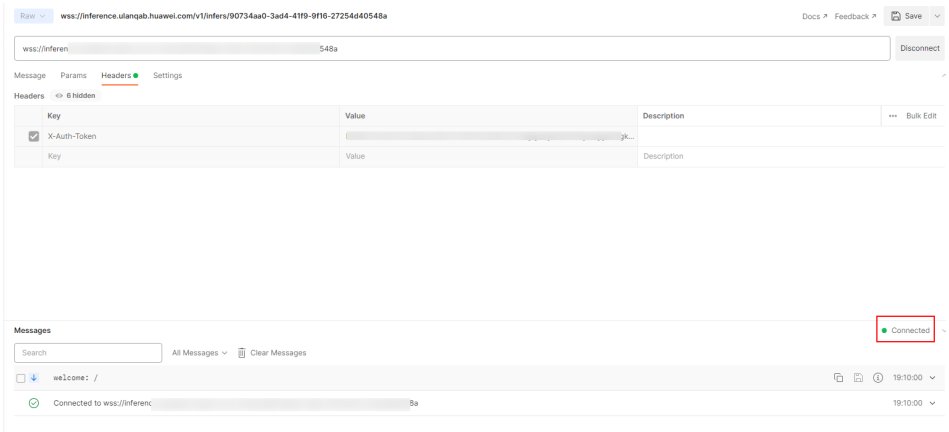


说明

- 如果信息正确，右下角连接状态处会显示：CONNECTED；
- 如果无法建立连接，如果是401状态码，检查认证信息；
- 如果显示WRONG_VERSION_NUMBER等关键字，检查自定义镜像的端口和ws跟wss的配置是否正确。

连接成功后结果如下：

图 8-33 连接成功



须知

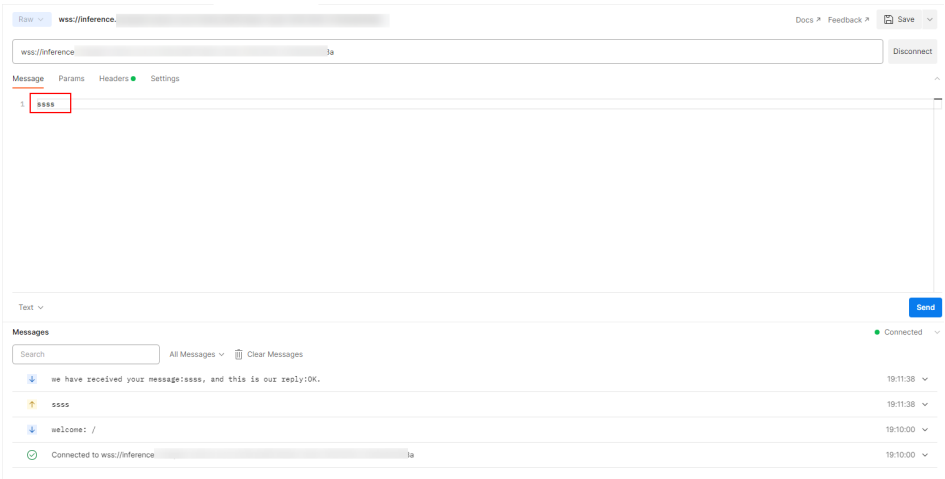
优先验证自定义镜像提供的websocket服务的情况，不同的工具实现的websocket服务会有不同，可能出现连接建立后维持不住，可能出现请求一次后连接就中断需要重新连接的情况，ModelArts平台只保证，未上ModelArts前自定义镜像的websocket的形态跟上了ModelArts平台后的websocket形态相同（除了地址跟认证方式不同）。

步骤2 WebSocket客户端和服务端双向传输数据

连接建立后，WebSocket使用TCP完成全双工通信。WebSocket的客户端可以往服务端发送数据，客户端有不同的实现，同一种语言也存在不同的lib包的实现，这里不考虑实现的不同种类。

客户端发送的内容在协议的角度不限定格式，Postman支持Text/Json/XML/HTML/Binary，以text为例，在输入框中输入要发送的文本，单击右侧中部的Send按钮即可将请求发往服务端，当文本内容过长，可能会导致postman工具卡住。

图 8-34 发送数据



----结束

8.8 从 0-1 制作自定义镜像并创建模型

针对ModelArts目前不支持的AI引擎，您可以针对该引擎构建自定义镜像，并将镜像导入ModelArts，创建为模型。本文详细介绍如何使用自定义镜像完成模型的创建，并部署成在线服务。

操作流程如下：

1. **本地构建镜像**：在本地制作自定义镜像包，镜像包规范可参考**创建AI应用的自定义镜像规范**。
2. **本地验证镜像并上传镜像至SWR服务**：验证自定义镜像的API接口功能，无误后将自定义镜像上传至SWR服务。
3. **将自定义镜像创建为模型**：将上传至SWR服务的镜像导入ModelArts的模型。
4. **将模型部署为在线服务**：将导入的模型部署上线。

本地构建镜像

以linux x86_x64架构的主机为例，您可以购买相同规格的ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录弹性云服务器**。镜像选择公共镜像，推荐使用ubuntu18.04的镜像。

图 8-35 创建 ECS 服务器-选择 X86 架构的公共镜像

1. 登录主机后，安装Docker，可参考**Docker官方文档**。也可执行以下命令安装docker。

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```

2. 获取基础镜像。本示例以Ubuntu18.04为例。

```
docker pull ubuntu:18.04
```

3. 新建文件夹“self-define-images”，在该文件夹下编写自定义镜像的“Dockerfile”文件和应用服务代码“test_app.py”。本样例代码中，应用服务代码采用了flask框架。

   文件结构如下所示

```
self-define-images/
  --Dockerfile
  --test_app.py
```

   - “Dockerfile”

```
From ubuntu:18.04
# 配置华为云的源，安装 python、python3-pip 和 Flask
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
  sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
&& \
  sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
&& \
  apt-get update && \
  apt-get install -y python3 python3-pip && \
  pip3 install  --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/
repository/pypi/simple  Flask

# 复制应用服务代码进镜像里面
COPY test_app.py /opt/test_app.py

# 指定镜像的启动命令
CMD python3  /opt/test_app.py
```

   - “test_app.py”

```
from flask import Flask, request
import json
app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("---------- in hello func ----------")
```

```
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("------------ in goodbye func ----------")
    return '\nGoodbye!\n'


@app.route('/', methods=['POST'])
def default_func():
    print("------------ in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```

4. 进入“self-define-images”文件夹，执行以下命令构建自定义镜像“test:v1”。

```
docker build -t test:v1 .
```

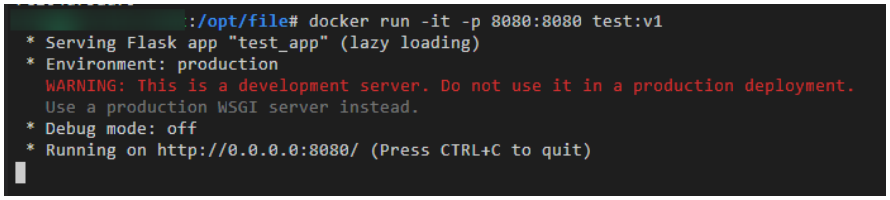
5. 您可以使用“docker images”查看您构建的自定义镜像。

本地验证镜像并上传镜像至 SWR 服务

1. 在本地环境执行以下命令启动自定义镜像

```
docker run -it -p 8080:8080 test:v1
```

图 8-36 启动自定义镜像

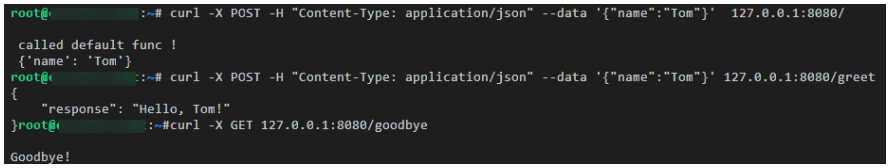


2. 另开一个终端，执行以下命令验证自定义镜像的三个API接口功能。

```
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}'  127.0.0.1:8080/
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
curl -X GET 127.0.0.1:8080/goodbye
```

如果验证自定义镜像功能成功，结果如下图所示。

图 8-37 校验接口



3. 上传自定义镜像至SWR服务。
4. 完成自定义镜像上传后，您可以在“容器镜像服务>我的镜像>自有镜像”列表中看到已上传镜像。

将自定义镜像创建为模型

参考**从容器镜像中选择元模型**导入元模型，您需要特别关注以下参数：

- 元模型来源：选择“从容器镜像中选择”
  - 容器镜像所在的路径：选择已制作好的自有镜像

    图 8-38 选择已制作好的自有镜像

  - 容器调用接口：指定模型启动的协议和端口号。请确保协议和端口号与自定义镜像中提供的协议和端口号保持一致。
  - 镜像复制：选填，选择是否将容器镜像中的模型镜像复制到ModelArts中。
  - 健康检查：选填，用于指定模型的健康检查。仅当自定义镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致模型创建失败。
- apis定义：选填，用于编辑自定义镜像的apis定义。模型apis定义需要遵循ModelArts的填写规范，参见**模型配置文件编写说明**。

  本样例的配置文件如下所示：

```
[{
      "url": "/",
      "method": "post",
      "request": {
         "Content-type": "application/json"
      },
      "response": {
         "Content-type": "application/json"
      }
   },
{
      "url": "/greet",
      "method": "post",
      "request": {
         "Content-type": "application/json"
      },
      "response": {
         "Content-type": "application/json"
      }
   },
{
      "url": "/goodbye",
      "method": "get",
      "request": {
         "Content-type": "application/json"
      },
      "response": {
         "Content-type": "application/json"
      }
   }
]
```

将模型部署为在线服务

1. 参考**部署为在线服务**将模型部署为在线服务。
2. 在线服务创建成功后，您可以在服务详情页查看服务详情。
3. 您可以通过“预测”页签访问在线服务。

图 8-39 访问在线服务

9 安全配置最佳实践

场景说明

安全性是华为云与您的共同责任。华为云负责云服务自身的安全，提供安全的云；作为租户，您需要合理使用云服务提供的安全能力对数据进行保护，安全地使用云。

本文提供了ModelArts使用过程中的安全最佳实践，旨在为提高整体安全能力提供可操作的规范性指导。根据该指导文档您可以持续评估ModelArts资源的安全状态，更好的组合使用ModelArts提供的多种安全能力，提高对ModelArts资源的整体安全防御能力，保护在ModelArts平台上的数据不泄露、不被篡改，以及数据传输过程中不泄露、不被篡改。

本文从以下几个维度给出建议，您可以评估ModelArts使用情况，并根据业务需要在本指导的基础上进行安全配置。

- **使用IP白名单的方式接入Notebook**
- **在生产环境下使用专属资源池**
- **自定义镜像使用非root用户运行**
- **开发过程不使用硬编码的凭证**
- **对不同的子用户，使用独立的委托**

使用 IP 白名单的方式接入 Notebook

ModelArts Standard的开发环境Notebook实例支持通过SSH方式直接连接，通过keypair方式进行认证。除此之外，对于安全性要求更强的用户，建议配置IP白名单的方式，进一步限制能接入该实例的终端节点。配置方式参考**Notebook SSH远程连接配置**章节中的“远程访问白名单”参数。

在生产环境下使用专属资源池

在使用训练、推理、开发环境时，建议在生产环境下使用专属资源池，它在提供独享的计算资源情况下，还可以提供更强更安全的资源隔离能力，专属资源池的使用请参考**创建专属资源池**。

在使用ModelArts进行AI全流程开发时，您可以选择使用两种不同的资源池。

公共资源池：公共资源池提供公共的大规模计算集群，根据用户作业参数分配使用，资源按作业隔离。按资源规格、使用时长及实例数计费，不区分任务（训练作业、部

署、开发）。公共资源池是ModelArts默认提供，不需另行创建或配置，您可以直接在AI开发过程中，直接选择公共资源池进行使用。

专属资源池：提供独享的计算资源，可用于Notebook、训练作业、部署模型。专属资源池不与其他用户共享，更加高效。

在使用专属资源池之前，您需要先购买一个专属资源池，然后在AI开发过程中选择此专属资源池。

自定义镜像使用非 root 用户运行

自定义镜像支持自行开发Dockerfile，并推送到SWR，出于权限控制范围的考虑，建议用户在自定义镜像时，显式定义默认运行的用户为非root用户，以降低容器运行时的安全风险。

在AI业务开发以及运行的过程中，一般都会有复杂的环境依赖需要进行调测并固化。面对开发中的开发环境的脆弱和多轨切换问题，在ModelArts的AI开发最佳实践中，通过容器镜像的方式，将运行环境进行固化，以这种方式不仅能够的进行依赖管理，而且可以方便的完成工作环境切换。配合ModelArts提供的云化容器资源使用，可以更加快速、高效地进行AI开发与模型实验的迭代等。

ModelArts Standard中使用自定义镜像请参见**自定义镜像使用场景**。

开发过程不使用硬编码的凭证

在使用ModelArts Standard Notebook进行算法开发时，如果要将此开发好的算法发布上生产环境，建议对代码中使用到的密码、AK/SK 、数据库连接、OBS连接、SWR连接信息等进行排查，不要使用固化的认证凭据，不方便后期算法更新维护。建议对上述敏感信息进行加密后保存在程序配置文件中。

对不同的子用户，使用独立的委托

要使用ModelArts的资源，需要得到用户的委托授权，为了控制各子用户之间权限，建议租户在ModelArts全局配置功能中给各子用户分配委托权限时，分开授权，不要多个子用户共用一个委托凭证。委托授权相关内容参考**创建IAM用户并授权使用ModelArts**。

