ModelArts

推理部署

文档版本 01

发布日期 2024-09-05

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。

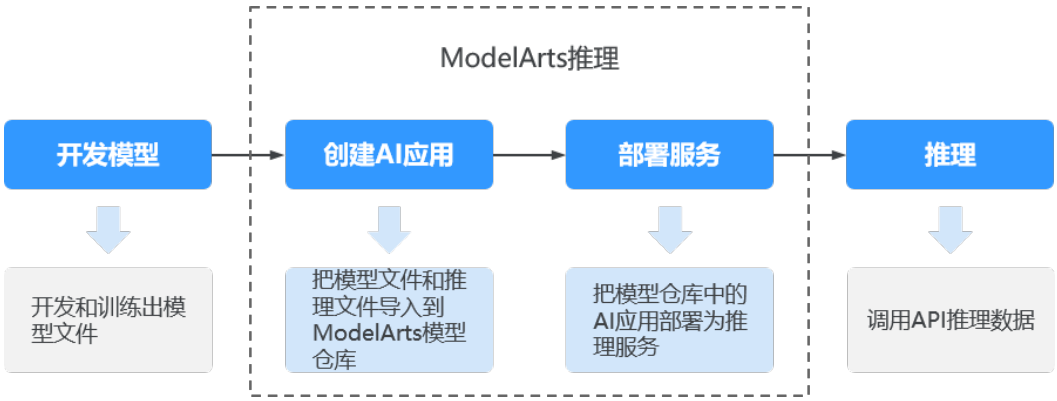
由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 推理简介

AI模型开发完成后，在ModelArts服务中可以将AI模型创建为AI应用，将AI应用快速部署为推理服务，您可以通过调用API的方式把AI推理能力集成到自己的IT平台。

图 1-1 推理简介



- 开发模型：模型开发可以在ModelArts服务中进行，也可以在您的本地开发环境进行，本地开发的模型需要上传到华为云OBS服务。
- 创建AI应用：把模型文件和推理文件导入到ModelArts的模型仓库中，进行版本化管理，并构建为可运行的AI应用。
- 部署服务：把AI应用在资源池中部署为容器实例，注册外部可访问的推理API。
- 推理：在您的应用中增加对推理API的调用，在业务流程中集成AI推理能力。

部署服务

在完成AI应用的创建后，可在“模型部署”页面对AI应用进行部署。ModelArts当前支持如下几种部署类型：

- **在线服务**

  将AI应用部署为一个Web Service，并且提供在线的测试UI与监控功能。
- **批量服务**

  批量服务可对批量数据进行推理，完成数据处理后自动停止。

2 管理 AI 应用

2.1 管理 AI 应用简介

AI开发和调优往往需要大量的迭代和调试，数据集、训练代码或参数的变化都可能会影响模型的质量，如不能统一管理开发流程元数据，可能会出现无法重现最优模型的现象。

ModelArts的AI应用可导入所有训练生成的元模型、上传至对象存储服务（OBS）中的元模型和容器镜像中的元模型，可对所有迭代和调试的AI应用进行统一管理。

约束与限制

- 自动学习项目中，在完成模型部署后，其生成的模型也将自动上传至AI应用列表中。但是自动学习生成的AI应用无法下载，只能用于部署上线。
- 创建AI应用、管理AI应用版本等功能目前是免费开放给所有用户，使用此功能不会产生费用。

创建 AI 应用的几种场景

- **从训练中选择**：在ModelArts中创建训练作业，并完成模型训练，在得到满意的模型后，可以将训练后得到的模型创建为AI应用，用于部署服务。
- **从对象存储服务（OBS）中选择**：如果您使用常用框架在本地完成模型开发和训练，可以将本地的模型按照模型包规范上传至OBS桶中，从OBS将模型导入至ModelArts中，创建为AI应用，直接用于部署服务。
- **从容器镜像中选择**：针对ModelArts目前不支持的AI引擎，可以通过自定义镜像的方式将编写的模型镜像导入ModelArts，创建为AI应用，用于部署服务。
- **从模板中选择**：相同功能的模型配置信息重复率高，将相同功能的配置整合成一个通用的模板，通过使用该模板，可以方便快捷的导入模型，创建为AI应用，而不用编写config.json配置文件。

AI 应用的功能描述

表 2-1 AI 应用相关功能

支持的功能	说明
创建AI应用	将训练后的模型导入至ModelArts创建为AI应用，便于进行统一管理，支持如下几种场景的导入方式，不同场景对应的操作指导请参见： • 从训练中选择元模型 • 从对象存储服务（OBS）中选择元模型 • 从容器镜像中选择元模型 • 从模板中选择元模型
查看AI应用详情	当AI应用创建成功后，您可以进入AI应用详情页查看AI应用的信息。
管理AI应用版本	为方便溯源和模型反复调优，在ModelArts中提供了AI应用版本管理的功能，您可以基于版本对AI应用进行管理。

推理支持的 AI 引擎

在ModelArts创建AI应用时，若使用预置镜像“从模板中选择”或“从OBS中选择”导入模型，则支持如下常用引擎及版本的模型包。

说明

- 标注“推荐”的Runtime来源于统一镜像，后续统一镜像将作为主流的推理基础镜像。统一镜像中的安装包更齐全，详细信息可以参见推理基础镜像列表。
- 推荐将旧版镜像切换为统一镜像，旧版镜像后续将会逐渐下线。
- 待下线的基本镜像不再维护。
- 统一镜像Runtime的命名规范：<AI引擎名字及版本> - <硬件及版本：cpu或cuda或cann> - <python版本> - <操作系统版本> - <CPU架构>
- 当前支持自定义模型启动命令，预置AI引擎都有默认的启动命令，如非必要无需改动

表 2-2 支持的常用引擎及其 Runtime 以及默认启动命令

模型使用的引擎类型	支持的运行环境（Runtime）	注意事项
TensorFlow	python3.6 python2.7（待下线） tf1.13-python3.6-gpu tf1.13-python3.6-cpu tf1.13-python3.7-cpu tf1.13-python3.7-gpu tf2.1-python3.7（待下线） tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64（推荐）	• python2.7、python3.6的运行环境搭载的TensorFlow版本为1.8.0。 • python3.6、python2.7、tf2.1-python3.7，表示该模型可同时在CPU或GPU运行。其他Runtime的值，如果后缀带cpu或gpu，表示该模型仅支持在CPU或GPU中运行。 • 默认使用的Runtime为python2.7。 • 默认启动命令：sh /home/mind/run.sh
Spark_MLlib	python2.7（待下线） python3.6（待下线）	• python2.7以及python3.6的运行环境搭载的Spark_MLlib版本为2.3.2。 • 默认使用的Runtime为python2.7。 • python2.7、python3.6只能用于运行适用于CPU的模型。 • 默认启动命令：bash /home/work/predict/bin/run.sh
Scikit_Learn	python2.7（待下线） python3.6（待下线）	• python2.7以及python3.6的运行环境搭载的Scikit_Learn版本为0.18.1。 • 默认使用的Runtime为python2.7。 • python2.7、python3.6只能用于运行适用于CPU的模型。 • 默认启动命令：bash /home/work/predict/bin/run.sh
XGBoost	python2.7（待下线） python3.6（待下线）	• python2.7以及python3.6的运行环境搭载的XGBoost版本为0.80。 • 默认使用的Runtime为python2.7。 • python2.7、python3.6只能用于运行适用于CPU的模型。 • 默认启动命令：bash /home/work/predict/bin/run.sh

模型使用的引擎类型	支持的运行环境（Runtime）	注意事项
PyTorch	python2.7（待下线） python3.6 python3.7 pytorch1.4-python3.7 pytorch1.5-python3.7（待下线） pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64（推荐）	• python2.7、python3.6、python3.7的运行环境搭载的PyTorch版本为1.0。 • python2.7、python3.6、python3.7、pytorch1.4-python3.7、pytorch1.5-python3.7，表示该模型可同时在CPU或GPU运行。 • 默认使用的Runtime为python2.7。 • 默认启动命令：sh /home/mind/run.sh
MindSpore	aarch64（推荐）	aarch64只能用于运行在Snt3芯片上。 • 默认启动命令：sh /home/mind/run.sh

2.2 创建 AI 应用

2.2.1 从训练中选择元模型

在ModelArts中创建训练作业，并完成模型训练，在得到满意的模型后，可以将训练后得到的模型导入至模型管理，方便统一管理，同时支持将模型快速部署上线为服务。

约束与限制

- 针对使用订阅算法的训练作业，无需推理代码和配置文件，其生成的模型可直接导入ModelArts。
- 使用容器化部署，导入的元模型有大小限制，详情请参见**导入AI应用对于镜像大小限制**。

前提条件

- 请确保训练作业已运行成功，且模型已存储至训练输出的OBS目录下（输入参数为train_url）。
- 针对使用常用框架或自定义镜像创建的训练作业，需根据**模型包规范介绍**，将推理代码和配置文件上传至模型的存储目录中。
- 确保您使用的OBS目录与ModelArts在同一区域。

创建 AI 应用操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI应用”，进入AI应用列表页面。
2. 单击左上角的“创建应用”，进入“创建应用”页面。
3. 在“创建应用”页面，填写相关参数。

a. 填写AI应用基本信息，详细参数说明请参见表2-3。

表 2-3 AI 应用基本信息参数说明

参数名称	说明
名称	AI应用名称。支持1～64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线。
版本	设置所创建AI应用的版本。第一次导入时，默认为0.0.1。 **说明** AI应用创建完成后，可以通过**创建新版本**，导入不同的元模型进行调优。
描述	AI应用的简要描述。

b. 填写元模型来源及其相关参数。当“元模型来源”选择“从训练中选择”时，其相关的参数配置请参见表2-4。

图 2-1 从训练中选择元模型

表 2-4 元模型来源参数说明

参数	说明
“元模型来源”	选择“从训练中选择”。 ● 在“选择训练作业”右侧下拉框中选择当前账号下已完成运行的训练作业。 ● “动态加载”：用于实现快速部署和快速更新模型。若勾选动态加载，则模型文件和运行时依赖仅在实际部署时拉取。当单个模型文件大小超过5GB时，必须配置“动态加载”。
“AI引擎”	元模型使用的推理引擎，选择训练作业后会自动匹配。
“推理代码”	推理代码自定义AI应用的推理处理逻辑。显示推理代码URL，您可以直接复制此URL使用。
“运行时依赖”	罗列选中模型对环境的依赖。例如依赖“tensorflow”，安装方式为“pip”，其版本必须为1.8.0及以上版本。

参数	说明
“AI应用说明”	为了帮助其他AI应用开发者更好的理解及使用您的AI应用，建议您提供AI应用的说明文档。单击“添加AI应用说明”，设置“文档名称”及其“URL”。AI应用说明最多支持3条。
“部署类型”	选择此AI应用支持部署服务的类型，部署上线时只支持部署为此处选择的部署类型，例如此处只选择在线服务，那您导入后只能部署为在线服务。

c. 确认信息填写无误，单击“立即创建”，完成AI应用的创建。

在AI应用列表中，您可以查看刚创建的AI应用及其对应的版本。当AI应用状态变更为“正常”时，表示AI应用导入成功。在此页面，您还可以创建新版本、快速部署服务等操作。

后续操作

部署服务：在“AI应用列表”中，单击AI应用的操作列的“部署”，在对应版本所在行，单击“操作”列的部署按钮，可以将AI应用部署上线为创建AI应用时所选择的部署类型。

2.2.2 从模板中选择元模型

相同功能的模型配置信息重复率高，将相同功能的配置整合成一个通用的模板，通过使用该模板，可以方便快捷的创建AI应用，而不用编写config.json配置文件。模板的详细说明请参见**模型模板简介**。

约束与限制

- 目前支持的模板请参见**支持的模板**，各模板相应的输入输出模式说明，请参见**支持的输入输出模式**。
- 创建和管理AI应用是免费的，不会产生费用。

前提条件

- 确保您已按照相应模板的模型包规范要求将模型上传至OBS。
- 确保您使用的OBS与ModelArts在同一区域。

创建 AI 应用操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI应用”，进入AI应用列表页面。
2. 单击左上角的“创建应用”，进入“创建应用”页面。
3. 在“创建应用”页面，填写相关参数。
   a. 填写AI应用基本信息，详细参数说明请参见**表2-5**。

表 2-5 AI 应用基本信息参数说明

参数名称	说明
名称	AI应用名称。支持1～64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线。
版本	设置所创建AI应用的版本。第一次导入时，默认为0.0.1。 **说明** AI应用创建完成后，可以通过**创建新版本**，导入不同的元模型进行调优。
描述	AI应用的简要描述。

b. 填写元模型来源及其相关参数。当“元模型来源”选择“从模板中选择”时，其相关的参数配置请参见**表2-6**。

图 2-2 从模板中选择元模型

表 2-6 元模型来源参数说明

参数	说明
“模型模板”	从已有的ModelArts模板列表中选择模板。例如，“TensorFlow图像分类模板”。 ModelArts还提供“类型”、“引擎”、“环境”三个筛选条件，帮助您更快找到想要的模板。如果这个三个筛选条件不能满足您的要求，可以使用关键词搜索，找到目标模板。支持的模板请参见**支持的模板**。

参数	说明
“模型目录”	指定模型存储的OBS路径。请根据您选择的“模型模板”，按照模板的模型输入要求，选择对应的模型存储的OBS路径。 OBS路径不能含有空格，否则创建AI应用会失败。 **说明** • 选择加密桶或者加密文件，会导入失败。 • 当训练作业执行多次时，将基于V001、V002等规则生成不同的版本目录，且生成的模型将存储在不同版本目录下的model文件夹。此处选择模型文件时，需指定对应版本目录下的model文件夹。
“动态加载”	用于实现快速部署和快速更新模型。若勾选“动态加载”，则模型文件和运行时依赖仅在实际部署时拉取。单个模型文件大小超过5GB，需要配置“动态加载”。
“输入输出模式”	如果您选择的模板允许覆盖其中的默认输入输出模式，您可以根据AI应用功能或业务场景在“输入输出模式”中，选择相应的输入输出模式。“输入输出模式”是对“config.json”中API（apis）的抽象，描述AI应用对外提供推理的接口。一个“输入输出模式”描述一个或多个API接口，每个模板对应于一个“输入输出模式”。 例如，“TensorFlow图像分类模板”，其支持的“输入输出模式”为“预置图像处理模式”，但该模板不允许修改其中的输入输出模式，所以您在页面上只能看到模板默认的输入输出模式，而不能选择其他模式。 支持的输入输出模式请参见**支持的输入输出模式**。
“AI应用说明”	为了帮助其他AI应用开发者更好的理解及使用您的AI应用，建议您提供AI应用的说明文档。单击“添加AI应用说明”，设置“文档名称”及其“URL”。AI应用说明支持增加3条。
“部署类型”	选择此AI应用支持部署服务的类型，部署上线时只支持部署为此处选择的部署类型，例如此处只选择在线服务导入后只能部署为在线服务。

c. 确认信息填写无误，单击“立即创建”，完成AI应用导入。

在AI应用列表中，您可以查看刚创建的AI应用及其对应的版本。当AI应用状态变更为“正常”时，表示AI应用创建成功。在此页面，您还可以创建新版本、快速部署服务等操作。

后续操作

部署服务：在“AI应用列表”中，单击AI应用的操作列的“部署”，在对应版本所在行，单击“操作”列的部署按钮，可以将AI应用部署上线为创建AI应用时所选择的部署类型。

2.2.3 从对象存储服务（OBS）中选择元模型

针对使用常用框架完成模型开发和训练的场景，可以将您的模型导入至ModelArts中，创建为AI应用，并进行统一管理。

约束与限制

- 针对创建AI应用的模型，需符合ModelArts的模型包规范，推理代码和配置文件也需遵循ModelArts的要求，详细说明请参见**模型包规范介绍**、**模型配置文件编写说明**、**模型推理代码编写说明**。
- 使用容器化部署，导入的元模型有大小限制，详情请参见**导入AI应用对于镜像大小限制**。

前提条件

- 已完成模型开发和训练，使用的AI引擎为ModelArts支持的类型和版本，详细请参见**推理支持的AI引擎**。
- 已完成训练的模型包，及其对应的推理代码和配置文件，且已上传至OBS目录中。
- 确保您使用的OBS与ModelArts在同一区域。

创建 AI 应用操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI应用”，进入AI应用列表页面。
2. 单击左上角的“创建应用”，进入“创建应用”页面。
3. 在“创建应用”页面，填写相关参数。
   a. 填写AI应用基本信息，详细参数说明请参见**表2-7**。

   表 2-7 AI 应用基本信息参数说明

参数名称	说明
名称	AI应用名称。支持1～64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线。
版本	设置所创建AI应用的版本。第一次导入时，默认为0.0.1。 **说明** AI应用创建完成后，可以通过**创建新版本**，导入不同的元模型进行调优。
描述	AI应用的简要描述。

   b. 填写元模型来源及其相关参数。当“元模型来源”选择“从对象存储服务（OBS）中选择”时，其相关的参数配置请参见**表2-8**。

   针对从OBS导入的元模型，ModelArts要求根据**模型包规范**，编写推理代码和配置文件，并将推理代码和配置文件放置元模型存储的“model”文件夹下。如果您选择的目录下不符合模型包规范，将无法创建AI应用。

图 2-3 从 OBS 中选择元模型

表 2-8 元模型来源参数说明

参数	说明
“选择元模型”	选择元模型存储的OBS路径。 OBS路径不能含有空格，否则创建AI应用会失败。
“AI引擎”	根据您选择的元模型存储路径，将自动关联出元模型使用的“AI引擎”。 如果“AI引擎”是Custom引擎时，需要配置容器调用接口，用于指定模型启动的协议和端口号。请求协议和端口号的缺省值是HTTPS和8080，端口和协议需要根据用户模型实际使用情况自行配置。
“容器调用接口”	模型提供的推理接口所使用的协议和端口号，需要根据模型实际定义的推理接口进行配置。 **说明** AI引擎选择Custom引擎时才会显示该参数，ModelArts提供的请求协议和端口号的缺省值是HTTPS和8080，端口和协议需要根据用户模型实际使用情况自行配置。

参数	说明
“健康检查”	用于指定模型的健康检查。使用Custom引擎时，会显示该参数。使用非Custom引擎时，选择了“AI引擎”和“运行环境”后，部分支持健康检查的引擎会显示该参数，请以实际界面显示为准。 当使用Custom引擎时，引擎包需要选择容器镜像，仅当容器镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致AI应用创建失败。 当前支持以下三种探针： • **启动探针：**用于检测应用实例是否已经启动。如果提供了启动探针(startup probe)，则禁用所有其他探针，直到它成功为止。如果启动探针失败，将会重启实例。如果没有提供启动探针，则默认状态为成功Success。 • **就绪探针：**用于检测应用实例是否已经准备好接收流量。如果就绪探针失败，即实例未准备好，会从服务负载均衡的池中剔除该实例，不会将流量路由到该实例，直到探测成功。 • **存活探针：**用于检测应用实例内应用程序的健康状态。如果存活探针失败，即应用程序不健康，将会自动重启实例。 3种探针的配置参数均为： • **检查方式：**仅支持“HTTP请求检查”。 • **健康检查URL**：健康检查的URL固定为“/health”。 • **健康检查周期（秒）**：填写1-2147483647之前的整数，单位为秒。 • **延迟时间（秒）：**实例启动后，延迟执行健康检查的时间。填写0-2147483647之间的整数，单位为秒，不能为空。 • **超时时间（秒）：** 每次检查的超时时间，填写0-2147483647之间的整数，单位为秒。 • **最大失败次数**：填写1-2147483647之间的整数。在服务启动阶段，当健康检查请求连续失败达到所填次数后，服务会进入异常状态；在服务运行阶段，当健康检查请求连续失败达到所填次数后，服务会进入告警状态。 **说明** 使用Custom引擎时需要符合自定义引擎规范，请参见**使用自定义引擎创建AI应用**。 当AI应用配置了健康检查，部署的服务在收到停止指令后，会延后3分钟才停止。
“动态加载”	用于实现快速部署和快速更新模型。若勾选“动态加载”，则模型文件和运行时依赖仅在实际部署时拉取。单个模型文件大小超过5GB，需要配置“动态加载”。
“运行时依赖”	罗列选中模型对环境的依赖。例如依赖“tensorflow”，安装方式为“pip”，其版本必须为1.8.0及以上版本。

参数	说明
“AI应用说明”	为了帮助其他AI应用开发者更好的理解及使用您的AI应用，建议您提供AI应用的说明文档。单击“添加AI应用说明”，设置“文档名称”及其“URL”。AI应用说明支持增加3条。
“配置文件”	系统默认关联您存储在OBS中的配置文件。打开开关，您可以直接在当前界面查看或编辑模型配置文件。 **说明** 该功能即将下线，后续请根据“AI引擎”、“运行时依赖”和“apis定义”修改模型的配置信息。
“部署类型”	选择此AI应用支持部署服务的类型，部署上线时只支持部署为此处选择的部署类型，例如此处只选择在线服务，那您导入后只能部署为在线服务。
“启动命令”	选填参数，指定模型的启动命令，您可以自定义该命令。 如果使用预置的AI引擎，若启动命令没有填写，会使用默认的启动命令，默认的启动命令见**表2-2**。若填写了启动命令，新填写的启动命令覆盖默认启动命令。 **说明** 包含字符$，|，>，<，`，!，\n，\，?，-v，--volume，--mount，--tmpfs，--privileged，--cap-add的启动命令，在AI应用发布时将会置空。
“apis定义”	提供AI应用对外Restfull api数据定义，用于定义AI应用的输入、输出格式。apis定义填写规范请参见**模型配置文件编写说明**中的apis参数说明，示例代码请参见**apis参数代码示例。**

c. 确认信息填写无误，单击“立即创建”，完成AI应用创建。

在AI应用列表中，您可以查看刚创建的AI应用及其对应的版本。当AI应用状态变更为“正常”时，表示AI应用创建成功。在此页面，您还可以创建新版本、快速部署服务等操作。

后续操作

部署服务：在“AI应用列表”中，单击AI应用的操作列的“部署”，在对应版本所在行，单击“操作”列的部署按钮，可以将AI应用部署上线为创建AI应用时所选择的部署类型。

2.2.4 从容器镜像中选择元模型

针对ModelArts目前不支持的AI引擎，您可以通过自定义镜像的方式将编写的模型导入ModelArts。

约束与限制

- 关于自定义镜像规范和说明，请参见模型镜像规范。
- 针对您开发并训练完成的模型，需要提供对应的模型配置文件，此文件需遵守ModelArts的填写规范，详情请参见**模型配置文件编写说明**。编写完成后，需将此文件上传至OBS指定目录下。

- 使用容器化部署，导入的元模型有大小限制，详情请参见**导入AI应用对于镜像大小限制**。

前提条件

确保您使用的OBS目录与ModelArts在同一区域。

创建 AI 应用操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI应用”，进入AI应用列表页面。
2. 单击左上角的“创建应用”，进入“创建应用”页面。
3. 在“创建应用”页面，填写相关参数。
   a. 填写AI应用基本信息，详细参数说明请参见**表2-9**。

表 2-9 AI 应用基本信息参数说明

参数名称	说明
名称	AI应用名称。支持1～64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线。
版本	设置所创建AI应用的版本。第一次导入时，默认为0.0.1。 **说明** AI应用创建完成后，可以通过**创建新版本**，导入不同的元模型进行调优。
描述	AI应用的简要描述。

   b. 填写元模型来源及其相关参数。当“元模型来源”选择“从容器镜像中选择”时，其相关的参数配置请参见**表2-10**。

图 2-4 从容器镜像中选择 AI 应用

表 2-10 元模型来源参数说明

参数	说明
“容器镜像所在的路径”	单击 从容器镜像中导入模型的镜像，其中，模型均为Image类型，且不再需要用配置文件中的“swr_location”来指定您的镜像位置。 制作自定义镜像的操作指导及规范要求，请参见模型镜像规范。 **说明** 您选择的模型镜像将共享给系统管理员，请确保具备共享该镜像的权限（不支持导入其他账户共享给您的镜像），部署上线时，ModelArts将使用该镜像部署成推理服务，请确保您的镜像能正常启动并提供推理接口。
“容器调用接口”	模型提供的推理接口所使用的协议和端口号，请根据模型实际定义的推理接口进行配置。 **说明** ModelArts提供的请求协议和端口号的缺省值是HTTP和8080。用户需根据实际的自定义镜像进行配置。
“镜像复制”	镜像复制开关，选择是否将容器镜像中的模型镜像复制到ModelArts中。 ● 关闭时，表示不复制模型镜像，可极速创建AI应用，更改或删除SWR源目录中的镜像会影响服务部署。 ● 开启时，表示复制模型镜像，无法极速创建AI应用，SWR源目录中的镜像更改或删除不影响服务部署。 **说明** 若使用他人共享的镜像，需要开启镜像复制功能，否则会导致创建AI应用失败。

参数	说明
“健康检查”	用于指定AI应用的健康检查。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AI应用创建失败。当前支持以下三种探针： ● **启动探针：**用于检测应用实例是否已经启动。如果提供了启动探针(startup probe)，则禁用所有其他探针，直到它成功为止。如果启动探针失败，将会重启实例。如果没有提供启动探针，则默认状态为成功Success。 ● **就绪探针：**用于检测应用实例是否已经准备好接收流量。如果就绪探针失败，即实例未准备好，会从服务负载均衡的池中剔除该实例，不会将流量路由到该实例，直到探测成功。 ● **存活探针：**用于检测应用实例内应用程序的健康状态。如果存活探针失败，即应用程序不健康，将会自动重启实例。 3种探针的配置参数均为： ● **检查方式：**可以选择“HTTP请求检查”或者“执行命令检查”。 ● **健康检查URL**：“检查方式”选择“HTTP请求检查”时显示，填写健康检查的URL，默认值为“/health”。 ● **健康检查命令：**“检查方式”选择“执行命令检查”时显示，填写健康检查的命令。 ● **健康检查周期（秒）**：填写1-2147483647之前的整数，单位为秒。 ● **延迟时间（秒）：**实例启动后，延迟执行健康检查的时间。填写0-2147483647之间的整数，单位为秒，不能为空。 ● **超时时间（秒）：** 每次检查的超时时间，填写0-2147483647之间的整数，单位为秒。 ● **最大失败次数**：填写1-2147483647之间的整数。在服务启动阶段，当健康检查请求连续失败达到所填次数后，服务会进入异常状态；在服务运行阶段，当健康检查请求连续失败达到所填次数后，服务会进入告警状态。 **说明** 当AI应用配置了健康检查，部署的服务在收到停止指令后，会延后3分钟才停止。
“AI应用说明”	为了帮助其他AI应用开发者更好的理解及使用您的AI应用，建议您提供AI应用的说明文档。单击“添加AI应用说明”，设置“文档名称”及其“URL”。AI应用说明支持增加3条。
“部署类型”	选择此AI应用支持部署服务的类型，部署上线时只支持部署为此处选择的部署类型，例如此处只选择在线服务，那您导入后只能部署为在线服务。

参数	说明
“启动命令”	指定模型的启动命令，您可以自定义该命令。 **说明** 包含字符$，|，>，<，`，!，\n，\，?，-v，--volume，--mount，--tmpfs，--privileged，--cap-add的启动命令，在AI应用发布时将会置空。
“apis定义”	提供AI应用对外Restfull api数据定义，用于定义AI应用的输入、输出格式。apis定义填写规范请参见**模型配置文件编写说明**中的apis参数说明，示例代码请参见**apis参数代码示例。**

c. 确认信息填写无误，单击“立即创建”，完成AI应用创建。

在AI应用列表中，您可以查看刚创建的AI应用及其对应的版本。当AI应用状态变更为“正常”时，表示AI应用创建成功。在此页面，您还可以进行创建新版本、快速部署服务等操作。

后续操作

部署服务：在“AI应用列表”中，单击AI应用的操作列的“部署”，在对应版本所在行，单击“操作”列的部署按钮，可以将AI应用部署上线为创建AI应用时所选择的部署类型。

2.3 查看 AI 应用列表

当AI应用创建成功后，您可在AI应用列表页查看所有创建的AI应用。AI应用列表页包含以下信息。

表 2-11 AI 应用列表

参数	说明
AI应用名称	AI应用的名称。
最新版本	AI应用的当前最新版本。
状态	AI应用当前状态。
部署类型	AI应用支持部署的服务类型。
版本数量	AI应用的版本数量。
请求模式	在线服务的请求模式。 • 同步请求：单次推理，可同步返回结果（约<60s）。例如: 图片、较小视频文件。 • 异步请求：单次推理，需要异步处理返回结果（约>60s）。例如: 实时视频推理、大视频文件。
创建时间	AI应用的创建时间。
描述	AI应用的描述。

参数	说明
操作	• 部署：将AI应用发布为在线服务、批量服务或边缘服务。 • 创建新版本：创建新的AI应用版本。参数配置除版本外，将默认选择上一个版本的配置信息，您可以对参数配置进行修改。 • 删除：删除对应的AI应用。 **说明** 如果AI应用的版本已经部署服务，需先删除关联的服务后再执行删除操作。AI应用删除后不可恢复，请谨慎操作。

单击AI应用的“版本数量”，可查看版本列表信息。

图 2-5 版本列表

版本列表中包含以下信息。

表 2-12 版本列表

参数	说明
版本	AI应用当前版本。
状态	AI应用当前状态。
部署类型	AI应用支持部署的服务类型。
AI应用大小	AI应用的大小。
模型来源	显示AI应用模型的来源。
创建时间	AI应用的创建时间。
描述	AI应用的描述。
操作	• 部署：将AI应用发布为在线服务、批量服务或边缘服务。 • 删除：针对AI应用的某一版本进行删除。

2.4 查看 AI 应用详情

当AI应用创建成功后，您可以进入AI应用详情页查看AI应用的信息。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“AI应用”，进入“自定义应用”列表页面。

2. 单击目标AI应用名称，进入AI应用详情页面。
   您可以查看AI应用的基本信息、模型精度，以及切换页签查看更多信息。

表 2-13 AI 应用基本信息

参数	说明
名称	AI应用的名称。
状态	AI应用当前状态。
版本	AI应用当前版本。
ID	AI应用的ID。
描述	单击编辑按钮，可以添加AI应用的描述。
部署类型	AI应用支持部署的服务类型。
元模型来源	显示元模型的来源，主要有从训练中选择、从对象存储服务（OBS）中选择、从容器镜像中选择。不同来源的元模型，AI应用显示的参数会不同。
训练作业名称	若元模型来源于训练作业，则显示关联的训练作业，单击训练作业名称可以直接跳转到训练作业详情页面。
训练作业版本	若元模型来源于训练作业且为旧版训练作业，显示训练作业版本。
元模型存储路径	若元模型来源于对象存储服务，显示元模型的存放路径。
容器镜像存储路径	若元模型来源于容器镜像，显示容器镜像存储路径。
AI引擎	若元模型来源于训练作业/对象存储服务，显示AI应用使用的AI引擎。
引擎包地址	若元模型来源于对象存储服务（AI引擎为Custom），显示引擎包地址。
运行环境	若元模型来源于训练作业/对象存储服务（AI引擎为预置引擎），显示元模型依赖的运行环境。
容器调用接口	若元模型来源于对象存储服务（AI引擎为Custom）/容器镜像，显示AI应用启动的协议和端口号。
推理代码	若元模型来源于训练作业且为旧版训练作业，则显示推理代码的存放路径。
镜像复制	若元模型来源于容器镜像，显示镜像复制功能状态。
动态加载	若元模型来源于训练作业/对象存储服务，显示AI应用是否支持动态加载。
大小	AI应用的大小。

参数	说明
健康检查	若元模型来源于对象存储服务/容器镜像，显示健康检查状态。当健康检查为开启时，会根据您启用的探针显示对应探针的参数设置情况。 • **启动探针：**用于检测应用实例是否已经启动。如果提供了启动探针(startup probe)，则禁用所有其他探针，直到它成功为止。如果启动探针失败，将会重启实例。如果没有提供启动探针，则默认状态为成功Success。 • **就绪探针：**用于检测应用实例是否已经准备好接收流量。如果就绪探针失败，即实例未准备好，会从服务负载均衡的池中剔除该实例，不会将流量路由到该实例，直到探测成功。 • **存活探针：**用于检测应用实例内应用程序的健康状态。如果存活探针失败，即应用程序不健康，将会自动重启实例。 每种探针下会显示以下字段：检查方式、健康检查URL（检查方式为“HTTP请求检查”时显示）、健康检查命令（检查方式为“执行命令检查”时显示）、健康检查周期、延迟时间、超时时间、最大失败次数。
AI应用说明	显示创建AI应用时添加的AI应用说明文档信息。
系统运行架构	显示系统运行架构。
推理加速卡类型	显示推理加速卡类型。

表 2-14 AI 应用页签详情

参数	说明
模型精度	显示该AI应用的模型召回率、精准率、准确率和F1值。
参数配置	可以查看AI应用的apis定义详情，以及AI应用的入参和出参。
运行时依赖	查看模型对环境的依赖。当构建任务失败后可以编辑运行时依赖，保存修改后将触发镜像重新构建。
事件	展示AI应用创建过程中的关键操作进展。 事件保存周期为3个月，3个月后自动清理数据。 查看AI应用的事件类型和事件信息，请参见**查看AI应用的事件**
使用约束	根据创建AI应用时的设置，显示部署服务的使用约束，如请求模式、启动命令、模型加密等。对于异步请求模式的AI应用，可显示输入模式、输出模式、服务启动参数和作业配置参数等参数。
关联服务	展示使用该AI应用部署的服务列表，单击服务名称可以直接跳转到服务详情页面。

2.5 管理 AI 应用版本

为方便溯源和AI应用反复调优，在ModelArts中提供了AI应用版本管理的功能，您可以基于版本对AI应用进行管理。

前提条件

已在ModelArts中创建AI应用。

创建新版本

在“AI应用”页面，单击操作列的“创建新版本”进入“创建新版本”页面，参数配置除版本外，将默认选择上一个版本的配置信息，您可以对参数配置进行修改，参数说明请参见**创建AI应用**。单击“立即创建”，完成新版本的创建操作。

删除版本

在“AI应用”页面，单击AI应用的“版本数量”，在展开的版本列表中，单击“操作”列的“删除”，即可删除对应的版本。

说明

如果AI应用的版本已经部署服务，需先删除关联的服务后再执行删除操作。版本删除后不可恢复，请谨慎操作。

删除 AI 应用

在“AI应用”页面，单击AI应用“操作”列的“删除”，即可删除对应的AI应用。

说明

如果AI应用的版本已经部署服务，需先删除关联的服务后再执行删除操作。AI应用删除后不可恢复，请谨慎操作。

2.6 查看 AI 应用的事件

创建AI应用的（从用户可看见创建AI应用任务开始）过程中，每一个关键事件点在系统后台均有记录，用户可随时在对应AI应用的详情页面进行查看。

方便用户更清楚的了解创建AI应用过程，遇到任务异常时，更加准确的排查定位问题。可查看的事件点包括：

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	开始导入模型。 Start model import.	-

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
异常	构建镜像失败。 Failed to build the image.	构建镜像失败原因较多，需根据具体的报错定位和处理问题。FAQ
异常	自定义镜像不支持指定依赖。 Customize model does not support dependencies.	自定义镜像导入不支持配置运行时依赖，在构建镜像的dockerfile文件中安装pip依赖包。FAQ
异常	非自定义镜像不支持指定swr_location字段。 Non-custom type models should not contain swr_location.	请删除模型配置文件config.json中的swr_location字段后重试。
异常	自定义镜像健康检查接口必须是xxx。 The health check url of custom image model must be %s.	请修改自定义镜像健康检查接口后重试。
正常	当前镜像构建任务状态为xxx。 The status of the image building task is %s.	-
异常	镜像xxx不存在名为xxx的标签。 Image %s does not have the %s tag.	请联系技术支持。
异常	模型配置文件包含非法参数值：xxx。 Invalid %s in config.json.	请删除模型配置文件中的非法参数后重试。
异常	获取镜像xxx的标签列表失败。 Failed to obtain the tag list of image %s.	请联系技术支持。
异常	xxx大于xxxG，无法导入。 %s [%s] is larger than %dG and cannot be imported.	模型或镜像大小超过限制，请精简模型或镜像后，重新导入。FAQ
异常	用户xxx没有OBS的obs:object:PutObjectAcl权限。 User %s does not have obs:object:PutObjectAcl permission	子用户没有OBS的obs:object:PutObjectAcl权限，为子用户添加委托权限。FAQ
异常	镜像构建任务超时。限制超时时间为xxx分钟。 Image building task timeout. The %s-minute limit is over.	imagePacker构建镜像有超时时间限制，请精简代码，提高编译效率。FAQ

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	模型描述已更新。 Model description updated.	-
正常	模型运行时依赖未更新。 Model running dependencies not updated.	-
正常	模型运行时依赖已更新。正在重新构建镜像 Model running dependencies updated. Rebuild the image.	-
异常	触发SWR限流，请稍后重试。 The throttling threshold of swr has been reached.	触发SWR限流，请稍后重试。
正常	系统升级中，请稍后重试。 System is upgrading, please try again later.	-
异常	获取源镜像失败。认证错误，token已失效。 Failed to access source image. Authenticate Error, token expired.	请联系技术支持。
异常	获取源镜像失败。检查该镜像是否存在。 Failed to access source image. Check whether the image exists.	请联系技术支持。
正常	源镜像大小计算完成。 Source image size calculated successfully.	-
正常	源镜像共享成功。 Source image shared successfully.	-
异常	构建镜像失败，因为触发了限流。请稍后重试。 Failed to build the image due to the threshold has been reached. Please try again later.	触发了限流，请稍后重试。
异常	发送构建镜像请求失败。 Failed to send image building request.	请联系技术支持。

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
异常	共享源镜像失败。请检查镜像是否存在或者是否有权限共享该镜像。 Failed to share source image. Check whether the image exists or whether you have the share permission on the image.	请检查镜像是否存在或者是否有权限共享该镜像。
正常	模型导入成功。 Model imported successfully.	-
正常	模型文件导入成功。 Model file imported successfully.	-
正常	模型大小计算完成。 Model size calculated successfully.	-
异常	模型导入失败。 Failed to import the model.	模型导入失败情况较多，请参考**FAQ**定位和处理。
异常	复制模型文件失败，请检查OBS权限是否正常。 Failed to copy model file due to obs exception. Please Check your obs access right.	请检查OBS权限是否正常。**FAQ**
异常	镜像构建任务调度失败。 Image building task scheduling failed.	请联系技术支持。
异常	启动镜像构建任务失败。 Failed to start the image building task.	请联系技术支持。
异常	罗马镜像构建完成，无法分享给资源租户。 The ROMA image is successfully built but cannot be shared to resource tenants.	请联系技术支持。
正常	镜像构建完成。 Image built successfully.	-
正常	启动镜像构建任务。 Start the image building task.	-
正常	启动环境镜像构建任务。 Start the env image building task.	-

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	收到构建模型环境镜像请求。 Received another env image building request of the model.	-
正常	收到构建模型镜像请求。 Received another image building request of the model.	-
正常	使用现有环境镜像。 Use cached env image.	-
异常	构建镜像失败。详细信息请查看构建日志。 Failed to build the image. For details, view the building log.	查看构建日志定位和处理问题。**FAQ**
异常	因系统内部原因构建镜像失败。请联系技术支持。 Failed to build the image due to system errors. Contact the administrator.	请联系技术支持。
异常	模型文件xxx大于5G，无法导入。 Model file %s is larger than 5G and cannot be imported.	模型文件xxx大于5G，请精简模型文件后重试，或者使用动态加载功能进行导入。**FAQ**
异常	因系统内部原因创建OBS桶失败，请联系技术支持。 Failed to create bucket due to system errors. Contact the administrator.	请联系技术支持。
异常	模型大小计算失败。子路径xxx在路径xxx下不存在。 Model size calculated failed.Can not find %s child directory in current model directory %s.	修改子路径为正确的路径后重试，或者联系技术支持。
异常	模型大小计算失败。xxx类型模型不存在路径xxx下。 Model size calculated failed.Can not find %s file in current model directory %s.	检查xxx类型模型的存储位置，修改为正确的路径后重试，或者联系技术支持。

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
提示	模型大小计算失败。多于一个xxx模型文件在路径xxx下。 Model size calculated failed.Find more than one %s file in current model directory %s.	-

创建AI应用的过程中，关键事件支持手动/自动刷新。

查看操作

1. 在ModelArts管理控制台的左侧导航栏中选择“AI应用”，在AI应用列表中，您可以单击AI应用名称，进入AI应用详情页面。
2. 在AI应用详情页面，切换到“事件”页签，查看事件信息。

3 部署 AI 应用（部署上线）

3.1 部署 AI 应用（在线服务）

3.1.1 部署为在线服务

AI应用准备完成后，您可以将AI应用部署为在线服务，对在线服务进行预测和调用。

约束与限制

单个用户最多可创建20个在线服务。

前提条件

- 数据已完成准备：已在ModelArts中创建状态“正常”可用的AI应用。
- 由于在线运行需消耗资源，确保账户未欠费。

操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型部署 > 在线服务”，默认进入“在线服务”列表。
2. 在“在线服务”列表中，单击左上角“部署”，进入“部署”页面。
3. 在“部署”页面，填写在线服务相关参数。
   a. 填写基本信息，详细参数说明请参见表3-1。

表 3-1 基本信息参数说明

参数名称	说明
“名称”	在线服务的名称，请按照界面提示规则填写。

参数名称	说明
“是否自动停止”	启用该参数并设置时间后，服务将在指定时间后自动停止。如果不启用此参数，在线服务将一直运行，同时一直收费，自动停止功能可以帮您避免产生不必要的费用。默认开启自动停止功能，且默认值为“1小时后”。 目前支持设置为“1小时后”、“2小时后”、“4小时后”、“6小时后”、“自定义”。如果选择“自定义”的模式，可在右侧输入框中输入1~24范围内的任意整数。
“描述”	在线服务的简要说明。

b. 填写资源池和AI应用配置等关键信息，详情请参见**表3-2**。

表 3-2 参数说明

参数名称	子参数	说明
“资源池”	“公共资源池”	公共资源池有CPU或GPU两种规格，不同规格的资源池，其收费标准不同，详情请参见**产品价格详情**。当前仅支持按需付费模式。
	“专属资源池”	在专属资源池规格中选择对应的规格进行使用。暂不支持选择创建了逻辑子池的物理池。 **说明** • 旧版“专属资源池”将逐渐迁移至新版“专属资源池”。 • 新用户和旧版“专属资源池”迁移完成的老用户在ModelArts管理控制台只能看到新版的“专属资源池”。 • 旧版“专属资源池”未迁移的老用户，可以看到两个专属资源池，其中“专属资源池 New”为新版的专属资源池。
“选择AI应用及配置”	“AI应用来源”	根据您的实际情况选择“自定义应用”或者“订阅应用”。
	“选择AI应用及版本”	选择状态“正常”的AI应用及版本。
	“分流”	设置当前实例节点的流量占比，服务调用请求根据该比例分配到当前版本上。 如您仅部署一个版本的AI应用，请设置为100%。如您添加多个版本进行灰度发布，多个版本分流之和设置为100%。

参数名称	子参数	说明
	“计算节点规格”	请根据界面显示的列表，选择可用的规格，置灰的规格表示当前环境无法使用。 如果公共资源池下规格为空数据，表示当前环境无公共资源。建议使用专属资源池，或者联系系统管理员创建公共资源池。 **说明** 使用所选规格部署服务时，会产生必要的系统消耗，因此服务实际占用的资源会略大于该规格。
	“计算节点个数”	设置当前版本AI应用的实例个数。如果节点个数设置为1，表示后台的计算模式是单机模式；如果节点个数设置大于1，表示后台的计算模式为分布式的。请根据实际编码情况选择计算模式。
	“环境变量”	设置环境变量，注入环境变量到容器实例。为确保您的数据安全，在环境变量中，请勿输入敏感信息，如明文密码。
	“部署超时时间”	用于设置单个模型实例的超时时间，包括部署和启动时间。默认值为20分钟，输入值必须在3到120之间。
	“添加AI应用版本进行灰度发布”	当选择的AI应用有多个版本时，您可以添加多个AI应用版本，并配置其分流占比，完成多版本和灵活流量策略的灰度发布，实现AI应用版本的平滑过渡升级。 **说明** 当前免费计算规格不支持多版本灰度发布。

参数名称	子参数	说明
	“存储挂载”	资源池为专属资源池时显示该参数。在服务运行时将存储卷以本地目录的方式挂载到计算节点（计算实例），模型或输入数据较大时建议使用。 SFS Turbo： ● 文件系统名称：选择对应的SFS Turbo极速文件。不支持选择跨区域（Region）的极速文件系统。 ● 挂载路径：指定容器内部的挂载路径，如“/sfs-turbo-mount/”。请选择全新目录，选择存量目录会覆盖存量文件。 **说明** ● 相同的文件系统只能挂载一次 ，且只能对应一个挂载路径，挂载路径均不可重复。最多可以挂载8个盘。 ● 使用专属资源池部署服务才允许使用存储挂载的能力，并且专属资源池需要打通VPC或关联SFS Turbo。 - 打通VPC为打通SFS Turbo所在VPC和专属资源池网络。 - 关联SFS Turbo：若SFS Turbo为HPC型的文件系统，可使用关联SFS Turbo功能。 ● 选择多挂载时请勿设置存在冲突的挂载路径如相同路径或相似路径如/obs-mount/与/obs-mount/tmp/等。 ● 选择SFS Turbo存储挂载后，请勿删除已经打通的VPC或解除SFS Turbo关联，否则会导致挂载功能无法使用。挂载时默认按客户端umask权限设置，为确保正常使用须在SFS Turbo界面绑定后端OBS存储后设置权限为777。
“服务流量限制”	-	服务流量限制是指每秒内一个服务能够被访问的次数上限。您可以根据实际需求设置每秒流量限制。
“升级为WebSocket”	-	设置在线服务是否部署为WebSocket服务。了解在线服务支持WebSocket，请参考**WebSocket在线服务全流程开发**。 **说明** ● 要求AI应用的元模型来源为从容器镜像中选择，并且镜像支持WebSocket。 ● 设置“升级为WebSocket”后，不支持设置“服务流量限制”。 ● “升级为WebSocket”参数配置，不支持修改。

参数名称	子参数	说明
“运行日志输出”	-	默认关闭，在线服务的运行日志仅存放在ModelArts日志系统，在服务详情页的“日志”支持简单查询。 若开启此功能，在线服务的运行日志会输出存放到云日志服务LTS。LTS自动创建日志组和日志流，默认缓存7天内的运行日志。如需了解LTS专业日志管理功能，请参见**云日志服务**。 **说明** • “运行日志输出”开启后，不支持关闭。 • LTS服务提供的日志查询和日志存储功能涉及计费，详细请参见**了解LTS的计费规则**。 • 请勿打印无用的audio日志文件，这会导致系统日志卡死，无法正常显示日志，可能会出现“Failed to load audio”的报错。
“支持APP认证”	“APP授权配置”	默认关闭。如需开启此功能，请参见**访问在线服务（APP认证）**了解详情并根据实际情况进行设置。

图 3-1 设置 AI 应用相关信息

c. 可选：配置高级选项。

表 3-3 高级选项参数说明

参数名称	说明
“标签”	ModelArts支持对接标签管理服务TMS，在ModelArts中创建资源消耗性任务（例如：创建Notebook、训练作业、推理在线服务）时，可以为这些任务配置标签，通过标签实现资源的多维分组管理。 标签详细用法请参见**ModelArts如何通过标签实现资源分组管理**。 **说明** 可以在标签输入框下拉选择TMS预定义标签，也可以自己输入自定义标签。预定义标签对所有支持标签功能的服务资源可见。租户自定义标签只对自己服务可见。

4. 确认填写信息无误后，根据界面提示完成在线服务的部署。部署服务一般需要运行一段时间，根据您选择的数据量和资源不同，部署时间将耗时几分钟到几十分钟不等。

   说明

   在线服务部署完成后，将立即启动。

   您可以前往在线服务列表，查看在线服务的基本情况。在线服务列表中，刚部署的服务“状态”为“部署中”，当在线服务的“状态”变为“运行中”时，表示服务部署完成。

3.1.2 查看服务详情

当AI应用部署为在线服务成功后，您可以进入“在线服务”页面，来查看服务详情。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>在线服务”，进入“在线服务”管理页面。
2. 单击目标服务名称，进入服务详情页面。

   您可以查看服务的“名称”、“状态”等信息，详情说明请参见**表3-4**。

表 3-4 在线服务配置

参数	说明
名称	在线服务名称。
状态	在线服务当前状态。
来源	在线服务的来源。
服务ID	在线服务的ID。
描述	您可以单击编辑按钮，添加服务描述。
资源池	当前服务使用的资源池规格。若使用公共资源池部署，则不显示该参数。
个性化配置	您可以为在线服务的不同版本设定不同配置条件，并支持携带自定义运行参数，丰富版本分流策略或同一版本内的不同运行配置。您可以打开个性化配置按钮，单击“查看配置”**修改服务个性化配置**。
服务流量限制	服务流量限制是指每秒内一个服务能够被访问的次数上限。
运行日志输出	默认关闭，在线服务的运行日志仅存放在ModelArts日志系统。 启用运行日志输出后，在线服务的运行日志会输出存放到云日志服务LTS。LTS自动创建日志组和日志流，默认缓存7天内的运行日志。如需了解LTS专业日志管理功能，请参见**云日志服务**。 **说明** • “运行日志输出”开启后，不支持关闭。 • LTS服务提供的日志查询和日志存储功能涉及计费，详细请参见**了解LTS的计费规则**。 • 请勿打印无用的audio日志文件，这会导致系统日志卡死，无法正常显示日志，可能会出现“Failed to load audio”的报错。

参数	说明
升级为WebSocket	是否升级为WebSocket服务。

3. 您可以进入在线服务的详情页面，通过切换页签查看更多详细信息，详情说明请参见**表3-5**。

表 3-5 在线服务详情

参数	说明
调用指南	展示API接口公网地址、AI应用信息、输入参数、输出参数。您可以通过复制API接口公网地址，调用服务。如果您支持APP认证方式，可以在调用指南查看API接口公网地址和授权管理详情，包括“应用名称”、“AppKey”、“AppSecret”等信息。您也可以在此处对APP应用进行“添加授权”或“解除授权”的操作。
预测	对在线服务进行预测。具体操作请参见**测试服务**。
实例	查看同步在线服务的实例信息。这里的实例个数和部署服务时设置的“计算节点个数”相对应，若修改服务或服务异常，实例数会有变化。若存在某个实例异常希望重建实例，您可单击“删除”按钮，该实例被删除后会自动新建一个相同计算规格的实例。
配置更新记录	展示“当前配置”详情和“历史更新记录”。 ● “当前配置”：展示AI应用名称、版本、状态、计算节点规格、分流、计算节点个数、部署超时时间、环境变量、存储挂载等信息。专属资源池部署的服务，同时展示资源池信息。 ● “历史更新记录”：展示历史AI应用相关信息。
监控信息	展示当前服务的“资源统计信息”和“AI应用调用次数统计”。 ● “资源统计信息”：包括CPU、内存、GPU、NPU的可用和已用信息。 ● “AI应用调用次数统计”：当前AI应用的调用次数，从AI应用状态为“已就绪”后开始统计。（websocket服务不显示）
事件	展示当前服务使用过程中的关键操作，比如服务部署进度、部署异常的详细原因、服务被启动、停止、更新的时间点等。 事件保存周期为1个月，1个月后自动清理数据。 查看服务的事件类型和事件信息，请参见**查看服务的事件**

参数	说明
日志	展示当前服务下每个AI应用的日志信息。包含最近5分钟、最近30分钟、最近1小时和自定义时间段。 自定义时间段您可以选择开始时间和结束时间。 当服务启用运行日志输出后，页面展示存放到云日志服务LTS中的日志信息。您可以单击“到LTS查看完整日志”查看全量的日志。 日志搜索规则说明： • 不支持带有分词符的字符串搜索（当前默认分词符有 ,'";=()[]{}@&<>/:\n\t\r）。 • 支持关键词精确搜索。关键词指相邻两个分词符之间的单词。 • 支持关键词模糊匹配搜索，例如输入“error”或“er?or”或“rro*”或“er*r”。 • 支持短语精确搜索。例如输入“Start to refresh”。 • 启用运行日志输出前，支持关键词的“与”、“或”组合搜索。格式为“query logs&&erro*”或“query logs||erro*”。启用运行日志输出后，支持关键词的“与”、“或”组合搜索。格式为“query logs AND erro*”或“query logs OR erro*”。
标签	展示服务已添加的标签。支持添加、修改、删除标签。 标签详细用法请参见**ModelArts如何通过标签实现资源分组管理**。
Cloud Shell	允许用户使用ModelArts控制台提供的CloudShell登录运行中在线服务实例容器，详情请见**CloudShell**。

修改服务个性化配置

服务个性化配置规则由配置条件、访问版本、自定义运行参数（包括配置项名称和配置项值）组成。

您可以为在线服务的不同版本设定不同配置条件，并支持携带自定义运行参数。

个性化配置规则的优先级与顺序相对应，从高到低设置。您可以通过拖动个性化配置规则的顺序更换优先级。

当匹配了某一规则后就不再继续下一规则的判断，最多允许配置10个条件。

表 3-6 个性化配置参数

参数	是否必选	说明
配置条件	必选	SPEL（Spring Expression Language）规则的表达式，当前仅支持字符型的“相等”、“matches”和hashCode函数计算。

参数	是否必选	说明
访问版本	必选	服务个性化配置规则对应的访问版本。当匹配到规则时，请求该版本的在线服务。
配置项名称	可选	自定义运行参数的Key值，不超过128个字符。 当需要通过Header（http消息头）携带自定义运行参数至在线服务时，可以配置。
配置项值	可选	自定义运行参数的Value值，不超过256个字符。 当需要通过Header（http消息头）携带自定义运行参数至在线服务时，可以配置。

可以设置以下三种场景：

- 如果在线服务部署多个版本用于灰度发布，可以使用个性化配置实现按用户分流。

  表 3-7 按内置变量配置条件

内置变量	说明
DOMAIN_NAME	调用预测请求的账号名。
DOMAIN_ID	调用预测请求的账号ID。
PROJECT_NAME	调用预测请求的项目名。
PROJECT_ID	调用预测请求的项目ID。
USER_NAME	调用预测请求的用户名。
USER_ID	调用预测请求的用户ID。

  “#”表示引用变量，匹配的字符串需要用单引号。

  ```
  #{内置变量} == '字符串'
  #{内置变量} matches '正则表达式'
  ```

  - 示例一：

    当调用预测请求的账号名为“zhangsan”时，匹配至指定版本。

    ```
    #DOMAIN_NAME == 'zhangsan'
    ```

  - 示例二：

    当调用预测请求的账号名以“op”开头时，匹配至指定版本。

    ```
    #DOMAIN_NAME matches 'op.*'
    ```

表 3-8 常用的正则匹配表达式

字符	描述
“.”	匹配除“\n”之外的任何单个字符串。需匹配包括“\n”在内的任何字符，请使用“(.|\n)”的模式。
“*”	匹配前面的子表达式零次或多次。例如，“zo*”能匹配“z”以及“zoo”。
“+”	匹配前面的子表达式一次或多次。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。
“?”	匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“does”或“does”中的“do”。
“^”	匹配输入字符串的开始位置。
“$”	匹配输入字符串的结束位置。
“{n}”	*n*是一个非负整数。匹配确定的*n*次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个“o”。
“x|y”	匹配x或y。例如，“z|food”能匹配“z”或“food”。“(z|f)ood”则匹配“zood”或“food”。
“[xyz]”	字符集合。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。

图 3-2 按用户分流

- 如果在线服务部署多个版本用于灰度发布，可以使用个性化配置实现通过Header来访问不同版本。

  您需要通过"#HEADER_"开头说明引用header作为条件

  ```
  #HEADER_{key} == '{value}'
  #HEADER_{key} matches '{value}'
  ```

  - 示例一：

    当预测的http请求的header中存在version，且值为0.0.1则符合条件。不存在此header或者值不为0.0.1都不符合条件。

    ```
    #HEADER_version == '0.0.1'
    ```

  - 示例二：

    当预测的http请求的header中存在testheader且值符合正以mock开头时，可匹配到这条规则。

```
#HEADER_testheader matches 'mock.*'
```

- 示例三：

  当预测的http请求的header中存在uid且其哈希值符合指定的分桶算法时，可匹配到这条规则。

```
#HEADER_uid.hashCode() % 100 < 10
```

图 3-3 通过 Header 访问不同版本

- 如果在线服务部署的版本支持使用不同的运行配置，您可以通过“配置项名称”和“配置项值”携带自定义运行参数至在线服务，实现不同用户使用不同运行配置。

  示例：

  用户zhangsan访问时，AI应用使用配置A；用户lisi访问时，AI应用使用配置B。当匹配到运行配置条件时，ModelArts会在请求里增加一个Header，传入自定义运行参数，其中Key是“配置项名称”，Value是“配置项值”。

图 3-4 个性化配置规则支持传入自定义运行参数。

3.1.3 测试服务

AI应用部署为在线服务成功后，您可以在“预测”页签进行代码调试或添加文件测试。根据AI应用定义的输入请求不同（JSON文本或文件），测试服务包括如下两种方式：

- **JSON文本预测**：如当前部署服务的AI应用，其输入类型指定的为JSON文本类，即不含有文件类型的输入，可以在“预测”页签输入JSON代码进行服务预测。
- **文件预测**：如当前部署服务的AI应用，其输入类型指定为文件类，可包含图片、音频或视频等场景，可以在“预测”页签添加图片进行服务预测。

说明

- 如果您的输入类型为图片，请注意测试服务单张图片输入应小于8MB。
- JSON文本预测，请求体的大小不超过8MB。
- 因APIG（API网关）的限制，单次预测的时间不能超过40S。
- 图片支持以下类型：“png”、“psd”、“jpg”、“jpeg”、“bmp”、“gif”、“webp”、“psd”、“svg”、“tiff”。
- 若服务部署时使用的是“Ascend”规格，则无法预测含有透明度的PNG图片，因为Ascend仅支持RGB-3通道的图片。
- 该功能为调测使用，实际生产建议使用API调用。根据鉴权方式的不同，可以根据实际情况选择**访问在线服务（Token认证）**、**访问在线服务（AK/SK认证）**或者**访问在线服务（APP认证）**。

了解服务的输入参数

针对您部署上线的服务，您可以在服务详情页面的“调用指南”中，了解本服务的输入参数，即上文提到的输入请求类型。

图 3-5 查看服务的调用指南

调用指南中的输入参数取决于您选择的AI应用来源：

- 如果您的元模型来源于自动学习或预置算法，其输入输出参数由ModelArts官方定义，请直接参考“调用指南”中的说明，并在预测页签中输入对应的JSON文本或文件进行服务测试。
- 如果您的元模型是自定义的，即推理代码和配置文件是自行编写的（**配置文件编写说明**），“调用指南”只是将您编写的配置文件进行了可视化展示。调用指南的输入参数与配置文件对应关系如下所示。

图 3-6 配置文件与调用指南的对应关系

- 如果您的元模型是采用模型模板导入，不同的模板指定了其对应的输入输出模式，请参见**模型模板简介**的相关说明。

JSON 文本预测

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>在线服务”，进入“在线服务”管理页面。
2. 单击目标服务名称，进入服务详情页面。在“预测”页签的预测代码下，输入预测代码，然后单击“预测”即可进行服务的预测。

文件预测

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>在线服务”，进入“在线服务”管理页面。
2. 单击目标服务名称，进入服务详情页面。在“预测”页签，单击“上传”，然后选择测试文件。文件上传成功后，单击“预测”即可进行服务的预测，如图3-7所示，输出标签名称，以及位置坐标和检测的评分。

图 3-7 图片预测

3.1.4 访问在线服务

3.1.4.1 访问在线服务简介

在线服务的状态处于“运行中”，则表示在线服务已部署成功，部署成功的在线服务，将为用户提供一个可调用的API，此API为标准Restful API。在集成至生产环境之前，需要对此API进行调测。

在线服务的API默认为HTTPS访问，同时还支持WebSocket访问。在线服务部署时如果选择了“升级为WebSocket”，服务部署完成后，API接口公网地址将是一个WebSocket协议地址。请参见**Websocket访问在线服务**。

当前ModelArts支持访问在线服务的认证方式有以下方式（均以HTTPS请求为例）：

- **Token认证**
- **AK/SK认证**
- **APP认证**

ModelArts支持通过以下几种方式调用API访问在线服务：

- **访问在线服务（公网访问通道）**
- **访问在线服务（VPC高速访问通道）**

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。

3.1.4.2 认证方式

3.1.4.2.1 访问在线服务（Token 认证）

若在线服务的状态处于“运行中”，则表示在线服务已部署成功，部署成功的在线服务，将为用户提供一个可调用的API，此API为标准Restful API。在集成至生产环境之前，需要对此API进行调测，您可以使用以下方式向在线服务发起预测请求：

- **方式一：使用图形界面的软件进行预测（以Postman为例）**。Windows系统建议使用Postman。
- **方式二：使用curl命令发送预测请求**。Linux系统建议使用curl命令。
- **方式三：使用Python语言发送预测请求**。
- **方式四：使用Java语言发送预测请求**。

前提条件

已经获取**用户Token**、**预测文件的本地路径**、**在线服务的调用地址**和**在线服务的输入参数**信息。

- 用户Token的获取请参见**获取Token认证**.。获取Token认证时，由于ModelArts生成的在线服务API不支持domain范围的token，因此需获取使用范围为project的Token信息，即scope参数的取值为project。
- 预测文件的本地路径既可使用绝对路径（如Windows格式"D:/test.png"，Linux格式"/opt/data/test.png"），也可以使用相对路径（如"./test.png"）。
- 在线服务的调用地址和输入参数信息，可以在控制台的“在线服务详情 > 调用指南”页面获取。

  “API接口公网地址”即在线服务的调用地址。当模型配置文件中apis定义了路径，调用地址后需拼接自定义路径。如：“{在线服务的调用地址}/predictions/poetry”。

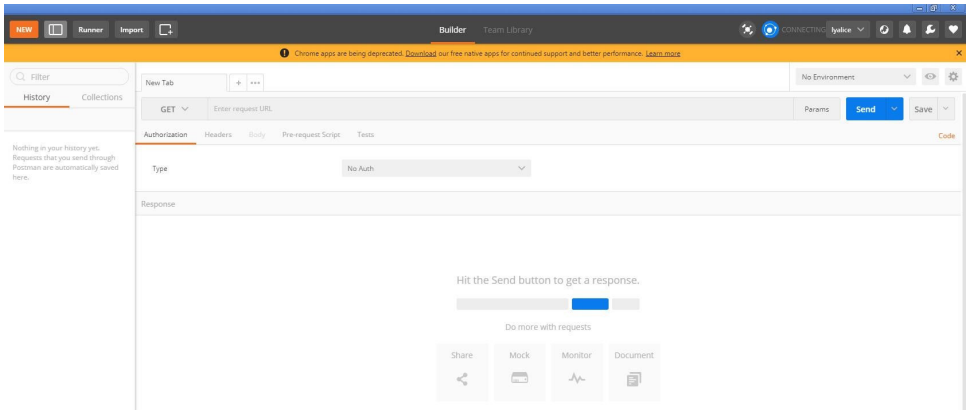
图 3-8 获取在线服务 API 接口地址和文件预测输入参数信息

图 3-9 获取在线服务 API 接口地址和文本预测输入参数信息

方式一：使用图形界面的软件进行预测（以 Postman 为例）

1. 下载Postman软件并安装，您也可以直接在Chrome浏览器添加Postman扩展程序（也可使用其他支持发送post请求的软件）。Postman推荐使用7.24.0版本。
2. 打开Postman，如图3-10所示。

图 3-10 Postman 界面

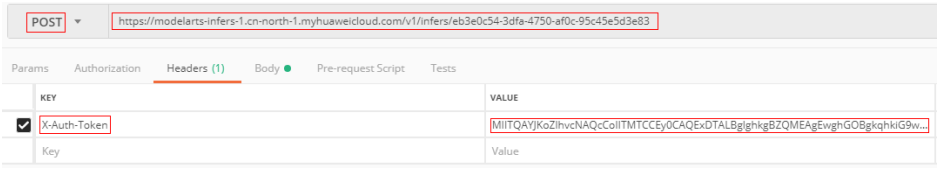


3. 在Postman界面填写参数，以图像分类举例说明。
   - 选择POST任务，将在线服务的调用地址复制到POST后面的方框。Headers页签的Key值填写为“X-Auth-Token”，Value值为用户Token。

     说明

     您也可以通过AK（Access Key ID）/SK（Secret Access Key)加密调用请求，具体可参见**用户AK-SK认证模式**。

图 3-11 参数填写

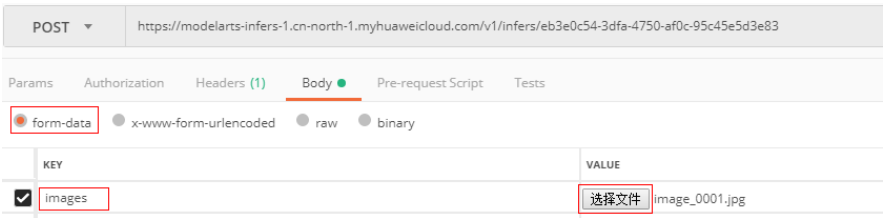


- 在Body页签，根据AI应用的输入参数不同，可分为2种类型：文件输入、文本输入。

  - **文件输入**

    选择“form-data”。在“KEY”值填写AI应用的入参，和在线服务的输入参数对应，比如本例中预测图片的参数为“images”。然后在“VALUE”值，选择文件，上传一张待预测图片（当前仅支持单张图片预测），如图3-12所示。

    图 3-12 填写 Body

  - **文本输入**

    选择“raw”，选择JSON(application/json)类型，在下方文本框中填写请求体，请求体样例如下：

    ```
    {
     "meta": {
      "uuid": "10eb0091-887f-4839-9929-cbc884f1e20e"
     },
     "data": {
      "req_data": [
       {
        "sepal_length": 3,
        "sepal_width": 1,
        "petal_length": 2.2,
        "petal_width": 4
       }
      ]
     }
    }
    ```

    其中，“meta”中可携带“uuid”，调用时传入一个“uuid”，返回预测结果时回传此“uuid”用于跟踪请求，如无此需要可不填写meta。“data”包含了一个“req_data”的数组，可传入单条或多条请求数据，其中每个数据的参数由AI应用决定，比如本例中的“sepal_length”、“sepal_width”等。

4. 参数填写完成，单击“send”发送请求，结果会在“Response”下的对话框里显示。

   - 文件输入形式的预测结果样例如图3-13所示，返回结果的字段值根据不同AI应用可能有所不同。
   - 文本输入形式的预测结果样例如图3-14所示，请求体包含“meta”及“data”。如输入请求中包含“uuid”，则输出结果中回传此“uuid”。如未输入，则为空。“data”包含了一个“resp_data”的数组，返回单条或多

条输入数据的预测结果，其中每个结果的参数由AI应用决定，比如本例中的“sepal_length”、“predictresult”等。

图 3-13 文件输入预测结果

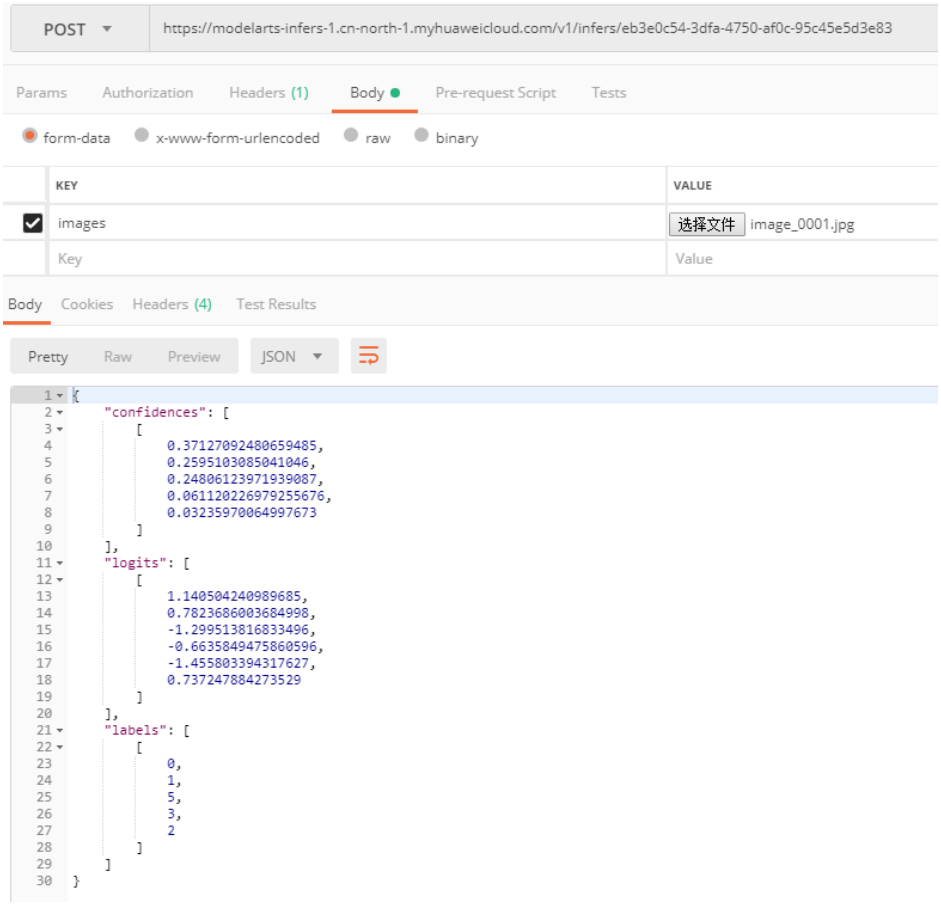
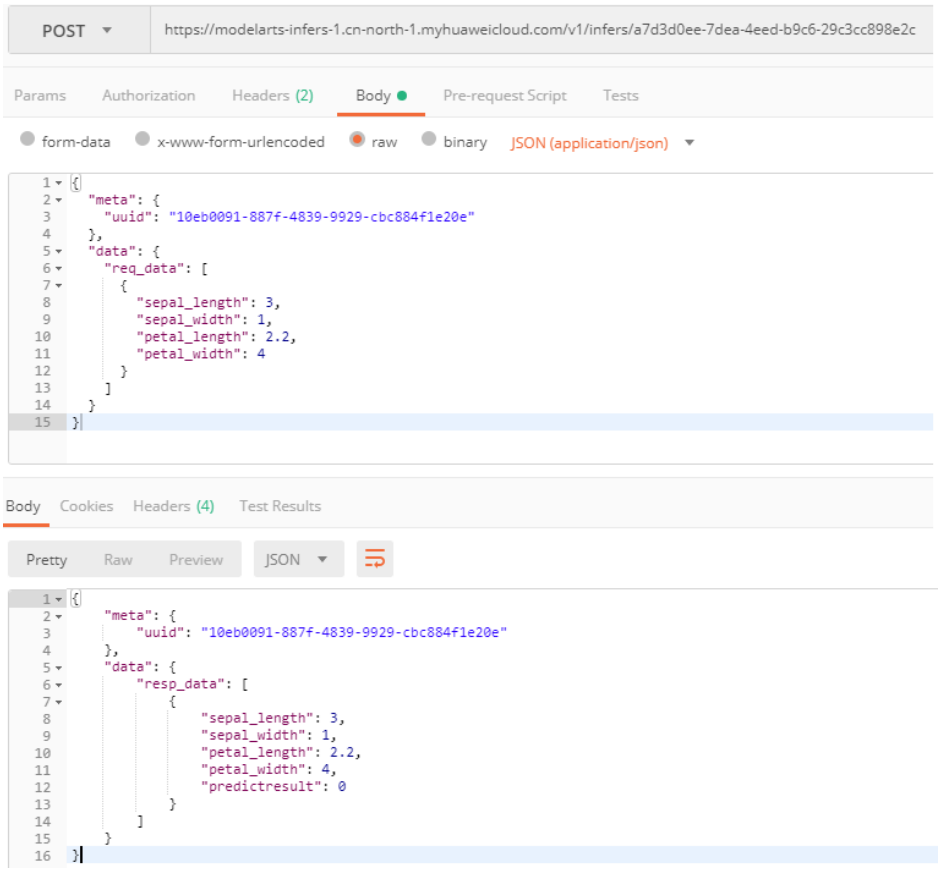


图 3-14 文本输入预测结果



方式二：使用 curl 命令发送预测请求

使用curl命令发送预测请求的命令格式也分为文件输入、文本输入两类。

- 文件输入
  curl -kv -F '**images**=@**图片路径**' -H '**X-Auth-Token**:**Token值**' -X POST **在线服务地址**
  - “-k”是指允许不使用证书到SSL站点。
  - “-F”是指上传数据的是文件，本例中参数名为“images”，这个名字可以根据具体情况变化，@后面是图片的存储路径。
  - “-H”是post命令的headers，Headers的Key值为“X-Auth-Token”，这个名字为固定的， Token值是获取的用户Token。
  - “POST”后面跟随的是在线服务的调用地址。

  curl命令文件输入样例：

  ```
  curl -kv -F 'images=@/home/data/test.png' -H 'X-Auth-Token:MIISkAY***80T9wHQ==' -X POST https://modelarts-infers-1.xxx/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
  ```

- 文本输入
  ```
  curl -kv -d '{"data":{"req_data":[{"sepal_length":3,"sepal_width":1,"petal_length":2.2,"petal_width":4}]}}' -H 'X-Auth-Token:MIISkAY***80T9wHQ==' -H 'Content-type: application/json' -X POST https://modelarts-infers-1.xxx/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
  ```

  “-d”是Body体的文本内容。

方式三：使用 Python 语言发送预测请求

1. 下载Python SDK并在开发工具中完成SDK配置。具体操作请参见**在Python环境中集成API请求签名的SDK**。
2. 创建请求体，进行预测请求。
   - **输入为文件格式**

     ```
     # coding=utf-8

     import requests

     if __name__ == '__main__':
         # Config url, token and file path.
         url = "在线服务的调用地址"
         token = "用户Token"
         file_path = "预测文件的本地路径"

         # Send request.
         headers = {
             'X-Auth-Token': token
         }
         files = {
             'images': open(file_path, 'rb')
         }
         resp = requests.post(url, headers=headers, files=files)

         # Print result.
         print(resp.status_code)
         print(resp.text)
     ```

     “files”中的参数名由在线服务的输入参数决定，需要和“类型”为“file”的输入参数“名称”保持一致。以**前提条件**里获取的文件预测输入参数“images”为例。
   - **输入为文本格式（json类型）**

     读取本地预测文件并进行base64编码的请求体示例如下：

     ```
     # coding=utf-8

     import base64
     import requests

     if __name__ == '__main__':
         # Config url, token and file path
         url = "在线服务的调用地址"
         token = "用户Token"
         file_path = "预测文件的本地路径"
         with open(file_path, "rb") as file:
             base64_data = base64.b64encode(file.read()).decode("utf-8")

         # Set body,then send request
         headers = {
             'Content-Type': 'application/json',
             'X-Auth-Token': token
         }
         body = {
             'image': base64_data
         }
         resp = requests.post(url, headers=headers, json=body)

         # Print result
         print(resp.status_code)
         print(resp.text)
     ```

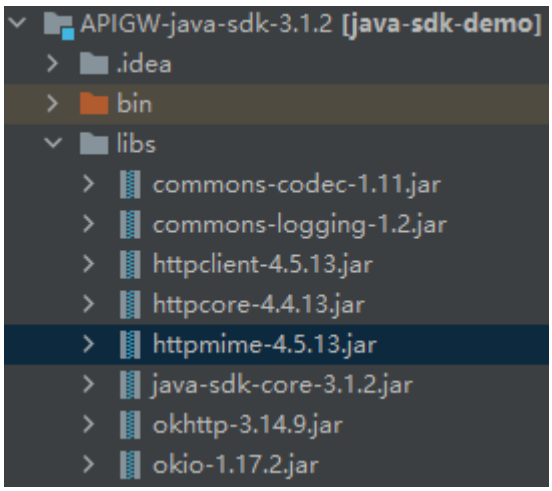
     “body”中的参数名由在线服务的输入参数决定，需要和“类型”为“string”的输入参数“名称”保持一致。以**前提条件**里获取的文本预测输入参数“image”为例。“body”中的base64_data值为string类型。

方式四：使用 Java 语言发送预测请求

1. 下载Java SDK并在开发工具中完成SDK配置。具体操作请参见**在Java环境中集成API请求签名的SDK**。
2. （可选）当预测请求的输入为文件格式时，Java工程依赖httpmime模块。
   a. 在工程“libs”中增加httpmime-x.x.x.jar。完整的Java依赖库如**图3-15**所示。

      httpmime-x.x.x.jar建议使用4.5及以上版本，下载地址：**https://mvnrepository.com/artifact/org.apache.httpcomponents/httpmime**。

      图 3-15 Java 依赖库

   b. httpmime-x.x.x.jar添加完成后，在Java工程的.classpath文件中，补充httpmime信息，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
<classpathentry kind="src" path="src"/>
<classpathentry kind="lib" path="libs/commons-codec-1.11.jar"/>
<classpathentry kind="lib" path="libs/commons-logging-1.2.jar"/>
<classpathentry kind="lib" path="libs/httpclient-4.5.13.jar"/>
<classpathentry kind="lib" path="libs/httpcore-4.4.13.jar"/>
<classpathentry kind="lib" path="libs/httpmime-x.x.x.jar"/>
<classpathentry kind="lib" path="libs/java-sdk-core-3.1.2.jar"/>
<classpathentry kind="lib" path="libs/okhttp-3.14.9.jar"/>
<classpathentry kind="lib" path="libs/okio-1.17.2.jar"/>
<classpathentry kind="output" path="bin"/>
</classpath>
```

3. 创建Java类，进行预测请求。
   - **输入为文件格式**

     Java的请求体示例如下：

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.Consts;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
```



```
import java.io.File;

public class MyTokenFile {

    public static void main(String[] args) {
        // Config url, token and filePath
        String url = "在线服务的调用地址";
        String token = "用户Token";
        String filePath = "预测文件的本地路径";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader("X-Auth-Token", token);

            // Add a body if you have specified the PUT or POST method. Special characters, such as the double quotation mark ("), contained in the body must be escaped.
            File file = new File(filePath);
            HttpEntity entity = MultipartEntityBuilder.create().addBinaryBody("images", file).setContentType(ContentType.MULTIPART_FORM_DATA).setCharset(Consts.UTF_8).build();
            httpPost.setEntity(entity);

            // Send post
            CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

“addBinaryBody”中的参数名由在线服务的输入参数决定，需要和“类型”为“file”的输入参数“名称”保持一致。此处以**前提条件**里获取的“images”为例。

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyTokenTest {

    public static void main(String[] args) {
        // Config url, token and body
        String url = "在线服务的调用地址";
        String token = "用户Token";
        String body = "{}";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
            httpPost.setHeader("X-Auth-Token", token);

            // Special characters, such as the double quotation mark ("), contained in the body
```

```
must be escaped.
            httpPost.setEntity(new StringEntity(body));

            // Send post.
            CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

“body”由具体文本格式决定，此处以json为例。

3.1.4.2.2 访问在线服务（AK/SK 认证）

若在线服务的状态处于“运行中”，则表示在线服务已部署成功。部署成功的在线服务，将为用户提供一个可调用的API，此API为标准Restful API。用户可以通过AK/SK签名认证方式调用API。

使用AK/SK认证时，您可以通过APIG SDK访问，也可以通过ModelArts SDK访问。使用ModelArts SDK访问参见**用户AK-SK认证模式**。本文档详细介绍如何通过APIG SDK访问在线服务，具体操作流程如下：

1. **获取AK/SK**
2. **获取在线服务信息**
3. 发送预测请求
   - **方式一：使用Python语言发送预测请求**
   - **方式二：使用Java语言发送预测请求**

说明

1. AK/SK签名认证方式，仅支持Body体12M以内，12M以上的请求，需使用Token认证。
2. 客户端须注意本地时间与时钟服务器的同步，避免请求消息头X-Sdk-Date的值出现较大误差。因为API网关除了校验时间格式外，还会校验该时间值与网关收到请求的时间差，如果时间差超过15分钟，API网关将拒绝请求。

获取 AK/SK

如果已生成过AK/SK，则可跳过此步骤，找到原来已下载的AK/SK文件，文件名一般为：credentials.csv。

如下图所示，文件包含了租户名（User Name），AK（Access Key Id），SK（Secret Access Key）。

图 3-16 credential.csv 文件内容

	A	B	C
1	User Name	Access Key Id	Secret Access Key
2	hu[redacted]dg	QTWA[redacted]UT2QVKYUC	MFyfvK41ba2[redacted]npdUKGpownRZlmVmHc

AK/SK生成步骤：

1. 注册并登录管理控制台。
2. 单击右上角的用户名，在下拉列表中单击“我的凭证”。

3. 单击“访问密钥”。
4. 单击“新增访问密钥”，进入“身份验证”页面。
5. 根据提示完成身份验证，下载密钥，并妥善保管。

获取在线服务信息

在调用接口时，需获取在线服务的调用地址，以及在线服务的输入参数信息。步骤如下：

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型部署 > 在线服务”，默认进入“在线服务”列表。
2. 单击目标服务名称，进入服务详情页面。
3. 在“在线服务”的详情页面，可以获取该服务的调用地址和输入参数信息。
   “API接口公网地址”即在线服务的调用地址。当模型配置文件中apis定义了路径，调用地址后需拼接自定义路径。如：“{在线服务的调用地址}/predictions/poetry”。

图 3-17 获取在线服务 API 接口地址和文件预测输入参数信息

图 3-18 获取在线服务 API 接口地址和文本预测输入参数信息

方式一：使用 Python 语言发送预测请求

1. 下载Python SDK并在开发工具中完成SDK配置。具体操作请参见**在Python环境中集成API请求签名的SDK**。
2. 创建请求体，进行预测请求。
   - **输入为文件格式**

```
# coding=utf-8

import requests
```

```
import os
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "在线服务的调用地址"
    # 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以ak和sk保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    ak = os.environ["HUAWEICLOUD_SDK_AK"]
    sk = os.environ["HUAWEICLOUD_SDK_SK"]
    file_path = "预测文件的本地路径"

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {"x-sdk-content-sha256": "UNSIGNED-PAYLOAD"}
    request = signer.HttpRequest(method, url, headers)

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = ak
    sig.Secret = sk
    sig.Sign(request)

    # Send request
    files = {'images': open(file_path, 'rb')}
    resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri, headers=request.headers, files=files)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

“file_path”为预测文件的本地路径，既可使用绝对路径（如Windows格式"D:/test.png"，Linux格式"/opt/data/test.png"），也可以使用相对路径（如"./test.png"）。

“files”参数的请求体样式为“files={"请求参数":("文件路径"，文件内容，“文件类型”)}”，参数填写可以参考**表3-9**。

表 3-9 files 参数说明

参数	是否必填	说明
请求参数	是	在线服务输入参数名称。
文件路径	否	上传文件的路径。
文件内容	是	上传文件的内容。
文件类型	否	上传文件类型。当前支持以下类型： • txt类型：text/plain • jpg/jpeg类型：image/jpeg • png类型：image/png

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
# coding=utf-8

import base64
```

```
import json
import os
import requests
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "在线服务的调用地址"
    # 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以ak和sk保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    ak = os.environ["HUAWEICLOUD_SDK_AK"]
    sk = os.environ["HUAWEICLOUD_SDK_SK"]
    file_path = "预测文件的本地路径"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {
        'Content-Type': 'application/json'
    }
    body = {
        'image': base64_data
    }
    request = signer.HttpRequest(method, url, headers, json.dumps(body))

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = ak
    sig.Secret = sk
    sig.Sign(request)

    # Send request
    resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri,
headers=request.headers, data=request.body)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

“body”中的参数名由在线服务的输入参数决定，需要和“类型”为“string”的输入参数“名称”保持一致。此处以“image”为例。“body”中的base64_data值为string类型。

方式二：使用 Java 语言发送预测请求

1. 下载Java SDK并在开发工具中完成SDK配置。
2. 创建Java类，进行预测请求。

   由于在APIG的Java SDK中，“request.setBody()”只支持String类型，所以只支持输入为文本格式的预测请求。如果输入的是文件格式，需要先进行base64编码转换成文本。

   - **输入为文件格式**

     此处以json格式为例介绍读取本地预测文件并进行base64编码的请求体，请求体示例如下：

```
package com.apig.sdk.demo;
import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.commons.codec.binary.Base64;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
```



```
import org.apache.http.util.EntityUtils;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
public class MyAkSkTest2 {
    public static void main(String[] args) {
        String url = "在线服务的调用地址";
       // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
       // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");
        String filePath = "预测文件的本地路径";
        try {
            // Create request
            Request request = new Request();
            // Set the AK/SK to sign and authenticate the request.
            request.setKey(ak);
            request.setSecret(sk);
            // Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
            request.setMethod(HttpPost.METHOD_NAME);
            // Add header parameters
            request.addHeader(HttpHeaders.CONTENT_TYPE, "application/json");
            // Set a request URL in the format of https://{Endpoint}/{URI}.
            request.setUrl(url);
            // build your json body
            String body = "{\"image\":\"" + getBase64FromFile(filePath) + "\"}";
            // Special characters, such as the double quotation mark ("), contained in the body must be escaped.
            request.setBody(body);
            // Sign the request.
            HttpRequestBase signedRequest = Client.sign(request);
            // Send request.
            CloseableHttpResponse response = HttpClients.createDefault().execute(signedRequest);
            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * Convert the file into a byte array and Base64 encode it
     * @return
     */
    private static String getBase64FromFile(String filePath) {
        // Convert the file into a byte array
        InputStream in = null;
        byte[] data = null;
        try {
            in = new FileInputStream(filePath);
            data = new byte[in.available()];
            in.read(data);
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Base64 encode
        return new String(Base64.encodeBase64(data));
    }
}
```

⚠ 注意

使用base64编码方式，需要在模型推理代码中增加对请求体解码的代码。

- **输入为文本格式（json类型）**

```
// Package name of the demo.
package com.apig.sdk.demo;

import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAkSkTest {

    public static void main(String[] args) {
        String url = "在线服务的调用地址";
       // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
       // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");

        try {
            // Create request
            Request request = new Request();

            // Set the AK/SK to sign and authenticate the request.
            request.setKey(ak);
            request.setSecret(sk);

            // Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
            request.setMethod(HttpPost.METHOD_NAME);

            // Add header parameters
            request.addHeader(HttpHeaders.CONTENT_TYPE, "application/json");

            // Set a request URL in the format of https://{Endpoint}/{URI}.
            request.setUrl(url);

            // Special characters, such as the double quotation mark ("), contained in the body must be escaped.
            String body = "{}";
            request.setBody(body);

            // Sign the request.
            HttpRequestBase signedRequest = Client.sign(request);

            // Send request.
            CloseableHttpResponse response = HttpClients.createDefault().execute(signedRequest);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

“body”由具体文本格式决定，此处以json为例。

3.1.4.2.3 访问在线服务（APP 认证）

部署在线服务支持开启APP认证，即ModelArts会为服务注册一个支持APP认证的接口，为此接口配置APP授权后，用户可以使用授权应用的AppKey+AppSecret或AppCode调用该接口。

针对在线服务的APP认证，具体操作流程如下。

1. **开启支持APP认证功能**：开启支持APP认证功能，选择已有APP应用或者创建新的APP应用。
2. **在线服务授权管理**：对创建的APP应用进行管理，包括查看、重置或删除应用，绑定或解绑应用对应的在线服务，获取“AppKey/AppSecret”或“AppCode”。
3. **APP认证鉴权**：调用支持APP认证的接口需要进行认证鉴权，支持两种鉴权方式（AppKey+AppSecret或AppCode），您可以选择其中一种进行认证鉴权。
4. 发送预测请求：
   - **方式一：使用Python语言通过AppKey+AppSecret认证鉴权方式发送预测请求**
   - **方式二：使用Java语言通过AppKey+AppSecret认证鉴权方式发送预测请求**
   - **方式三：使用Python语言通过AppCode认证鉴权方式发送预测请求**
   - **方式四：使用Java语言通过AppCode认证鉴权方式发送预测请求**

前提条件

- 数据已完成准备：已在ModelArts中创建状态“正常”可用的AI应用。
- 由于在线运行需消耗资源，确保账户未欠费。
- 已获取预测文件的本地路径，可使用绝对路径（如Windows格式"D:/test.png"，Linux格式"/opt/data/test.png"）或相对路径（如"./test.png"）。

开启支持 APP 认证功能

在部署为在线服务时，您可以开启支持APP认证功能。或者针对已部署完成的在线服务，您可以修改服务，开启支持APP认证功能。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署 > 在线服务”，进入在线服务管理页面。
2. 开启支持APP认证功能。
   - 在部署为在线服务时，即“部署”页面，填写部署服务相关参数时，开启支持APP认证功能。
   - 针对已部署完成的在线服务，进入在线服务管理页面，单击目标服务名称“操作”列的“修改”按钮，进入修改服务页面开启支持APP认证功能。

   图 3-19 部署页面开启支持 APP 认证功能

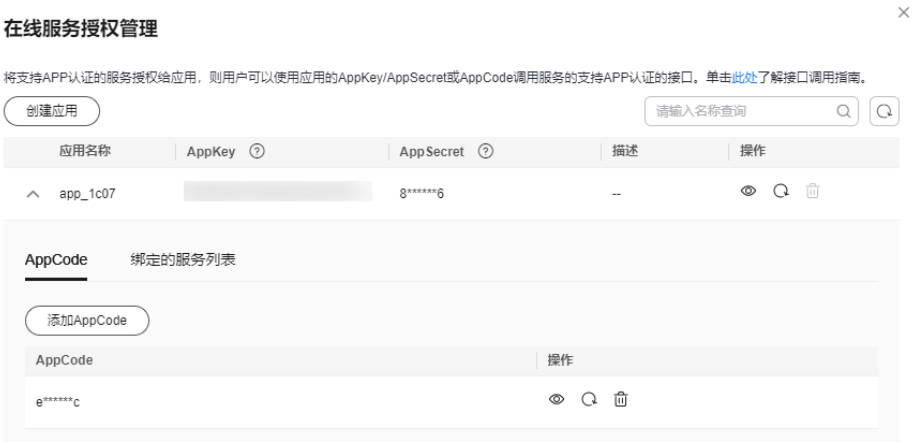
   

3. 选择APP授权配置。从下拉列表中选择您需要配置的APP应用，如果没有可选项，您可以通过如下方式创建应用。
   - 单击右侧“创建应用”，填写应用名称和描述之后单击“确定”完成创建。其中应用名称默认以“app_”开头，您也可以自行修改。
   - 进入“模型部署>在线服务”页面，单击“授权管理”，进入“在线服务授权管理”页面，选择“创建应用”，详请参见**在线服务授权管理**。
4. 开启支持APP认证功能后，将支持APP认证的服务授权给应用，用户可以使用创建的“AppKey/AppSecret”或“AppCode”调用服务的支持APP认证的接口。

在线服务授权管理

如果您需要使用支持APP认证功能，建议您在部署在线服务之前进行授权管理操作完成应用创建。进入“模型部署>在线服务”页面，单击“授权管理”，进入“在线服务授权管理”对话框。在此页面您可以实现应用的创建和管理，包括查询明文、重置或删除应用，解绑应用对应的在线服务，获取“AppKey/AppSecret”或“AppCode”。

图 3-20 在线服务授权管理



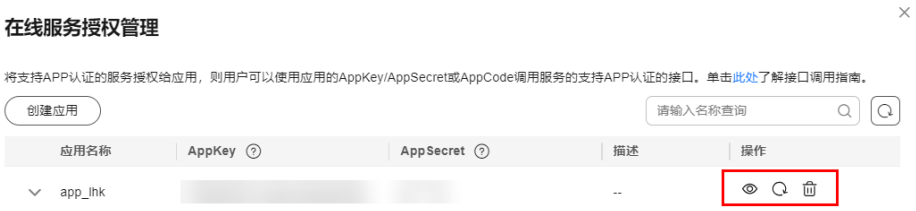
- **创建应用**

  选择“创建应用”，填写应用名称和描述之后单击“确定”完成创建。其中应用名称默认以“app_”开头，您也可以自行修改。

- **查看、重置或删除应用**

  您可以单击目标应用名称操作列的按钮完成应用的查询明文、重置或删除。创建完成后自动生成“AppKey/AppSecret”以供您后续调取接口进行APP鉴权使用。

  图 3-21 查询明文、重置或删除

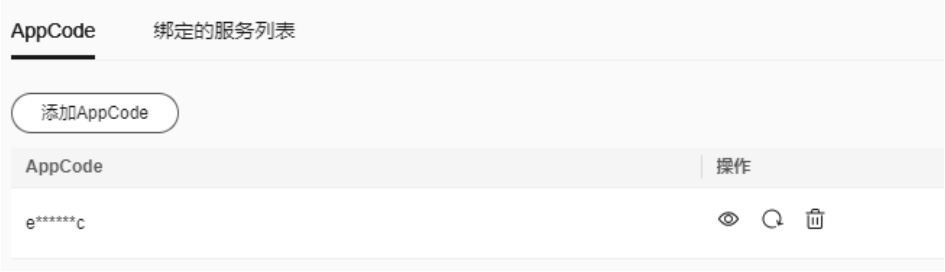
- **解绑服务**

  您可以单击目标应用名称前方的，在下拉列表中展示绑定的服务列表，即该应用对应的在线服务列表。单击操作列的“解绑”取消绑定，将不再支持调用该接口。

- **获取AppKey/AppSecret或AppCode**

  调用接口需要进行APP鉴权，在创建APP应用时自动生成“AppKey/AppSecret”，您可以在“在线服务授权管理”对话框中单击APP应用操作列的 查看完整的**AppSecret**。单击应用名称前方的展开下拉列表，通过单击“添加AppCode”自动生成“AppCode”，您可以单击操作列的 查看完整的**AppCode**。

图 3-22 添加 AppCode



APP 认证鉴权

当支持APP认证功能的在线服务运行成功处于“运行中”状态，就可以对服务进行调用 。在调用之前您需要进行APP认证鉴权。

当使用APP认证，且开启了简易认证模式，API请求既可以选择使用Appkey和AppSecret做签名和校验，也可以选择使用AppCode进行简易认证（ModelArts默认启用简易认证）。推荐使用AppKey/AppSecret认证，其安全性比AppCode认证要高。

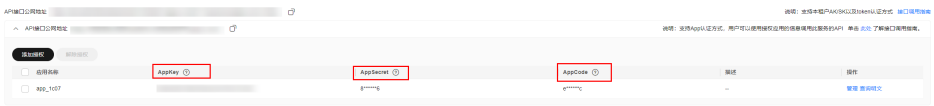
- **AppKey/AppSecret认证**：通过AppKey与AppSecret对请求进行加密签名，可标识发送方并防止请求被修改。使用AppKey/AppSecret认证时，您需要使用专门的签名SDK对请求进行签名。
  - AppKey：APP访问密钥ID。与私有访问密钥关联的唯一标识符；访问密钥ID和私有访问密钥一起使用，对请求进行加密签名。
  - AppSecret：APP私有访问密钥，即与访问密钥ID结合使用的密钥，对请求进行加密签名，可标识发送方，并防止请求被修改。

  AppKey进行简易认证时，即在调用API的时候，在HTTP请求头部消息增加一个参数“apikey”（参数值为“AppKey”），实现快速认证。
- **AppCode认证**：通过AppCode认证通用请求。

  AppCode认证就是在调用API的时候，在HTTP请求头部消息增加一个参数“X-Apig-AppCode”（参数值为“AppCode”），而不需要对请求内容签名，API网关也仅校验AppCode，不校验请求签名，从而实现快速响应。

您可以在服务详情页的“调用指南”页签（如图3-23）或者在线服务授权管理页面（如图3-20）获取API接口和AppKey/AppSecret和AppCode。请注意使用图中红框所示的API接口公网地址。当模型配置文件中apis定义了路径，调用地址后需拼接自定义路径。如：“{在线服务的调用地址}/predictions/poetry”。

图 3-23 获取 API 的接口地址



方式一：使用 Python 语言通过 AppKey+AppSecret 认证鉴权方式发送预测请求

1. 下载Python SDK并在开发工具中完成SDK配置。
2. 创建请求体，进行预测请求。
   - **输入为文件格式**

```
# coding=utf-8

import requests
```

```
import os
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "在线服务的调用地址"
    # 认证用的app_key和app_secret硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以app_key和app_secret保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_KEY和HUAWEICLOUD_APP_SECRET。
    app_key = os.environ["HUAWEICLOUD_APP_KEY"]
    app_secret= os.environ["HUAWEICLOUD_APP_SECRET"]
    file_path = "预测文件的本地路径"

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {"x-sdk-content-sha256": "UNSIGNED-PAYLOAD"}
    request = signer.HttpRequest(method, url, headers)

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = app_key
    sig.Secret = app_secret
    sig.Sign(request)

    # Send request
    files = {'images': open(file_path, 'rb')}
    resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri, headers=request.headers, files=files)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

“files”参数的请求体样式为“files={"请求参数":("文件路径"，文件内容，“文件类型”)}”，参数填写可以参考**表3-10**。

表 3-10 files 参数说明

参数	是否必填	说明
请求参数	是	在线服务输入参数名称。
文件路径	否	上传文件的路径。
文件内容	是	上传文件的内容。
文件类型	否	上传文件类型。当前支持以下类型： • txt类型：text/plain • jpg/jpeg类型：image/jpeg • png类型：image/png

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
# coding=utf-8

import base64
import json
import os
import requests
from apig_sdk import signer
```

```
if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "在线服务的调用地址"
    # 认证用的app_key和app_secret硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以app_key和app_secret保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_KEY和HUAWEICLOUD_APP_SECRET。
    app_key = os.environ["HUAWEICLOUD_APP_KEY"]
    app_secret= os.environ["HUAWEICLOUD_APP_SECRET"]
    file_path = "预测文件的本地路径"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {
        'Content-Type': 'application/json'
    }
    body = {
        'image': base64_data
    }
    request = signer.HttpRequest(method, url, headers, json.dumps(body))

    # Create sign, set the AppKey&AppSecret to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = app_key
    sig.Secret = app_secret
    sig.Sign(request)

    # Send request
    resp = requests.request(request.method, request.scheme + "://" + request.host + request.uri, headers=request.headers, data=request.body)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

“body”中的参数名由在线服务的输入参数决定，需要和“类型”为“string”的输入参数“名称”保持一致。此处以“image”为例。“body”中的base64_data值为string类型。

方式二：使用 Java 语言通过 AppKey+AppSecret 认证鉴权方式发送预测请求

1. 下载Java SDK并在开发工具中完成SDK配置。
2. 创建Java类，进行预测请求。

   由于在APIG的Java SDK中，“request.setBody()”只支持String类型，所以只支持输入为文本格式的预测请求。

   此处以json格式为例介绍读取本地预测文件并进行base64编码的请求体：

```
// Package name of the demo.
package com.apig.sdk.demo;

import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAkSkTest {

    public static void main(String[] args) {
        String url = "在线服务的调用地址";
       // 认证用的appKey和appSecret硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件
```



```
或者环境变量中密文存放，使用时解密，确保安全；
    // 本示例以appKey和appSecret保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_KEY和HUAWEICLOUD_APP_SECRET。
     String appKey = System.getenv("HUAWEICLOUD_APP_KEY");
    String appSecret = System.getenv("HUAWEICLOUD_APP_SECRET");
     String body = "{}";

    try {
        // Create request
        Request request = new Request();

        // Set the AK/AppSecret to sign and authenticate the request.
        request.setKey(appKey);
        request.setSecret(appSecret);

        // Specify a request method, such as GET, PUT, POST, DELETE, HEAD, and PATCH.
        request.setMethod(HttpPost.METHOD_NAME);

        // Add header parameters
        request.addHeader(HttpHeaders.CONTENT_TYPE, "application/json");

        // Set a request URL in the format of https://{Endpoint}/{URI}.
        request.setUrl(url);

        // Special characters, such as the double quotation mark ("), contained in the body must be escaped.
        request.setBody(body);

        // Sign the request.
        HttpRequestBase signedRequest = Client.sign(request);

        // Send request.
        CloseableHttpResponse response = HttpClients.createDefault().execute(signedRequest);

        // Print result
        System.out.println(response.getStatusLine().getStatusCode());
        System.out.println(EntityUtils.toString(response.getEntity()));
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

“body”由具体文本格式决定，此处以json为例。

方式三：使用 Python 语言通过 AppCode 认证鉴权方式发送预测请求

1. 下载Python SDK并在开发工具中完成SDK配置。
2. 创建请求体，进行预测请求。
   - **输入为文件格式**

     ```
     # coding=utf-8

     import requests
     import os

     if __name__ == '__main__':
         # Config url, app code and file path.
         url = "在线服务的调用地址"
         # 认证用的app_code硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
         # 本示例以app_code保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_CODE。
         app_code = os.environ["HUAWEICLOUD_APP_CODE"]
         file_path = "预测文件的本地路径"

         # Send request.
         headers = {
             'X-Apig-AppCode': app_code
     ```



```
}
files = {
    'images': open(file_path, 'rb')
}
resp = requests.post(url, headers=headers, files=files)

# Print result
print(resp.status_code)
print(resp.text)
```

“files”中的参数名由在线服务的输入参数决定，需要和“类型”为“file”的输入参数“名称”保持一致。此处以“images”为例。

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
# coding=utf-8

import base64
import requests
import os

if __name__ == '__main__':
    # Config url, app code and request body.
    url = "在线服务的调用地址"
    # 认证用的app_code硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    # 本示例以app_code保存在环境变量中来实现身份验证为例,运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_CODE。
    app_code = os.environ["HUAWEICLOUD_APP_CODE"]
    file_path = "预测文件的本地路径"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Send request
    headers = {
        'Content-Type': 'application/json',
        'X-Apig-AppCode': app_code
    }
    body = {
        'image': base64_data
    }
    resp = requests.post(url, headers=headers, json=body)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

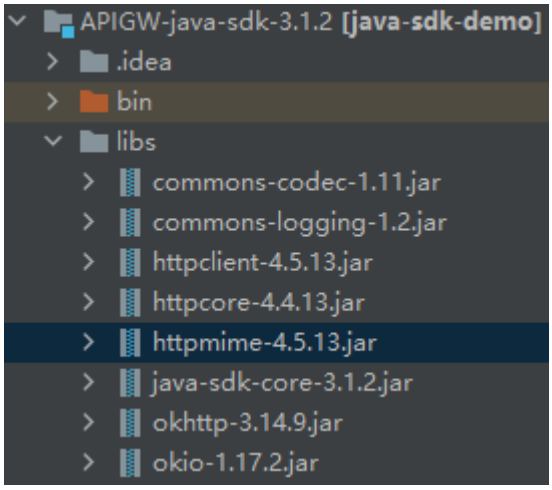
“body”中的参数名由在线服务的输入参数决定，需要和“类型”为“string”的输入参数“名称”保持一致。此处以“image”为例。“body”中的base64_data值为string类型。

方式四：使用 Java 语言通过 AppCode 认证鉴权方式发送预测请求

1. 下载Java SDK并在开发工具中完成SDK配置。
2. （可选）当预测请求的输入为文件格式时，Java工程依赖httpmime模块。
   a. 在工程“libs”中增加httpmime-x.x.x.jar。完整的Java依赖库如图3-24所示。

      httpmime-x.x.x.jar建议使用4.5及以上版本，下载地址：https://mvnrepository.com/artifact/org.apache.httpcomponents/httpmime。

图 3-24 Java 依赖库



b. httpmime-x.x.x.jar添加完成后，在Java工程的.classpath文件中，补充httpmime信息，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
<classpathentry kind="src" path="src"/>
<classpathentry kind="lib" path="libs/commons-codec-1.11.jar"/>
<classpathentry kind="lib" path="libs/commons-logging-1.2.jar"/>
<classpathentry kind="lib" path="libs/httpclient-4.5.13.jar"/>
<classpathentry kind="lib" path="libs/httpcore-4.4.13.jar"/>
<classpathentry kind="lib" path="libs/httpmime-x.x.x.jar"/>
<classpathentry kind="lib" path="libs/java-sdk-core-3.1.2.jar"/>
<classpathentry kind="lib" path="libs/okhttp-3.14.9.jar"/>
<classpathentry kind="lib" path="libs/okio-1.17.2.jar"/>
<classpathentry kind="output" path="bin"/>
</classpath>
```

3. 创建Java类，进行预测请求。
   - **输入为文件格式**

     Java的请求体示例如下：

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.Consts;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import java.io.File;

public class MyAppCodeFile {

    public static void main(String[] args) {
        String url = "在线服务的调用地址";
        // 认证用的appCode硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
        // 本示例以appCode保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_CODE。
        String appCode = System.getenv("HUAWEICLOUD_APP_CODE");
        String filePath = "预测文件的本地路径";

        try {
            // Create post
```



```
        HttpPost httpPost = new HttpPost(url);

        // Add header parameters
        httpPost.setHeader("X-Apig-AppCode", appCode);

        // Special characters, such as the double quotation mark ("), contained in the body must be escaped.
        File file = new File(filePath);
        HttpEntity entity = MultipartEntityBuilder.create().addBinaryBody("images", file).setContentType(ContentType.MULTIPART_FORM_DATA).setCharset(Consts.UTF_8).build();
        httpPost.setEntity(entity);

        // Send post
        CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

        // Print result
        System.out.println(response.getStatusLine().getStatusCode());
        System.out.println(EntityUtils.toString(response.getEntity()));
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

“addBinaryBody”中的参数名由在线服务的输入参数决定，需要和“类型”为“file”的输入参数“名称”保持一致。此处以“images”为例。

- **输入为文本格式（json类型）**

  读取本地预测文件并进行base64编码的请求体示例如下：

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAppCodeTest {

  public static void main(String[] args) {
    String url = "在线服务的调用地址";
    // 认证用的appCode硬编码到代码中或者明文存储都有很大的安全风险,建议在配置文件或者环境变量中密文存放,使用时解密,确保安全;
    // 本示例以appCode保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_APP_CODE。
    String appCode = System.getenv("HUAWEICLOUD_APP_CODE");
    String body = "{}";

    try {
      // Create post
      HttpPost httpPost = new HttpPost(url);

      // Add header parameters
      httpPost.setHeader(HttpHeaders.CONTENT_TYPE, "application/json");
      httpPost.setHeader("X-Apig-AppCode", appCode);

      // Special characters, such as the double quotation mark ("), contained in the body must be escaped.
      httpPost.setEntity(new StringEntity(body));

      // Send post
      CloseableHttpResponse response = HttpClients.createDefault().execute(httpPost);

      // Print result
      System.out.println(response.getStatusLine().getStatusCode());
      System.out.println(EntityUtils.toString(response.getEntity()));
    } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
```

“body”由具体文本格式决定，此处以json为例。

3.1.4.3 访问方式

3.1.4.3.1 访问在线服务（公网访问通道）

背景描述

ModelArts推理默认使用公网访问在线服务。在线服务部署成功后，将为用户提供一个可调用的API，此API为标准Restful API。您可以在服务详情页面，调用指南页签中查看API接口公网地址。

图 3-25 API 接口公网地址

访问在线服务

公网访问在线服务有以下认证方式，API调用请参见认证详情：

- **访问在线服务（Token认证）**
- **访问在线服务（AK/SK认证）**
- **访问在线服务（APP认证）**

3.1.4.3.2 访问在线服务（VPC 高速访问通道）

背景说明

访问在线服务的实际业务中，用户可能会存在如下需求：

- 高吞吐量、低时延
- TCP或者RPC请求

因此，ModelArts提供了VPC直连的高速访问通道功能以满足用户的需求。

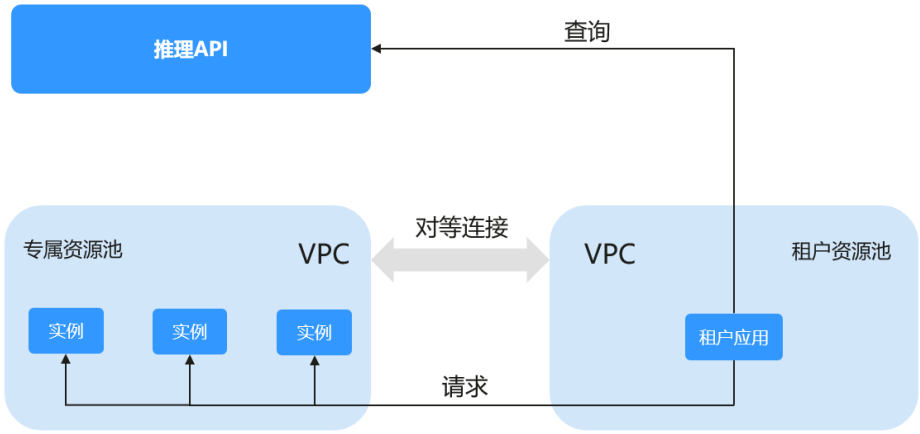
使用VPC直连的高速访问通道，用户的业务请求不需要经过推理平台，而是直接经VPC对等连接发送到实例处理，访问速度更快。

说明

由于请求不经过推理平台，所以会丢失以下功能：

- 认证鉴权
- 流量按配置分发
- 负载均衡
- 告警、监控和统计

图 3-26 VPC 直连的高速访问通道示意图



准备工作

使用专属资源池部署在线服务，服务状态为“运行中”。

须知

- 只有专属资源池部署的服务才支持VPC直连的高速访问通道。
- VPC直连的高速访问通道，目前只支持访问在线服务。
- 因流量限控，获取在线服务的IP和端口号次数有限制，每个主账号租户调用次数不超过2000次/分钟，每个子账号租户不超过20次/分钟。
- 目前仅支持自定义镜像导入模型，部署的服务支持高速访问通道。

操作步骤

使用VPC直连的高速访问通道访问在线服务，基本操作步骤如下：

1. **将专属资源池的网络打通VPC**
2. **VPC下创建弹性云服务器**
3. **获取在线服务的IP和端口号**
4. **通过IP和端口号直连应用**

步骤1 将专属资源池的网络打通VPC

登录ModelArts控制台，进入“AI专属资源池 > 弹性集群 Cluster”找到服务部署使用的专属资源池，单击“名称/ID”，进入资源池详情页面，查看网络配置信息。返回专

属资源池列表，选择“网络”页签，找到专属资源池关联的网络，打通VPC。打通VPC网络后，网络列表和资源池详情页面将显示VPC名称，单击后可以跳转至VPC详情页面。

图 3-27 查看网络配置

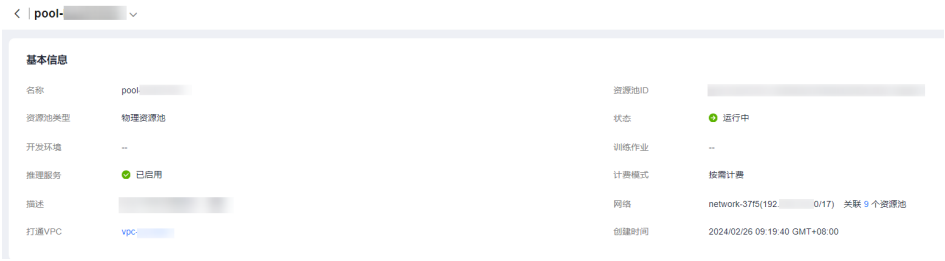


图 3-28 打通 VPC

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 3-29 购买弹性云服务器时选择 VPC

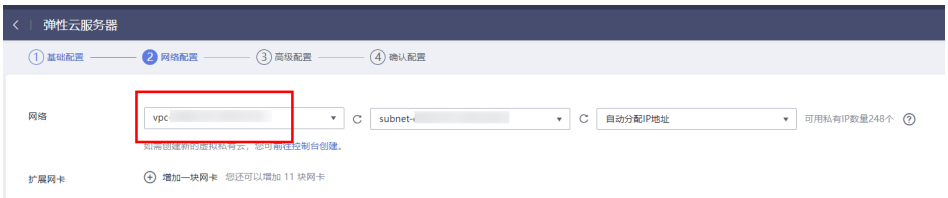
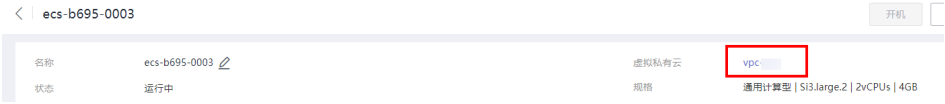


图 3-30 查看虚拟私有云配置信息



步骤3 获取在线服务的IP和端口号

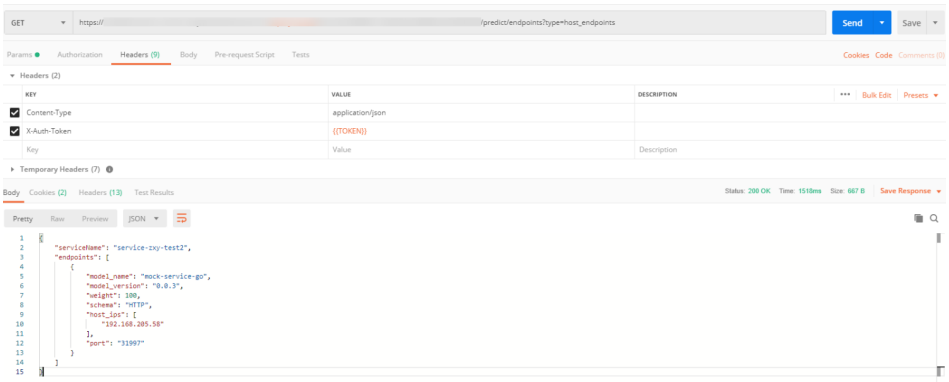
可以通过使用图形界面的软件（以Postman为例）获取服务的IP和端口号，也可以登录弹性云服务器（ECS），创建Python环境运行代码，获取服务IP和端口号。

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 3-31 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 3-32 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

3.1.4.4 WebSocket 访问在线服务

背景说明

WebSocket是一种网络传输协议，可在单个TCP连接上进行全双工通信，位于OSI模型的应用层。WebSocket协议在2011年由IETF标准化为RFC 6455，后由RFC 7936补充规范。Web IDL中的WebSocket API由W3C标准化。

WebSocket使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在WebSocket API中，浏览器和服务器只需要完成一次握手，两者之间就可以建立持久性的连接，并进行双向数据传输。

前提条件

- 在线服务部署时需选择“升级为WebSocket”。
- 在线服务中的AI应用导入选择的镜像需支持WebSocket协议。

约束与限制

- WebSocket协议只支持部署在线服务。
- 只支持自定义镜像导入AI应用部署的在线服务。

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**

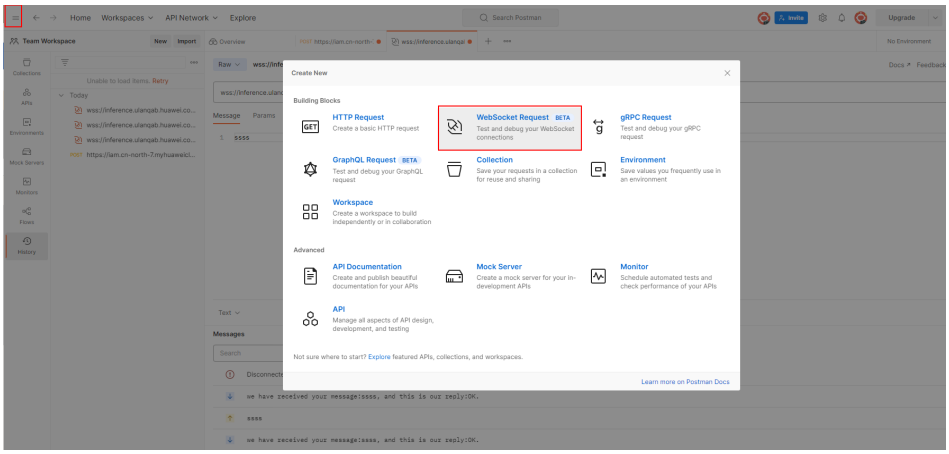
WebSocket服务调用步骤如下（以图形界面的软件Postman进行预测，token认证为例）：

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 3-33 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。

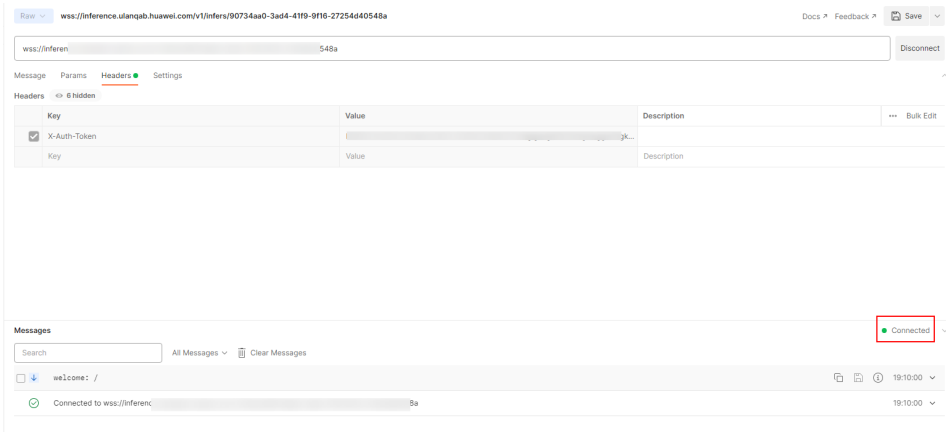
图 3-34 获取 API 接口调用公网地址

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 3-35 连接成功



须知

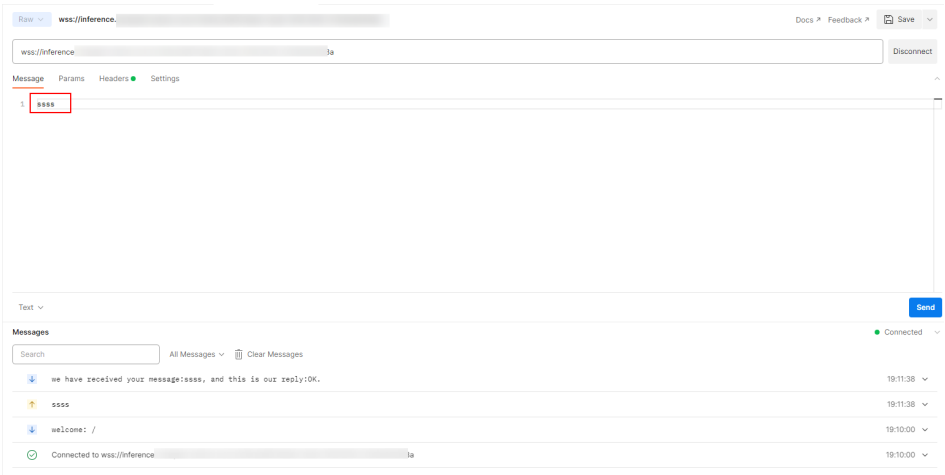
优先验证自定义镜像提供的websocket服务的情况，不同的工具实现的websocket服务会有不同，可能出现连接建立后维持不住，可能出现请求一次后连接就中断需要重新连接的情况，ModelArts平台只保证，未上ModelArts前自定义镜像的websocket的形态跟上了ModelArts平台后的websocket形态相同（除了地址跟认证方式不同）。

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 3-36 发送数据



----结束

3.1.4.5 Server-Sent Events 访问在线服务

背景说明

Server-Sent Events（SSE）是一种服务器向客户端推送数据的技术，它是一种基于HTTP的推送技术，服务器可以向客户端推送事件。这种技术通常用于实现服务器向客户端推送实时数据，例如聊天应用、实时新闻更新等。

SSE主要解决了客户端与服务器之间的单向实时通信需求（例如ChatGPT回答的流式输出），相较于WebSocket（双向实时），它更加轻量级且易于实现。

前提条件

在线服务中的AI应用导入选择的镜像需支持SSE协议。

约束与限制

- SSE协议只支持部署在线服务。
- 只支持自定义镜像导入AI应用部署的在线服务。

SSE 在线服务调用

SSE协议本身不提供额外的认证方式，和HTTP请求方式一致。

可以使用ModelArts提供的以下认证方式：

- **token认证**
- **AK/SK**
- **APP认证**

SSE服务调用如下（以图形界面的软件Postman进行预测，token认证为例）：

图 3-37 SSE 服务调用

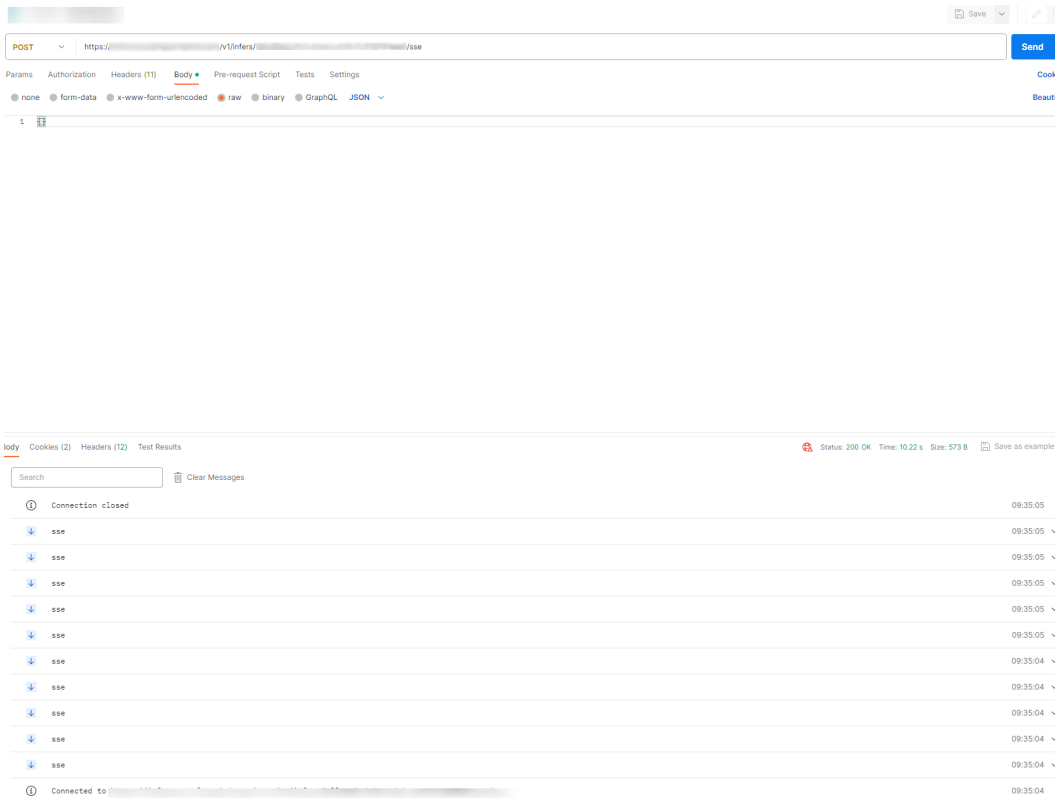


图 3-38 响应头 Content-Type

说明

正常情况下，可以观察到响应头Content-Type为text/event-stream;charset=UTF-8。

3.1.5 集成在线服务

针对已完成调测的API，可以将在线服务API集成至生产环境中应用。

前提条件

确保在线服务一直处于“运行中”状态，否则会导致生产环境应用不可用。

集成方式

ModelArts在线服务提供的API是一个标准的Restful API，可使用HTTPS协议访问。ModelArts提供了SDK用于调用在线服务API，SDK调用方式请参见《SDK参考》>“场景1：部署在线服务Predictor的推理预测”。

除此之外，您还可以使用常见的开发工具及开发语言调用此接口，建议通过互联网搜索并获取调用标准Restful API的指导。

3.1.6 CloudShell

使用场景

允许用户使用ModelArts控制台提供的CloudShell登录运行中在线服务实例容器。

约束限制

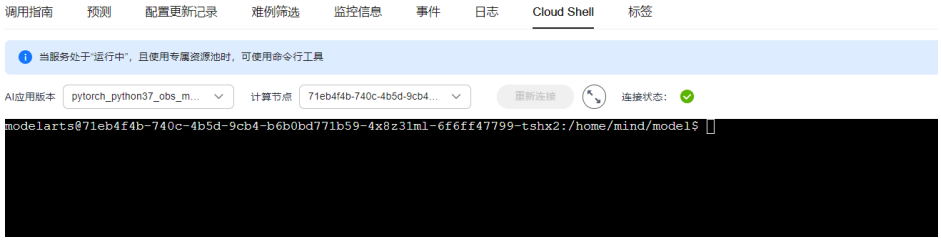
- 只支持专属资源池部署的在线服务使用CloudShell访问容器。
- 在线服务必须处于“运行中”状态，才支持CloudShell访问容器。

如何使用 CloudShell

1. 登录ModelArts控制台，左侧菜单选择“模型部署 > 在线服务”。
2. 在线服务列表页面单击“名称/ID”，进入在线服务详情页面。
3. 单击CloudShell页签，选择AI应用版本和计算节点，当连接状态变为 时，即登录实例容器成功。

   若遇到异常情况服务器主动断开或超过10分钟未操作自动断开，此时可单击“重新连接”重新登录实例容器。

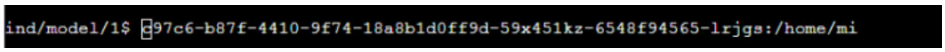
图 3-39 CloudShell 界面



说明

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

图 3-40 路径异常



3.2 部署 AI 应用（批量服务）

3.2.1 部署为批量服务

AI应用准备完成后，您可以将AI应用部署为批量服务。在“模型部署>批量服务”界面，列举了用户所创建的批量服务。

前提条件

- 数据已完成准备：已在ModelArts中创建状态“正常”可用的AI应用。
- 准备好需要批量处理的数据，并上传至OBS目录。
- 已在OBS创建至少1个空的文件夹，用于存储输出的内容。

背景信息

- 用户最多可创建1000个批量服务。
- 根据AI应用定义的输入请求不同（JSON文本或文件），不同的AI应用输入，需要填写的参数不同。当AI应用输入为JSON文件时，则需要根据配置文件生成映射文件；如果AI应用输入为文件时，则不需要。
- 批量服务只支持使用公共资源池，暂不支持使用专属资源池。

操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“模型部署 > 批量服务”，默认进入“批量服务”列表。
2. 在批量服务列表中，单击左上角“部署”，进入“部署”页面。
3. 在部署页面，填写批量服务相关参数。
   a. 填写基本信息。基本信息包含“名称”、“描述”。其中“名称”默认生成。例如：service-bc0d，您也可以根据实际情况填写“名称”和“描述”信息等。
   b. 填写服务参数。包含资源池、AI应用配置等关键信息，详情请参见表3-11。

表 3-11 参数说明

参数名称	说明
“AI应用来源”	根据您的实际情况选择“自定义应用”或者“订阅应用”。

参数名称	说明
“选择AI应用及版本”	选择状态“正常”的AI应用及版本。
“输入数据目录位置”	选择输入数据的OBS路径，即您上传数据的OBS目录。只能选择文件夹或“.manifest”文件。“.manifest”文件规范请参见**Manifest文件规范**。 **说明** • 输入数据为图片时，建议单张图片小于12MB。 • 输入数据格式为csv时，建议不要包含中文。 • 输入数据格式为csv时，建议文件大小不超过12MB。 • 若单张图片/csv文件超过文件12MB，会提示报错，建议调整文件大小使其符合要求，或联系技术支持人员调整文件大小限制。
“请求路径”	批量服务中调用AI应用的接口URL，表示服务的请求路径，此值来自AI应用配置文件中apis的url字段。
“映射关系”	如果AI应用输入是json格式时，系统将根据此AI应用对应的配置文件自动生成映射关系。如果AI应用的输入是文件，则不需要映射关系。 自动生成的映射关系文件，填写每个参数对应到csv单行数据的字段索引，索引index从0开始计数。 映射关系生成规则：映射规则来源于模型配置文件“config.json”中输入参数（request）。当“type”定义为“string/number/integer/boolean”基本类型时，需要配置映射规则参数，即index参数。请参见**映射关系示例**了解其规则。 index必须是从0开始的正整数，当index设置不规则不符时，最终的请求将忽略此参数。配置映射规则后，其对应的csv数据必须以英文半角逗号分隔。
“输出数据目录位置”	选择批量预测结果的保存位置，可以选择您创建的空文件夹。
“计算节点规格”	系统将根据您的AI应用匹配提供可用的计算资源。请在下拉框中选择可用资源，如果资源标识为售罄，表示暂无此资源。 例如，模型来源于自动学习项目，则计算资源将自动关联自动学习规格供使用。
“计算节点个数”	设置当前版本AI应用的实例个数。如果节点个数设置为1，表示后台的计算模式是单机模式；如果节点个数设置大于1，表示后台的计算模式为分布式的。请根据实际编码情况选择计算模式。
“环境变量”	设置环境变量，注入环境变量到容器实例。为确保您的数据安全，在环境变量中，请勿输入敏感信息，如明文密码。
“部署超时时间”	用于设置单个模型实例的超时时间，包括部署和启动时间。默认值为20分钟，输入值必须在3到120之间。

参数名称	说明
“运行日志输出”	默认关闭，批量服务的运行日志仅存放在ModelArts日志系统，在服务详情页的“日志”支持简单查询。 若开启此功能，批量服务的运行日志会输出存放到云日志服务LTS。LTS自动创建日志组和日志流，默认缓存7天内的运行日志。如需了解LTS专业日志管理功能，请参见**云日志服务**。 **说明** • “运行日志输出”开启后，不支持关闭。 • LTS服务提供的日志查询和日志存储功能涉及计费，详细请参见**了解LTS的计费规则**。 • 请勿打印无用的audio日志文件，这会导致系统日志卡死，无法正常显示日志，可能会出现“Failed to load audio”的报错。

4. 完成参数填写后，根据界面提示完成批量服务的部署。部署服务一般需要运行一段时间，根据您选择的数据量和资源不同，部署时间将耗时几分钟到几十分钟不等。

   说明

   批量服务部署完成后，将立即启动，运行过程中将按照您选择的资源按需计费。

   您可以前往批量服务列表，查看批量服务的基本情况。在批量服务列表中，刚部署的服务“状态”为“部署中”，当批量服务的“状态”变为“运行完成”时，表示服务部署完成。

Manifest 文件规范

推理平台批量服务支持使用manifest文件，manifest文件可用于描述数据的输入输出。

输入manifest文件样例

- 文件名：“test.manifest”
- 文件内容：

```
{"source": "obs://test/data/1.jpg"}
{"source": "s3://test/data/2.jpg"}
{"source": "https://infers-data.obs.cn-north-1.myhuaweicloud.com:443/xgboosterdata/data.csv?AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz1uDzwve8GpY%3D&x-obs-security-token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-ZaMSxHvl68kKLAy5feYvLDM..."}
```

- 文件要求：
  a. 文件名后缀需为“.manifest”；
  b. 文件内容是多行JSON，每行JSON描述一个输入数据，需精确到文件，不能是文件夹；
  c. JSON内容需定义一个source字段，字段值是OBS的文件地址，有2种表达形式：
     i. 桶路径“<obs path>{{桶名}}/{{对象名}}/文件名”，适用于访问自己名下的OBS数据；您可以访问OBS服务的对象获取路径。<obs path>可以为“obs://”或“s3://”。

ii. OBS生成的分享链接，包含签名信息。适用于访问其他人的OBS数据。分享链接有有效时间限制，请在有效时间内操作。

输出manifest文件样例

批量服务的输出结果目录会有一个manifest文件。

- 假设用户输出结果路径为/test-bucket/test/，则结果存放位置如下：

```
OBS桶/目录名
├── test-bucket
│   ├── test
│   │   ├── infer-result-{{task_id}}.manifest
│   │   ├── infer-result
│   │   │   ├── 1.jpg_result.txt
│   │   │   ├── 2.jpg_result.txt
```

- infer-result-0.manifest文件内容：

```
{"source": "obs://obs-data-bucket/test/data/1.jpg","result":"SUCCESSFUL","inference-loc": "obs://test-bucket/test/infer-result/1.jpg_result.txt"}
{"source": "s3://obs-data-bucket/test/data/2.jpg","result":"FAILED","error_message": "Download file failed."}
{"source ": "https://infers-data.obs.xxx.com:443/xgboosterdata/2.jpg?AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz1uDzwve8GpY%3D&x-obs-security-token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-ZaMSxHvl68kKLAy5feYvLDMNZWxzhBZ6Q-3HcoZMh9gISwQOVBwm4ZytB_m8sg1fL6isU7T3CnoL9jmvDGgT9VBC7dC1EyfSJrUcqfB_N0ykCsfrA1Tt_IQYZFDu_HyqVk-GunUcTVdDfWlCV3TrYcpmznZjliAnYUO89kAwCYGeRZsCsC0ePu4PHMsBvYV9gWmN9AUZIDn1sfRL4voBpwQnp6tnAgHW49y5a6hP2hCAoQ-95SpUriJ434QlymoeKfTHVMKOeZxZea-JxOvevOCGI5CcGehEJaz48sgH81UiHzl21zocNB_hpPfus2jY6KPglEJxMv6Kwmro-ZBXWuSJUDOnSYXI-3ciYjg9-h10b8W3sW1mOTFCWNGoWsd74it7l_5-7UUhoIeyPByO_REwkur2FOJsuMpGlRaPyglZxXm_jfdLFXobYtzZhbul4yWXga6oxTOkfcwykTOYH0NPoPRt5MYGYweOXXxFs3d5w2rd0y7p0QYhyTzIkk5CIz7FlWNapFISL7zdhsl8RfchTqESq94KgkeqatSF_iIvnYMW2r8P8x2k_eb6NJ7U_q5ztMbO9oWEcfr0D2f7n7Bl_nb2HIB_H9tjzKvqwngaimYhBbMRPfibvttW86GiwVP8vrC27FOn39Be9z2hSfJ_8pHej0yMlyNqZ481FQ5vWT_vFV3JHM-7I1ZB0_hIdaHfItm-J69cTfHSEOzt7DGaMIES1o7U3w%3D%3D","result":"SUCCESSFUL","inference-loc": "obs://test-bucket/test/infer-result/2.jpg_result.txt"}
```

- 文件格式：
  a. 文件名为“infer-result-{{task_id}}.manifest”，task_id为批量任务id，批量服务对应唯一的批量任务id。
  b. 当处理文件数目较多时，可能会有多个manifest文件，后缀相同，均为“.manifest”，文件名以后缀区分，例如“infer-result-{{task_id}}_1.manifest”等。
  c. manifest同一目录下会创建infer-result-{{task_id}}目录存放文件处理结果。
  d. 文件内容是多行JSON，每行JSON描述一个输入数据的对应输出结果。
  e. JSON内容包含多个字段。
     i. source：输入数据描述，与输入的manifest一致。
     ii. result：文件处理结果，值为SUCCESSFUL或FAILED，分别代表成功与失败。
     iii. inference-loc：输出结果路径，result为SUCCESSFUL时有此字段，格式为“obs://{{桶名}}/{{对象名}}”。
     iv. error_message：错误信息，result为FAILED时有此字段。

映射关系示例

如下示例展示了配置文件、映射规则、csv数据以及最终推理请求的关系。

假设，您的模型所用配置文件，其apis参数如下所示：


```
[
  {
    "method": "post",
    "url": "/",
    "request": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "data": {
            "type": "object",
            "properties": {
              "req_data": {
                "type": "array",
                "items": [
                  {
                    "type": "object",
                    "properties": {
                      "input_1": {
                        "type": "number"
                      },
                      "input_2": {
                        "type": "number"
                      },
                      "input_3": {
                        "type": "number"
                      },
                      "input_4": {
                        "type": "number"
                      }
                    }
                  }
                ]
              }
            }
          }
        }
      }
    }
  }
]
```

此时，其对应的映射关系如下所示。ModelArts管理控制台将从配置文件中自动解析映射关系，如果您调用ModelArts API时，需要自行根据规则编写映射关系。

```
{
  "type": "object",
  "properties": {
    "data": {
      "type": "object",
      "properties": {
        "req_data": {
          "type": "array",
          "items": [
            {
              "type": "object",
              "properties": {
                "input_1": {
                  "type": "number",
                  "index": 0
                },
                "input_2": {
                  "type": "number",
                  "index": 1
                },
                "input_3": {
                  "type": "number",
                  "index": 2
                },
                "input_4": {
```



```
                        "type": "number",
                        "index": 3
                    }
                }
            }
        ]
    }
  }
 }
}
```

用户需要进行推理的数据，即CSV数据，格式如下所示。数据必须以英文逗号隔开。

```
5.1,3.5,1.4,0.2
4.9,3.0,1.4,0.2
4.7,3.2,1.3,0.2
```

根据定义好的映射关系，最终推理请求样例如下所示，与在线服务使用的格式类似：

```
{
  "data": {
    "req_data": [{
      "input_1": 5.1,
      "input_2": 3.5,
      "input_3": 1.4,
      "input_4": 0.2
    }]
  }
}
```

3.2.2 查看批量服务详情

当AI应用部署为批量服务成功后，您可以进入“批量服务”页面，来查看服务详情。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>批量服务”，进入“批量服务”管理页面。
2. 单击目标服务名称，进入服务详情页面。
   您可以查看服务的“名称”、“状态”等信息，详情说明请参见**表3-12**。

表 3-12 批量服务参数

参数	说明
名称	批量服务名称。
服务ID	批量服务的ID。
状态	批量服务当前状态。
任务ID	批量服务的任务ID。
计算节点规格	批量服务的节点规格。
计算节点个数	批量服务的节点个数。
任务开始时间	本次批量服务的任务开始时间。
环境变量	批量服务创建时填写的环境变量。
任务结束时间	本次批量服务的任务结束时间。
描述	您可以单击编辑按钮，添加服务描述。

参数	说明
输入数据目录位置	本次批量服务中，输入数据的OBS路径。
输出数据目录位置	本次批量服务中，输出数据的OBS路径。
AI应用名称&版本	本次批量服务所使用的AI应用名称及版本。
运行日志输出	默认关闭，批量服务的运行日志仅存放在ModelArts日志系统。 启用运行日志输出后，批量服务的运行日志会输出存放到云日志服务LTS。LTS自动创建日志组和日志流，默认缓存7天内的运行日志。如需了解LTS专业日志管理功能，请参见**云日志服务**。 **说明** • “运行日志输出”开启后，不支持关闭。 • LTS服务提供的日志查询和日志存储功能涉及计费，详细请参见**了解LTS的计费规则**。 • 请勿打印无用的audio日志文件，这会导致系统日志卡死，无法正常显示日志，可能会出现“Failed to load audio”的报错。

3. 您可以进入批量服务的详情页面，通过切换页签查看更多详细信息，详情说明请参见**表3-13**。

表 3-13 批量服务页签

参数	说明
事件	展示当前服务使用过程中的关键操作，比如服务部署进度、部署异常的详细原因、服务被启动、停止、更新的时间点等。 事件保存周期为1个月，1个月后自动清理数据。 查看服务的事件类型和事件信息，请参见**查看服务的事件**

参数	说明
日志	展示当前服务下每个AI应用的日志信息。包含最近5分钟、最近30分钟、最近1小时和自定义时间段。 自定义时间段您可以选择开始时间和结束时间。 当服务启用运行日志输出后，页面展示存放到云日志服务LTS中的日志信息。您可以单击“到LTS查看完整日志”查看全量的日志。 日志搜索规则说明： ● 不支持带有分词符的字符串搜索（当前默认分词符有 ,'";=()[]{}@&<>/:\n\t\r）。 ● 支持关键词精确搜索。关键词指相邻两个分词符之间的单词。 ● 支持关键词模糊匹配搜索，例如输入“error”或“er?or”或“rro*”或“er*r”。 ● 支持短语精确搜索。例如输入“Start to refresh”。 ● 启用运行日志输出前，支持关键词的“与”、“或”组合搜索。格式为“query logs&&erro*”或“query logs||erro*”。启用运行日志输出后，支持关键词的“与”、“或”组合搜索。格式为“query logs AND erro*”或“query logs OR erro*”。

3.2.3 查看批量服务预测结果

当您在部署批量服务时，会选择输出数据目录位置，您可以查看“运行完成”状态的批量服务运行结果。

操作步骤

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署>批量服务”，进入“批量服务”管理页面。
2. 单击状态为“运行完成”的目标服务名称，进入服务详情页面。
   - 您可以查看服务的“名称”、“状态”、“服务ID”、“输入数据目录位置”、“输出数据目录位置”和“描述”。
   - 您也可以通过单击描述右侧的  ，对描述信息进行编辑。
3. 从“输出数据目录位置”参数右侧获取详细OBS地址，前往此OBS目录，可以获取批量服务预测结果，包括预测结果文件和AI应用预测结果。

   若预测成功，目录下有预测结果文件和AI应用预测结果；若预测失败，目录下只有预测结果文件。
   - 预测结果文件：文件格式为“xxx.manifest”，里面包含文件路径、预测结果等信息。
   - AI应用预测结果输出：
     - 当输入为图片时，每张图片输出一个结果，输出结果格式为“图片名_result.txt”。例如：IMG_20180919_115016.jpg_result.txt。

- 当输入为音频时，每个音频输出一个结果，输出结果格式为“音频名_result.txt”。例如：1-36929-A-47.wav_result.txt。
- 当输入为表格数据时，输出结果格式为“表格名_result.txt”。例如：train.csv_result.txt。

3.3 修改服务

对于已部署的服务，您可以修改服务的基本信息以匹配业务变化，更换AI应用的版本号，实现服务升级。

您可以通过如下两种方式修改服务的基本信息：

方式一：通过服务管理页面修改服务信息

方式二：通过服务详情页面修改服务信息

前提条件

服务已部署成功，“部署中”的服务不支持修改服务信息进行升级。

约束限制

- 服务升级关系着业务实现，不当的升级操作会导致升级期间业务中断的情况，请谨慎操作。
- ModelArts支持部分场景下在线服务进行无损滚动升级。按要求进行升级前准备，做好验证，即可实现业务不中断的无损升级。

表 3-14 支持无损滚动升级的场景

创建AI应用的元模型来源	服务使用的是公共资源池	服务使用的是专属资源池
从训练中选择元模型	不支持	不支持
从容器镜像中选择元模型	不支持	支持，创建AI应用的自定义镜像需要满足创建AI应用的自定义镜像规范。
从OBS中选择元模型	不支持	不支持

方式一：通过服务管理页面修改服务信息

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。
2. 在服务列表中，单击目标服务操作列的“修改”，修改服务基本信息，然后根据提示提交修改任务。

   当修改了服务的某些参数配置时，系统会自动重启服务使修改生效。在提交修改服务任务时，若涉及重启，会有弹窗提醒。

   - 在线服务参数说明请参见**部署为在线服务**。修改在线服务还需要配置“最大无效实例数”设置并行升级的最大节点数，升级阶段节点无效。

- 批量服务参数说明请参见**部署为批量服务**。

方式二：通过服务详情页面修改服务信息

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。
2. 单击目标服务名称，进入服务详情页面。
3. 您可以通过单击页面右上角“修改”，修改服务基本信息，然后根据提示提交修改任务。

   当修改了服务的某些参数配置时，系统会自动重启服务使修改生效。在提交修改服务任务时，若涉及重启，会有弹窗提醒。

   - 在线服务参数说明请参见**部署为在线服务**。修改在线服务还需要配置“最大无效实例数”设置并行升级的最大节点数，升级阶段节点无效。
   - 批量服务参数说明请参见**部署为批量服务**。

3.4 启动、停止、删除、重启服务

启动服务

您可以对处于“运行完成”、“异常”和“停止”状态的服务进行启动操作，“部署中”状态的服务无法启动。启动服务，当服务处于“运行中”状态后，ModelArts将开始计费。您可以通过如下方式启动服务：

- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。您可以单击“操作”列的“启动”，启动服务。
- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。单击目标服务名称，进入服务详情页面。您可以单击页面右上角“启动”，启动服务。

说明

部署方式为ModelArts边缘节点和ModelArts边缘资源池的服务不支持启动。

停止服务

停止服务，ModelArts将停止计费。您可以通过如下方式停止服务：

- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。您可以单击“操作”列的“停止”（在线服务在操作列选择“更多 > 停止”），停止服务。
- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。单击目标服务名称，进入服务详情页面。您可以单击页面右上角“停止”，停止正在运行中服务。

说明

部署方式为ModelArts边缘节点和ModelArts边缘资源池的服务不支持停止。

删除服务

如果服务不再使用，您可以删除服务释放资源。

登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入目标服务类型管理页面。

- 在线服务
  - 单击在线服务列表“操作”列的“更多>删除”删除服务。
  - 勾选在线服务列表中的服务，然后单击列表左上角“删除”按钮，批量删除服务。
  - 单击目标服务名称，进入服务详情页面，单击右上角“删除”删除服务。
- 批量服务
  - 单击批量服务列表“操作”列的“删除”，删除服务。
  - 勾选批量服务列表中的服务，然后单击列表左上角“删除”按钮，批量删除服务。
  - 单击目标服务名称，进入服务详情页面，单击右上角“删除”按钮进行删除。

说明

- 删除操作无法恢复，请谨慎操作。
- 没有委托授权时，无法删除服务。
- 如果在线服务开启了“运行日志输出”，删除服务时，推荐同时删除LTS中的日志以及日志流，避免LTS日志流超过限额产生额外费用，如后续不再使用，建议删除。

重启服务

只有当在线服务处于“运行中”或“告警”状态时，才可进行重启操作。批量服务、边缘服务不支持重启。您可以通过如下方式重启在线服务：

- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入在线服务列表页面。您可以单击“操作”列的“更多>重启”，重启服务。
- 登录ModelArts管理控制台，在左侧菜单栏中选择“模型部署”，进入在线服务列表页面。单击目标服务名称，进入服务详情页面。您可以单击页面右上角“重启”，重启在线服务。

说明

部署方式为ModelArts边缘节点和ModelArts边缘资源池的服务不支持重启。

3.5 查看服务的事件

服务的（从用户可看见部署服务任务开始）整个生命周期中，每一个关键事件点在系统后台均有记录，用户可随时在对应服务的详情页面进行查看。

方便用户更清楚的了解服务部署和运行过程，遇到任务异常时，更加准确的排查定位问题。可查看的事件点包括：

表 3-15 事件

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	开始部署服务。 Start to deploy service.	-
异常	资源不足，等待资源释放。 Lack of resources, transform state to waiting.	等待资源释放后重试。
异常	xxx资源不足，服务调度失败。补充信息：xxx %s %s Schedule failed due to insufficient resources. Retry later. %s nodes are available: %s Insufficient memory.	根据补充信息，了解资源不足详情，参考FAQ处理。
正常	开始构建镜像。 Start to build image.	-
异常	构建模型(xxx) 镜像失败，构建日志:\nxxx。 Failed to build image for model (%s %s), docker build log:\n%s.	根据构建日志定位和处理问题。
异常	构建镜像失败。 Failed to build image.	请联系技术支持。
正常	构建镜像完成。 Image built successfully.	-
异常	xxx服务失败。错误信息：xxx Failed to %s service, retry later. Error message: %s	请根据错误信息定位和处理问题。
异常	更新服务失败，执行回滚操作。 Failed to update service, rollback it.	请联系技术支持。
正常	服务更新中。 Updating service.	-
正常	服务启动中。 Starting service.	-
正常	服务停止中。 Stopping service.	-
正常	服务已停止。 Service stopped.	-
正常	自动停止开关已关闭。 Auto-stop switched off.	-

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	自动关闭功能开启，服务将在xs后停止。 Auto-stop switched on, service will be stopped in %d %s.	-
正常	到达自动停止时间，服务停止。 Service stopped automatically because due time is reached.	-
异常	配额超限，服务停止。 Service stopped automatically because over quota.	请联系技术支持。
异常	自动停止服务失败，错误信息: xxx Failed to stop service automatically, error message: %s	请根据错误信息定位和处理问题。
正常	删除资源池(xxx)上服务实例。 Model in node(%s) deleted.	-
正常	停止资源池(xxx)上服务实例。 Model in node(%s) stopped.	-
异常	批量服务失败，请稍后重试。错误信息：xxx Failed to %s batch service, retry later. Error message: %s.	请根据错误信息定位和处理问题。
正常	服务运行完成。 Service stopped automatically after running.	-
异常	停止服务失败，错误信息：xxx Failed to stopped service, error message: %s	请根据错误信息定位和处理问题。
正常	订阅许可即将超期：xxx Impending expiration notice: %s	-
正常	服务xxx启动成功。 Service %s started successfully.	-
异常	启动服务xxx失败。 Service %s started failed.	启动服务失败情况较多，请参考FAQ定位和处理。
异常	部署服务超时，错误信息：xxx Deploying timeout, details: %s	请根据错误信息定位和处理问题。

事件类型	事件信息（“XXX”表示占位符，以实际返回信息为准）	解决方案
正常	更新服务失败，执行回滚操作成功。 Failed to update service, rollback succeeded.	-
异常	更新服务失败，执行回滚操作失败。 Failed to update service, rollback failed.	请联系技术支持。

服务部署和运行过程中，关键事件支持手动/自动刷新。

查看操作

1. 在ModelArts管理控制台的左侧导航栏中选择“模型部署 > 在线服务|批量服务|边缘服务”，在服务列表中，您可以单击名称/ID，进入服务详情页面。
2. 在服务详情页面，切换到“事件”页签，查看事件信息。

4 推理规范说明

4.1 模型包规范

4.1.1 模型包规范介绍

创建AI应用时，如果是从OBS中导入元模型，则需要符合一定的模型包规范。

说明

- 模型包规范适用于单模型场景，若是多模型场景（例如含有多个模型文件）推荐使用自定义镜像方式。
- ModelArts推理平台不支持的AI引擎，推荐使用自定义镜像方式。
- 请参考创建AI应用的自定义镜像规范和从0-1制作自定义镜像并创建AI应用，制作自定义镜像。
- 更多的自定义脚本代码示例，请参考**自定义脚本代码示例**。

模型包里面必须包含“model”文件夹，“model”文件夹下面放置模型文件，模型配置文件，模型推理代码文件。

- **模型文件**：在不同模型包结构中模型文件的要求不同，具体请参见**模型包结构示例**。
- **模型配置文件**：模型配置文件必须存在，文件名固定为“config.json”，有且只有一个，模型配置文件编写请参见**模型配置文件编写说明**。
- **模型推理代码文件**：模型推理代码文件是必选的。文件名固定为“customize_service.py”，此文件有且只能有一个，模型推理代码编写请参见**模型推理代码编写说明**。
  - customize_service.py依赖的py文件可以直接放model目录下，推荐采用相对导入方式导入自定义包。
  - customize_service.py依赖的其他文件可以直接放model目录下，需要采用绝对路径方式访问。绝对路径获取请参考**绝对路径如何获取**。

ModelArts针对多种引擎提供了样例及其示例代码，您可以参考样例编写您的配置文件和推理代码，详情请参见**ModelArts样例列表**。ModelArts也提供了常用AI引擎对应的自定义脚本示例，请参见**自定义脚本代码示例**。

若您在导入元模型过程中遇到问题，可**联系华为云技术支持**协助解决故障。

模型包结构示例

- TensorFlow模型包结构

  发布该模型时只需要指定到“ocr”目录。

```
OBS桶/目录名
├── ocr
|   ├── model 必选： 固定子目录名称，用于放置模型相关文件
|   |   ├── <<自定义python包>> 可选：用户自有的Python包，在模型推理代码中可以直接引用
|   |   ├── saved_model.pb 必选: protocol buffer格式文件，包含该模型的图描述
|   |   ├── variables 对*.pb模型主文件而言必选；固定子目录名称，包含模型的权重偏差等信息
|   |   |   ├── variables.index 必选
|   |   |   ├── variables.data-00000-of-00001 必选
|   |   ├──config.json 必选：模型配置文件，文件名称固定为config.json, 只允许放置一个
|   |   ├──customize_service.py 必选：模型推理代码，文件名称固定为customize_service.py, 只允许放置
一个
customize_service.py依赖的文件可以直接放model目录下
```

- PyTorch模型包结构

  发布该模型时只需要指定到“resnet”目录。

```
OBS桶/目录名
├── resnet
|   ├── model 必选： 固定子目录名称，用于放置模型相关文件
|   |   ├── <<自定义Python包>> 可选：用户自有的Python包，在模型推理代码中可以直接引用
|   |   ├── resnet50.pth 必选，pytorch模型保存文件，保存为“state_dict”，存有权重变量等信息。
|   |   ├──config.json 必选：模型配置文件，文件名称固定为config.json, 只允许放置一个
|   |   ├──customize_service.py 必选：模型推理代码，文件名称固定为customize_service.py, 只允许放置
一个，customize_service.py依赖的文件可以直接放model目录下
```

- Custom模型包结构，与您自定义镜像中AI引擎有关。例如自定义镜像中的AI引擎为TensorFlow，则模型包采用TensorFlow模型包结构。

4.1.2 模型配置文件编写说明

模型开发者发布模型时需要编写配置文件config.json。模型配置文件描述模型用途、模型计算框架、模型精度、推理代码依赖包以及模型对外API接口。

配置文件格式说明

配置文件为JSON格式，参数说明如表4-1所示。

表 4-1 参数说明

参数	是否必选	参数类型	描述
model_algorithm	是	String	模型算法，表示该模型的用途，由模型开发者填写，以便使用者理解该模型的用途。只能以英文字母开头，不能包含中文以及&!'\"<>=，不超过36个字符。常见的模型算法有image_classification（图像分类）、object_detection（物体检测）、predict_analysis（预测分析）等。

参数	是否必选	参数类型	描述
model_type	是	String	模型AI引擎，表明模型使用的计算框架，支持常用AI框架和“Image”。 • 可选的常用AI框架请参见**推理支持的AI引擎**。 • 当model_type设置为Image，表示以自定义镜像方式创建AI应用，此时swr_location为必填参数。Image镜像制作规范可参见模型镜像规范。
runtime	否	String	模型运行时环境，系统默认使用python2.7。runtime可选值与model_type相关，当model_type设置为Image时，不需要设置runtime，当model_type设置为其他常用框架时，请选择您使用的引擎所对应的运行时环境。目前支持的运行时环境列表请参见**推理支持的AI引擎**。 需要注意的是，如果您的模型需指定CPU或GPU上运行时，请根据runtime的后缀信息选择，当runtime中未包含cpu或gpu信息时，请仔细阅读“推理支持的AI引擎”中每个runtime的说明信息。
metrics	否	object数据结构	模型的精度信息，包括平均数、召回率、精确率、准确率，metrics object数据结构说明如**表4-2**所示。 结果会显示在AI应用详情页面的“模型精度”模块。
apis	否	api数据结构数组	表示模型接收和返回的请求样式，为结构体数据。 即模型可对外提供的Restful API数组，API数据结构如**表4-3**所示。示例代码请参见**apis参数代码示例**。 • “model_type”为“Image”时，即自定义镜像的模型场景，“apis”可根据镜像实际对外暴露的请求路径在“apis”中声明不同路径的API。 • “model_type”不为“Image”时，“apis”只能声明一个请求路径为“/”的API，因为系统预置的AI引擎仅暴露一个请求路径为“/”的推理接口。
dependencies	否	dependency结构数组	表示模型推理代码需要依赖的包，为结构体数据。 模型开发者需要提供包名、安装方式、版本约束。目前只支持pip安装方式。dependency结构数组说明如**表4-6**所示。 如果模型包内没有推理代码customize_service.py文件，则该字段可不填。自定义镜像模型不支持安装依赖包。 **说明** “dependencies”参数支持多个“dependency”结构数组，以list格式填入，默认安装包存在先后依赖关系（即写在前面的先安装，写在后面的后安装），且支持线下wheel包安装（wheel包必须与模型文件放在同一目录）。示例请参考**导入模型时安装包依赖配置文件如何书写?**

参数	是否必选	参数类型	描述
health	否	health数据结构	镜像健康接口配置信息，只有“model_type”为“Image”时才需填写。 如果在滚动升级时要求不中断业务，那么必须提供健康检查的接口供ModelArts调用。health数据结构如表4-8所示。

表 4-2 metrics object 数据结构说明

参数	是否必选	参数类型	描述
f1	否	Number	平均数。精确到小数点后17位，超过17位时，取前17位数值。
recall	否	Number	召回率。精确到小数点后17位，超过17位时，取前17位数值。
precision	否	Number	精确率。精确到小数点后17位，超过17位时，取前17位数值。
accuracy	否	Number	准确率。精确到小数点后17位，超过17位时，取前17位数值。

表 4-3 api 数据结构说明

参数	是否必选	参数类型	描述
url	否	String	请求路径。默认值为“/”。自定义镜像的模型（即model_type为Image时）需要根据镜像内实际暴露的请求路径填写“url”。非自定义镜像模型（即model_type不为Image时）时，“url”只能为“/”。
method	否	String	请求方法。默认值为“POST”。
request	否	Object	请求体，request结构说明如表4-4所示。
response	否	Object	响应体，response结构说明如表4-5所示。

表 4-4 request 结构说明

参数	是否必选	参数类型	描述
Content-type	在线服务-非必选 批量服务-必选	String	data以指定内容类型发送。默认值为“application/json”。 一般情况包括如下两种内容类型： ● “application/json”，发送json数据。 ● “multipart/form-data”，上传文件。 **说明** 针对机器学习类模型，仅支持“application/json”
data	在线服务-非必选 批量服务-必选	String	请求体以json schema描述。参数说明请参考**官方指导**。

表 4-5 response 结构说明

参数	是否必选	参数类型	描述
Content-type	在线服务-非必选 批量服务-必选	String	data以指定内容类型发送。默认值为“application/json”。 **说明** 针对机器学习类模型，仅支持“application/json”
data	在线服务-非必选 批量服务-必选	String	响应体以json schema描述。参数说明请参考**官方指导**。

表 4-6 dependency 结构数组说明

参数	是否必选	参数类型	描述
installer	是	String	安装方式，当前只支持“pip”。
packages	是	package结构数组	依赖包集合，package结构数组说明如**表4-7**所示。

表 4-7 package 结构数组说明

参数	是否必选	参数类型	描述
package_name	是	String	依赖包名称。不能含有中文及特殊字符&!'"<>=。
package_version	否	String	依赖包版本，如果不强依赖于版本号，则该项不填。不能含有中文及特殊字符&!'"<>=。
restraint	否	String	版本限制条件，当且仅当“package_version”存在时必填，可选“EXACT/ATLEAST/ATMOST”。 ● “EXACT”表示安装给定版本。 ● “ATLEAST”表示安装版本不小于给定版本。 ● “ATMOST”表示安装包版本不大于给定版本。 **说明** ● 如果对版本有明确要求，优先使用“EXACT”；如果使用“EXACT”与系统安装包有冲突，可以选择“ATLEAST” ● 如果对版本没有明确要求，推荐不填写“restraint”、“package_version”，只保留“package_name”参数

表 4-8 health 数据结构说明

参数	是否必选	参数类型	描述
check_method	是	String	健康检查方式。可选“HTTP/EXEC”。 ● HTTP：HTTP请求检查 ● EXEC：执行命令检查。
command	否	String	健康检查命令。健康检查方式为EXEC时必选。
url	否	String	健康检查接口请求路径。健康检查方式为HTTP时必选。
protocol	否	String	健康检查接口请求协议，默认为http。健康检查方式为HTTP时必选。
initial_delay_seconds	否	String	健康检查初始化延迟时间。

参数	是否必选	参数类型	描述
timeout_seconds	否	String	健康检查超时时间。
period_seconds	是	String	健康检查周期。填写大于0且小于等于2147483647的整数，单位为秒。
failure_threshold	是	String	健康检查最大失败次数。填写大于0且小于等于2147483647的整数。

apis 参数代码示例

```
[{
    "url": "/",
    "method": "post",
    "request": {
        "Content-type": "multipart/form-data",
        "data": {
            "type": "object",
            "properties": {
                "images": {
                    "type": "file"
                }
            }
        }
    },
    "response": {
        "Content-type": "applicaton/json",
        "data": {
            "type": "object",
            "properties": {
                "mnist_result": {
                    "type": "array",
                    "item": [
                        {
                            "type": "string"
                        }
                    ]
                }
            }
        }
    }
}]
```

目标检测模型配置文件示例

如下代码以TensorFlow引擎为例，您可以根据实际使用的引擎类型修改model_type参数后使用。

- 模型输入

  key：images

  value：图片文件

- 模型输出

  ```
  {
      "detection_classes": [
          "face",
          "arm"
  ```



```
    ],
    "detection_boxes": [
        [
            33.6,
            42.6,
            104.5,
            203.4
        ],
        [
            103.1,
            92.8,
            765.6,
            945.7
        ]
    ],
    "detection_scores": [0.99, 0.73]
}
```

- 配置文件

```
{
    "model_type": "TensorFlow",
    "model_algorithm": "object_detection",
    "runtime": "tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [{
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "application/json",
            "data": {
                "type": "object",
                "properties": {
                    "detection_classes": {
                        "type": "array",
                        "items": [{
                            "type": "string"
                        }]
                    },
                    "detection_boxes": {
                        "type": "array",
                        "items": [{
                            "type": "array",
                            "minItems": 4,
                            "maxItems": 4,
                            "items": [{
                                "type": "number"
                            }]
                        }]
                    },
                    "detection_scores": {
                        "type": "array",
                        "items": [{
                            "type": "number"
                        }]
```



```
                }
            }
        }
    }
  }],
  "dependencies": [{
    "installer": "pip",
    "packages": [{
          "restraint": "EXACT",
          "package_version": "1.15.0",
          "package_name": "numpy"
        },
        {
          "restraint": "EXACT",
          "package_version": "5.2.0",
          "package_name": "Pillow"
        }
    ]
  }]
}
```

图像分类模型配置文件示例

如下代码以TensorFlow引擎为例，您可以根据实际使用的引擎类型修改model_type参数后使用。

- 模型输入

  key：images

  value：图片文件

- 模型输出

```
{
  "predicted_label": "flower",
  "scores": [
    ["rose", 0.99],
    ["begonia", 0.01]
  ]
}
```

- 配置文件

```
{
  "model_type": "TensorFlow",
  "model_algorithm": "image_classification",
  "runtime": "tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64",
  "metrics": {
    "f1": 0.345294,
    "accuracy": 0.462963,
    "precision": 0.338977,
    "recall": 0.351852
  },
  "apis": [{
    "url": "/",
    "method": "post",
    "request": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "images": {
            "type": "file"
          }
        }
      }
    },
    "response": {
      "Content-type": "application/json",
      "data": {
```



```
            "type": "object",
            "properties": {
                "predicted_label": {
                    "type": "string"
                },
                "scores": {
                    "type": "array",
                    "items": [{
                        "type": "array",
                        "minItems": 2,
                        "maxItems": 2,
                        "items": [
                            {
                                "type": "string"
                            },
                            {
                                "type": "number"
                            }
                        ]
                    }]
                }
            }
        }
    }],
    "dependencies": [{
        "installer": "pip",
        "packages": [{
                "restraint": "ATLEAST",
                "package_version": "1.15.0",
                "package_name": "numpy"
            },
            {
                "restraint": "",
                "package_version": "",
                "package_name": "Pillow"
            }
        ]
    }]
}
```

如下代码以MindSpore引擎为例，您可以根据实际使用的引擎类型修改model_type参数后使用。

- 模型输入

  key：images

  value：图片文件

- 模型输出

  ```
  "[[-2.404526   -3.0476532  -1.9888215   0.45013925 -1.7018927   0.40332815\n  -7.1861157
  11.290332   -1.5861531   5.7887416 ]]"
  ```

- 配置文件

  ```
  {
      "model_algorithm": "image_classification",
      "model_type": "MindSpore",
      "runtime": "mindspore_2.1.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b",
      "metrics": {
          "f1": 0.124555,
          "recall": 0.171875,
          "precision": 0.0023493892851938493,
          "accuracy": 0.00746268656716417
      },
      "apis": [{
              "url": "/",
              "method": "post",
              "request": {
                  "Content-type": "multipart/form-data",
  ```



```
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "applicaton/json",
            "data": {
                "type": "object",
                "properties": {
                    "mnist_result": {
                        "type": "array",
                        "item": [{
                            "type": "string"
                        }]
                    }
                }
            }
        }
    }
  ],
  "dependencies": []
}
```

预测分析模型配置文件示例

如下代码以TensorFlow引擎为例，您可以根据实际使用的引擎类型修改model_type参数后使用。

- 模型输入

```
{
    "data": {
        "req_data": [
            {
                "buying_price": "high",
                "maint_price": "high",
                "doors": "2",
                "persons": "2",
                "lug_boot": "small",
                "safety": "low",
                "acceptability": "acc"
            },
            {
                "buying_price": "high",
                "maint_price": "high",
                "doors": "2",
                "persons": "2",
                "lug_boot": "small",
                "safety": "low",
                "acceptability": "acc"
            }
        ]
    }
}
```

- 模型输出

```
{
    "data": {
        "resp_data": [
            {
                "predict_result": "unacc"
            },
            {
                "predict_result": "unacc"
```



```
        }
      ]
    }
}
```

- 配置文件

  说明

  代码中request结构和response结构中的data参数是json schema数据结构。data/properties里面的内容对应“模型输入”和“模型输出”。

```
{
  "model_type": "TensorFlow",
  "model_algorithm": "predict_analysis",
  "runtime": "tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64",
  "metrics": {
    "f1": 0.345294,
    "accuracy": 0.462963,
    "precision": 0.338977,
    "recall": 0.351852
  },
  "apis": [
    {
      "url": "/",
      "method": "post",
      "request": {
        "Content-type": "application/json",
        "data": {
          "type": "object",
          "properties": {
            "data": {
              "type": "object",
              "properties": {
                "req_data": {
                  "items": [
                    {
                      "type": "object",
                      "properties": {}
                    }
                  ],
                  "type": "array"
                }
              }
            }
          }
        }
      },
      "response": {
        "Content-type": "application/json",
        "data": {
          "type": "object",
          "properties": {
            "data": {
              "type": "object",
              "properties": {
                "resp_data": {
                  "type": "array",
                  "items": [
                    {
                      "type": "object",
                      "properties": {}
                    }
                  ]
                }
              }
            }
          }
        }
      }
```



```
    }
  ],
  "dependencies": [
    {
      "installer": "pip",
      "packages": [
        {
          "restraint": "EXACT",
          "package_version": "1.15.0",
          "package_name": "numpy"
        },
        {
          "restraint": "EXACT",
          "package_version": "5.2.0",
          "package_name": "Pillow"
        }
      ]
    }
  ]
}
```

自定义镜像类型的模型配置文件示例

模型输入和输出与**目标检测模型配置文件示例**类似。

- 模型预测输入为**图片类型**时，request请求示例如下：

  该实例表示模型预测接收一个参数名为images、参数类型为file的预测请求，在推理界面会显示文件上传按钮，以文件形式进行预测。

```
{
  "Content-type": "multipart/form-data",
  "data": {
    "type": "object",
    "properties": {
      "images": {
        "type": "file"
      }
    }
  }
}
```

- 模型预测输入为**json数据类型**时，request请求示例如下：

  该实例表示模型预测接收json请求体，只有一个参数名为input、参数类型为string的预测请求，在推理界面会显示文本输入框，用于填写预测请求。

```
{
  "Content-type": "application/json",
  "data": {
    "type": "object",
    "properties": {
      "input": {
        "type": "string"
      }
    }
  }
}
```

完整请求示例如下：

```
{
  "model_algorithm": "image_classification",
  "model_type": "Image",
  "metrics": {
    "f1": 0.345294,
    "accuracy": 0.462963,
    "precision": 0.338977,
    "recall": 0.351852
  },
```



```
  "apis": [{
    "url": "/",
    "method": "post",
    "request": {
      "Content-type": "multipart/form-data",
      "data": {
        "type": "object",
        "properties": {
          "images": {
            "type": "file"
          }
        }
      }
    },
    "response": {
      "Content-type": "application/json",
      "data": {
        "type": "object",
        "required": [
          "predicted_label",
          "scores"
        ],
        "properties": {
          "predicted_label": {
            "type": "string"
          },
          "scores": {
            "type": "array",
            "items": [{
              "type": "array",
              "minItems": 2,
              "maxItems": 2,
              "items": [{
                  "type": "string"
                },
                {
                  "type": "number"
                }
              ]
            }]
          }
        }
      }
    }
  }]
}
```

机器学习类型的模型配置文件示例

以下代码以XGBoost为例。

- 模型输入：

```
{
  "req_data": [
    {
      "sepal_length": 5,
      "sepal_width": 3.3,
      "petal_length": 1.4,
      "petal_width": 0.2
    },
    {
      "sepal_length": 5,
      "sepal_width": 2,
      "petal_length": 3.5,
      "petal_width": 1
    },
    {
      "sepal_length": 6,
```



```
            "sepal_width": 2.2,
            "petal_length": 5,
            "petal_width": 1.5
        }
    ]
}
```

- 模型输出：

```
{
    "resp_data": [
        {
            "predict_result": "Iris-setosa"
        },
        {
            "predict_result": "Iris-versicolor"
        }
    ]
}
```

- 配置文件：

```
{
    "model_type": "XGBoost",
    "model_algorithm": "xgboost_iris_test",
    "runtime": "python2.7",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [
        {
            "url": "/",
            "method": "post",
            "request": {
                "Content-type": "application/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "req_data": {
                            "items": [
                                {
                                    "type": "object",
                                    "properties": {}
                                }
                            ],
                            "type": "array"
                        }
                    }
                }
            },
            "response": {
                "Content-type": "applicaton/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "resp_data": {
                            "type": "array",
                            "items": [
                                {
                                    "type": "object",
                                    "properties": {
                                        "predict_result": {}
                                    }
                                }
                            ]
                        }
                    }
                }
```



```
        }
      }
    ]
}
```

使用自定义依赖包的模型配置文件示例

如下示例中，定义了1.16.4版本的numpy的依赖环境。

```
{
    "model_algorithm": "image_classification",
    "model_type": "TensorFlow",
    "runtime": "python3.6",
    "apis": [
        {
            "url": "/",
            "method": "post",
            "request": {
                "Content-type": "multipart/form-data",
                "data": {
                    "type": "object",
                    "properties": {
                        "images": {
                            "type": "file"
                        }
                    }
                }
            },
            "response": {
                "Content-type": "applicaton/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "mnist_result": {
                            "type": "array",
                            "item": [
                                {
                                    "type": "string"
                                }
                            ]
                        }
                    }
                }
            }
        }
    ],
    "metrics": {
        "f1": 0.124555,
        "recall": 0.171875,
        "precision": 0.00234938928519385,
        "accuracy": 0.00746268656716417
    },
    "dependencies": [
        {
            "installer": "pip",
            "packages": [
                {
                    "restraint": "EXACT",
                    "package_version": "1.16.4",
                    "package_name": "numpy"
                }
            ]
        }
    ]
}
```


4.1.3 模型推理代码编写说明

本章节介绍了在ModelArts中模型推理代码编写的通用方法及说明，针对常用AI引擎的自定义脚本代码示例（包含推理代码示例），请参见**自定义脚本代码示例**。本文在编写说明下方提供了一个TensorFlow引擎的推理代码示例以及一个在推理脚本中自定义推理逻辑的示例。

ModelArts推理因API网关（APIG）的限制，模型单次预测的时间不能超过40S，模型推理代码编写需逻辑清晰，代码简洁，以此达到更好的推理效果。

推理代码编写指导

1. 在模型代码推理文件“customize_service.py”中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如**表4-9**所示。导入语句所涉及的Python包在ModelArts环境中已配置，用户无需自行安装。

表 4-9 各模型类型的父类名称和导入语句

模型类型	父类	导入语句
TensorFlow	TfServingBaseService	from model_service.tfserving_model_service import TfServingBaseService
PyTorch	PTServingBaseService	from model_service.pytorch_model_service import PTServingBaseService
MindSpore	SingleNodeService	from model_service.model_service import SingleNodeService

2. 可以重写的方法有以下几种。

表 4-10 重写方法

方法名	说明
__init__(self, model_name, model_path)	初始化方法，适用于深度学习框架模型。该方法内加载模型及标签等（pytorch和caffe类型模型必须重写，实现模型加载逻辑）。
__init__(self, model_path)	初始化方法，适用于机器学习框架模型。该方法内初始化模型的路径（self.model_path）。在Spark_MLlib中，该方法还会初始化SparkSession（self.spark）。
_preprocess(self, data)	预处理方法，在推理请求前调用，用于将API接口输入的用户原始请求数据转换为模型期望输入数据。
_inference(self, data)	实际推理请求方法（不建议重写，重写后会覆盖ModelArts内置的推理过程，运行自定义的推理逻辑）。
_postprocess(self, data)	后处理方法，在推理请求完成后调用，用于将模型输出转换为API接口输出。

说明

- 用户可以选择重写preprocess和postprocess方法，以实现API输入数据的预处理和推理输出结果的后处理。
- 重写模型父类的初始化方法init可能导致AI应用“运行异常”。

3. 可以使用的属性为模型所在的本地路径，属性名为“self.model_path”。另外pyspark模型在“customize_service.py”中可以使用“self.spark”获取SparkSession对象。

说明

推理代码中，需要通过绝对路径读取文件。模型所在的本地路径可以通过self.model_path属性获得。

- 当使用TensorFlow、Caffe、MXNet时，self.model_path为模型文件目录路径，读取文件示例如下：

```
# model目录下放置label.json文件，此处读取
with open(os.path.join(self.model_path, 'label.json')) as f:
    self.label = json.load(f)
```

- 当使用PyTorch、Scikit_Learn、pyspark时，self.model_path为模型文件路径，读取文件示例如下：

```
# model目录下放置label.json文件，此处读取
dir_path = os.path.dirname(os.path.realpath(self.model_path))
with open(os.path.join(dir_path, 'label.json')) as f:
    self.label = json.load(f)
```

4. 预处理方法、实际推理请求方法和后处理方法中的接口传入“data”当前支持两种content-type，即“multipart/form-data”和“application/json”。
   - “multipart/form-data”请求

```
curl -X POST \
  <modelarts-inference-endpoint> \
  -F image1=@cat.jpg \
  -F images2=@horse.jpg
```

对应的传入data为

```
[
  {
    "image1":{
      "cat.jpg":"<cat.jpg file io>"
    }
  },
  {
    "image2":{
      "horse.jpg":"<horse.jpg file io>"
    }
  }
]
```

   - “application/json”请求

```
curl -X POST \
  <modelarts-inference-endpoint> \
  -d '{
   "images":"base64 encode image"
  }'
```

对应的传入data为python dict

```
{
  "images":"base64 encode image"
}
```

TensorFlow 的推理脚本示例

TensorFlow MnistService示例如下。更多TensorFlow推理代码示例请参考**TensorFlow**、**TensorFlow 2.1**。其他引擎推理代码请参考**PyTorch**、**Caffe**。

- 推理代码

```
from PIL import Image
import numpy as np
from model_service.tfserving_model_service import TfServingBaseService

class MnistService(TfServingBaseService):

    def _preprocess(self, data):
        preprocessed_data = {}

        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1, 784))
                preprocessed_data[k] = image1

        return preprocessed_data

    def _postprocess(self, data):

        infer_output = {}

        for output_name, result in data.items():

            infer_output["mnist_result"] = result[0].index(max(result[0]))

        return infer_output
```

- 请求

```
curl -X POST \ 在线服务地址 \ -F images=@test.jpg
```

- 返回

```
{"mnist_result": 7}
```

在上面的代码示例中，完成了将用户表单输入的图片的大小调整，转换为可以适配模型输入的shape。首先通过Pillow库读取“32×32”的图片，调整图片大小为“1×784”以匹配模型输入。在后续处理中，转换模型输出为列表，用于Restful接口输出展示。

XGBoost 的推理脚本示例

更多机器学习引擎的推理代码请参考**Pyspark**、**Scikit Learn**。

```
# coding:utf-8
import collections
import json
import xgboost as xgb
from model_service.python_model_service import XgSklServingBaseService


class UserService(XgSklServingBaseService):

    # request data preprocess
    def _preprocess(self, data):
        list_data = []
        json_data = json.loads(data, object_pairs_hook=collections.OrderedDict)
        for element in json_data["data"]["req_data"]:
            array = []
            for each in element:
                array.append(element[each])
                list_data.append(array)
        return list_data

    #   predict
    def _inference(self, data):
        xg_model = xgb.Booster(model_file=self.model_path)
        pre_data = xgb.DMatrix(data)
```



```
        pre_result = xg_model.predict(pre_data)
        pre_result = pre_result.tolist()
        return pre_result

    # predict result process
    def _postprocess(self, data):
        resp_data = []
        for element in data:
            resp_data.append({"predict_result": element})
        return resp_data
```

自定义推理逻辑的推理脚本示例

首先，需要在配置文件中，定义自己的依赖包，详细示例请参见**使用自定义依赖包的模型配置文件示例**。然后通过如下示例代码，实现了“saved_model”格式模型的加载推理。

说明

当前推理基础镜像使用的python的logging模块，采用的是默认的日志级别Warnning，即当前只有warning级别的日志可以默认查询出来。如果想要指定INFO等级的日志能够查询出来，需要在代码中指定logging的输出日志等级为INFO级别。

```
# -*- coding: utf-8 -*-
import json
import os
import threading
import numpy as np
import tensorflow as tf
from PIL import Image
from model_service.tfserving_model_service import TfServingBaseService
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class MnistService(TfServingBaseService):
    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model_inputs = {}
        self.model_outputs = {}

        # label文件可以在这里加载,在后处理函数里使用
        # label.txt放在OBS和模型包的目录

        # with open(os.path.join(self.model_path, 'label.txt')) as f:
        #     self.label = json.load(f)

        # 非阻塞方式加载saved_model模型，防止阻塞超时
        thread = threading.Thread(target=self.get_tf_sess)
        thread.start()

    def get_tf_sess(self):
        # 加载saved_model格式的模型
        # session要重用，建议不要用with语句
        sess = tf.Session(graph=tf.Graph())
        meta_graph_def = tf.saved_model.loader.load(sess, [tf.saved_model.tag_constants.SERVING],
self.model_path)
        signature_defs = meta_graph_def.signature_def
        self.sess = sess
        signature = []

        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)
        if len(signature) == 1:
            model_signature = signature[0]
```



```
    else:
        logger.warning("signatures more than one, use serving_default signature")
        model_signature = tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY

    logger.info("model signature: %s", model_signature)

    for signature_name in meta_graph_def.signature_def[model_signature].inputs:
        tensorinfo = meta_graph_def.signature_def[model_signature].inputs[signature_name]
        name = tensorinfo.name
        op = self.sess.graph.get_tensor_by_name(name)
        self.model_inputs[signature_name] = op

    logger.info("model inputs: %s", self.model_inputs)

    for signature_name in meta_graph_def.signature_def[model_signature].outputs:
        tensorinfo = meta_graph_def.signature_def[model_signature].outputs[signature_name]
        name = tensorinfo.name
        op = self.sess.graph.get_tensor_by_name(name)
        self.model_outputs[signature_name] = op

    logger.info("model outputs: %s", self.model_outputs)

def _preprocess(self, data):
    # https两种请求形式
    # 1. form-data文件格式的请求对应：data = {"请求key值":{"文件名":<文件io>}}
    # 2. json格式对应：data = json.loads("接口传入的json体")
    preprocessed_data = {}

    for k, v in data.items():
        for file_name, file_content in v.items():
            image1 = Image.open(file_content)
            image1 = np.array(image1, dtype=np.float32)
            image1.resize((1, 28, 28))
            preprocessed_data[k] = image1

    return preprocessed_data

def _inference(self, data):
    feed_dict = {}
    for k, v in data.items():
        if k not in self.model_inputs.keys():
            logger.error("input key %s is not in model inputs %s", k, list(self.model_inputs.keys()))
            raise Exception("input key %s is not in model inputs %s" % (k, list(self.model_inputs.keys())))
        feed_dict[self.model_inputs[k]] = v

    result = self.sess.run(self.model_outputs, feed_dict=feed_dict)
    logger.info('predict result : ' + str(result))
    return result

def _postprocess(self, data):
    infer_output = {"mnist_result": []}
    for output_name, results in data.items():

        for result in results:
            infer_output["mnist_result"].append(np.argmax(result))

    return infer_output

def __del__(self):
    self.sess.close()
```

说明

对于ModelArts不支持的结构模型或者多模型加载，需要__init__方法中自己指定模型加载的路径。示例代码如下：

```
# -*- coding: utf-8 -*-
import os
from model_service.tfserving_model_service import TfServingBaseService

class MnistService(TfServingBaseService):
    def __init__(self, model_name, model_path):
        # 获取程序当前运行路径，即model文件夹所在的路径
        root = os.path.dirname(os.path.abspath(__file__))
        # test.onnx为待加载模型文件的名称，需要放在model文件夹下
        self.model_path = os.path.join(root, test.onnx)

        # 多模型加载，例如：test2.onnx
        # self.model_path2 = os.path.join(root, test2.onnx)
```

4.2 模型模板

4.2.1 模型模板简介

相同功能的模型配置信息重复率高，将相同功能的配置整合成一个通用的模板，通过使用该模板，可以方便快捷的导入模型创建AI应用，而不用编写config.json配置文件。简单来说，模板将AI引擎以及模型配置模板化，每种模板对应于1种具体的AI引擎及1种推理模式，借助模板，可以快速导入模型到ModelArts创建AI应用。

背景信息

模板分两大类型：通用类型，非通用类型。

- 非通用类型模板，针对特定的场景所定制的，固定输入输出模式，不可覆盖，如“TensorFlow图像分类模板”，固定使用预置图像处理模式。
- 通用模板，搭载特定的AI引擎以及运行环境，内置的输入输出模式为未定义模式，即不定义具体的输入输出格式，用户需根据模型功能或业务场景重新选择新的输入输出模式来覆盖内置的未定义模式，如图像分类模型应选择预置图像处理模式，而目标检测模型则应选择预置物体检测模式。

  说明

  使用未定义模式的模型将无法部署批量服务。

如何使用模板

您需要先将模型包上传至OBS。模型文件应存放在model目录下，通过该模板创建AI应用时，您需要选择到model这一目录。具体使用方式请参见**从模板中选择元模型**。

支持的模板

- **TensorFlow图像分类模板**
- **TensorFlow-py27通用模板**
- **TensorFlow-py36通用模板**
- **MXNet-py27通用模板**

- MXNet-py36通用模板
- PyTorch-py27通用模板
- PyTorch-py36通用模板
- Caffe-CPU-py27通用模板
- Caffe-GPU-py27通用模板
- Caffe-CPU-py36通用模板
- Caffe-GPU-py36通用模板
- ARM-Ascend模板

支持的输入输出模式

- 预置物体检测模式
- 预置图像处理模式
- 预置预测分析模式
- 未定义模式

4.2.2 模板说明

4.2.2.1 TensorFlow 图像分类模板

简介

搭载TensorFlow1.8引擎，运行环境为“python2.7”，适合导入以“SavedModel”格式保存的TensorFlow图像分类模型。该模板使用平台预置的图像处理模式，模式详情参见**预置图像处理模式**，推理时向模型输入一张“key”为“images”的待处理图片，所以需确保您的模型能处理“key”为“images”的输入。使用该模板导入模型时请选择到包含模型文件的“model”目录。

模板输入

存储在OBS上的TensorFlow模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

预置图像处理模式，不可覆盖，即创建模型时不支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
├── 模型文件              //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
```

```
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

TensorFlow模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
    ├── <<自定义python包>>    可选，用户自有的Python包，在模型推理代码中可以直接引用。
    ├── saved_model.pb        必选，protocol buffer格式文件，包含该模型的图描述。
    ├── variables             对“*.pb”模型主文件而言必选。文件夹名称必须为“variables”，包含模型的权重偏差等信息。
        ├── variables.index                必选
        ├── variables.data-00000-of-00001     必选
    ├──customize_service.py   必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.2 TensorFlow-py27 通用模板

简介

搭载TensorFlow1.8 AI引擎，运行环境为“python2.7”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的TensorFlow模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
├── 模型文件              //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

TensorFlow模型包结构

发布该模型时只需要指定到“model”目录。


```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
    ├── <<自定义python包>>    可选，用户自有的Python包，在模型推理代码中可以直接引用。
    ├── saved_model.pb        必选，protocol buffer格式文件，包含该模型的图描述。
    ├── variables             对“*.pb”模型主文件而言必选。文件夹名称必须为“variables”，包含模型的权重偏差等信息。
        ├── variables.index                 必选
        ├── variables.data-00000-of-00001    必选
    ├──customize_service.py   必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.3 TensorFlow-py36 通用模板

简介

搭载TensorFlow1.8 AI引擎，运行环境为“python3.6”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的TensorFlow模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
├── 模型文件              //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
├── 自定义Python包        //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

TensorFlow模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
    ├── <<自定义python包>>    可选，用户自有的Python包，在模型推理代码中可以直接引用。
    ├── saved_model.pb        必选，protocol buffer格式文件，包含该模型的图描述。
    ├── variables             对“*.pb”模型主文件而言必选。文件夹名称必须为“variables”，包含模型的权重偏差等信息。
        ├── variables.index                 必选
        ├── variables.data-00000-of-00001    必选
    ├──customize_service.py   必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.4 MXNet-py27 通用模板

简介

搭载MXNet1.2.1 AI引擎，运行环境为“python2.7”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的MXNet模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
├── 模型文件                //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

MXNet模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
    ├── <<自定义python包>>      可选，用户自有的Python包，在模型推理代码中可以直接引用。
    ├── resnet-50-symbol.json    必选，模型定义文件，包含模型的神经网络描述，
    ├── resnet-50-0000.params    必选，模型变量参数文件，包含参数和权重信息。
    ├──customize_service.py      必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.5 MXNet-py36 通用模板

简介

搭载MXNet1.2.1 AI引擎，运行环境为“python3.6”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的MXNet模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
├── 模型文件                //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

MXNet模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
    ├── <<自定义python包>>      可选，用户自有的Python包，在模型推理代码中可以直接引用。
    ├── resnet-50-symbol.json    必选，模型定义文件，包含模型的神经网络描述，
    ├── resnet-50-0000.params    必选，模型变量参数文件，包含参数和权重信息。
    ├──customize_service.py      必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.6 PyTorch-py27 通用模板

简介

搭载PyTorch1.0AI引擎，运行环境为“python2.7”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的PyTorch模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
├── 模型文件                //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

PyTorch模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
    ├── <<自定义Python包>>     可选，用户自有的Python包，在模型推理代码中可以直接引用。
    ├── resnet50.pth          必选，pytorch模型保存文件，存有权重变量等信息。
    ├──customize_service.py    必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.7 PyTorch-py36 通用模板

简介

搭载PyTorch1.0AI引擎，运行环境为“python3.6”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的PyTorch模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
├── 模型文件                //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
```

```
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

PyTorch模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
   ├── <<自定义Python包>>     可选，用户自有的Python包，在模型推理代码中可以直接引用。
   ├── resnet50.pth          必选，pytorch模型保存文件，存有权重变量等信息。
   ├──customize_service.py    必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.8 Caffe-CPU-py27 通用模板

简介

搭载Caffe1.0 CPU版AI引擎，运行环境为“python2.7”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录那一层。

模板输入

存储在OBS上的Caffe模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
├
├── 模型文件                //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

Caffe模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
   ├── <<自定义python包>>     可选，用户自有的Python包，在模型推理代码中可以直接引用。
   ├── deploy.prototxt        必选，caffe模型保存文件，存有模型网络结构等信息。
```

```
|── resnet.caffemodel      必选，caffe模型保存文件，存有权重变量等信息。
|── customize_service.py   必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.9 Caffe-GPU-py27 通用模板

简介

搭载Caffe1.0 GPU版AI引擎，运行环境为“python2.7”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的Caffe模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
|── 模型文件               //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
|── 自定义Python包         //可选，用户自有的Python包，在模型推理代码中可以直接引用。
|── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

Caffe模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
|── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
|   |── <<自定义python包>>     可选，用户自有的Python包，在模型推理代码中可以直接引用。
|   |── deploy.prototxt        必选，caffe模型保存文件，存有模型网络结构等信息。
|   |── resnet.caffemodel      必选，caffe模型保存文件，存有权重变量等信息。
|   |── customize_service.py   必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.10 Caffe-CPU-py36 通用模板

简介

搭载Caffe1.0 CPU版AI引擎，运行环境为“python3.6”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的Caffe模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
├── 模型文件                //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

Caffe模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
|── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
   |── <<自定义python包>>     可选，用户自有的Python包，在模型推理代码中可以直接引用。
   |── deploy.prototxt        必选，caffe模型保存文件，存有模型网络结构等信息。
   |── resnet.caffemodel      必选，caffe模型保存文件，存有权重变量等信息。
   |── customize_service.py   必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.11 Caffe-GPU-py36 通用模板

简介

搭载Caffe1.0 GPU版AI引擎，运行环境为“python3.6”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的Caffe模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
├── 模型文件                //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

Caffe模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
   ├── <<自定义python包>>     可选，用户自有的Python包，在模型推理代码中可以直接引用。
   ├── deploy.prototxt        必选，caffe模型保存文件，存有模型网络结构等信息。
   ├── resnet.caffemodel      必选，caffe模型保存文件，存有权重变量等信息。
   ├── customize_service.py   必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.2.12 ARM-Ascend 模板

简介

搭载MindSpore AI引擎，运行环境为“python3.5”，内置输入输出模式为未定义模式，请根据模型功能或业务场景重新选择合适的输入输出模式。使用该模板导入模型时请选择到包含模型文件的model目录。

模板输入

存储在OBS上的om模型包，确保您使用的OBS目录与ModelArts在同一区域。模型包的要求请参见**模型包示例**。

对应的输入输出模式

未定义模式，可覆盖，即创建模型时支持选择其他输入输出模式。

模型包规范

- 模型包必须存储在OBS中，且必须以“model”命名。“model”文件夹下面放置模型文件、模型推理代码。
- 模型推理代码文件必选，其文件名必须为“customize_service.py”，“model”文件夹下有且只能有1个推理代码文件，模型推理代码编写请参见**模型推理代码编写说明**。
- 使用模板导入的模型包结构如下所示：

```
model/
|
├── 模型文件                //必选，不同的框架，其模型文件格式不同，详细可参考模型包示例。
```

```
├── 自定义Python包          //可选，用户自有的Python包，在模型推理代码中可以直接引用。
├── customize_service.py  //必选，模型推理代码，文件名称必须为“customize_service.py”，否则不视为推理代码。
```

模型包示例

om模型包结构

发布该模型时只需要指定到“model”目录。

```
OBS桶/目录名
├── model    必选，文件夹名称必须为“model”，用于放置模型相关文件。
    ├── <<自定义python包>>    可选，用户自有的Python包，在模型推理代码中可以直接引用。
    ├── model.om              必选，protocol buffer格式文件，包含该模型的图描述。
    ├──customize_service.py   必选，模型推理代码，文件名称必须为“customize_service.py”，有且只有1个推理代码文件。“customize_service.py”依赖的“py”文件可以直接放“model”目录下。
```

4.2.3 输入输出模式说明

4.2.3.1 预置物体检测模式

输入

系统预置物体检测输入输出模式，适用于物体检测的模型，使用该模式的模型被标识为物体检测模型。预测请求路径“/”，请求协议为“HTTP”，请求方法为“POST”，调用方需采用“multipart/form-data”内容类型，以“key”为“images”，“type”为“file”的格式输入待处理图片。选择该模式时需确保您的模型能处理key为images的输入数据。

输出

推理结果以“JSON”体的形式返回，具体字段请参见表4-11。

表 4-11 参数说明

字段名	类型	描述
detection_classes	字符串数组	输出物体的检测类别列表，如["yunbao","cat"]
detection_boxes	数组，元素为浮点数数组	输出物体的检测框坐标列表，坐标表示为 $[y_{min}, x_{min}, y_{max}, x_{max}]$
detection_scores	浮点数数组	输出每种检测列表的置信度，用来衡量识别的准确度。

推理结果的“JSON Schema”表示如下：

```
{
    "type": "object",
    "properties": {
        "detection_classes": {
            "items": {
```



```
            "type": "string"
         },
         "type": "array"
      },
      "detection_boxes": {
         "items": {
            "minItems": 4,
            "items": {
               "type": "number"
            },
            "type": "array",
            "maxItems": 4
         },
         "type": "array"
      },
      "detection_scores": {
         "items": {
            "type": "string"
         },
         "type": "array"
      }
   }
}
```

请求样例

该模式下的推理方式均为输入一张待处理图片，推理结果以“JSON”格式返回。示例如下：

- 页面预测

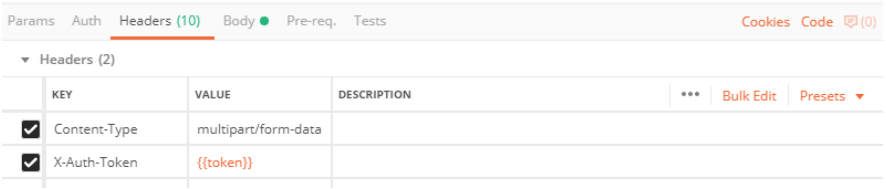
  在服务详情的“预测”页签，上传需要检测的图片，单击“预测”即可获取检测结果。

- Postman调REST接口预测

  部署上线成功后，您可以从服务详情页的调用指南中获取预测接口地址，预测步骤如下：

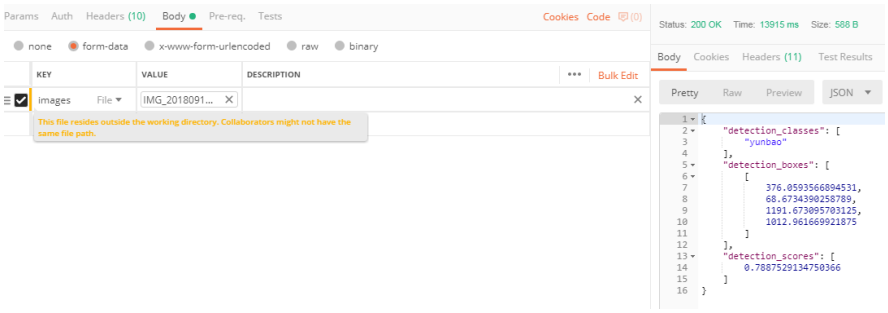
  - 选择“Headers”设置请求头部，“Content-Type”的值设为“multipart/form-data”，“X-Auth-Token”的值设为用户实际获取的token值。

    图 4-1 设置请求头部

  - 选择“Body”设置请求体，“key”选择为“images”，选择为“File”类型，接着通过选择文件按钮选择需要处理的图片，最后单击“send”，发送您的预测请求。

图 4-2 设置请求体



4.2.3.2 预置图像处理模式

输入

系统预置图像处理输入输出模式，适用于图像分类、物体检测和图像语义分割等图像处理模型。预测请求路径“/”，请求协议为“HTTPS”，请求方法为“POST”，调用方需采用“multipart/form-data”内容类型，以“key”为“images”，“type”为“file”的格式输入待处理图片。选择该模式时需确保您的模型能处理key为images的输入数据。

输出

推理结果以“JSON”体的形式返回，“JSON”的具体字段由模型决定。

请求样例

该模式下的推理方式均为输入一张待处理图片，响应的“JSON”根据模型改变而改变。示例如下：

- 页面预测
- Postman调REST接口预测

  部署上线成功后，您可以从服务详情页的调用指南中获取预测接口地址。选择“Body”设置请求体，“key”选择为“images”，选择为“File”类型，接着通过选择文件按钮选择需要处理的图片，最后单击“send”，发送您的预测请求。

图 4-3 调用 REST 接口

4.2.3.3 预置预测分析模式

输入

系统预置预测分析输入输出模式，适用于预测分析的模型，使用该模式的模型将被标识为预测分析模型。预测请求路径“/”，请求协议为“HTTP”，请求方法为“POST”，调用方需采用“application/json”内容类型，发送预测请求，请求体以“JSON”格式表示，“JSON”字段说明请参见表4-12。选择该模式时需确保您的模

型能处理符合该输入“JSON Schema”格式的输入数据。“JSON Schema”格式说明请参考**官方指导**。

表 4-12 JSON 字段说明

字段名	类型	描述
data	Data结构	包含预测数据。“Data结构”说明请参见**表4-13**。

表 4-13 Data 结构说明

字段名	类型	描述
req_data	ReqData结构数组	预测数据列表。

“ReqData”，是“Object”类型，表示预测数据，数据的具体组成结构由业务场景决定。使用该模式的模型，其自定义的推理代码中的预处理逻辑应能正确处理模式所定义的输入数据格式。

预测请求的“JSON Schema”表示如下：

```
{
    "type": "object",
    "properties": {
        "data": {
            "type": "object",
            "properties": {
                "req_data": {
                    "items": [{
                        "type": "object",
                        "properties": {}
                    }],
                    "type": "array"
                }
            }
        }
    }
}
```

输出

预测结果以“JSON”格式返回，“JSON”字段说明请参见**表4-14**。

表 4-14 JSON 字段说明

字段名	类型	描述
data	Data结构	包含预测数据。“Data结构”说明请参见**表4-15**。

表 4-15 Data 结构说明

字段名	类型	描述
resp_data	RespData结构数组	预测结果列表。

与“ReqData”一样，“RespData”也是“Object”类型，表示预测结果，其具体组成结构由业务场景决定。建议使用该模式的模型，其自定义的推理代码中的后处理逻辑应输出符合模式所定义的输出格式的数据。

预测结果的“JSON Schema”表示如下：

```
{
    "type": "object",
    "properties": {
        "data": {
            "type": "object",
            "properties": {
                "resp_data": {
                    "type": "array",
                    "items": [{
                        "type": "object",
                        "properties": {}
                    }]
                }
            }
        }
    }
}
```

请求样例

该模式下的推理方式均为输入“JSON”格式的待预测数据，预测结果以“JSON”格式返回。示例如下：

- 页面预测

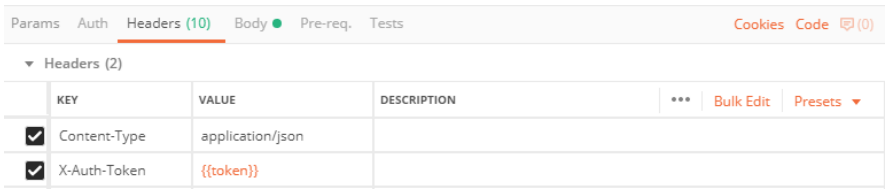
  在服务详情的“预测”页签，输入预测代码，单击“预测”即可获取检测结果。

- Postman调REST接口预测

  部署上线成功后，您可以从服务详情页的调用指南中获取预测接口地址，预测步骤如下：

  - 选择“Headers”设置请求头部，“Content-Type”的值设为“application/json”，“X-Auth-Token”的值设为用户实际获取的token值。

    图 4-4 预测设置请求头部

  - 选择“Body”设置请求体，编辑需要预测的数据，最后单击“send”，发送您的预测请求。

4.2.3.4 未定义模式

描述

未定义的模式，即不定义具体的输入输出格式，请求的输入输出完全由模型决定。当现有的输入输出模式不适合模型的场景时，才考虑选择该模式。使用未定义模式导入的模型无法部署批量服务，同时服务的预测界面可能无法正常工作。

输入

不限。

输出

不限。

请求样例

未定义模式没有特定的请求样例，请求的输入输出完全由模型决定。

4.3 自定义脚本代码示例

4.3.1 TensorFlow

TensorFlow存在两种接口类型，keras接口和tf接口，其训练和保存模型的代码存在差异，但是推理代码编写方式一致。

训练模型（keras接口）：

```
from keras.models import Sequential
model = Sequential()
from keras.layers import Dense
import tensorflow as tf

# 导入训练数据集
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

print(x_train.shape)

from keras.layers import Dense
from keras.models import Sequential
import keras
from keras.layers import Dense, Activation, Flatten, Dropout

# 定义模型网络
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(units=5120,activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(units=10, activation='softmax'))

# 定义优化器，损失函数等
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```



```
model.summary()
# 训练
model.fit(x_train, y_train, epochs=2)
# 评估
model.evaluate(x_test, y_test)
```

保存模型（keras接口）：

```
from keras import backend as K

# K.get_session().run(tf.global_variables_initializer())

# 定义预测接口的inputs和outputs
# inputs和outputs字典的key值会作为模型输入输出tensor的索引键
# 模型输入输出定义需要和推理自定义脚本相匹配
predict_signature = tf.saved_model.signature_def_utils.predict_signature_def(
    inputs={"images" : model.input},
    outputs={"scores" : model.output}
)

# 定义保存路径
builder = tf.saved_model.builder.SavedModelBuilder('./mnist_keras/')

builder.add_meta_graph_and_variables(

    sess = K.get_session(),
    # 推理部署需要定义tf.saved_model.tag_constants.SERVING标签
    tags=[tf.saved_model.tag_constants.SERVING],
    """
    signature_def_map：items只能有一个，或者需要定义相应的key为
    tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY
    """
    signature_def_map={
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
            predict_signature
    }

)
builder.save()
```

训练模型（tf接口）：

```
from __future__ import print_function

import gzip
import os
import urllib

import numpy
import tensorflow as tf
from six.moves import urllib

# 训练数据来源于yann lecun官方网站http://yann.lecun.com/exdb/mnist/
SOURCE_URL = 'http://yann.lecun.com/exdb/mnist/'
TRAIN_IMAGES = 'train-images-idx3-ubyte.gz'
TRAIN_LABELS = 'train-labels-idx1-ubyte.gz'
TEST_IMAGES = 't10k-images-idx3-ubyte.gz'
TEST_LABELS = 't10k-labels-idx1-ubyte.gz'
VALIDATION_SIZE = 5000


def maybe_download(filename, work_directory):
    """Download the data from Yann's website, unless it's already here."""
    if not os.path.exists(work_directory):
        os.mkdir(work_directory)
    filepath = os.path.join(work_directory, filename)
    if not os.path.exists(filepath):
        filepath, _ = urllib.request.urlretrieve(SOURCE_URL + filename, filepath)
        statinfo = os.stat(filepath)
        print('Successfully downloaded %s %d bytes.' % (filename, statinfo.st_size))
```



```
    return filepath


def _read32(bytestream):
    dt = numpy.dtype(numpy.uint32).newbyteorder('>')
    return numpy.frombuffer(bytestream.read(4), dtype=dt)[0]


def extract_images(filename):
    """Extract the images into a 4D uint8 numpy array [index, y, x, depth]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2051:
            raise ValueError(
                'Invalid magic number %d in MNIST image file: %s' %
                (magic, filename))
        num_images = _read32(bytestream)
        rows = _read32(bytestream)
        cols = _read32(bytestream)
        buf = bytestream.read(rows * cols * num_images)
        data = numpy.frombuffer(buf, dtype=numpy.uint8)
        data = data.reshape(num_images, rows, cols, 1)
        return data


def dense_to_one_hot(labels_dense, num_classes=10):
    """Convert class labels from scalars to one-hot vectors."""
    num_labels = labels_dense.shape[0]
    index_offset = numpy.arange(num_labels) * num_classes
    labels_one_hot = numpy.zeros((num_labels, num_classes))
    labels_one_hot.flat[index_offset + labels_dense.ravel()] = 1
    return labels_one_hot


def extract_labels(filename, one_hot=False):
    """Extract the labels into a 1D uint8 numpy array [index]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2049:
            raise ValueError(
                'Invalid magic number %d in MNIST label file: %s' %
                (magic, filename))
        num_items = _read32(bytestream)
        buf = bytestream.read(num_items)
        labels = numpy.frombuffer(buf, dtype=numpy.uint8)
        if one_hot:
            return dense_to_one_hot(labels)
        return labels


class DataSet(object):
    """Class encompassing test, validation and training MNIST data set."""

    def __init__(self, images, labels, fake_data=False, one_hot=False):
        """Construct a DataSet. one_hot arg is used only if fake_data is true."""

        if fake_data:
            self._num_examples = 10000
            self.one_hot = one_hot
        else:
            assert images.shape[0] == labels.shape[0], (
                    'images.shape: %s labels.shape: %s' % (images.shape,
                                                           labels.shape))
            self._num_examples = images.shape[0]

            # Convert shape from [num examples, rows, columns, depth]
            # to [num examples, rows*columns] (assuming depth == 1)
```



```
            assert images.shape[3] == 1
            images = images.reshape(images.shape[0],
                                    images.shape[1] * images.shape[2])
            # Convert from [0, 255] -> [0.0, 1.0].
            images = images.astype(numpy.float32)
            images = numpy.multiply(images, 1.0 / 255.0)
        self._images = images
        self._labels = labels
        self._epochs_completed = 0
        self._index_in_epoch = 0

    @property
    def images(self):
        return self._images

    @property
    def labels(self):
        return self._labels

    @property
    def num_examples(self):
        return self._num_examples

    @property
    def epochs_completed(self):
        return self._epochs_completed

    def next_batch(self, batch_size, fake_data=False):
        """Return the next `batch_size` examples from this data set."""
        if fake_data:
            fake_image = [1] * 784
            if self.one_hot:
                fake_label = [1] + [0] * 9
            else:
                fake_label = 0
            return [fake_image for _ in range(batch_size)], [
                fake_label for _ in range(batch_size)
            ]
        start = self._index_in_epoch
        self._index_in_epoch += batch_size
        if self._index_in_epoch > self._num_examples:
            # Finished epoch
            self._epochs_completed += 1
            # Shuffle the data
            perm = numpy.arange(self._num_examples)
            numpy.random.shuffle(perm)
            self._images = self._images[perm]
            self._labels = self._labels[perm]
            # Start next epoch
            start = 0
            self._index_in_epoch = batch_size
            assert batch_size <= self._num_examples
        end = self._index_in_epoch
        return self._images[start:end], self._labels[start:end]


def read_data_sets(train_dir, fake_data=False, one_hot=False):
    """Return training, validation and testing data sets."""

    class DataSets(object):
        pass

    data_sets = DataSets()

    if fake_data:
        data_sets.train = DataSet([], [], fake_data=True, one_hot=one_hot)
        data_sets.validation = DataSet([], [], fake_data=True, one_hot=one_hot)
        data_sets.test = DataSet([], [], fake_data=True, one_hot=one_hot)
        return data_sets
```



```
    local_file = maybe_download(TRAIN_IMAGES, train_dir)
    train_images = extract_images(local_file)

    local_file = maybe_download(TRAIN_LABELS, train_dir)
    train_labels = extract_labels(local_file, one_hot=one_hot)

    local_file = maybe_download(TEST_IMAGES, train_dir)
    test_images = extract_images(local_file)

    local_file = maybe_download(TEST_LABELS, train_dir)
    test_labels = extract_labels(local_file, one_hot=one_hot)

    validation_images = train_images[:VALIDATION_SIZE]
    validation_labels = train_labels[:VALIDATION_SIZE]
    train_images = train_images[VALIDATION_SIZE:]
    train_labels = train_labels[VALIDATION_SIZE:]

    data_sets.train = DataSet(train_images, train_labels)
    data_sets.validation = DataSet(validation_images, validation_labels)
    data_sets.test = DataSet(test_images, test_labels)
    return data_sets

training_iteration = 1000

modelarts_example_path =  './modelarts-mnist-train-save-deploy-example'

export_path = modelarts_example_path + '/model/'
data_path = './'

print('Training model...')
mnist = read_data_sets(data_path, one_hot=True)
sess = tf.InteractiveSession()
serialized_tf_example = tf.placeholder(tf.string, name='tf_example')
feature_configs = {'x': tf.FixedLenFeature(shape=[784], dtype=tf.float32), }
tf_example = tf.parse_example(serialized_tf_example, feature_configs)
x = tf.identity(tf_example['x'], name='x')  # use tf.identity() to assign name
y_ = tf.placeholder('float', shape=[None, 10])
w = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
sess.run(tf.global_variables_initializer())
y = tf.nn.softmax(tf.matmul(x, w) + b, name='y')
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
values, indices = tf.nn.top_k(y, 10)
table = tf.contrib.lookup.index_to_string_table_from_tensor(
    tf.constant([str(i) for i in range(10)]))
prediction_classes = table.lookup(tf.to_int64(indices))
for _ in range(training_iteration):
    batch = mnist.train.next_batch(50)
    train_step.run(feed_dict={x: batch[0], y_: batch[1]})
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
print('training accuracy %g' % sess.run(
    accuracy, feed_dict={
        x: mnist.test.images,
        y_: mnist.test.labels
    }))
print('Done training!')
```

保存模型（tf 接口）

```
# 导出模型
# 模型需要采用saved_model接口保存
print('Exporting trained model to', export_path)
builder = tf.saved_model.builder.SavedModelBuilder(export_path)

tensor_info_x = tf.saved_model.utils.build_tensor_info(x)
tensor_info_y = tf.saved_model.utils.build_tensor_info(y)
```



```
# 定义预测接口的inputs和outputs
# inputs和outputs字典的key值会作为模型输入输出tensor的索引键
# 模型输入输出定义需要和推理自定义脚本相匹配
prediction_signature = (
    tf.saved_model.signature_def_utils.build_signature_def(
        inputs={'images': tensor_info_x},
        outputs={'scores': tensor_info_y},
        method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME))

legacy_init_op = tf.group(tf.tables_initializer(), name='legacy_init_op')
builder.add_meta_graph_and_variables(
    # tag设为serve/tf.saved_model.tag_constants.SERVING
    sess, [tf.saved_model.tag_constants.SERVING],
    signature_def_map={
        'predict_images':
            prediction_signature,
    },
    legacy_init_op=legacy_init_op)

builder.save()

print('Done exporting!')
```

推理代码（keras 接口和 tf 接口）

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表4-9**。本案例中调用父类“_inference(self, data)”推理请求方法，因此下文代码中不需要重写方法。

```
from PIL import Image
import numpy as np
from model_service.tfserving_model_service import TfServingBaseService


class MnistService(TfServingBaseService):

    # 预处理中处理用户HTTPS接口输入匹配模型输入
    # 对应上述训练部分的模型输入为{"images":<array>}
    def _preprocess(self, data):

        preprocessed_data = {}
        images = []
        # 对输入数据进行迭代
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1,784))
                images.append(image1)
        # 返回numpy array
        images = np.array(images,dtype=np.float32)
        # 对传入的多个样本做batch处理，shape保持和训练时输入一致
        images.resize((len(data), 784))
        preprocessed_data['images'] = images
        return preprocessed_data

    # 对应的上述训练部分保存模型的输出为{"scores":<array>}
    # 后处理中处理模型输出为HTTPS的接口输出
    def _postprocess(self, data):
        infer_output = {"mnist_result": []}
        # 迭代处理模型输出
        for output_name, results in data.items():
            for result in results:
                infer_output["mnist_result"].append(result.index(max(result)))
        return infer_output
```


4.3.2 TensorFlow 2.1

训练并保存模型

```
from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    # 对输出层命名output，在模型推理时通过该命名取结果
    tf.keras.layers.Dense(10, activation='softmax', name="output")
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

tf.keras.models.save_model(model, "./mnist")
```

推理代码：

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表4-9**。

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tfserving_model_service import TfServingBaseService

logger = logging.getLogger()
logger.setLevel(logging.INFO)


class MnistService(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

        # label文件可以在这里加载,在后处理函数里使用
        # label.txt放在obs和模型包的目录

        # with open(os.path.join(self.model_path, 'label.txt')) as f:
        #     self.label = json.load(f)
        # 非阻塞方式加载saved_model模型，防止阻塞超时
        thread = threading.Thread(target=self.load_model)
        thread.start()

    def load_model(self):
        # load saved_model 格式的模型
        self.model = tf.saved_model.load(self.model_path)
```



```
    signature_defs = self.model.signatures.keys()

    signature = []
    # only one signature allowed
    for signature_def in signature_defs:
        signature.append(signature_def)

    if len(signature) == 1:
        model_signature = signature[0]
    else:
        logging.warning("signatures more than one, use serving_default signature from %s", signature)
        model_signature = tf.saved_model.DEFAULT_SERVING_SIGNATURE_DEF_KEY

    self.predict = self.model.signatures[model_signature]

def _preprocess(self, data):
    images = []
    for k, v in data.items():
        for file_name, file_content in v.items():
            image1 = Image.open(file_content)
            image1 = np.array(image1, dtype=np.float32)
            image1.resize((28, 28, 1))
            images.append(image1)

    images = tf.convert_to_tensor(images, dtype=tf.dtypes.float32)
    preprocessed_data = images

    return preprocessed_data

def _inference(self, data):

    return self.predict(data)

def _postprocess(self, data):

    return {
        "result": int(data["output"].numpy()[0].argmax())
    }
```

4.3.3 PyTorch

训练模型

训练模型自定义脚本代码示例：

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms

# 定义网络结构
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 输入第二维需要为784
        self.hidden1 = nn.Linear(784, 5120, bias=False)
        self.output = nn.Linear(5120, 10, bias=False)

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = F.relu((self.hidden1(x)))
        x = F.dropout(x, 0.2)
        x = self.output(x)
        return F.log_softmax(x)
```



```
def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                       100. * batch_idx / len(train_loader), loss.item()))

def test( model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

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

device = torch.device("cpu")

batch_size=64

kwargs={}

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor()
                   ])),
    batch_size=batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=False, transform=transforms.Compose([
        transforms.ToTensor()
    ])),
    batch_size=1000, shuffle=True, **kwargs)

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
optimizer = optim.Adam(model.parameters())

for epoch in range(1, 2 + 1):
    train(model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
```

保存模型：

```
# 必须采用state_dict的保存方式，支持异地部署
torch.save(model.state_dict(), "pytorch_mnist/mnist_mlp.pt")
```

推理代码：

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表4-9**。

```
from PIL import Image
import log
```



```
from model_service.pytorch_model_service import PTServingBaseService
import torch.nn.functional as F

import torch.nn as nn
import torch
import json

import numpy as np

logger = log.getLogger(__name__)

import torchvision.transforms as transforms

# 定义模型预处理
infer_transformation = transforms.Compose([
    transforms.Resize((28,28)),
    # 需要处理成pytorch tensor
    transforms.ToTensor()
])


import os


class PTVisionService(PTServingBaseService):

    def __init__(self, model_name, model_path):
        # 调用父类构造方法
        super(PTVisionService, self).__init__(model_name, model_path)
        # 调用自定义函数加载模型
        self.model = Mnist(model_path)
        # 加载标签
        self.label = [0,1,2,3,4,5,6,7,8,9]
        # 亦可通过文件标签文件加载
        # model目录下放置label.json文件，此处读取
        dir_path = os.path.dirname(os.path.realpath(self.model_path))
        with open(os.path.join(dir_path, 'label.json')) as f:
            self.label = json.load(f)


    def _preprocess(self, data):

        preprocessed_data = {}
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # 灰度处理
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
                        input_batch.append(infer_transformation(image1).cuda())
                    else:
                        input_batch.append(infer_transformation(image1))
            input_batch_var = torch.autograd.Variable(torch.stack(input_batch, dim=0), volatile=True)
            print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var

        return preprocessed_data

    def _postprocess(self, data):
        results = []
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

    def _inference(self, data):
```



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

        return result

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.hidden1 = nn.Linear(784, 5120, bias=False)
        self.output = nn.Linear(5120, 10, bias=False)

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = F.relu((self.hidden1(x)))
        x = F.dropout(x, 0.2)
        x = self.output(x)
        return F.log_softmax(x)



def Mnist(model_path, **kwargs):
    # 生成网络
    model = Net()
    # 加载模型
    if torch.cuda.is_available():
        device = torch.device('cuda')
        model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
    else:
        device = torch.device('cpu')
        model.load_state_dict(torch.load(model_path, map_location=device))
    # CPU或者GPU映射
    model.to(device)
    # 声明为推理模式
    model.eval()

    return model
```

4.3.4 Caffe

训练并保存模型

“lenet_train_test.prototxt”文件

```
name: "LeNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
```



```
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
```



```
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
```



```
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

“lenet_solver.prototxt”文件

```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 1000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# solver mode: CPU or GPU
solver_mode: CPU
```

执行训练

```
./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt
```

训练后生成“caffemodel”文件，然后将“lenet_train_test.prototxt”文件改写成部署用的lenet_deploy.prototxt，修改输入层和输出层。

```
name: "LeNet"
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 1 dim: 1  dim: 28 dim: 28 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
```



```
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
```



```
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "prob"
  type: "Softmax"
  bottom: "ip2"
  top: "prob"
}
```

推理代码

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表4-9**。

```
from model_service.caffe_model_service import CaffeBaseService

import numpy as np

import os, json

import caffe

from PIL import Image


class LenetService(CaffeBaseService):

    def __init__(self, model_name, model_path):
        # 调用父类推理方法
        super(LenetService, self).__init__(model_name, model_path)

        # 设置预处理
        transformer = caffe.io.Transformer({'data': self.net.blobs['data'].data.shape})
        # 转换为NCHW格式
        transformer.set_transpose('data', (2, 0, 1))
        # 归一化处理
        transformer.set_raw_scale('data', 255.0)

        # batch size设为1，只支持一张图片的推理
        self.net.blobs['data'].reshape(1, 1, 28, 28)
        self.transformer = transformer

       # 设置类别标签
        self.label = [0,1,2,3,4,5,6,7,8,9]
```



```
def _preprocess(self, data):

    for k, v in data.items():
        for file_name, file_content in v.items():
            im = caffe.io.load_image(file_content, color=False)
            # 图片预处理
            self.net.blobs['data'].data[...] = self.transformer.preprocess('data', im)

            return

def _postprocess(self, data):

    data = data['prob'][0, :]
    predicted = np.argmax(data)
    predicted = {"predicted" : str(predicted) }

    return predicted
```

4.3.5 XGBoost

训练并保存模型

```
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split

# Prepare training data and setting parameters
iris = pd.read_csv('/home/ma-user/work/iris.csv')
X = iris.drop(['variety'],axis=1)
y = iris[['variety']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234565)
params = {
    'booster': 'gbtree',
    'objective': 'multi:softmax',
    'num_class': 3,
    'gamma': 0.1,
    'max_depth': 6,
    'lambda': 2,
    'subsample': 0.7,
    'colsample_bytree': 0.7,
    'min_child_weight': 3,
    'silent': 1,
    'eta': 0.1,
    'seed': 1000,
    'nthread': 4,
}
plst = params.items()
dtrain = xgb.DMatrix(X_train, y_train)
num_rounds = 500
model = xgb.train(plst, dtrain, num_rounds)
model.save_model('/tmp/xgboost.m')
```

训练前请先下载iris.csv数据集，解压后上传至Notebook本地路径/home/ma-user/work/。iris.csv数据集下载地址：**https://gist.github.com/netj/8836201**。Notebook上传文件操作请参见上传本地文件至Notebook中。

保存完模型后，需要上传到OBS目录才能发布。发布时需要带上config.json配置和推理代码customize_service.py。config.json编写请参考**模型配置文件编写说明**，推理代码请参考**推理代码**。

推理代码

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表4-9**。

```
# coding:utf-8
import collections
import json
import xgboost as xgb
from model_service.python_model_service import XgSklServingBaseService
class UserService(XgSklServingBaseService):

    # request data preprocess
    def _preprocess(self, data):
        list_data = []
        json_data = json.loads(data, object_pairs_hook=collections.OrderedDict)
        for element in json_data["data"]["req_data"]:
            array = []
            for each in element:
                array.append(element[each])
            list_data.append(array)
        return list_data

    #   predict
    def _inference(self, data):
        xg_model = xgb.Booster(model_file=self.model_path)
        pre_data = xgb.DMatrix(data)
        pre_result = xg_model.predict(pre_data)
        pre_result = pre_result.tolist()
        return pre_result

    # predict result process
    def _postprocess(self,data):
        resp_data = []
        for element in data:
            resp_data.append({"predictresult": element})
        return resp_data
```

4.3.6 Pyspark

训练并保存模型

```
from pyspark.ml import Pipeline, PipelineModel
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import LogisticRegression

# 创建训练数据，此处通过tuples创建
# Prepare training data from a list of (label, features) tuples.
training = spark.createDataFrame([
    (1.0, Vectors.dense([0.0, 1.1, 0.1])),
    (0.0, Vectors.dense([2.0, 1.0, -1.0])),
    (0.0, Vectors.dense([2.0, 1.3, 1.0])),
    (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "features"])

# 创建训练实例，此处使用逻辑回归算法进行训练
# Create a LogisticRegression instance. This instance is an Estimator.
lr = LogisticRegression(maxIter=10, regParam=0.01)

# 训练逻辑回归模型
# Learn a LogisticRegression model. This uses the parameters stored in lr.
model = lr.fit(training)

# 保存模型到本地目录
# Save model to local path.
model.save("/tmp/spark_model")
```

保存完模型后，需要上传到OBS目录才能发布。发布时需要带上config.json配置和推理代码customize_service.py。config.json编写请参考**模型配置文件编写说明**，推理代码请参考**推理代码**。

推理代码

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表4-9**。

```
# coding:utf-8
import collections
import json
import traceback

import model_service.log as log
from model_service.spark_model_service import SparkServingBaseService
from pyspark.ml.classification import LogisticRegression

logger = log.getLogger(__name__)


class UserService(SparkServingBaseService):
    # 数据预处理
    def _preprocess(self, data):
        logger.info("Begin to handle data from user data...")
        # 读取数据
        req_json = json.loads(data, object_pairs_hook=collections.OrderedDict)
        try:
            # 将数据转换成spark dataframe格式
            predict_spdf = self.spark.createDataFrame(pd.DataFrame(req_json["data"]["req_data"]))
        except Exception as e:
            logger.error("check your request data does meet the requirements ?")
            logger.error(traceback.format_exc())
            raise Exception("check your request data does meet the requirements ?")
        return predict_spdf

    # 模型推理
    def _inference(self, data):
        try:
            # 加载模型文件
            predict_model = LogisticRegression.load(self.model_path)
            # 对数据进行推理
            prediction_result = predict_model.transform(data)
    except Exception as e:
            logger.error(traceback.format_exc())
            raise Exception("Unable to load model and do dataframe transformation.")
        return prediction_result

    # 数据后处理
    def _postprocess(self, pre_data):
        logger.info("Get new data to respond...")
        predict_str = pre_data.toPandas().to_json(orient='records')
        predict_result = json.loads(predict_str)
        return predict_result
```

4.3.7 Scikit Learn

训练并保存模型

```
import json
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.externals import joblib
iris = pd.read_csv('/home/ma-user/work/iris.csv')
X = iris.drop(['variety'],axis=1)
y = iris[['variety']]
# Create a LogisticRegression instance and train model
logisticRegression = LogisticRegression(C=1000.0, random_state=0)
logisticRegression.fit(X,y)
```

```
# Save model to local path
joblib.dump(logisticRegression, '/tmp/sklearn.m')
```

训练前请先下载iris.csv数据集，解压后上传至Notebook本地路径/home/ma-user/work/。iris.csv数据集下载地址：**https://gist.github.com/netj/8836201**。Notebook上传文件操作请参见上传本地文件至Notebook中。

保存完模型后，需要上传到OBS目录才能发布。发布时需要带上“config.json”配置以及“customize_service.py”，定义方式参考**模型包规范介绍**。

推理代码

在模型代码推理文件customize_service.py中，需要添加一个子类，该子类继承对应模型类型的父类，各模型类型的父类名称和导入语句如请参考**表4-9**。

```
# coding:utf-8
import collections
import json
from sklearn.externals import joblib
from model_service.python_model_service import XgSklServingBaseService

class UserService(XgSklServingBaseService):

    # request data preprocess
    def _preprocess(self, data):
        list_data = []
        json_data = json.loads(data, object_pairs_hook=collections.OrderedDict)
        for element in json_data["data"]["req_data"]:
            array = []
            for each in element:
                array.append(element[each])
                list_data.append(array)
        return list_data

    # predict
    def _inference(self, data):
        sk_model = joblib.load(self.model_path)
        pre_result = sk_model.predict(data)
        pre_result = pre_result.tolist()
        return pre_result

    # predict result process
    def _postprocess(self,data):
        resp_data = []
        for element in data:
            resp_data.append({"predictresult": element})
        return resp_data
```


5 云监控平台 ModelArts 监控

5.1 ModelArts 支持的监控指标

功能说明

为使用户更好地掌握自己的ModelArts在线服务和对应模型负载的运行状态，云服务平台提供了云监控。您可以使用该服务监控您的ModelArts在线服务和对应模型负载，执行自动实时监控、告警和通知操作，帮助您更好地了解服务和模型的各项性能指标。

命名空间

SYS.ModelArts

监控指标

表 5-1 ModelArts 支持的监控指标

指标ID	指标名称	指标含义	取值范围	测量对象	监控周期
cpu_usage	CPU使用率	该指标用于统计ModelArts用户服务的CPU使用率。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
mem_usage	内存使用率	该指标用于统计ModelArts用户服务的内存使用率。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟

指标ID	指标名称	指标含义	取值范围	测量对象	监控周期
gpu_util	GPU使用率	该指标用于统计ModelArts用户服务的GPU使用情况。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
gpu_mem_usage	GPU显存使用率	该指标用于统计ModelArts用户服务的GPU显存使用情况。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
npu_util	NPU使用率	该指标用于统计ModelArts用户服务的NPU使用情况。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
npu_mem_usage	NPU显存使用率	该指标用于统计ModelArts用户服务的NPU显存使用情况。 单位：百分比。	≥ 0%	ModelArts模型负载	1分钟
successfully_called_times	调用成功次数	统计ModelArts用户调用服务的成功次数。 单位：次/分钟。	≥Count/min	ModelArts模型负载 ModelArts在线服务	1分钟
failed_called_times	调用失败次数	统计ModelArts用户调用服务的失败次数。 单位：次/分钟。	≥Count/min	ModelArts模型负载 ModelArts在线服务	1分钟
total_called_times	调用总次数	统计ModelArts用户调用服务的次数。 单位：次/分钟。	≥Count/min	ModelArts模型负载 ModelArts在线服务	1分钟
disk_read_rate	磁盘读取速率	统计ModelArts用户服务的磁盘读取速率 单位：bit/min	≥bit/min	ModelArts模型负载	1分钟
disk_write_rate	磁盘写入速率	统计ModelArts用户服务的磁盘写入速率 单位：bit/min	≥bit/min	ModelArts模型负载	1分钟

指标ID	指标名称	指标含义	取值范围	测量对象	监控周期
send_bytes_rate	上行速率	统计ModelArts用户服务的 出方向网络流速。 单位：bit/min	≥bit/min	ModelArts模型负载	1分钟
recv_bytes_rate	下行速率	统计ModelArts用户服务的 入方向网络流速。	≥bit/min	ModelArts模型负载	1分钟
req_count_2xx	2xx响应次数	统计api接口2xx响应的次数	≥Count/min	ModelArts在线服务	1分钟
req_count_4xx	4xx异常次数	统计api接口返回4xx错误的次数	≥Count/min	ModelArts在线服务	1分钟
req_count_5xx	5xx异常次数	统计api接口返回5xx错误的次数	≥Count/min	ModelArts在线服务	1分钟
avg_latency	平均延迟毫秒数	统计api接口平均响应延时时间	≥ms	ModelArts在线服务	1分钟

对于有多个测量维度的测量对象，使用接口查询监控指标时，所有测量维度均为必选。

- 查询单个监控指标时，多维度dim使用样例：
  dim.0=service_id,530cd6b0-86d7-4818-837f-935f6a27414d&dim.1="model_id,3773b058-5b4f-4366-9035-9bbd9964714a。
- 批量查询监控指标时，多维度dim使用样例：
  "dimensions": [
  {
  "name": "service_id",
  "value": "530cd6b0-86d7-4818-837f-935f6a27414d"
  }
  {
  "name": "model_id",
  "value": "3773b058-5b4f-4366-9035-9bbd9964714a"
  }
  ]

维度

表 5-2 维度说明

Key	Value
service_id	在线服务ID。
model_id	模型负载ID。

5.2 设置告警规则

操作场景

通过设置ModelArts在线服务和模型负载告警规则，用户可自定义监控目标与通知策略，及时了解ModelArts在线服务和模型负载状况，从而起到预警作用。

设置ModelArts服务和模型的告警规则包括设置告警规则名称、监控对象、监控指标、告警阈值、监控周期和是否发送通知等参数。本节介绍了设置ModelArts服务和模型告警规则的具体方法。

说明

只有“运行中”的在线服务，支持对接CES监控。

前提条件

- 已创建ModelArts在线服务。
- 已在云监控服务创建ModelArts监控服务。登录“云监控服务”控制台，在“自定义监控”页面，根据界面提示创建ModelArts监控服务。

操作步骤

设置告警规则有多种方式。您可以根据实际应用场景，选择设置告警规则的方式。

- 对ModelArts服务设置告警规则
- 对单个服务设置告警规则
- 对模型版本设置告警规则
- 对服务或模型版本的单个指标设置告警规则

方式一：对整个 ModelArts 服务设置告警规则

1. 登录管理控制台。
2. 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。
3. 在左侧导航栏，选择“告警 > 告警规则”页面，单击“创建告警规则”。
4. 在“创建告警规则”页面，“资源类型”选择“ModelArts”，“维度”选择“服务”，“触发规则”选择“自定义创建”，设置告警策略，完成其他信息填写后，单击“立即创建”。

方式二：对单个服务设置告警规则

1. 登录管理控制台。
2. 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。
3. 在左侧导航栏，选择“云服务监控 > ModelArts”。
4. 选择需要添加告警规则的在线服务名称，单击操作列的“创建告警规则”。
5. 在“创建告警规则”界面，根据界面提示设置ModelArts在线服务和模型负载的告警规则。

方式三：对单个版本设置告警规则

1. 登录管理控制台。
2. 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。
3. 在左侧导航栏，选择“云服务监控 > ModelArts”。
4. 单击在线服务名称前面的小三角，展示模型版本列表，选择需要设置告警规则的模型版本，单击操作列的“创建告警规则”。
5. 在“创建告警规则”界面，根据界面提示设置模型负载的告警规则。

方式四：对服务或模型版本的单个指标设置告警规则

1. 登录管理控制台。
2. 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。
3. 在左侧导航栏，选择“云服务监控 > ModelArts”。
4. 单击在线服务名称或单击在线服务名称前面的小三角，展示模型版本列表，单击模型版本名称，查看告警规则详情。
5. 在告警规则详情页，单击单个指标右上角的加号按钮，对服务或模型版本的单个指标设置告警规则。

5.3 查看监控指标

操作场景

云服务平台提供的云监控，可以对ModelArts在线服务和模型负载运行状态进行日常监控。您可以通过管理控制台，直观地查看ModelArts在线服务和模型负载的各项监控指标。由于监控数据的获取与传输会花费一定时间，因此，云监控显示的是当前时间5～10分钟前的状态。如果您的在线服务刚创建完成，请等待5～10分钟后查看监控数据。

前提条件

- ModelArts在线服务正常运行。
- 已在云监控页面设置告警规则，具体操作请参见**设置告警规则**。
- 在线服务已正常运行一段时间（约10分钟）。
- 对于新创建的在线服务，需要等待一段时间，才能查看上报的监控数据和监控视图。

- 故障、删除状态的在线服务，无法在云监控中查看其监控指标。当在线服务再次启动或恢复后，即可正常查看。

对接云监控之前，用户无法查看到未对接资源的监控数据。具体操作，请参见**设置告警规则**。

操作步骤

1. 登录管理控制台。
2. 在“服务列表”中选择“管理与监管 > 云监控服务”，进入“云监控服务”管理控制台。
3. 在左侧导航栏，选择“云服务监控 > ModelArts”。
4. 查看监控图表。
   - 查看在线服务监控图表：单击目标在线服务“操作”列的“查看监控指标”。
   - 查看模型负载监控图标：单击目标在线服务左侧的  ，在下拉列表中选择模型负载“操作”列的“查看监控指标”。
5. 在监控区域，您可以通过选择时长，查看对应时间的监控数据。

   当前支持查看近1小时、近3小时和近12小时的监控数据，查看更长时间范围监控曲线，请在监控视图中单击进入大图模式查看。