ModelArts

推理部署用户指南

文档版本 05

发布日期 2022-06-07

华为技术有限公司

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。

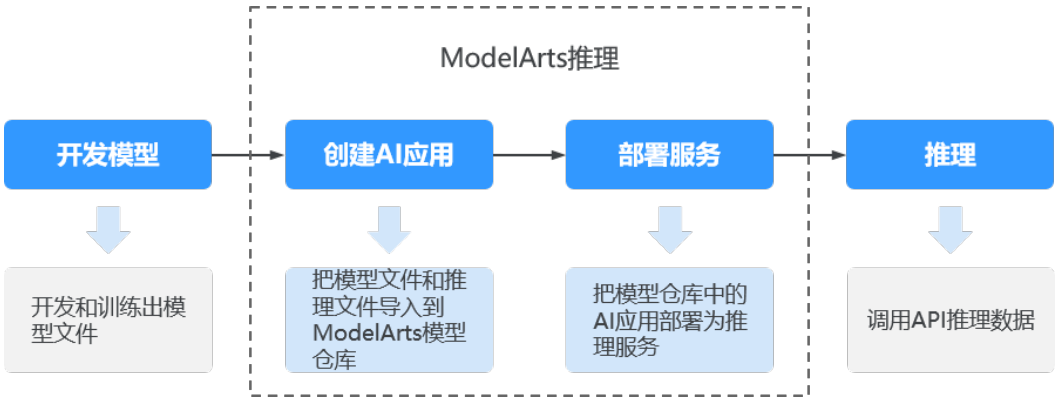
由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

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平台中。

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“部署上线”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。
- **边缘服务**

通过智能边缘平台，在边缘节点将AI应用部署为一个Web Service。

2 管理 AI 应用

2.1 管理 AI 应用简介

说明

为了更方便用户使用和清晰的获取文档帮助，《 AI工程师用户指南 》将按照AI开发流程调整拆分为多本文档，在一级导航中呈现，并从用户使用场景对各模块文档进行了优化和改进。“模型管理”的内容已拆分至《 **推理部署** 》,《 AI工程师用户指南 》里的“模型管理”即将下线。

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。

ModelArtsAI应用管理可导入所有训练生成的元模型，可对所有迭代和调试的AI应用进行统一管理。

使用限制和说明

- 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应用管理列表中。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、模型转换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。
- **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，可以将模型导入至ModelArts中，创建为AI应用，直接用于部署服务。

AI 应用管理的功能描述

表 2-1 AI 应用管理相关功能

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OBS导入元模型（使用模板） • 制作模型镜像并导入 • 从OBS导入元模型（手动配置）
管理AI应用	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。
发布AI应用	针对在ModelArts创建的AI应用，支持将参赛模型提交至比赛项目。

2.2 创建 AI 应用

2.2.1 简介

ModelArts支持您导入本地或者训练作业产生的模型，创建为AI应用。导入模型时，需满足模型包规范。模型包规范内容包括模型、配置模型配置信息config.json、模型推理代码等，详细请参考模型包规范介绍。

- 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创建为AI应用，直接用于部署服务。
- 从OBS导入元模型（使用模板）：您可以将本地的模型按照模型包规范上传至OBS桶中。由于相同功能的模型配置信息重复率高，ModelArts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，模型模板请参考模型模板。
- 从OBS导入元模型（手动配置）：您可以将本地的模型按照模型包规范上传至OBS桶中，从OBS将您的模型导入至ModelArts中，创建为AI应用，用于部署服务。
- 制作模型镜像并导入：针对ModelArts目前不支持的AI引擎，支持自制模型镜像，上传至SWR服务，从SWR中导入模型镜像至ModelArts中，创建为AI应用，用于部署服务。

2.2.2 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。

背景信息

- 如果使用ModelArts训练作业生成的模型，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。
- 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应用管理 > AI应用”，进入AI应用列表页面。
2. 单击左上角的“创建”，进入“创建AI应用”页面。
3. 在“创建AI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2-2**。

表 2-2 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。
标签	AI应用标签，最多支持5个。
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2-3**。

图 2-1 从训练中选择元模型

表 2-3 元模型来源参数说明

参数	说明
“元模型来源”	选择“从训练中选择>训练作业”或者“从训练中选择>训练作业（New）”。 ● 在“选择训练作业”右侧下拉框中选择当前帐号下已完成运行的训练作业及其“版本”。 ● “动态加载”：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，用于实现快速部署和快速更新模型。 **说明** 当前ModelArts同时存在新版训练管理和旧版训练管理功能。旧版训练管理功能仅对部分存量用户可见，新用户不可见。
“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。
“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。当前支持“在线服务”、“批量服务”和“边缘服务”。

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、发布AI应用等操作。

后续操作

部署服务：在“AI应用列表”中，单击AI应用名称左侧的小三角，打开此AI应用下的所有版本。在对应版本所在行，单击“操作”列的“部署”，在下拉框中选择部署类型，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 OBS 导入元模型（使用模板）

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。

背景信息

- 由于相同功能的模型配置信息重复率高，ModelArts将相同功能的配置整合成一个通用的模板，用户通过使用该模板，可以方便快捷的创建AI应用。模板的详细说明请参见**模型模板简介**。

- 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**。
- 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。
- 创建和管理AI应用是免费的，不会产生费用。

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应用管理 > AI应用”，进入AI应用列表页面。
2. 单击左上角的“创建”，进入“创建AI应用”页面。
3. 在“创建AI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2-4**。

表 2-4 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。
标签	AI应用标签，最多支持5个。
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2-5**。

图 2-2 从模板中选择元模型

表 2-5 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**。
“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”	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，用于实现快速部署和快速更新模型。
“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。当前支持“在线服务”、“批量服务”和“边缘服务”。

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、发布AI应用等操作。

后续操作

部署服务：在“AI应用列表”中，单击AI应用名称左侧的小三角，打开此AI应用下的所有版本。在对应版本所在行，单击“操作”列的“部署”，在下拉框中选择部署类型，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 OBS 导入元模型（手动配置）

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。

使用前必读

- 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。ModelArts支持如下常用引擎及其支持的Runtime范围。

表 2-6 支持的常用引擎及其 Runtime

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 tf1.13-python2.7-gpu tf1.13-python2.7-cpu tf1.13-python3.6-gpu tf1.13-python3.6-cpu tf1.13-python3.7-cpu tf1.13-python3.7-gpu tf2.1-python3.7	• 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。
MXNet	python3.7 python3.6 python2.7	• python2.7、python3.6、python3.7的运行环境搭载的MXNet版本为1.2.1。 • python2.7、python3.6、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。

模型使用的引擎类型	支持的运行环境（Runtime）	注意事项
Caffe	python2.7 python3.6 python3.7 python2.7-gpu python3.6-gpu python3.7-gpu python2.7-cpu python3.6-cpu python3.7-cpu	● python2.7、python3.6、python3.7、python2.7-gpu、python3.6-gpu、python3.7-gpu、python2.7-cpu、python3.6-cpu、python3.7-cpu的运行环境搭载的Caffe版本为1.0.0。 ● python2.7、python3.6、python3.7只能用于运行适用于CPU的模型。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。推荐使用python2.7-gpu、python3.6-gpu、python3.7-gpu、python2.7-cpu、python3.6-cpu、python3.7-cpu的Runtime。 ● 默认使用的Runtime为python2.7。
Spark_MLlib	python2.7 python3.6	● 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。
Scikit_Learn	python2.7 python3.6	● 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。
XGBoost	python2.7 python3.6	● 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。

模型使用的引擎类型	支持的运行环境（Runtime）	注意事项
PyTorch	python2.7 python3.6 python3.7 pytorch1.4-python3.7 pytorch1.5-python3.7	• 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。

- 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**。
- 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应用管理 > AI应用”，进入AI应用列表页面。
2. 单击左上角的“创建”，进入“创建AI应用”页面。
3. 在“创建AI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。
标签	AI应用标签，最多支持5个。
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（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”。
“动态加载”	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，用于实现快速部署和快速更新模型。
“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。
“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。当前支持“在线服务”、“批量服务”和“边缘服务”。
“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。

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、发布AI应用等操作。

后续操作

部署服务：在“AI应用列表”中，单击AI应用名称左侧的小三角，打开此AI应用下的所有版本。在对应版本所在行，单击“操作”列的“部署”，在下拉框中选择部署类型，可以将AI应用部署上线为导入AI应用时所选择的部署类型。

2.2.5 制作模型镜像并导入

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。

使用前必读

- 关于自定义镜像规范和说明，请参见**模型镜像规范**。
- 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应用管理 > AI应用”，进入AI应用列表页面。
2. 单击左上角的“创建”，进入“创建AI应用”页面。
3. 在“创建AI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。
标签	AI应用标签，最多支持5个。
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管理员，请确保具备共享该镜像的权限（不支持导入其他帐户共享给您的镜像），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”	可选参数，用于指定模型启动的协议和端口号。
“健康检查”	可选参数，用于指定模型的健康检查。仅当自定义镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致AI应用创建失败。 ● **健康检查URL**：填写健康检查的URL。 ● **健康检查周期**：填写大于0的整数，单位为秒。 ● **健康检查最大失败次数**：填写大于0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。当前支持“在线服务”、“批量服务”和“边缘服务”。
“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。

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、发布AI应用等操作。

后续操作

部署服务：在“AI应用列表”中，单击AI应用名称左侧的小三角，打开此AI应用下的所有版本。在对应版本所在行，单击“操作”列的“部署”，在下拉框中选择部署类型，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 订阅模型

2.3.1 从 Gallery 订阅模型

在AI Gallery中，支持订阅官方发布或者他人分享的模型，订阅后的模型，可推送至ModelArts模型管理中，进行统一管理。

说明

我的订阅模型与云服务订阅模型的区别：

- 在管理控制台中，模型管理所在位置不同。我的订阅统一管理在“AI应用管理>AI应用>我的订阅”页面中，而云服务订阅模型管理在“AI应用管理>云服务订阅AI应用”页面中。
- 模型来源不同。我的订阅，模型来源于AI Gallery；云服务订阅模型，模型来源于其他AI服务开发的模型。

我的订阅模型列表

在ModelArts的“AI应用管理>AI应用>我的订阅”页面中，罗列了从AI Gallery订阅的所有模型。如果需要更多模型，可单击“查找AI应用”，跳转至“AI Gallery”选择更多。

图 2-5 我的订阅模型列表

我的订阅模型，可通过如下几个操作获得：

1. 订阅

   在“AI Gallery”中，作为买家，完成模型订阅。订阅模型的操作指导请参见**买家指导（订阅模型）**。
2. **将订阅模型部署为服务**

将订阅模型部署为服务

针对我的订阅模型，支持将模型一键部署为服务。

1. 在“我的订阅”列表中，单击模型名称左侧的小三角，展开模型的详情和模型版本。
2. 在“版本列表”中，单击“部署”，选择对应的服务类型。

图 2-6 部署

- 若您选择部署的是商用模型，则选择服务类型后会弹出“修改配额”窗口，根据需要选择配额后单击“确定”即可跳转至“部署”页面。

  说明

  由于商用模型支持同时购买多种配额模式的资产，所以仅部署商用模型时需要进行配额选择。免费模型仅一种配额模式无需选择。

图 2-7 修改配额

- 若您选择部署的非商业模型，系统自动跳转至“部署”页面。

3. 在部署页面中，无需再选择模型及其版本，参考**部署模型**的操作指导完成其他参数填写，即可部署为您需要的服务。

2.3.2 从 AI 服务订阅模型

针对其他AI服务开发的AI应用，您可以订阅AI应用，并在ModelArts管理控制台中，快速部署为边缘服务。

在“云服务订阅AI应用”中，AI应用具备几种状态：“正常”、“未支付”、“已过期”。

- “正常”：表示已付费，可以使用的AI应用。请关注其剩余可用天数。
- “未支付”：表示已购买未支付费用，或者未购买，此AI应用暂时无法部署。购买后，AI应用才能变为正常状态。
- “已过期”：表示已经超过可用天数，AI应用暂时无法部署。续费后，AI应用才能变为正常状态。

使用限制

- 只支持部署为边缘服务。
- 当前仅支持部分服务的AI应用，将持续不断开放新的服务。

购买 AI 应用

当在其他AI服务完成AI应用订阅，订阅后的AI应用将呈现在ModelArts的“AI应用管理>AI应用>云服务订阅AI应用”中。在完成购买操作后，AI应用才可用于部署。

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“AI应用管理>AI应用>云服务订阅AI应用”。
2. 在“云服务订阅AI应用”页面，选择“未支付”状态的AI应用，单击操作列的“购买”。
3. 在“购买AI应用”页面，填写“高级配置”、设置“购买时长”，然后单击“下一步”。
4. 在“规格确认”页面，确认信息无误后，单击“去支付”。
5. 在“支付”页面，确认需支付的“金额”，选择“支付方式”，然后单击“确认支付”。

   支付成功后，您可以通过界面提示链接，快速返回订阅模型的AI服务控制台（如人脸识别服务控制台），或者返回订阅列表页面。

部署 AI 应用

对于状态为“正常”的AI应用，可将AI应用部署为边缘服务。

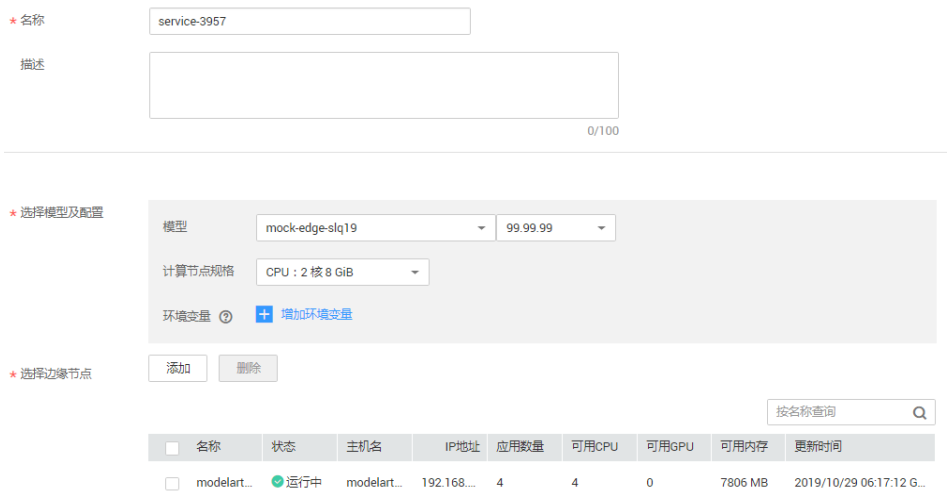
1. 在“订阅列表”页面，选择“正常”状态的AI应用，单击操作列的“部署”。
2. 在部署页面，设置边缘服务的名称和描述，同时，选择AI应用配置以及边缘节点。

表 2-11 参数说明

参数名称	说明
“名称”	边缘服务的名称。
“描述”	边缘服务的简要描述。
“选择AI应用及配置”	此处显示在“订阅列表”中对应的AI应用及其版本号。

参数名称	说明
“计算节点规格”	支持如下几种规格。 ● “CPU：2核 8GiB” ● “CPU：2核 8GiB GPU：1*P4” ● “自定义规格”，如果选择自定义规格，可以在参数下方设置您所需的“CPU”、“Memory”、“GPU”或“Ascend”。其中，“GPU”和“Ascend”只能二选一。
“环境变量”	设置环境变量，注入环境变量到容器实例。
“选择边缘节点”	边缘节点是您自己的边缘计算设备，用于运行边缘应用，处理您的数据，并安全、便捷地和云端应用进行协同。 单击选择边缘节点“添加”，在弹出的“添加节点”对话框中选择节点。选择您已创建的节点后，单击“确定”。

图 2-8 部署为边缘服务



3. 确认信息填写无误后，单击“立即创建”。

   边缘服务创建完成后，可跳转至“部署上线 > 边缘服务”，查看服务的“状态”，当“状态”变为“运行中”时，表示服务已部署成功。如何访问边缘服务，操作指导请参见**访问边缘服务**。

续费

针对“已过期”或“正常”的AI应用，您都可以为其续费。

1. 在“订阅列表”页面，选择“已过期”或“正常”状态的AI应用，单击操作列的“续费”。
2. 在“续费”页面，选择续费时长，然后单击“去支付”。
3. 在“支付”页面，确认需支付的“金额”，选择“支付方式”，然后单击“确认付款”。

支付成功后，您可以通过界面提示链接，快速返回订阅AI应用的AI服务控制台（如人脸识别服务控制台），或者返回订阅列表页面。

退订

针对“正常”的AI应用，如果您不再需要订阅此AI应用，可以执行退订操作。如果AI应用已部署至边缘服务，不支持退订操作。如果需要执行退订，请先删除使用此AI应用部署的边缘服务。

1. 在“订阅列表”页面，选择“正常”状态的AI应用，单击操作列的“退订”。
2. 在“退订资源”页面，选择退订原因，然后单击“退订”。
3. 在弹出的对话框中，确认信息无误后，单击“是”，完成退订操作。

   退订操作提交成功后，您可以通过界面提示链接，查看退订的处理进度。

2.4 管理 AI 应用

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应用管理 > 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，参见**创建AI应用**操作中的参数说明填写相关参数，单击“立即创建”，完成新版本的创建操作。

删除版本

在“AI应用管理 > AI应用”页面，单击AI应用名称左侧的小三角展开版本列表，在版本列表中，单击“操作”列“删除”，即可删除对应的版本。

说明

版本删除后不可恢复，请谨慎操作。

2.5 发布 AI 应用

针对ModelArts中创建的AI应用，支持以下三种发布方式：

- **发布至AI Gallery**

  AI Gallery是在ModelArts的基础上构建的开发者生态社区，提供算法、模型、数据集等内容的共享，为高校科研机构、AI应用开发商、解决方案集成商、企业级个人开发者等群体，提供安全、开放的共享，加速AI资产的开发与落地。

  发布至AI Gallery的资产是免费的，只需要支付在使用过程中消耗的硬件资源，硬件资源费用将根据实际使用情况由华为云ModelArts管理控制台向使用方收取。

- **发布至AI云市场**

华为云云市场是软件及服务交易交付平台。云市场AI专区汇聚优质的人工智能服务提供商，提供丰富的人工智能解决方案、应用、API及算法模型，助力用户快速部署、接入、调用相关应用，方便地购买和使用算法模型。

发布至云市场AI专区的模型为商业售卖资产。买家需购买商品的使用配额进行有偿使用。

说明

云市场仅支持企业级卖家发布商品，请在首次发布商品前**入驻华为云市场**成为企业级卖家。

- **发布至AI大赛**

  华为云人工智能大赛面向开发者组织了一些开发者大赛，您可以在ModelArts开发模型，并将参赛模型提交至对应的比赛项目。

前提条件

已在ModelArts中导入模型。且至少存在一个版本。

发布至 AI Gallery

ModelArts提供了“AI Gallery”功能，方便将个人的模型等共享给所有ModelArts用户，您也可以从“AI Gallery”获取他人共享的内容，快速完成构建。在您完成模型的训练和导入之后，您可以将自己的模型分享至“AI Gallery”，进行知识共享。

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI应用管理 > AI应用”，进入AI应用列表页面。
2. 单击AI应用左侧的小三角展开版本列表，单击“操作”列的“发布 ”进入发布页面。
3. 在发布弹出框中，单击“前往AI Gallery”进入AI Gallery。

图 2-9 前往 AI Gallery

4. 进入AI Gallery后，请参考**免费分享模型**。

发布至 AI 云市场

华为云云市场有完整的售前、交易、售后的保障体系，用户无需担忧交易，省心省力。在您完成模型的训练和导入之后，您可以将自己的模型发布至AI 云市场进行商品售卖。请在首次发布商品前**入驻华为云市场**成为企业级卖家。

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI应用管理 > AI应用”，进入AI应用列表页面。
2. 单击AI应用左侧的小三角展开版本列表，单击“操作”列的“发布”进入发布页面。
3. 在发布弹出框中，单击“前往AI 云市场”进入云市场。

图 2-10 前往云市场

4. 进入云市场后，请参考**发布AI资产类商品操作指导**。

发布至 AI 大赛

华为云人工智能大赛面向开发者组织了一些开发者大赛，您可以在ModelArts开发AI应用，并将参赛AI应用提交至对应的比赛项目。

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI应用管理 > AI应用”，进入AI应用列表页面。
2. 单击AI应用左侧的小三角展开版本列表，单击“操作”列的“发布”进入发布页面。
3. 在发布弹出框中，填写“比赛项目”，确认AI应用信息无误后，单击“提交作品”完成提交。

图 2-11 提交参赛 AI 应用

3 部署 AI 应用（在线服务）

3.1 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。

说明

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，确保帐户未欠费。

操作步骤

1. 登录ModelArts管理控制台，在左侧导航栏中选择“部署上线 > 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。
“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。

参数名称	说明
“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**价格详情说明**。当前仅支持按需付费模式。
“资源池”	“专属资源池”	创建专属资源池请参见**创建专属资源池**。您可以在资源池规格中选择对应的规格进行使用。
“选择AI应用及配置”	“AI应用来源”	根据您的实际情况选择“我的AI应用”或者“我的订阅”。
	“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%。
	“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。
	“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。
	“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
“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。
“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。
“订阅消息”	-	订阅消息使用消息通知服务，在事件列表中选择需要监控的资源池状态，在事件发生时发送消息通知。

图 3-1 设置 AI 应用相关信息

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，运行过程中将按照您选择的资源按需计费。

   您可以前往在线服务列表，查看在线服务的基本情况。在在线服务列表中，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。

3.2 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“部署上线>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3-3**。

表 3-3 在线服务配置

参数	说明
名称	在线服务名称。

参数	说明
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。
调用失败次数/总次数	服务调用次数从创建后开始统计。 如修改AI应用数量，或在AI应用处于“未就绪”时对服务发起调用，不计入统计。
网络配置	如果您使用专属资源池，显示专属资源池的自定义网络配置。
描述	您可以单击编辑按钮，添加服务描述。
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。

3. 您可以在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3-4**。

表 3-4 在线服务详情

参数	说明
调用指南	展示API接口地址、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测试。具体操作请参见**测试服务**。
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”：AI应用名称、版本、状态、分流、 ● “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。
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。

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。

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 3-5 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”和“matches”。
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 3-6 按内置变量配置条件

内置变量	说明
DOMAIN_NAME	调用预测请求的帐号名。
DOMAIN_ID	调用预测请求的帐号ID。

内置变量	说明
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：

  当调用预测请求的帐号名为“zhangsan”时，匹配至指定版本。

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

- 示例二：

  当调用预测请求的帐号名以“op”开头时，匹配至指定版本。

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

表 3-7 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”。

更多正则表达式请参考**表达式全集**。

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.*'
    ```

    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。

3.3 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：

1. **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。
2. **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。

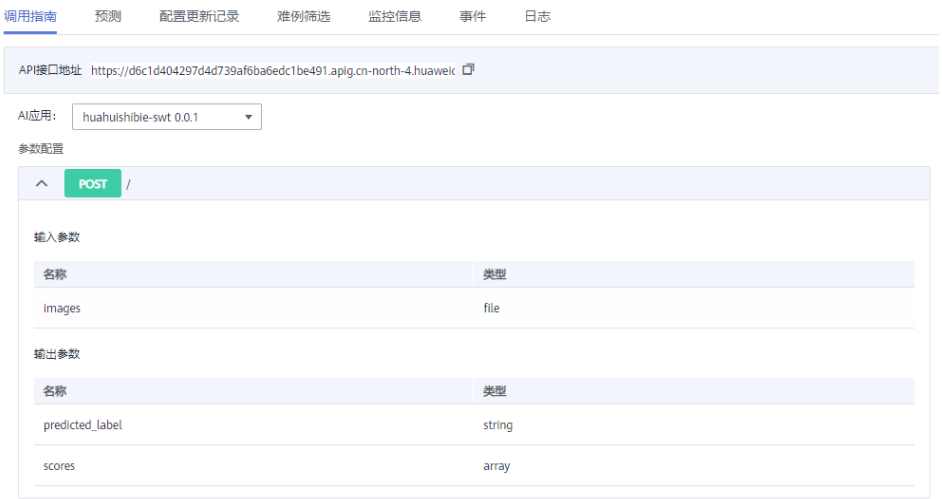
说明

- 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。
- 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

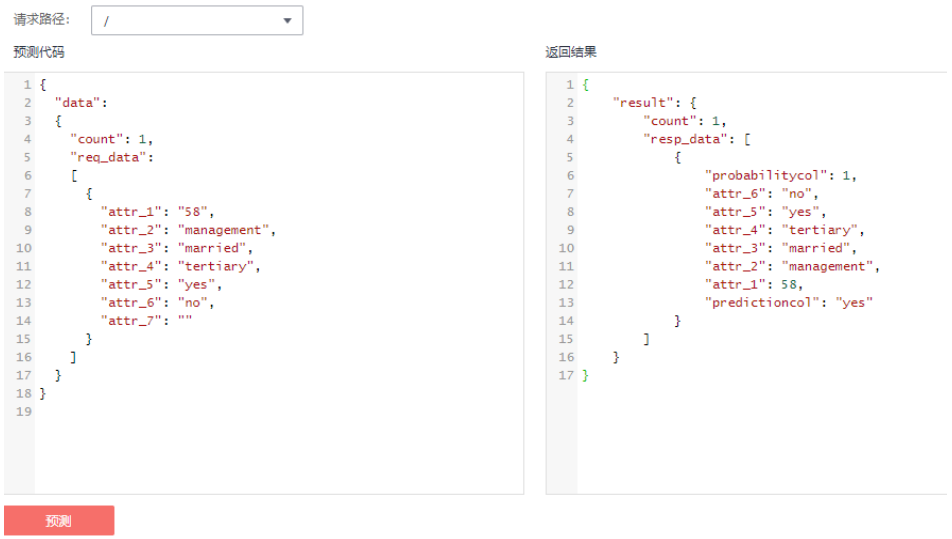
- 如果您的元模型是采用模型模板导入，不同的模板指定了其对应的输入输出模式，请参见**模型模板简介**的相关说明。

JSON 文本预测

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“部署上线>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，如**图3-7**所示，attr_7为需要目标列，predictioncol为目标列attr_7的预测结果。

   JSON文本类的预测代码和返回结果样例，可参见**银行存款预测样例**。此样例是使用自动学习功能训练的元模型，其输入类型为ModelArts官方定义，不可更改。

图 3-7 预测代码



说明

输入数据中attr_7的值可任意填写，或为空，不会影响预测结果。

文件预测

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“部署上线>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3-8所示，输出标签名称，以及位置坐标和检测的评分。

   文件类的预测代码和返回结果样例，可参见**花卉识别样例**。此样例是使用预置算法训练的元模型，其输入类型为ModelArts官方定义，不可更改，如需自定义的元模型，请参见**手写数字识别样例**。

图 3-8 图片预测

3.4 访问在线服务

3.4.1 访问在线服务（Token 认证）

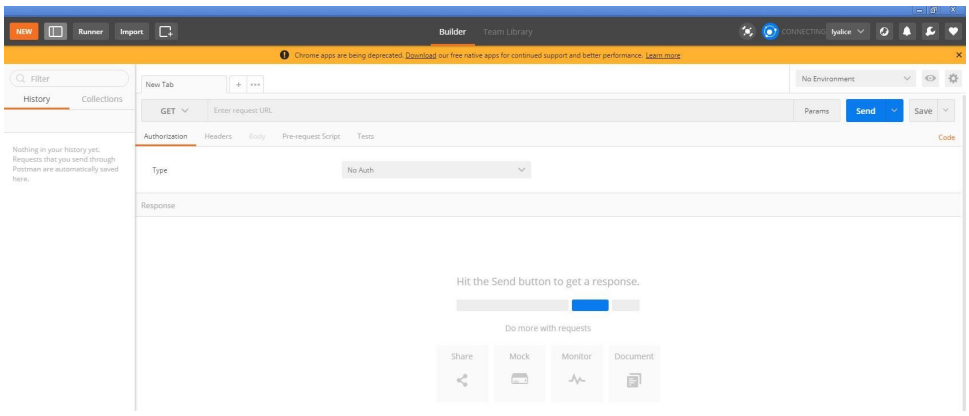
若在线服务的状态处于“运行中”，则表示在线服务已部署成功，部署成功的在线服务，将为用户提供一个可调用的API，此API为标准Restful API。在集成至生产环境之前，需要对此API进行调测，您可以使用以下方式向在线服务发起预测请求：

- **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脚本发送预测请求**。

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（也可使用其它支持发送post请求的软件）。Postman推荐使用7.24.0版本。
2. 打开Postman，如图3-9所示。

图 3-9 Postman 界面



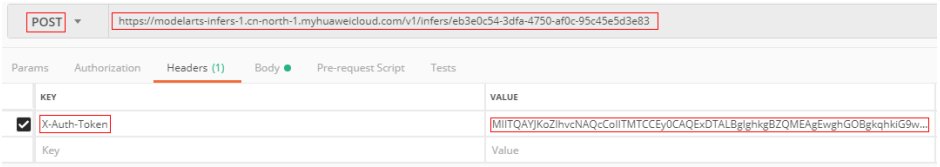
3. 在Postman界面填写参数，以图像分类举例说明。
   - 选择POST任务，将在线服务的调用地址（通过在线服务详情界面-调用指南页签查看）复制到POST后面的方框。Headers页签的Key值填写为“X-Auth-Token”，Value值为您获取到的Token。

     关于如何获取token，请参考**获取用户Token**。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。

     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**SDK参考>Session鉴权>用户AK-SK认证模式**。

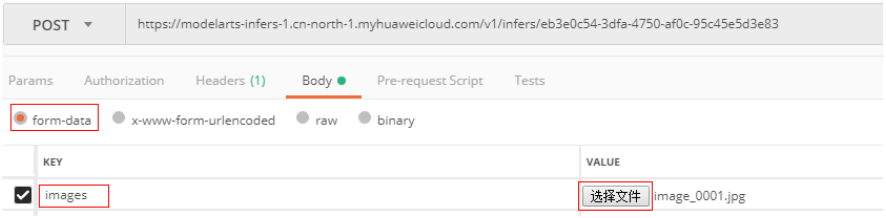
     图 3-10 参数填写

   - 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，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3-11**所示。

       图 3-11 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3-12**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3-13**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 3-12 文件输入预测结果

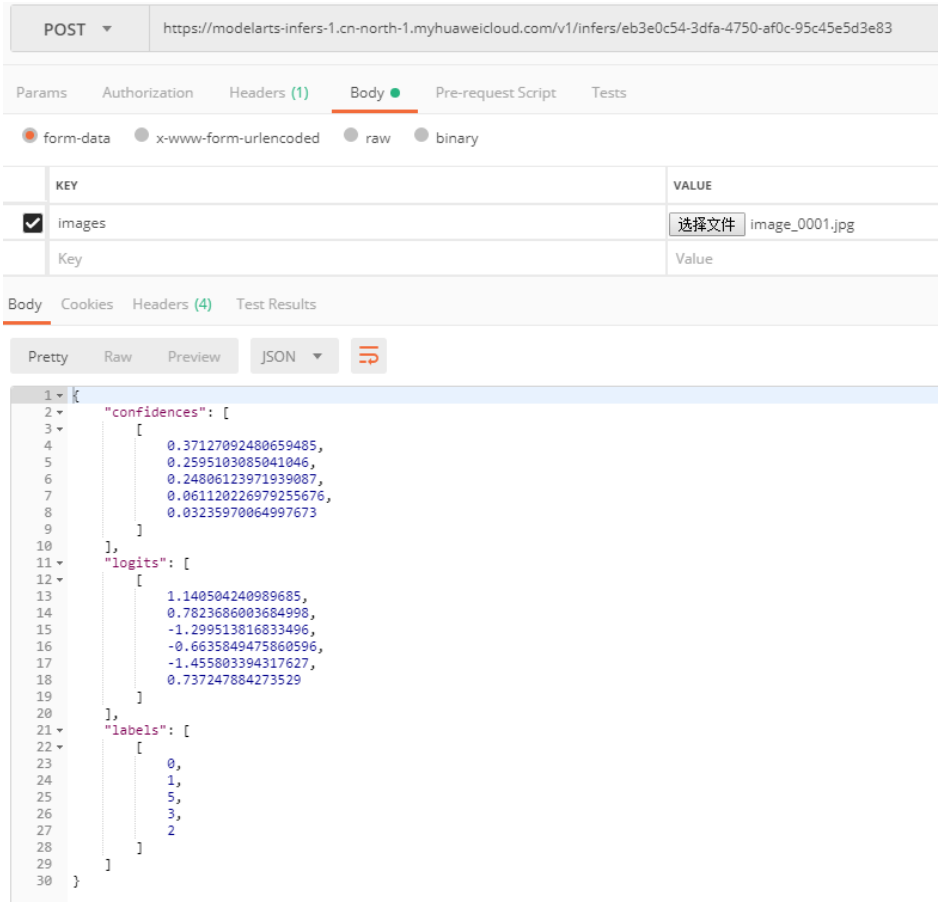
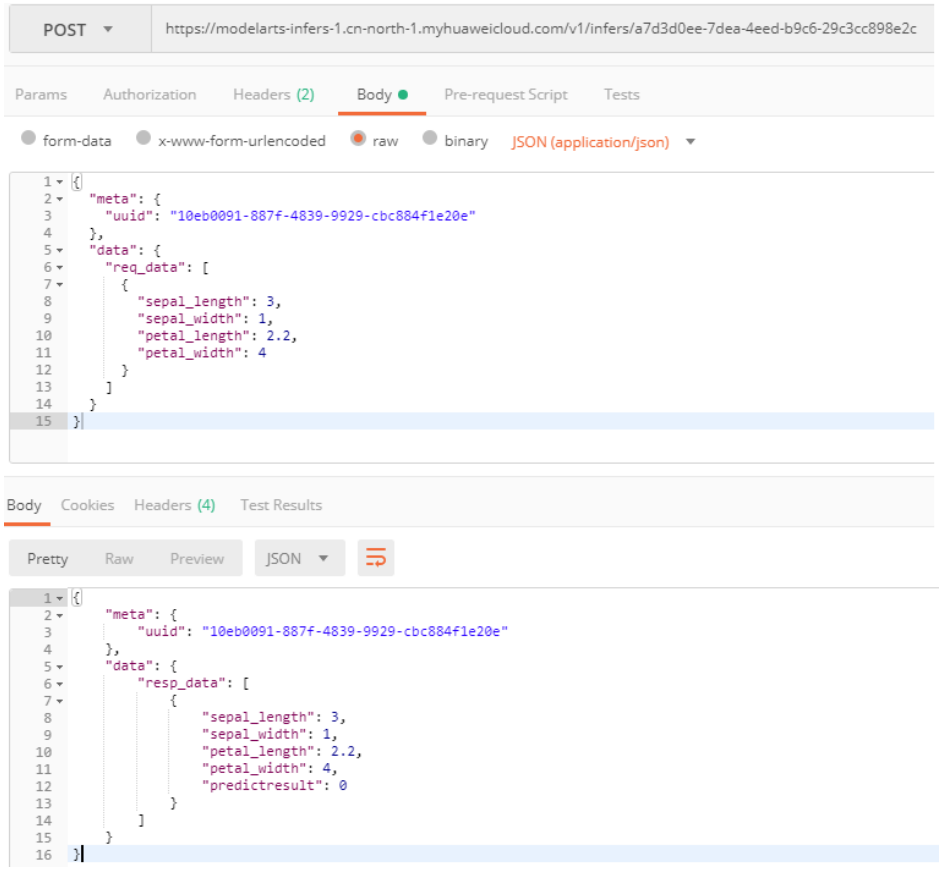


图 3-13 文本输入预测结果



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值是用户获取到的token值（关于如何获取token，请参见**获取请求认证**）。
  - “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 脚本发送预测请求

使用python脚本发送预测请求，也可以处理文件输入和文本输入。

- 以常用的图片文件输入为例：

```
# -*- coding:utf-8 -*-
import requests
import threading
import json

def inference(file_name, num):
  while num > 0:
    num -= 1

    try:
        method = "POST"
        headers = {
            "X-Auth-Token": "MIISkAY***80T9wHQ=="
        }
        url = "https://modelarts-infers-1.xxx/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83"
        files = {
            'images': (file_name, open(file_name, 'rb'), "multipart/form-data")
        }
        resp = requests.request(method, url, headers=headers, files=files, verify=False)
        if 200 != int(resp.status_code):
            raise Exception(
                'inference failed. the status code is:' + str(resp.status_code) + str(resp.content) +
str(resp.headers))
    except Exception as e:
        raise e

if __name__ == '__main__':
    thread = threading.Thread()
    predict = threading.Thread(target=inference, args=("./test.png", 20))  #预测
    predict.start()
    predict.join()  #等待线程结束
```

  - “X-Auth-Token”后面填写的是token值，如何获取token请参见**获取请求认证**。
  - “url”后面填写的是在线服务的调用地址。
  - “args”后面填写需要输入的图片以及执行多少次后停止预测。

- 以文本输入为例：

```
    body = {
        "input": xxx
    }
    data = json.dumps(body)
    resp = requests.request(method, url, data=data, headers=headers, verify=False)
```

  “input”后面填写文本内容。

3.4.2 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URI**
3. **以Python语言为例准备环境**
4. **调用API示例**

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
   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 3-14 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。

图 3-15 访问密钥获取页面示意

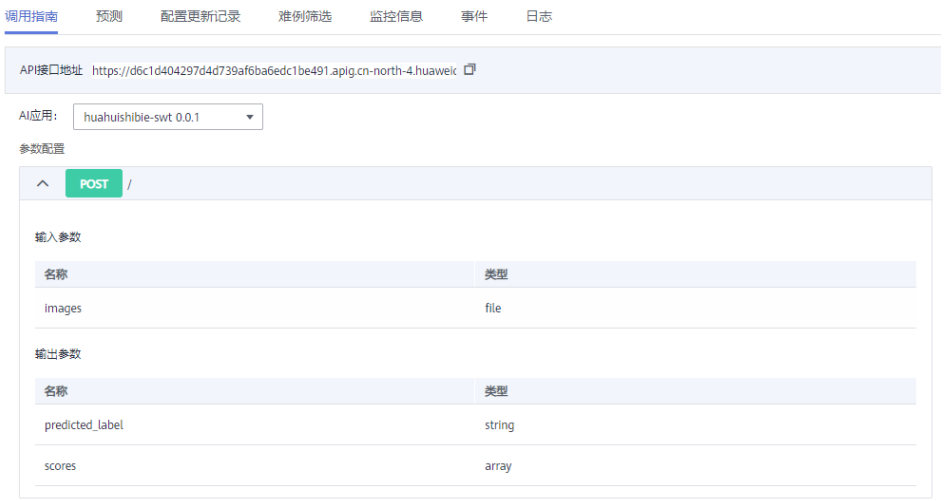
5. 输入验证码，单击“确定”，下载密钥，请妥善保管。

获取 URI

在调用接口时，需获取API接口地址。步骤如下：

1. 登录ModelArts管理控制台，在左侧导航栏中选择“部署上线 > 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API接口。

图 3-16 获取 API 接口



以 Python 语言为例准备环境

此章节内容以Python语言为例介绍APIG SDK使用方式，其他语言的SDK下载与使用示例请参见《API网关 APIG开发指南》。

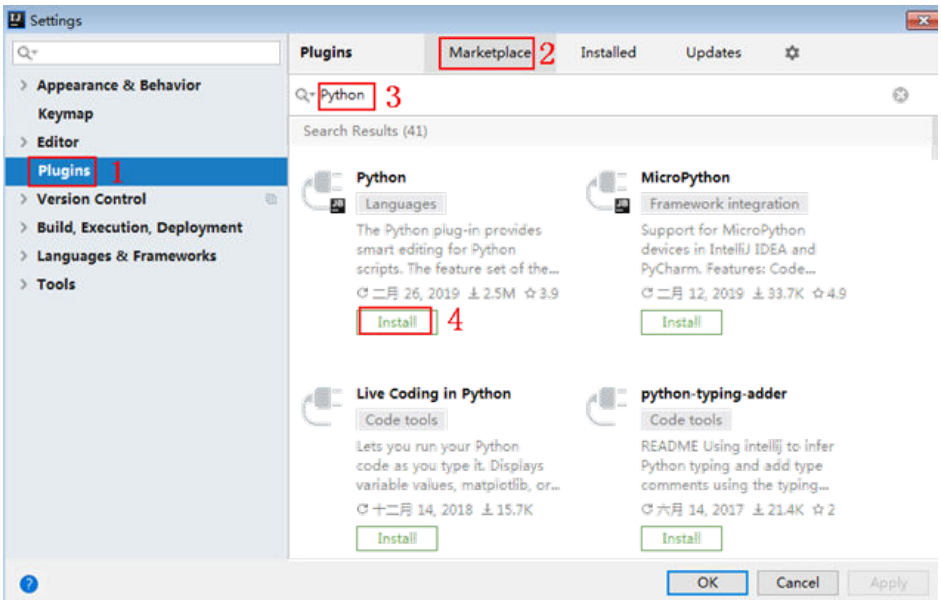
1. 从Python官网获取并安装Python安装包（支持使用2.7.9或3.X版本）。Python安装完成后，您可以执行命令**pip install requests**，通过Python通用包管理工具pip安装“requests”库。

   说明

   如果使用pip安装requests库遇到证书错误，请下载并使用Python执行此文件，升级pip，然后再执行以上命令安装。

2. 从IntelliJ IDEA官网获取并安装IntelliJ IDEA。在IntelliJ IDEA中安装Python插件， 如图3-17所示。

图 3-17 安装 Python 插件



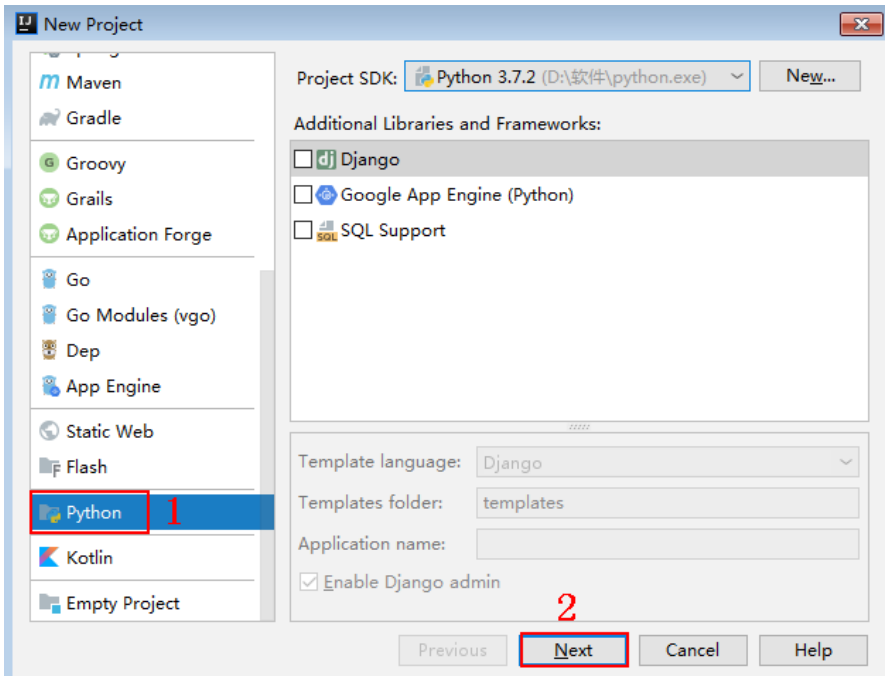
3. 获取SDK。下载SDK，获取“ApiGateway-python-sdk.zip”压缩包，解压后目录结构如下：

表 3-8 SDK 压缩包解压目录

名称	说明
apig_sdk__init__.py	SDK代码
apig_sdk\signer.py	
main.py	示例代码
backend_signature.py	后端签名示例代码
licenses\license-requests	第三方库license文件

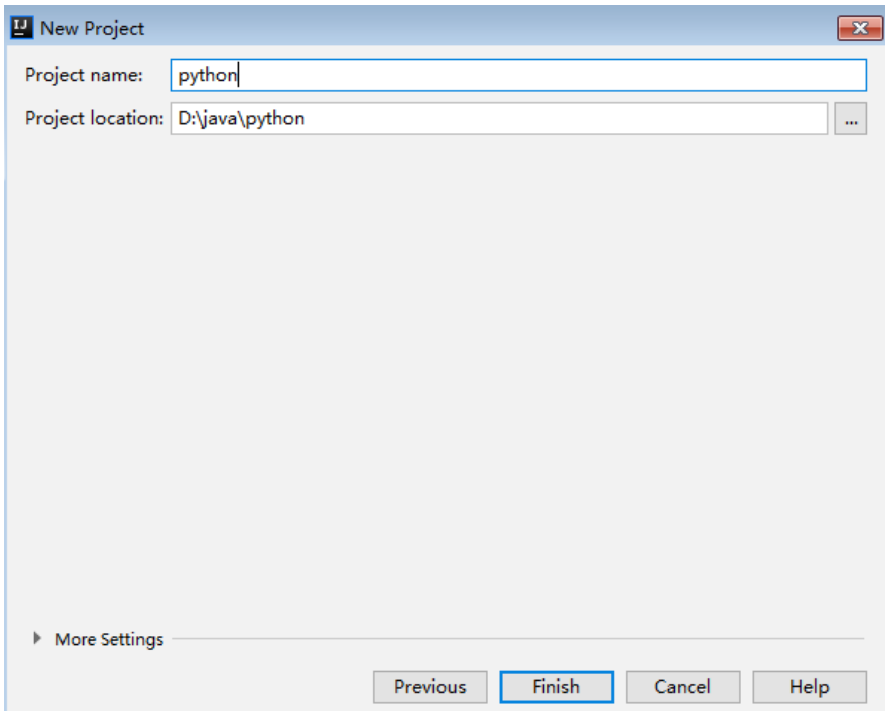
4. 新建工程。
   a. 打开IDEA，选择菜单“File > New > Project”。在弹出的“New Project”对话框中选择“Python”，单击“Next”。

图 3-18 新建工程



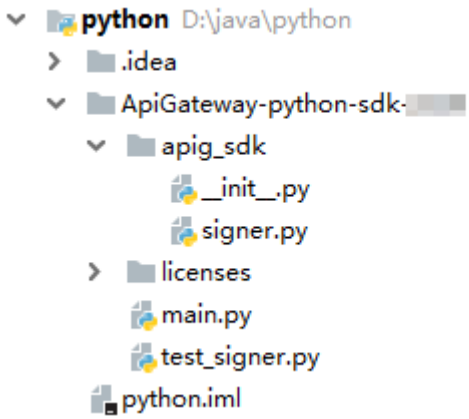
   b. 再次单击“Next”，弹出以下对话框。单击“...”，在弹出的对话框中选择解压后的SDK路径，单击“Finish”完成工程创建。

图 3-19 选择解压后的 SDK 路径



5. 完成工程创建后，目录结构如下。其中“main.py”为示例代码，请根据实际情况修改参数后使用。

图 3-20 新建工程的目录结构



调用 API 示例

1. 在工程中引入“apig_sdk”。

```
from apig_sdk import signer
import requests
```

2. 生成一个新的Signer，填入“AK”和“SK”，获取方式请参见**获取AK/SK**。

```
sig = signer.Signer()
sig.Key = "UATBQ1PQO1D5ORNVCDAA"
sig.Secret = "ap6H7L******0QvHCNk"
```

3. 生成一个Request对象，指定方法名、请求uri、header和body。

```
r= signer.HttpRequest(method, uri, header, body)
```

表 3-9 HttpRequest 参数说明

参数	子参数	是否必填	说明
method	-	是	可填“GET”、“POST”、“PUT”、“DELETE”
uri	-	是	填入在线服务的API接口，获取接口方式参见**获取URI**。
header	x-stage	是	接口发布环境，必填，当前仅支持“RELEASE”。
	Content-Type	否	内容类型，当前支持“application/json”。“multipart/form-data”形式请求体请参考**表3-10**。
	x-sdk-content-sha256	否	签名方式，可填“UNSIGNED-PAYLOAD”，表示不对body进行签名认证。 当body为文件时，该参数必填。
body	-	否	支持json格式，示例：“{\"xxx\":\"xxx\"}”

a. 当请求体为json格式

```
r = signer.HttpRequest("POST",
                       "https://
1684994b180244de9d141c00d3e52c73.apig.exampleRegion.huaweicloudapis.com/v1/infers/
exampleServiceId",
                       {"x-stage": "RELEASE","Content-Type":"application/json"},"{\"xxx\":\"xxx\"}")
```

b. 当请求体为文件格式，您需要构造multipart/form-data形式的请求体。

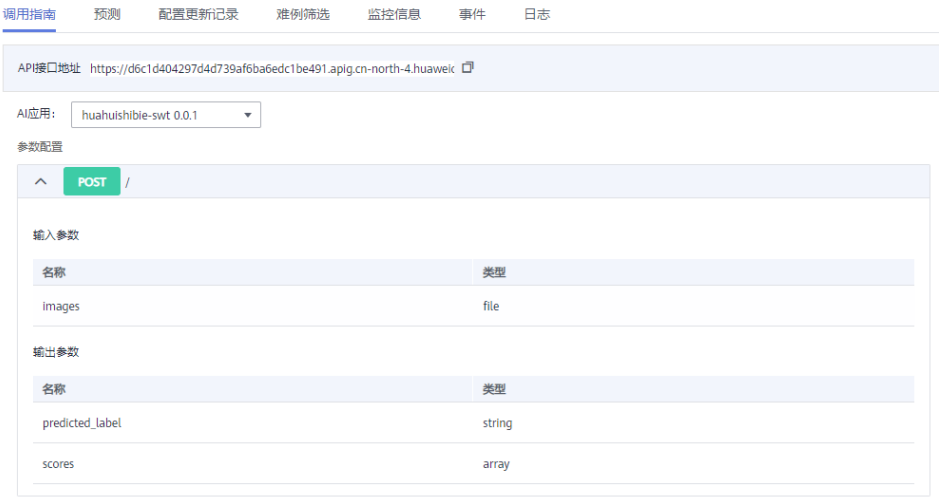
请求体样式：*files={"请求参数":("文件路径"，文件内容，“文件类型”，请求头)}*

表 3-10 files 参数

参数	说明
请求参数	在线服务输入参数名称。
文件路径	上传文件的路径。
文件内容	上传文件的内容。
文件类型	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
请求头	建议填“{}”，请求头在“HttpRequest”参数“header”中填入。

如果您访问请求参数为images在线服务，示例如下。

图 3-21 访问在线服务




```
r = signer.HttpRequest("POST","https://63fb035aeef34368880448a94cb7f440.apig.cn-north-4.huaweicloudapis.com/v1/infers/76c41384-23ab-45f9-a66e-892e7bc2be53",
{"x-stage": "RELEASE", "x-sdk-content-sha256": "UNSIGNED-PAYLOAD"})
files = {"images": ("flower.png", open("flower.png", "rb"), "image/png", {})}
```

4. 进行签名，执行此函数会在请求参数中添加用于签名的“X-Sdk-Date”头和“Authorization”头。

```
sig.Sign(r)
```

5. 调用API，查看访问结果。

```
resp = requests.request(method,url, headers, data, files)
```

表 3-11 request 参数说明

参数	说明
method	填入签名后Request对象的请求方法。
url	填入签名后Request对象的请求地址。
headers	填入签名后Request对象的headers。
data	填入Request对象的body请求体，仅支持Json格式。
files	填入multipart/form-data形式的请求体。

a. 当请求体为json格式

```
resp = requests.request(r.method, r.scheme + "://" + r.host + r.uri, headers=r.headers, data=r.body)
print(resp.status_code, resp.reason)
print(resp.content)
```

b. 当请求体为图片格式

```
resp = requests.request(r.method, r.scheme + "://" + r.host + r.uri, headers=r.headers, data={},
files=files)
print(resp.status_code, resp.reason)
print(resp.content)
```

3.4.3 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。

- **开启支持APP认证功能**：开启支持APP认证并创建应用。
- **在线服务授权管理**：对创建的应用进行管理，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”。
- **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，您可以选择其中一种进行认证鉴权。
- **以Python语言为例准备环境**：在APP认证鉴权完成之后，您可以准备调用环境进行接口调用。
- **调用API示例**：使用授权应用信息调用该接口进行预测。

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，确保帐户未欠费。

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“部署上线 > 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 3-22 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“部署上线>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“部署上线>在线服务”页面，单击“授权管理”，进入“在线服务授权管理”页面。在此页面您可以实现应用的创建和管理，包括查看、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 3-23 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看、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。

- **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，在创建应用时自动生成“AppKey/AppSecret”，“AppCode”也可以通过单击“+添加AppCode”自动生成。

  图 3-24 添加 AppCode

APP 认证鉴权

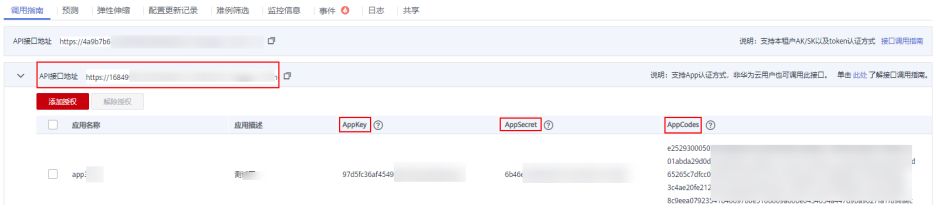
当支持APP认证功能的在线服务运行成功处于“运行中”状态，就可以对服务进行调用 。在调用之前您需要进行APP认证鉴权。

调用支持APP认证的接口有如下两种认证方式，您可以选择其中一种进行认证鉴权。推荐使用AppKey/AppSecret认证，其安全性比AppCode认证要高。

- AppCode认证：通过AppCode认证通用请求。
- AppKey/AppSecret认证：通过AppKey（APP访问密钥ID）与AppSecret（APP私有访问密钥），对请求进行加密签名，可标识发送方，并防止请求被修改。

您可以在服务详情页的“调用指南”页签（如图3-25）或者在线服务授权管理页面（如图3-23）获取API接口和AppKey/AppSecret和AppCode。请注意使用图中红框所示的API接口地址。

图 3-25 获取 API 的接口地址



AppCode认证

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。

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。示例代码如下：

```
import requests
url = 'https://e27faf2cc66a42fbad2dce18747962e5.apig.cn-north-4.huaweicloudapis.com/v1/infers/xxxxxx/goodbye'
headers={'Content-Type':'application/json',
        'X-Apig-Appcode':'dd19xxxxxx'}
response=requests.request('GET', url, headers=headers)
print(response.text)
```

说明

- 当APP不支持AppCode时，可使用AppKey进行简易认证，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。
- 使用AppCode认证访问的更多方式可参考**访问在线服务（Token认证）**。

AppKey/AppSecret认证

AppKey/AppSecret认证就是通过应用的AppKey和AppSecret进行签名认证。

- 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：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。

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。

以 Python 语言为例准备环境

在APP认证鉴权完成之后，您可以准备调用环境进行接口调用。此章节内容以Python语言为例介绍SDK使用方式，其他语言的SDK下载与使用示例请参见**《API网关 APIG开发指南》**。

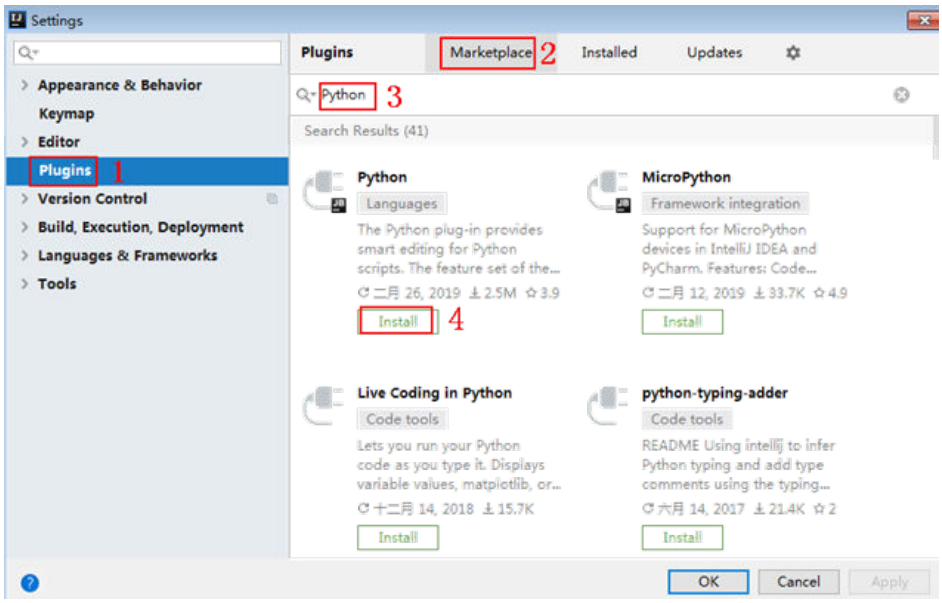
1. 获取API的接口地址、请求方法、AppKey和AppSecret。获取方式请参见**APP认证鉴权**。
2. 从**Python官网**获取并安装Python安装包（支持使用2.7.9或3.X版本）。Python安装完成后，您可以执行命令**pip install requests**，通过Python通用包管理工具pip安装“requests”库。

   说明

   如果使用pip安装requests库遇到证书错误，请下载并使用Python执行**此文件**，升级pip，然后再执行以上命令安装。

3. 从**IntelliJ IDEA官网**获取并安装IntelliJ IDEA。在IntelliJ IDEA中安装Python插件， 如**图3-26**所示。

   图 3-26 安装 Python 插件

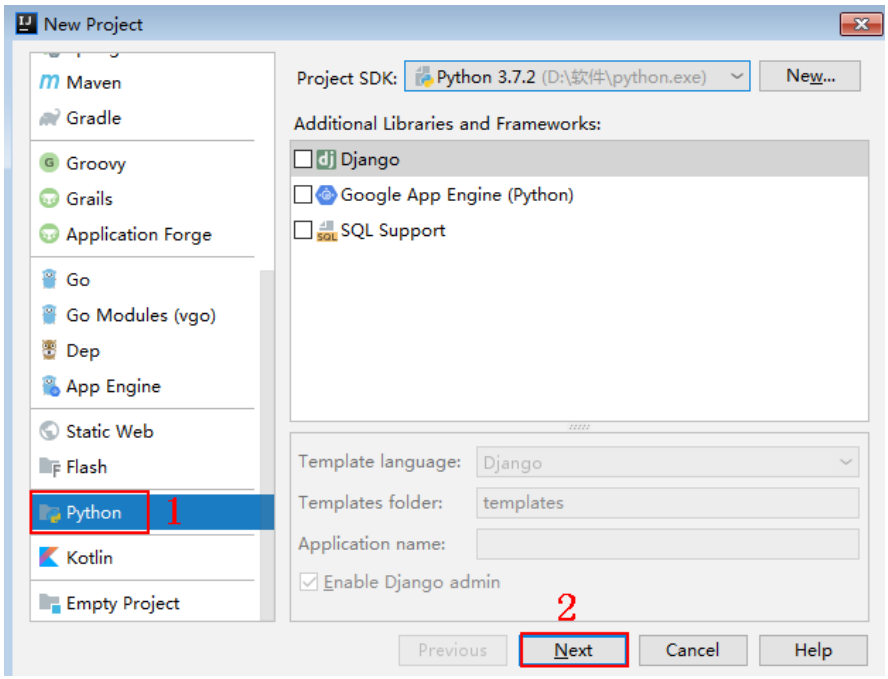
4. 获取SDK。**下载SDK**，获取“ApiGateway-python-sdk.zip”压缩包，解压后目录结构如下：

   表 3-12 SDK 压缩包解压目录

名称	说明
apig_sdk__init__.py	SDK代码
apig_sdk\signer.py	
main.py	示例代码
backend_signature.py	后端签名示例代码
licenses\license-requests	第三方库license文件

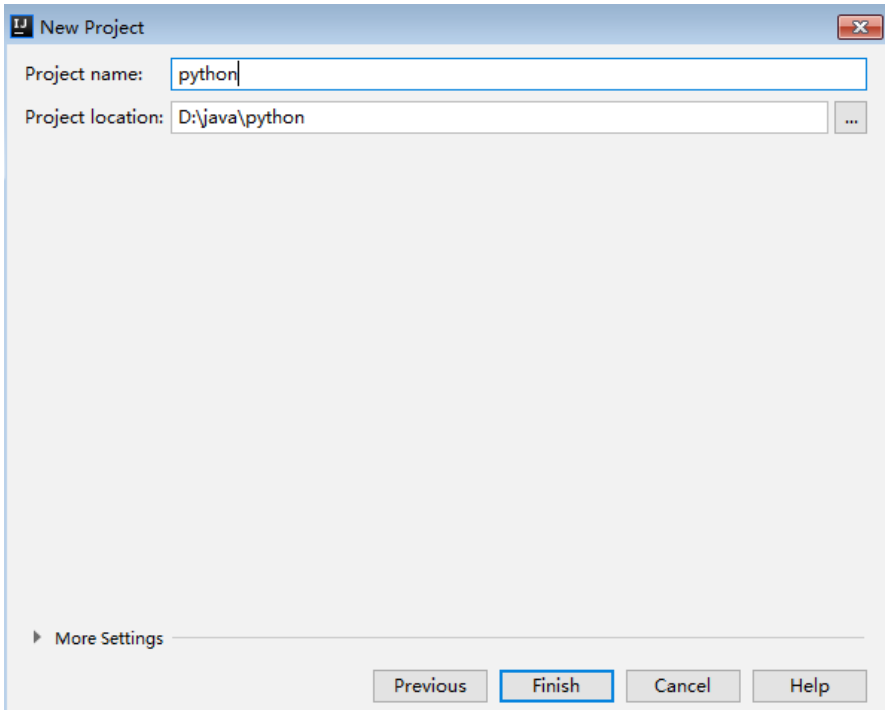
5. 新建工程。
   a. 打开IDEA，选择菜单“File > New > Project”。在弹出的“New Project”对话框中选择“Python”，单击“Next”。

图 3-27 新建工程



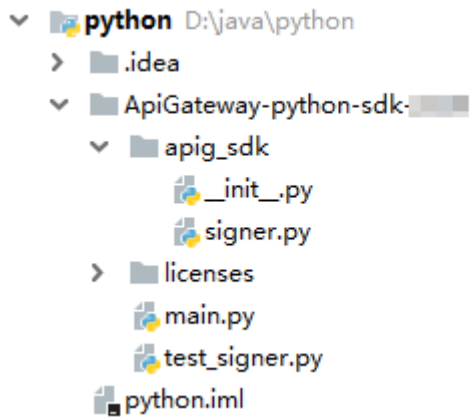
   b. 再次单击“Next”，弹出以下对话框。单击“...”，在弹出的对话框中选择解压后的SDK路径，单击“Finish”完成工程创建。

图 3-28 选择解压后的 SDK 路径



6. 完成工程创建后，目录结构如下。其中“main.py”为示例代码，请根据实际情况修改参数后使用。

图 3-29 新建工程的目录结构



调用 API 示例

1. 在工程中引入“apig_sdk”。

```
from apig_sdk import signer
import requests
```

2. 生成一个新的Signer，填入“AppKey”和“AppSecret”，获取方式请参见**APP认证鉴权**。

```
sig = signer.Signer()
sig.Key = "4f5f626b-073f-402f-a1e0-e52171c6100c"
sig.Secret = "******"
```

3. 生成一个Request对象，指定方法名、请求uri、header和body。

```
r= signer.HttpRequest(method, uri, header, body)
```

表 3-13 HttpRequest 参数说明

参数	子参数	是否必填	说明
method	-	是	可填“GET”、“POST”、“PUT”、“DELETE”
uri	-	是	填入在线服务的API接口，获取接口方式参见**APP认证鉴权**
header	x-stage	是	接口发布环境，必填，当前仅支持“RELEASE”。
	Content-Type	否	内容类型，当前支持“application/json”。“multipart/form-data”形式请求体请参考**表3-14**。
	x-sdk-content-sha256	否	签名方式，可填“UNSIGNED-PAYLOAD”，表示不对body进行签名认证。 当body为文件时，该参数必填。

参数	子参数	是否必填	说明
body	-	否	支持json格式，示例： "{\"xxx\":\"xxx\"}"。

a. 当请求体为json格式

```
r = signer.HttpRequest("POST",
                       "https://
1684994b180244de9d141c00d3e52c73.apig.exampleRegion.huaweicloudapis.com/v1/infers/
exampleServiceId",
                       {"x-stage": "RELEASE","Content-Type":"application/json"},{\"xxx\":\"xxx\"}")
```

b. 当请求体为图片格式，您需要构造multipart/form-data形式的请求体。

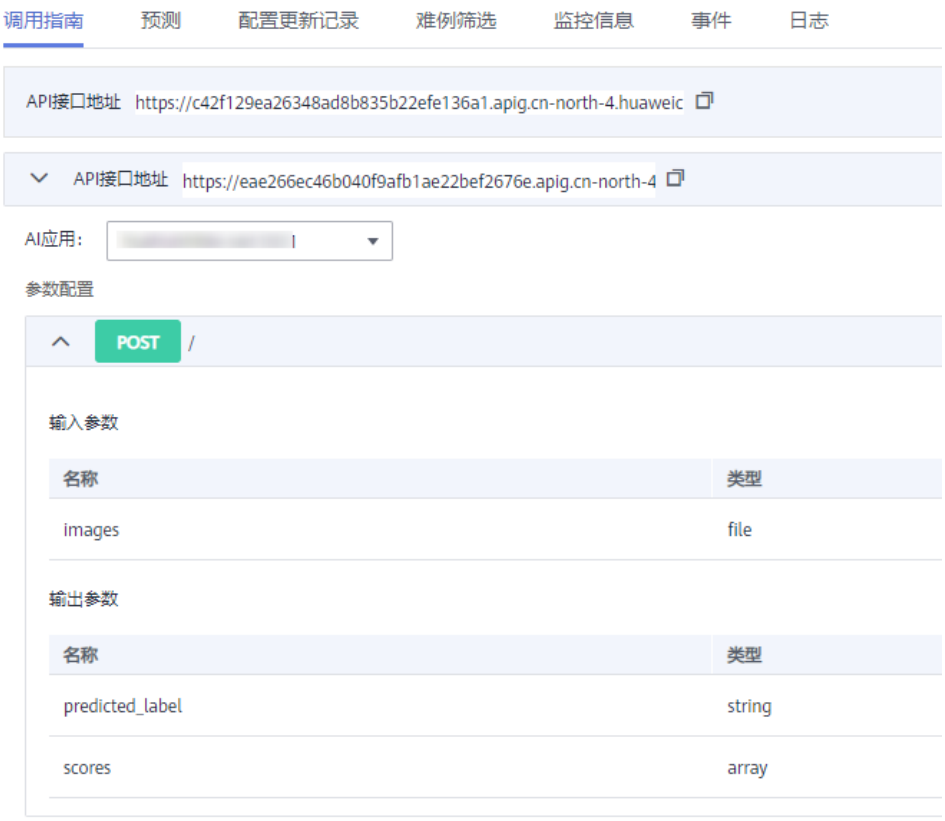
请求体样式：*files={"请求参数":("文件路径"，文件内容，“文件类型”，请求头)}*

表 3-14 files 参数

参数	说明
请求参数	在线服务输入参数名称。
文件路径	上传文件的路径。
文件内容	上传文件的内容。
文件类型	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
请求头	建议填“{}”，请求头在“HttpRequest”参数“header”中填入。

如果您访问请求参数为images，输入格式为file类型的在线服务，示例如下。

图 3-30 访问在线服务




```
r = signer.HttpRequest("POST","https://63fb035aeef34368880448a94cb7f440.apig.cn-north-4.huaweicloudapis.com/v1/infers/76c41384-23ab-45f9-a66e-892e7bc2be53",
{"x-stage": "RELEASE", "x-sdk-content-sha256": "UNSIGNED-PAYLOAD"})
files = {"images": ("flower.png", open("flower.png", "rb"), "image/png", {})}
```

4. 进行签名，执行此函数会在请求参数中添加用于签名的“X-Sdk-Date”头和“Authorization”头。

```
sig.Sign(r)
```

5. 调用API，查看访问结果。

```
resp = requests.request(method,url, headers, data, files)
```

表 3-15 request 参数说明

参数	说明
method	填入签名后Request对象的请求方法。
url	填入签名后Request对象的请求地址。
headers	填入签名后Request对象的headers。
data	填入Request对象的body请求体，仅支持Json格式。
files	填入multipart/form-data形式的请求体。

a. 当请求体为json格式

```
resp = requests.request(r.method, r.scheme + "://" + r.host + r.uri, headers=r.headers, data=r.body)
print(resp.status_code, resp.reason)
print(resp.content)
```

b. 当请求体为图片格式

```
resp = requests.request(r.method, r.scheme + "://" + r.host + r.uri, headers=r.headers, data={},
files=files)
print(resp.status_code, resp.reason)
print(resp.content)
```

3.5 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，本文不再赘述，建议通过互联网搜索并获取调用标准Restful API的指导。

4 部署 AI 应用（批量服务）

4.1 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“部署上线>批量服务”界面，列举了用户所创建的批量服务。您可以在右上方搜索框中输入服务名称，单击 进行查询。

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“部署上线 > 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。
   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4-1。

表 4-1 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“我的AI应用”或者“我的订阅”。
“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10MB。 • 输入数据格式为csv时，建议不要包含中文。如需使用中文，请将文件编码格式设置为UTF-8编码。您可以使用代码方式转换文件编码格式，也可以将csv文件用记事本方式打开，在另存为弹出的窗口页面设置编码格式。
“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必须从0开始的正整数，当index设置不符合规则时,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。请下拉框中选择可用资源，如果资源标识为售罄，表示此暂无此资源。 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。

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。

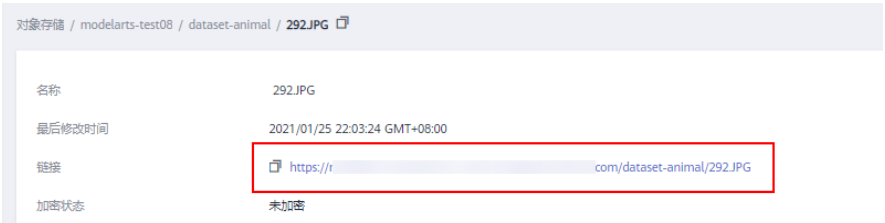
输入manifest文件样例

- 文件名：“test.manifest”
- 文件内容：

```
{"source": "<obs path>/test/data/1.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. “<obs path>/{{桶名}}/{{对象名}}”，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。

     图 4-1 数据的 OBS 路径

     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。

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-{{index}}.manifest
│   │   ├── infer-result
```

```
|   |   |   ├── 1.jpg_result.txt
|   |   |   ├── 2.jpg_result.txt
```

- infer-result-0.manifest文件内容：

```
{"source": "<obs path>/obs-data-bucket/test/data/1.jpg","inference-loc": "<obs path>/test-bucket/test/infer-result/1.jpg_result.txt"}
{"source ": "https://infers-data.obs.cn-north-1.myhuaweicloud.com:443/xgboosterdata/2.jpg?AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz1uDzwve8GpY%3D&x-obs-security-token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-ZaMSxHvl68kKLAy5feYvLDMNZWxzhBZ6Q-3HcoZMh9gISwQOVBwm4ZytB_m8sg1fL6isU7T3CnoL9jmvDGgT9VBC7dC1EyfSJrUcqfB...",  "inference-loc": "obs://test-bucket/test/infer-result/2.jpg_result.txt"}
```

- 文件格式：
  a. 文件名为“infer-result-{{index}}.manifest”，index为实例序号，批量服务运行多少个实例就会产生多少个manifest文件。
  b. manifest同一目录下会创建infer-result目录存放文件处理结果。
  c. 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。
  d. 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. inference-loc：输出结果路径，格式为“<obs path>/{{桶名}}/{{对象名}}”。

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：

```
[
  {
    "protocol": "http",
    "method": "post",
    "url": "/",
    "request": {
      "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
        }]
    }
}
```

4.2 查看批量服务预测结果

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

操作步骤

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“部署上线>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。

5 部署 AI 应用（边缘服务）

5.1 部署为边缘服务

AI应用准备完成后，您可以将AI应用部署为边缘服务。在“部署上线>边缘服务”界面，列举了用户所创建的边缘服务。您可以在右上方搜索框中输入服务名称，单击 进行查询。边缘服务依赖智能边缘平台（IEF），部署前需要在智能边缘平台上创建边缘节点。

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。
- 已在IEF上创建边缘节点。如果您未创建边缘节点，具体操作请参见**创建边缘节点**。
- 由于在线运行需消耗资源，确保帐户未欠费。

背景信息

- 边缘服务目前还处于限时免费阶段，运行中的边缘服务，并不会产生费用。
- 用户最多可创建1000个边缘服务。

部署边缘服务

1. 登录ModelArts管理控制台，在左侧导航栏中选择“部署上线>边缘服务”，默认进入“边缘服务”列表。
2. 在边缘服务列表中，单击左上角“部署”，进入“部署”页面。
3. 在部署页面，填写边缘服务相关参数。
   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。
   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5-1**。

表 5-1 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“我的AI应用”或者“我的订阅”。
“选择AI应用及版本”	选择状态“正常”的AI应用及版本。
“计算节点规格”	支持如下几种规格。 ● “CPU：2核 8GiB”：适合纯CPU类型的负载运行的AI应用。 ● “CPU：2核 8GiB GPU：1*P4”：适合CPU+GPU类型AI应用的运行，带有1个Nvidia P4卡。 ● “自定义规格”，如果选择自定义规格，可以在参数下方设置您所需的“CPU”、“内存配额”、“GPU”或“Ascend”。其中，“GPU”和“Ascend”只能二选一。
“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。 默认设置外部接口协议为https，用户可以通过修改“MODELARTS_SSL_ENABLED”环境变量设置接口协议为http。 MODELARTS_SSL_ENABLED = false
“部署方式”	可选择“节点”或“节点组”。 ● 如果您在IEF创建的是边缘节点，则选择“节点”。IEF相关说明请参见**边缘节点**。 ● 如果您在IEF创建的是铂金版实例和边缘节点组，则选择“节点组”。需指定对应的铂金版“资源实例”和“部署实例个数”。IEF相关说明请参见**边缘节点组**。
“选择边缘节点”	边缘节点是您自己的边缘计算设备，用于运行边缘应用，处理您的数据，并安全、便捷地和云端应用进行协同。 单击选择边缘节点“添加”，在弹出的“添加节点”对话框中选择节点。选择您已创建的节点后，单击“确定”。

4. 完成参数填写后，根据界面提示完成边缘服务的部署。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。

   您可以前往边缘服务列表，查看边缘服务的基本情况。在边缘服务列表中，刚部署的服务“状态”为“部署中”，当边缘服务的“状态”变为“运行中”时，表示服务部署完成。

部署边缘服务（Atlas 500）

如果您纳管至IEF的设备是Atlas 500智能小站，则需要将训练好的模型部署至Atlas 500设备中。在开始操作之前，您需要了解如下几点要求。

- 对AI应用的要求：仅支持om模型或tflite模型，即支持部署在Ascend或ARM资源的模型。针对不满足格式的模型，必须经过模型转换操作，将模型转换成对应格式。模型转换操作和限制，请参见**压缩和转换模型**。
- 关于固件升级：如果您使用AI Gallery中新版预置算法训练得到的模型。当前此算法仅适配C32固件,不支持低级版本和C7X版本。因此将此模型部署至Atlas 500设备时，需下载并升级固件，请参考《Atlas500产品C32固件升级操作指导》升级Atlas 500设备。如果您使用部署的模型适配Atlas 500原有的固件，则无需升级固件。
- 固件下载升级仅适用于Atlas 500。
- 对于预置算法训练所得的模型，要求使用AI Gallery中的预置算法，且算法支持Ascend 310进行推理。

将AI应用部署至Atlas 500，请参考如下步骤进行操作。

1. 登录ModelArts管理控制台，在左侧导航栏中选择“部署上线>边缘服务”，默认进入“边缘服务”列表。
2. 在边缘服务列表中，单击左上角“部署”，进入“部署”页面。
3. 在部署页面，填写边缘服务相关参数，然后单击“下一步”。
   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，建议根据实际业务填写有意义的名称。
   b. 填写边缘服务参数，详情请参见**表5-2**。

表 5-2 部署至 Atlas 500 的参数说明

参数名称	说明
“AI应用来源”	根据您的实际情况选择“我的AI应用”或者“我的订阅”。
“选择AI应用及版本”	从下拉列表中选择可用的AI应用及版本。 **说明** 选择的模型是om或tflite格式的，即经过模型转换，然后使用“ARM-Ascend模板”导入至ModelArts创建为AI应用。
“计算节点规格”	选择符合要求的AI应用后，计算节点规格默认支持如下两种： ● ARM：3 核 3 GiB Ascend: 1 * Ascend 310 ● 自定义规格：可自行设置CPU、内存和Ascend个数。由于Atlas 500只有1个Ascend，设置为Ascend后，数量需设置为1。
“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。
“部署方式”	选择“节点”。

参数名称	说明
“选择边缘节点”	边缘节点是您自己的边缘计算设备，用于运行边缘应用，处理您的数据，并安全、便捷地和云端应用进行协同。 单击选择边缘节点右侧的“添加”，在弹出的“添加节点”对话框中，选择在IEF中纳管的Atlas 500节点，然后单击“确定”。 ModelArts系统会进行自动识别和匹配，如果纳管的设备未升级至符合要求的固件，则需根据界面提示完成C32固件升级。反之，则不需要进行固件升级。

图 5-1 选择 AI 应用及边缘节点

4. （可选）Atlas 500设备升级C32固件。
   a. 如图5-1提示，在节点列表下方单击“升级C32固件”，在弹出的对话框中，仔细阅读升级说明，勾选“我已阅读并同意以上内容”，然后单击“下载”，将固件版本及升级指导下载至本地。文件名称为“atlas500_C32_Firmware.zip”。
   b. 解压“atlas500_C32_Firmware.zip”文件，打开Atlas500产品C32固件升级操作指导.doc文件，根据指导完成Atlas 500的固件升级操作。
   c. 待Atlas 500升级后，重新部署边缘服务。

      刷新ModelArts管理控制台页面，根据1~3步骤，重新填写部署边缘服务的信息，此时选择升级后的Atlas 500，则不会再出现升级提示。
5. 完成参数填写后，单击“立即创建”，完成边缘服务的部署。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。

   您可以前往边缘服务列表，查看边缘服务的基本情况。在边缘服务列表中，刚部署的服务“状态”为“部署中”，当边缘服务的“状态”变为“运行中”时，表示服务部署完成。部署完成后，您可以登录Atlas 500查看部署完成的应用。

5.2 访问边缘服务

访问边缘服务

当边缘服务和边缘节点的状态都处于“运行中”状态，表示边缘服务已在边缘节点成功部署。

您可以通过以下两种方式，在能够访问到边缘节点的网络环境中，对部署在边缘节点上的边缘服务发起预测请求。

- **方式一：使用图形界面的软件进行预测（以Postman为例）**
- **方式二：使用curl命令发送预测请求**

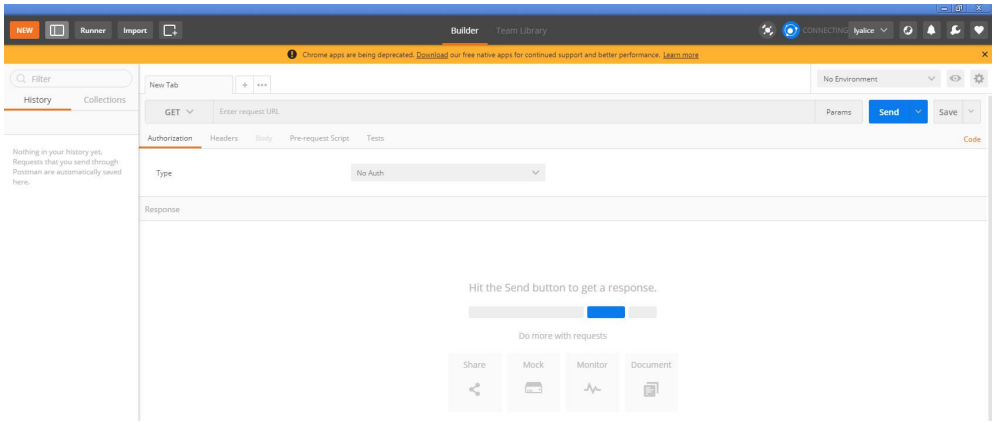
说明

通过方式一和方式二，边缘服务无需安全认证即可访问。如果用户需要设置安全认证，请选择使用自定义镜像创建的AI应用，具体操作请参考**官方指导**。在制作自定义镜像时，可参考**Flask文档**使用Flask设计RESTful的认证。

方式一：使用图形界面的软件进行预测（以 Postman 为例）

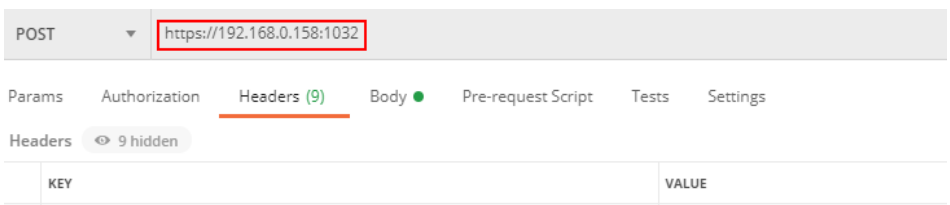
1. 下载**Postman**软件并安装，您可以直接在Chrome浏览器添加Postman扩展程序（也可使用其它支持发送post请求的软件）。
2. 打开Postman，如**图5-2**所示。

图 5-2 Postman 软件界面



3. 在Postman界面填写参数，以图像分类举例说明。
   - 选择POST任务，将某个边缘节点的调用地址（通过边缘服务详情界面-节点信息页签查看）复制到POST后面的方框。

图 5-3 POST 参数填写

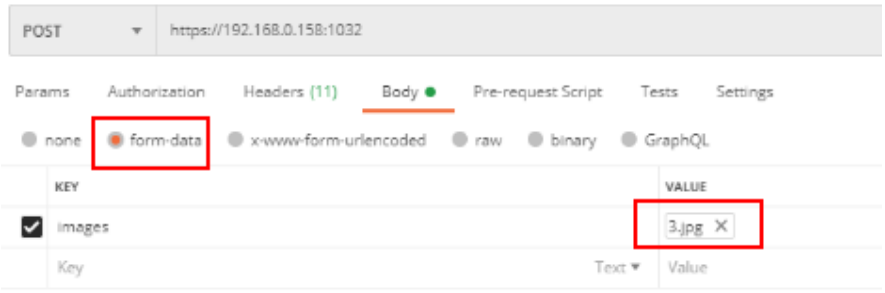


- 在Body页签，根据AI应用的输入参数不同，可分为“文件输入”或“文本输入”。

  - **文件输入**

    选择“form-data”。在“KEY”值填写AI应用的入参，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）。

    图 5-4 填写 Body 配置

  - **文本输入**

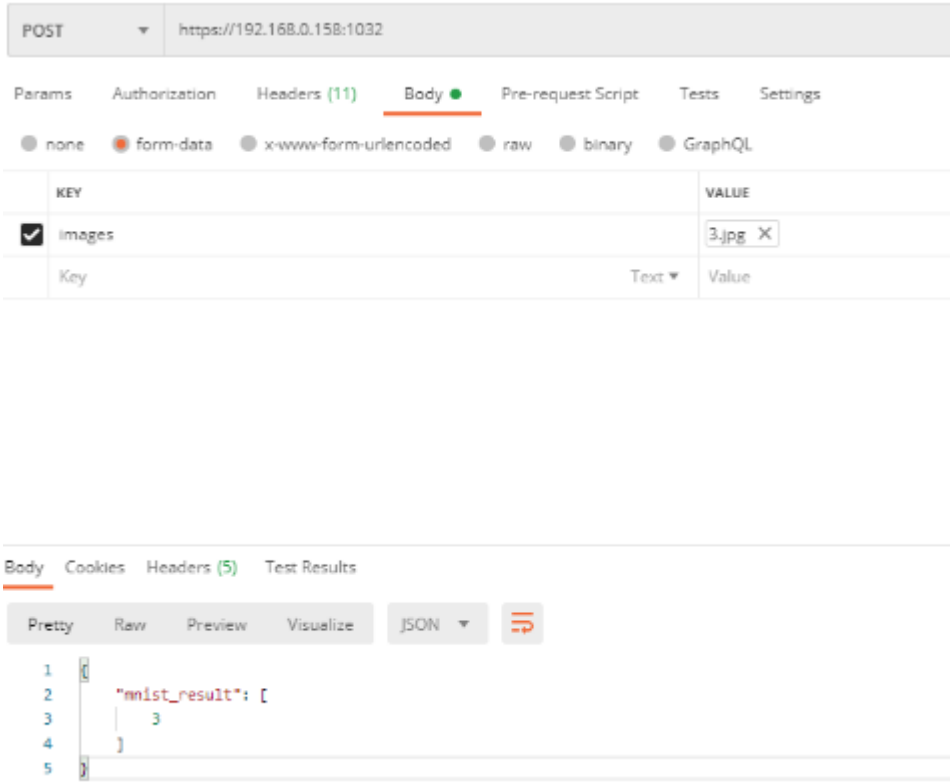
    选择“raw”，选择JSON(application/json)类型，在下方文本框中填写请求体，请求体样例如下。

    ```
    {
    "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”，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。

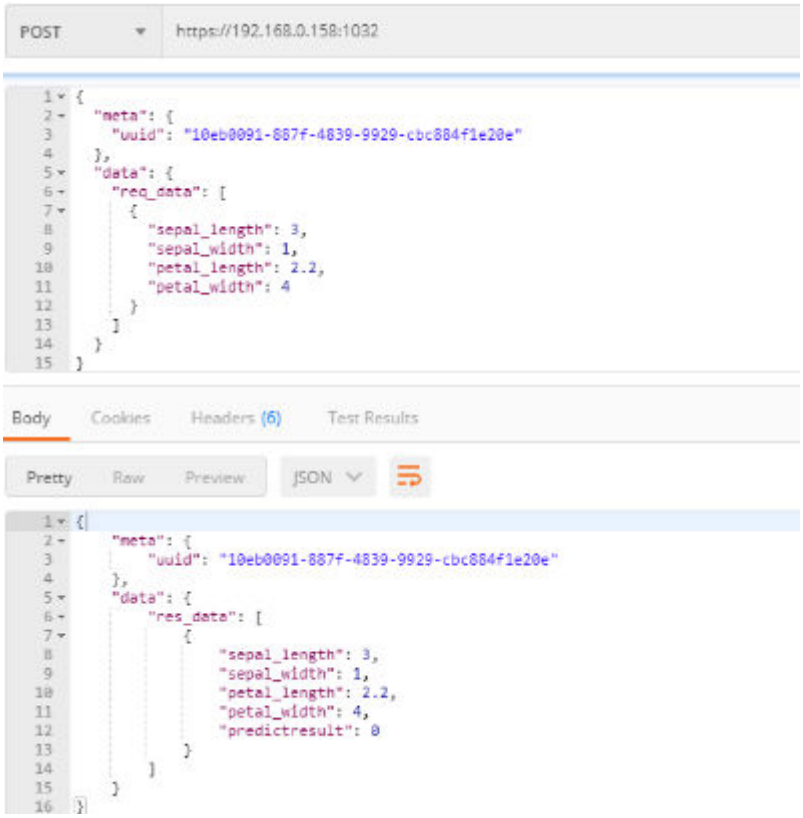
4. 参数填写完成，点击“Send”发送请求，结果会在Response下的对话框里显示。
   - 文件输入形式的预测结果样例如图5-5所示，返回结果的字段值根据不同AI应用可能有所不同。

图 5-5 边缘服务文件输入预测结果



- 文本输入形式的预测结果样例如图5-6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“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。

图 5-6 边缘服务文本输入预测结果



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

1. 文件输入

   curl -F '**images**=@**图片路径**'-X POST **边缘节点服务地址** -k

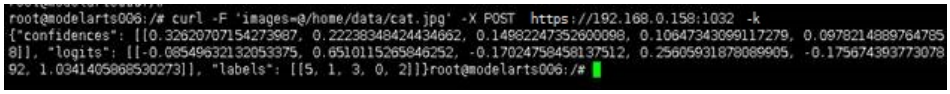
   - “-F”是指上传数据的是文件，本例中参数名为**images**，这个名字可以根据具体情况变化，@后面是图片的存储路径。
   - “POST”后面跟随的是边缘节点的调用地址。

   curl命令文件输入预测样例：

   ```
   curl -F 'images=@/home/data/cat.jpg' -X POST https://192.168.0.158:1032 -k
   ```

   预测结果如图5-7所示。

   图 5-7 curl 命令文件输入预测结果

2. 文本输入

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

```
"petal_width": 4
}
]
}
}' -X POST <边缘节点服务地址> -k
```

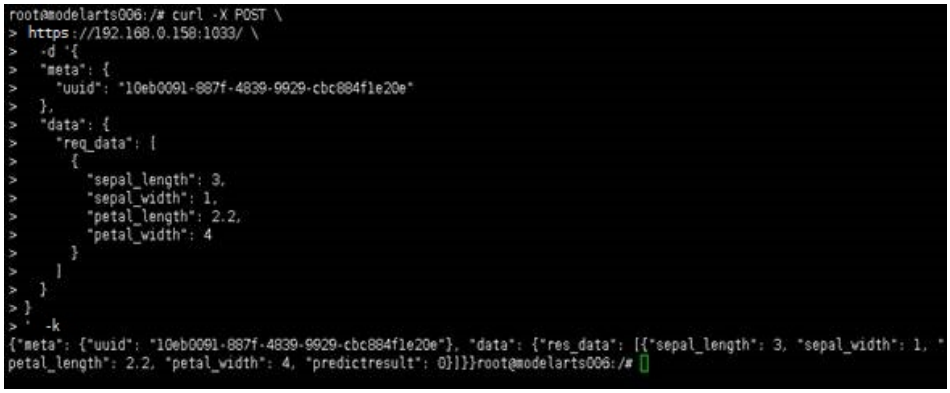
- “-d”是Body体的文本内容，如AI应用为文本输入，则需要用此参数。

curl命令文本输入预测样例：

```
curl -d '{
"meta": {
"uuid": "10eb0091-887f-4839-9929-cbc884f1e20e"
},
"data": {
"req_data": [
{
"sepal_length": 3,
"sepal_width": 1,
"petal_length": 2.2,
"petal_width": 4
}
]
}
}' -X POST https://192.168.0.158:1033 -k
```

预测结果如图5-8所示。

图 5-8 curl 命令文本输入预测结果



6 升级服务

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，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

已存在部署完成的服务。

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 6-1 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
从模板中选择元模型	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“部署上线”，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，然后单击“确定”完成修改。
   - 在线服务参数说明请参见**部署为在线服务**。
   - 批量服务参数说明请参见**部署为批量服务**。
   - 边缘服务参数说明请参见**部署为边缘服务**。

   说明

   “部署中”状态的服务无法进行修改。

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“部署上线”，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，然后单击“确定”完成修改。
   - 在线服务参数说明请参见**部署为在线服务**。
   - 批量服务参数说明请参见**部署为批量服务**。
   - 边缘服务参数说明请参见**部署为边缘服务**。

7 启动或停止服务

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“部署上线”，进入目标服务类型管理页面。您可以单击“操作”列的“启动”（在线服务在操作列选择“更多 > 启动”），启动服务。
- 登录ModelArts管理控制台，在左侧菜单栏中选择“部署上线”，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。

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“部署上线”，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“部署上线”，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。

8 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“部署上线”，进入目标服务类型管理页面。

- 在线服务，您可以单击“操作”列的“更多>删除”删除服务。
- 批量服务和边缘服务，

说明

删除操作无法恢复，请谨慎操作。

9 推理规范说明

9.1 模型包规范

9.1.1 模型包规范介绍

说明

为了更方便用户使用和清晰的获取文档帮助，《AI工程师用户指南》将按照AI开发流程调整拆分为多本文档，在一级导航中呈现，并从用户使用场景对各模块文档进行了优化和改进。“模型包规范”的内容已拆分至《**推理部署**》,《AI工程师用户指南》里的“模型包规范介绍”即将下线。

在AI应用管理创建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

模型包框架适用于单模型场景，若是多模型场景（例如含有多个权重文件）推荐使用自定义镜像方式。

模型包里面必需包含“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。

- 模型文件在不同模型包结构中要求不同，具体请参见**模型包示例**。
- 模型配置文件必需存在，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**。
- 模型推理代码文件是可选的。如果需要此文件，则文件名固定为“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（Python Import）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**。

模型包示例

- 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 可选：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
```

- MXNet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
|   |   |—— 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
|   |   |—— 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 可选：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
```

- Image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
|   |   |—— config.json 必选：模型配置文件（需要配置swr镜像地址），文件名称固定为config.json, 只允许放置一个
```

- pyspark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
|   |   |—— spark_model 必选: 模型文件夹，包含pyspark保存的模型内容
|   |   |—— 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 可选：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
```

- Caffe模型包结构

  用户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
```



```
| | |—— 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
| | |—— 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 可选：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
```

- XGBoost模型包结构

  用户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
| | |—— *.m必选: 模型文件，命名以.m结尾
| | |—— 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 可选：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
```

- Scikit_Learn模型包结构

  用户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
| | |—— *.m必选: 模型文件，命名以.m结尾
| | |—— 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 可选：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
```

9.1.2 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。模型配置文件描述模型用途、模型计算框架、模型精度、推理代码依赖包以及模型对外API接口。

配置文件格式说明

配置文件为JSON格式，参数说明如表9-1所示。

表 9-1 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。
model_type	是	String	模型AI引擎，表明模型使用的计算框架，可选的框架有TensorFlow、MXNet、Caffe、MindSpore、Image。 其中，Image并不是一个常用AI框架，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。

参数	是否必选	参数类型	描述
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及其对应开发环境。目前支持的运行环境列表请参见**支持的常用引擎及其Runtime**。 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**支持的常用引擎及其Runtime**中每个runtime的说明信息。
swr_location	否	String	SWR镜像地址。 • 如果您使用“从容器镜像中选择”的方式导入自定义镜像元模型，“swr_location”参数无需配置。 • 如果您使用“从对象存储服务(OBS)中选择”的方式导入自定义镜像元模型（不推荐该方式），且model_type设置为Image时，“swr_location”参数必填。“swr_location”为docker镜像在SWR上的地址，表示直接使用SWR的docker镜像发布模型。
metrics	否	object数据结构	模型的精度信息，包括平均数、召回率、精确率、准确率，metrics object数据结构说明如**表9-2**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9-3**所示。 • “model_type”为“Image”时，即自定义镜像的模型场景，“apis”可根据镜像实际对外暴露的请求路径在“apis”中声明不同路径的API。 • “model_type”不为“Image”时，“apis”只能声明一个请求路径为“/”的API，因为系统预置的AI引擎仅暴露一个请求路径为“/”的推理接口。

参数	是否必选	参数类型	描述
dependencies	否	dependency结构数组	表示模型推理代码需要依赖的包，为结构体数据。 模型开发者需要提供包名、安装方式、版本约束。目前只支持pip安装方式。dependency结构数组说明如**表9-6**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ist格式填入，适用于安装包存在先后依赖关系。示例请参考**使用OBS或者容器镜像导入模型时安装包存在前后依赖关系，模型配置文件应该如何说明?**
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，那么必需提供健康检查的接口供ModelArts调用。health数据结构如**表9-8**所示。

表 9-2 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 9-3 api 数据结构说明

参数	是否必选	参数类型	描述
protocol	否	String	请求协议。填写http或者https均可，请根据您的自定义镜像填写此参数。如果您使用OBS导入元模型，默认协议为https。其他字段的详细指导可参考**目标检测模型配置文件示例**解读。

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9-4所示。
response	否	Object	响应体，response结构说明如表9-5所示。

表 9-4 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 9-5 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 9-6 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9-7所示。

表 9-7 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 9-8 health 数据结构说明

参数	是否必选	参数类型	描述
url	是	String	健康检查接口请求路径。
protocol	否	String	健康检查接口请求协议，当前仅支持http。

参数	是否必选	参数类型	描述
initial_delay_seconds	否	String	实例启动后，延迟initial_delay_seconds秒再执行健康检查。
timeout_seconds	否	String	健康检查超时时间

目标检测模型配置文件示例

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

- 模型输入

  key：images

  value：图片文件

- 模型输出

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

- 配置文件

```
{
    "model_type": "TensorFlow",
    "model_algorithm": "object_detection",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [{
        "protocol": "http",
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
```



```
                "images": {
                    "type": "file"
                }
            }
        }
    },
    "response": {
        "Content-type": "multipart/form-data",
        "data": {
            "type": "object",
            "properties": {
                "detection_classes": {
                    "type": "array",
                    "items": [{
                        "type": "string"
                    }]
                },
                "detection_boxes": {
                    "type": "array",
                    "items": [{
                        "type": "array",
                        "minItems": 4,
                        "maxItems": 4,
                        "items": [{
                            "type": "number"
                        }]
                    }]
                },
                "detection_scores": {
                    "type": "array",
                    "items": [{
                        "type": "number"
                    }]
                }
            }
        }
    }
  }],
  "dependencies": [{
      "installer": "pip",
      "packages": [{
            "restraint": "EXACT",
            "package_version": "1.15.0",
            "package_name": "numpy"
        },
        {
            "restraint": "EXACT",
            "package_version": "5.2.0",
            "package_name": "Pillow"
        }
    ]
  }]
}
```

图像分类模型配置文件示例

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

- 模型输入

  key：images

  value：图片文件

- 模型输出

  ```
  {
  ```



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

- 配置文件

```
{
    "model_type": "TensorFlow",
    "model_algorithm": "image_classification",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [{
        "protocol": "http",
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "multipart/form-data",
            "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"
        }
    ]
  }]
}
```

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

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



```
  "apis": [
    {
      "protocol": "http",
      "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": "multipart/form-data",
        "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": [{
        "protocol": "http",
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "multipart/form-data",
            "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XGBoost为例：

- 模型输入：

```
{
    "data": {
        "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：

```
{
    "data": {
        "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": [
  {
    "protocol": "http",
    "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": "applicaton/json",
      "data": {
        "type": "object",
        "properties": {
          "resp_data": {
            "type": "array",
            "items": [
              {
                "type": "object",
                "properties": {
                  "predict_result": {
                    "type": "number"
                  }
                }
              }
            ]
          }
        }
      }
    }
  }
]
}
```

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": [{
        "procotol": "http",
        "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": 0.0023493892851938493,
      "accuracy": 0.00746268656716417
  },
  "dependencies": [{
      "installer": "pip",
      "packages": [{
            "restraint": "EXACT",
            "package_version": "1.16.4",
            "package_name": "numpy"
        }
      ]
  }]
}
```

9.1.3 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。

推理代码编写指导

1. 所有的自定义Python代码必须继承自BaseService类，不同类型的模型父类导入语句如**表9-9**所示。

表 9-9 BaseService 类导入语句

模型类型	父类	导入语句
TensorFlow	TfServingBaseService	from model_service.tfserving_model_service import TfServingBaseService
MXNet	MXNetBaseService	from mms.model_service.mxnet_model_service import MXNetBaseService
PyTorch	PTServingBaseService	from model_service.pytorch_model_service import PTServingBaseService
Caffe	CaffeBaseService	from model_service.caffe_model_service import CaffeBaseService

2. 可以重写的方法有以下几种。

   表 9-10 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，以实现数据的API输入的预处理和推理结果输出的后处理。
   - 重写BaseService继承类的初始化方法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

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

- 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 mnist_service(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 user_Service(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**。然后通过如下所示示例代码，实现了“saved_model”格式模型的加载推理。

```
# -*- 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

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()
```

9.2 模型模板

9.2.1 模型模板简介

说明

为了更方便用户使用和清晰的获取文档帮助，《AI工程师用户指南》将按照AI开发流程调整拆分为多本文档，在一级导航中呈现，并从用户使用场景对各模块文档进行了优化和改进。“模型模板”的内容已拆分至《**推理部署**》,《AI工程师用户指南》里的“模型模板”即将下线。

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OBS导入元模型（使用模板）**。

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**

9.2.2 模板说明

9.2.2.1 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```


9.2.2.2 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```

9.2.2.3 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```

9.2.2.4 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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     可选，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。
```

9.2.2.5 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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     可选，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。
```

9.2.2.6 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```

9.2.2.7 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```

9.2.2.8 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```

9.2.2.9 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```

9.2.2.10 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```

9.2.2.11 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```

9.2.2.12 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。
- 模型推理代码文件不是必选文件，如果有，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  //可选，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   可选，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。
```

9.2.3 输入输出模式说明

9.2.3.1 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9-11**。

表 9-11 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

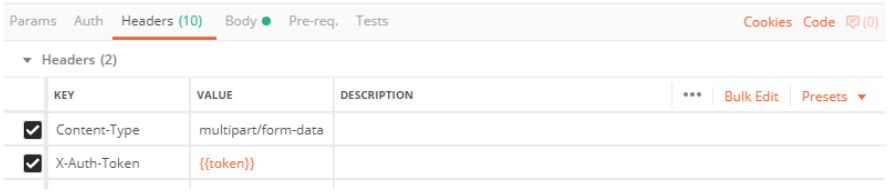
  在服务详情的“预测”页签，上传需要检测的图片，单击“预测”即可获取检测结果。

- 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：

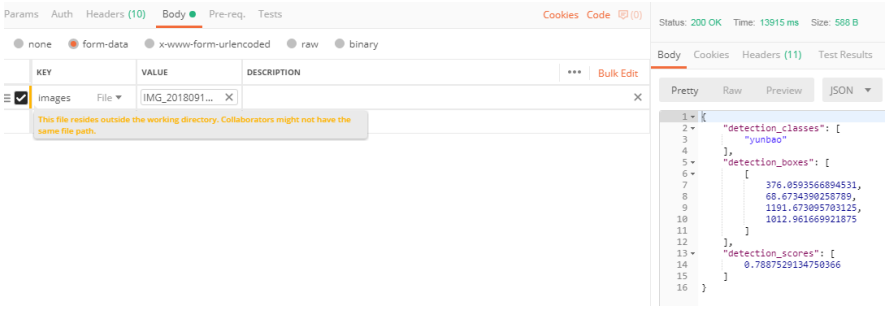
  - 选择“Headers”设置请求头部，“Content-Type”的值设为“multipart/form-data”，“X-Auth-Token”的值设为用户实际获取的token值。

    图 9-1 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 9-2 设置请求体

9.2.3.2 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 9-3 调用 REST 接口

9.2.3.3 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9-12**。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 9-12 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9-13**。

表 9-13 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9-14。

表 9-14 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9-15。

表 9-15 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

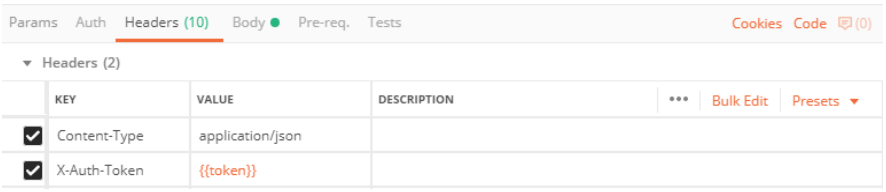
  在服务详情的“预测”页签，输入预测代码，单击“预测”即可获取检测结果。

- 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 9-4 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。

9.2.3.4 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。

9.3 自定义脚本代码示例

9.3.1 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）

推理代码需要继承自BaseService类，不同类型的模型父类导入语句如请参考**表9-9**。

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


class mnist_service(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

# inference调用父类处理逻辑

# 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
```

9.3.2 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

推理代码需要继承自BaseService类，不同类型的模型父类导入语句如请参考**表9-9**。

```
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 mnist_service(TfServingBaseService):

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

        # 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())
    }
```

9.3.3 PyTorch

训练模型

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

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

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

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

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

    test_loss /= len(test_loader.dataset)

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

device = torch.device("cpu")

batch_size=64

kwargs={}
```



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

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

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

保存模型

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

推理代码

推理代码需要继承自BaseService类，不同类型的模型父类导入语句如请参考**表9-9**。

```
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
```


9.3.4 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

推理代码需要继承自BaseService类，不同类型的模型父类导入语句如请参考**表9-9**。


```
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
```

9.3.5 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

推理代码需要继承自BaseService类，不同类型的模型父类导入语句如请参考**表9-9**。

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

9.3.6 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

推理代码需要继承自BaseService类，不同类型的模型父类导入语句如请参考**表9-9**。

```
# 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 user_Service(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
```

9.3.7 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

推理代码需要继承自BaseService类，不同类型的模型父类导入语句如请参考**表9-9**。

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

class user_Service(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
```

