ALIBABA CLOUD

阿里云

机器学习PAI

机器学习公共云合集

文档版本：20220117

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.新功能更新记录

1.1. 公告

本文为您介绍PAI的更新记录。

2020年12月15日（PAI-DSW服务升级）

为了更好地服务PAI-DSW的用户群体，2020年12月15日，PAI-DSW在华北2（北京）地域进行服务升级。PAI-DSW整体改动为三个版本：个人版、GPU特价版及探索者版。新版本与原V1入门版和V2专业版的对应关系如下：

- 原V1入门版的GPU预付费改动为GPU特价版。
- 原V1入门版的CPU后付费改动为个人版1.0待迁移。
- 原V1入门版的GPU后付费改动为个人版1.0待迁移。
- 原V2专业版改动为个人版2.0待迁移。

以上版本的实例使用不会受到影响，您仍然可以启动或关闭实例。个人版、GPU特价板及探索者版的详细说明请参见概述。非常感谢您的支持！

1.2. 文档最新动态

1.2.1. 2021年

本文为您介绍机器学习PAI版本文档2021年更新的最新动态，方便您及时了解PAI版本新增特性及功能变更。机器学习PAI的更多更新信息请参见产品首页的产品动态模块。

2021年11月更新记录

时间	特性	类别	描述	产品文档
2021.11.30	PAI-EAS API	新增功能	PAI-EAS上线API，您可以通过API管理在线服务及资源。	PAI-EAS API
2021.11.16	音视觉算法	新增功能	PAI-Designer新增视觉相关组件。	• 数据转tfrecord • 图像分类训练 • 图像检测训练 • 图像分割训练 • 视频分类训练 • 图像自监督训练 • 端到端的文字识别训练 • 通用视频预测 • 通用图像预测 • 文字检测识别组件
2021.11.16	关联角色	新增功能	新增PAI-EAS关联角色的权限说明及如何删除该角色。	PAI-EAS服务关联角色

2021年10月更新记录

时间	特性	类别	描述	产品文档
2021.10.19	基本概念	优化体验	从管理视角、AI开发视角及PAI产品模块三方面介绍涉及的基本概念。	基本概念
2021.10.19	资源管理	新增功能	在使用PAI功能之前，建议开通并购买用于训练的PAI-DLC和MaxCompute计算资源，以及用于推理的PAI-EAS资源组。资源仪表中可以查看您对每种资源组的持有情况。	资源管理
2021.10.19	AI工作空间（新版）	新增功能	工作空间是PAI的顶层概念，为企业和团队提供统一的计算资源管理及人员权限管理能力，为AI开发者提供支持团队协作的全流程开发工具以及AI资产管理能力。介绍工作空间相关的基础操作。	• 创建工作空间 • 管理成员
2021.10.19	准备工作	新增功能	介绍接入新版AI工作空间后，您使用PAI功能之前需要进行的准备工作。	• 开通并创建默认工作空间 • 开通并授权依赖的阿里云产品
2021.10.11	推理优化	新增功能	新增使用Blade优化基于TensorFlow的ResNet50模型的优化案例。	ResNet50优化案例：使用Blade优化基于TensorFlow的ResNet50模型

2021年9月更新记录

时间	特性	类别	描述	产品文档
2021.09.27	相似图像匹配与图像检索解决方案	新增功能	针对图像检索业务场景，PAI提供了端到端的相似图像匹配和图像检索解决方案。本文介绍如何基于未标注的数据构建图像自监督模型，助力您快速搭建相似图像匹配和图像检索业务系统，进而实现以图搜图。	相似图像匹配与图像检索解决方案
2021.09.23	推理优化	新增功能	新增使用Blade优化输入为Dynamic Shape的ResNet50。	Dynamic Shape优化案例：使用Blade优化输入为Dynamic Shape的ResNet50

时间	特性	类别	描述	产品文档
2021.09.16	推理优化	新增功能	新增使用Blade优化基于TensorFlow的BERT模型案例。	BERT模型优化案例：使用Blade优化基于TensorFlow的BERT模型
2021.09.15	风控解决方案	新增功能	在线上业务的内容生产过程中，为了及时识别其中的高风险内容，PAI提供了图像内容风控解决方案。	图像内容风控解决方案
2021.09.10	风控解决方案	新增功能	本文介绍如何基于人工智能算法，快速构建符合业务场景的风控模型，助力您快速识别高风险内容，进而对其进行拦截。	文本内容风控解决方案
2021.09.09	智能标注（iTAG）	新增功能	相比旧版智能标注，智能标注（iTAG）是一款智能化数据标注平台，支持图像、文本、视频、音频等多种数据类型的标注以及多模态的混合标注，提供了丰富的标注内容组件和题目组件，您可以直接使用平台预置的标注模板，也可以根据自己的场景自定义模板进行数据标注。	概述
2021.09.09	数据集管理（新版）	新增功能	数据集管理（新版）支持四种注册数据集的方式，您需要结合数据来源及应用场景选择合适的注册方式。	创建数据集

2021年8月更新记录

时间	特性	类别	描述	产品文档
2021.08.27	音视觉算法	新增功能	PAI-Studio 2.0新增ASR相关组件。	• EasyASR数据集构建 • EasyASR语音识别训练 • EasyASR语音分类训练 • EasyASR离线预测（MaxCompute） • EasyASR离线预测（DLC）

时间	特性	类别	描述	产品文档
2021.08.13	推理优化	新增功能	新增三篇使用Blade优化RetinaNet模型案例。	RetinaNet优化案例1：使用Blade优化RetinaNet（Detectron2）模型、RetinaNet优化案例2：结合Blade和Custom C++ Operator优化模型、RetinaNet优化案例3：结合Blade和TensorRT Plugin优化模型
2021.08.13	用户增长插件	新增功能	新增通过API创建短信模板、创建发送短信计划。	通过API创建短信模板、创建短信发送计划

2021年7月更新记录

时间	特性	类别	描述	产品文档
2021.7.25	通过Python SDK使用自定义组件	体验优化	调整通过Python SDK使用自定义组件的文档架构。	通过Python SDK使用自定义组件
2021.07.01	通过Python SDK创建PAI-DSW实例	新增功能	适用于需要对实例整体生命周期进行管理的开发者。例如，您需要将PAI-DSW系统中的资源整合到其他系统中，可以通过PAI-DSW的OpenAPI SDK实现。	通过Python SDK创建实例

2021年6月更新记录

时间	特性	类别	描述	产品文档
2021.06.30	AI行业插件	优化体验	根据控制台，刷新视觉模型插件和通用模型插件的操作指南。	AI行业插件
2021.06.25	模型预热	新增功能	为了解决模型初次请求耗时较长的问题，PAI-EAS提供了模型预热功能，使模型服务在上线之前得到预热，从而实现模型服务上线后即可进入正常服务状态。	模型预热
2021.06.17	Golang SDK使用说明	新增功能	推荐使用PAI-EAS提供的官方SDK进行服务调用，从而有效减少编写调用逻辑的时间并提高调用稳定性。本文介绍官方Golang SDK接口详情，并以常见类型的输入输出为例，提供了使用Golang SDK进行服务调用的完整程序示例。	Golang SDK使用说明

时间	特性	类别	描述	产品文档
2021.06.11	使用自定义镜像部署模型服务	新增功能	在业务开发过程中，通常存在复杂的环境依赖。如果通过Processor的形式进行模型服务部署，则需要将依赖的共享库统一打包到Processor中，无法通过yum install命令将依赖包直接安装至系统路径下，灵活性较差。因此PAI-EAS推出了使用自定义镜像部署模型服务。	使用自定义镜像部署模型服务
2021.06.02	通过Python SDK使用自定义组件	新增功能	PAI Pipeline Service中可以使用PAI提供的组件（Operator）处理数据和训练模型，也可以使用自定义组件处理具体任务。本文介绍如何在PAI Pipeline Service中，通过Python SDK创建并使用自定义组件。	通过Python SDK使用自定义组件

2021年5月更新记录

时间	特性	类别	描述	产品文档
2021.05.27	通过Python SDK调用PAI Pipeline service	新增功能	PAI不仅支持通过可视化建模（PAI-Studio 2.0）拖拉拽的形式构建数据处理、数据验证及模型训练的机器学习工作流，而且支持使用Python SDK构建相同的工作流，并在PAI Pipeline Service中运行。	通过Python SDK调用PAI Pipeline Service
2021.05.26	服务调用SDK	新增功能	推荐使用PAI-EAS提供的官方SDK进行服务调用，从而有效减少编写调用逻辑的时间并提高调用稳定性。	Java SDK使用说明、Python SDK使用说明
2021.05.19	文到图检索模型	新增功能	文到图检索模型是基于自然场景训练的文本到图片检索模型，即为模型输入文本，模型返回与该文本相似的图片。部署模型后，您可以通过POST方式访问模型服务。	概述
2021.05.12	数据配置	优化体验	PAI-DLC支持添加对象存储OSS的数据集。	数据配置
2021.05.11	NLP模型压缩镜像	新增功能	PAI官方提供了NLP模型压缩镜像，助力您快捷地进行文本类模型的训练和优化。	使用NLP模型压缩镜像训练模型

2021年4月更新记录

时间	特性	类别	描述	产品文档
2021.04.30	PAI-DLC客户端工具	新增功能	您可以通过客户端工具管理PAI-DLC中的任务和数据，提供了用户认证、创建、删除、提交、停止及查询系列命令。	命令列表
2021.04.25	AI工作空间	新增功能	AI工作空间可以统一管理PAI相关的计算资源和人员权限。例如，管理成员、为成员添加角色权限及添加计算资源。	AI工作空间（旧版）
2021.04.25	AI资产管理	新增功能	该模块统一管理PAI相关的数据集、算法、模型及镜像。	创建数据集
2021.04.07	PAI-DLC公共镜像列表	优化体验	新增镜像的版本详情。	公共镜像列表
2021.04.06	搭建NLP智能客服	新增功能	针对App内大量问题咨询场景，PAI提供搭建NLP智能客服的解决方案。本文介绍如何基于BERT模型，快速构建符合业务的NLP文本分类模型，从而实现智能客服QA平台。	NLP意图智能识别解决方案

2021年3月更新记录

时间	特性	类别	描述	产品文档
2021.03.18	PAI-EAS模型在线服务	体验优化	通过PAI-EAS，您可以将模型快速部署为RESTful API，再通过HTTP请求的方式调用该服务。	概述
2021.03.11	场景分类模型	新增功能	该模型能够识别图片内容反映的各种室内外场景，例如天空、沙滩、蓝天、厨房及音乐厅等。介绍该模型的输入格式、输出格式及测试示例。	场景分类模型
2021.03.04	图像智能处理类模型	新增功能	新增电商直播中文语音识别模型、中文语音检测模型及背景音乐检测模型，介绍该模型的输入格式、输出格式及测试示例。	语音智能处理类模型
2021.03.03	自然语言处理（NLP）类模型	新增功能	新增通用情绪识别模型、通用文本匹配模型、通用问答匹配模型、BERT英文文本向量化模型，介绍该模型的输入格式、输出格式及测试示例。	自然语言处理（NLP）类模型
2021.03.02	预置Processor使用说明	新增功能	新增TensorFlow1.15 Processor和PyTorch1.6 Processor，介绍它们的使用方法。	TensorFlow1.15 Processor（内置PAI-Blade敏捷版优化引擎）

2021年2月更新记录

时间	特性	类别	描述	产品文档
2021.02.26	PAI-Blade模型推理（敏捷版）	新增功能	PAI-Blade敏捷版结合了多种优化技术，对训练完成的模型进行优化，从而达到最优的推理性能。同时，PAI-Blade敏捷版提供的C++ SDK可以部署优化后的模型推理，帮助您快速的将模型应用于生产。	概述
2021.02.20	EasyTransfer/EasyASR	新增功能	介绍如何通过PAI-DSW使用EasyTransfer或EasyASR算法库。	使用EasyTransfer进行文本分类使用EasyASR进行语音识别
2021.02.18	人脸模型	新增功能	新增人脸属性模型和人脸检索模型。	人脸属性模型、人脸检索模型
2021.02.05	图像智能处理类模型	新增功能	新增前景分割模型，介绍该模型的输入格式、输出格式及测试示例。	前景分割模型
2021.02.05	多模态智能处理类模型	新增功能	新增多模态智能处理相关的模型，介绍该模型的输入格式、输出格式及测试示例。	图片文本匹配模型
2021.02.05	自然语言处理（NLP）类模型	新增功能	新增通用情感分析模型和电商评论情感分析模型，介绍该模型的输入格式、输出格式及测试示例。	自然语言处理（NLP）类模型

2021年1月更新记录

时间	特性	类别	描述	产品文档
2021.01.14	视频智能处理类模型	新增功能	新增通用视频分类和视频集锦生成模型，介绍该模型的输入格式、输出格式及测试示例。	视频智能处理类模型
2021.01.11	预置Processor使用说明	新增功能	新增EasyNLP Processor、EasyVision Processor及MediaFlow Processor。	EasyNLP Processor
2021.01.05	图像智能处理类模型	新增功能	新增通用OCR模型和人脸相似度比对模型，介绍该模型的输入格式、输出格式及测试示例。	通用OCR模型

1.2.2. 2020年

本文为您介绍机器学习PAI版本文档2020年更新的最新动态，方便您及时了解PAI版本新增特性及功能变更。机器学习PAI的更多更新信息请参见产品首页的产品动态模块。

2020年12月更新记录

时间	特性	类别	描述	产品文档
2020.12.30	PAI-DSW创建实例	优化体验	使用PAI-DSW进行Notebook建模前，您需要创建PAI-DSW实例。本文介绍如何创建PAI-DSW实例。	创建实例
2020.12.30	使用PAI-DSW开发环境	优化体验	介绍新版PAI-DSW开发环境，包括界面介绍、使用预置案例及管理第三方库。	使用开发环境
2020.12.22	自然语言处理（NLP）类模型	新增功能	新增文本内容反垃圾模型，介绍该模型的输入格式、输出格式及测试示例。	文本内容反垃圾模型
2020.12.17	命令使用说明	优化体验	介绍通过PAI-EAS客户端工具管理服务及资源组的常用命令。	命令使用说明
2020.12.15	离线预测通用说明	新增功能	介绍离线预测通用的输入数据格式及PAI命令参数。	离线预测通用说明
2020.12.15	图像物体检测	新增功能	介绍如何通过PAI-EasyVision使用训练好的模型进行离线物体检测。	PAI图像物体检测
2020.12.15	PAI-DSW新版概述	新增功能	介绍PAI-DSW个人版、GPU特价版及探索者版的功能特点、实例规格及可用区。	概述
2020.12.14	Whale分布式深度学习框架	新增功能	Whale可以帮助您便捷地进行分布式并行训练，支持各种并行策略混合训练，同时提供多种通信优化方法。介绍使用Whale实现分布式并行策略的经典案例、Whale API及超参配置。	快速开始

2020年11月更新记录

时间	特性	类别	描述	产品文档
2020.11.27	使用EasyRec构建推荐模型	新增功能	以读取MaxCompute表数据为例，介绍如何使用EasyRec进行模型训练、配置任务例行化及部署模型。	使用EasyRec构建推荐模型

2020年10月更新记录

时间	特性	类别	描述	产品文档
2020.10.26	PAI-Studio可视化建模	体验优化	介绍PAI-Studio提供的机器学习算法组件、深度学习框架、AutoML及周期性调度。	PAI-Studio可视化建模

时间	特性	类别	描述	产品文档
2020.10.14	使用PAI-EasyVision进行目标检测	新增功能	PAI-EasyVision（视觉智能增强算法包）提供多种模型的训练及预测功能，旨在帮助计算机视觉应用开发者方便快捷地构建视觉模型并应用于生产。本文以目标检测为例，介绍如何在PAI-DSW中使用PAI-EasyVision。	使用EasyVision进行目标检测
2020.10.14	使用WebIDE在线调试代码	新增功能	本文以排查PAI-DSW中提供的Sample Notebook问题为例，介绍如何通过PAI-DSW中的WebIDE，在线调试Notebook中运行的Python代码。	使用WebIDE在线调试代码

2020年9月更新记录

时间	特性	类别	描述	产品文档
2020.09.29	PAI-EAS模型在线服务	体验优化	通过PAI-EAS，您可以将模型快速部署为RESTful API，再通过HTTP请求的方式调用该服务。PAI-EAS提供的弹性扩缩和蓝绿部署等功能，可以支撑您以较低的资源成本获取高并发且稳定的在线算法模型服务。	PAI-EAS模型在线服务
2020.09.29	数加智能生态市场	体验优化	数加智能生态市场是一个大数据与AI领域的“淘宝”交易平台，旨在促进大数据与AI技术产品的开发创新与应用。不仅可以帮助开发者基于DataWorks和PAI开发应用，并将应用售卖给用户，而且可以帮助有业务需求的客户在市场中找到解决自己问题的方案。	数加智能生态市场
2020.09.28	常见问题	体验优化	介绍如何解决使用PAI过程中经常遇到的问题。	常见问题
2020.09.27	DeepFM算法	新增功能	介绍如何使用TensorFlow实现分布式DeepFM算法。	使用TensorFlow实现分布式DeepFM算法
2020.09.27	标注模板	新增功能	智能标注新增文本类和视频类标注模板，并将图像类的单标签和多标签图像分类标注模板合并为图像分类标注模板。	标注模板
2020.09.16	PAI-ModelHub公共模型仓库	新增功能	介绍PAI-ModelHub提供的图像智能类和NLP类模型。	PAI-ModelHub公共模型仓库

2020年8月更新记录

时间	特性	类别	描述	产品文档
2020.08.27	基于分箱组件实现连续特征离散化	新增功能	介绍如何使用分箱组件进行连续特征离散化。	基于分箱组件实现连续特征离散化
2020.08.11	定时自动部署模型服务	新增功能	介绍如何搭配使用PAI-EAS和DateWorks，进行模型服务定时部署。	定时自动部署模型服务
2020.08.11	PAI-DSW Notebook建模	优化体验	介绍PAI-DSW的开发环境、使用说明及经典案例。	概述

2020年7月更新记录

时间	特性	类别	描述	产品文档
2020.07.13	FM算法实现推荐模型	体验优化	介绍如何通过FM算法，快速构建推荐模型。	FM算法实现推荐模型
2020.07.10	TableRecordDataset	新增功能	介绍如何使用TableRecordDataset接口按照行读取MaxComepute表数据并构建数据流。	TableRecordDataset
2020.07.10	AdagradDecay Optimizer	新增功能	介绍如何使用AdagradDecay Optimizer进行超大规模训练。	AdagradDecay Optimizer
2020.07.10	EmbeddingVariable	新增功能	使用EmbeddingVariable进行超大规模训练，不仅可以保证模型特征无损，而且可以节约内存资源。	EmbeddingVariable
2020.07.10	分布式通信框架gRPC++	新增功能	介绍如何使用分布式通信框架gRPC++进行分布式训练。	分布式通信框架gRPC++
2020.07.10	TableWriter API	新增功能	介绍如何使用TableWriter API对MaxComepute表进行读写。	TableWriter API
2020.07.10	分布式训练框架StarServer	新增功能	介绍如何使用分布式训练框架StarServer进行分布式训练。	分布式训练框架StarServer

2020年6月更新记录

时间	特性	类别	描述	产品文档
2020.06.28	PAI-AutoLearning自动学习	体验优化	介绍如何进行通用模型和视觉模型训练。	PAI-AutoLearning
2020.06.24	产品定价	体验优化	介绍如何购买PAI及计费规则。	产品定价
2020.06.19	产品简介	体验优化	介绍产品架构、产品优势、产品基本概念及地域和可用区。	产品简介

时间	特性	类别	描述	产品文档
2020.06.19	快速入门	体验优化	以PAI-Studio为例，介绍如何快速构建一个完整的实验。	快速入门
2020.06.18	PAI-DLC云原生深度学习训练平台	体验优化	介绍使用PAI-DLC进行深度学习的准备工作及如何管理集群和深度学习任务。	PAI-DLC
2020.06.17	PAI数据准备	体验优化	介绍如何注册数据集及标注数据。	智能标注

1.3. 版本历史

本文为您及时同步PAI的版本发布信息。

2021年3月

功能名称	功能描述	发布时间	发布地域	相关文档
自然语言处理（NLP）类模型	ModelHub中新增通用情绪识别模型、通用文本匹配模型、通用问答匹配模型、BERT英文文本向量化模型。	2021年3月03日	• 华北2（北京） • 华东2（上海） • 华东1（杭州） • 华南1（深圳） • 中国（香港）	自然语言处理（NLP）类模型
图像智能处理类模型	ModelHub中新增电商直播中文语音识别模型、中文语音检测模型及背景音乐检测模型。	2021年3月04日	图像智能处理类模型	

2021年2月

功能名称	功能描述	发布时间	发布地域	相关文档
人脸模型	新增人脸属性模型和人脸检索模型。人脸检索服务支持图像数据库的搭建和检索，提供数据库层接口和数据库层接口。	2021年2月18日	• 华北2（北京） • 华东2（上海） • 华东1（杭州） • 华南1（深圳） • 中国（香港）	人脸属性模型
PAI-Blade模型推理（敏捷版）	PAI-Blade敏捷版结合了多种优化技术，对训练完成的模型进行优化，从而达到最优的推理性能。同时，PAI-Blade敏捷版提供的C++ SDK可以部署优化后的模型推理，帮助您快速的将模型应用于生产。	2021年2月26日	不涉及	概述

2021年1月

功能名称	功能描述	发布时间	发布地域	相关文档
视频集锦生成模型	PAI-ModelHub提供的视频集锦生成模型，可以选取视频中的精彩集锦，生成一个5s的视频片段。	2021年01月14日	华东2（上海）	视频集锦生成
通用视频分类	PAI-ModelHub提供了通用视频分类模型，该模型基于UCF101数据集，采用ResNet3D架构。	2021年01月14日	• 华北2（北京） • 华东2（上海） • 华东1（杭州） • 华南1（深圳）	通用视频分类
MediaFlow Processor	PAI-EAS提供的MediaFlow Processor是通用的编排引擎，可以进行视频、音频及图像分析处理。	2021年01月11日	• 华北2（北京） • 华东2（上海） • 华东1（杭州） • 华南1（深圳）	MediaFlow Processor
EasyVision Processor	PAI-EAS提供的EasyVision Processor可以加载EasyVision框架训练得到的深度学习模型。	2021年01月11日	与PAI-EAS支持的地域相同，详情请参见PAI-EAS。	EasyVision Processor
EasyNLP Processor	PAI-EAS提供的EasyNLP Processor可以加载EasyTransfer框架训练得到的深度学习NLP模型。	2021年01月11日	• 华北2（北京） • 华东2（上海） • 华东1（杭州） • 华南1（深圳）	EasyNLP Processor
人脸相似度比对模型	PAI-ModelHub提供了人脸相似度比对模型，该模型采用ResNet50。	2021年01月05日	与PAI-EAS支持的地域相同，详情请参见PAI-EAS。	人脸相似度比对模型
通用OCR模型	PAI-ModelHub提供了通用OCR模型，该模型采用PAI自研的端到端OCR模型，具有文本检测、识别的功能。	2021年01月05日	与PAI-EAS支持的地域相同，详情请参见PAI-EAS。	通用OCR模型

2020年12月

功能名称	功能描述	发布时间	发布地域	相关文档
PAI-DSW	PAI-DSW进行服务升级，整体改动为三个版本：个人版、GPU特价版及探索者版。	2020年12月15日	• 华北2（北京） • 华东2（上海） • 华东1（杭州） • 华南1（深圳）	概述
文本内容反垃圾模型	PAI-ModelHub提供了BERT分类模型进行文本内容反垃圾，输入为单句。该模型的结构与新闻分类模型相同。	2020年12月22日		文本内容反垃圾模型

功能名称	功能描述	发布时间	发布地域	相关文档
Whale分布式深度学习框架	Whale可以帮助您便捷地进行分布式并行训练，支持各种并行策略混合训练，同时提供多种通信优化方法。	2020年12月14日		快速开始

2.大数据和AI体验教程

初级课程 | 进阶课程 | 实战课程 | 客户证言 | 大咖视角 | 技术解析

实操演示课程
PAI视觉算法组件介绍-图像分类
- 初级课程（最新课程）
- 实操时长：19分钟22秒

开始学习

实操演示课程
PAI视觉算法组件介绍-物体检测
- 初级课程（最新课程）
- 实操时长：26分钟40秒

开始学习

实操演示课程
如何在PAI-Studio上传自定义算法
- 初级课程（最新课程）
- 实操时长：1分钟52秒

开始学习

实操演示课程
如何使用PAI-EAS部署API服务
- 初级课程（最新课程）
- 实操时长：20分钟52秒

开始学习

实操演示课程
PAI的视觉模型平台插件
- 初级课程（最新课程）
- 实操时长：2分钟41秒

开始学习

实操演示课程
使用NAS提交单机PyTorch迁移学习任务
- 初级课程（最新课程）
- 实操时长：4分钟58秒

开始学习

实操演示课程
EASCMD客户端管理PAI-EAS服务
- 初级课程（最新课程）
- 实操时长：3分钟44秒

开始学习

实操演示课程
在PAI里进行镜像管理
- 初级课程（最新课程）
- 实操时长：2分钟18秒

开始学习

实操演示课程
如何将Studio模型部署到调度的任务
- 初级课程（最新课程）

实操演示课程
如何使用AutoML进行自动调参训练
- 初级课程（最新课程）

- 实操时长：2分钟26秒

开始学习

- 实操时长：2分钟35秒

开始学习

实操演示课程

如何查看logview并进行日志分析

- 初级课程（最新课程）
- 实操时长：1分钟52秒

开始学习

实操演示课程

如何在PAI-EAS定时自动部署模型服务

- 初级课程（最新课程）
- 实操时长：6分钟23秒

开始学习

实操演示课程

ModelHub公共模型仓库查看

- 初级课程（最新课程）
- 实操时长：5分钟13秒

开始学习

实操演示课程

如何部署PS-LR模型？

- 初级课程（最新课程）
- 实操时长：3分钟23秒

开始学习

实操演示课程

DLC深度学习训练平台

- 初级课程（最新课程）
- 实操时长：3分钟5秒

开始学习

实操演示课程

通过MaxCompute和OSS上传数据

- 初级课程（最新课程）
- 实操时长：3分钟7秒

开始学习

实操演示课程

在PAI-EAS中通过VPC高速直连调用

- 初级课程（最新课程）
- 实操时长：6分钟12秒

开始学习

实操演示课程

阿里云机器学习PAI-DSW

- 初级课程（最新课程）
- 实操时长：8分钟2秒

开始学习

实操演示课程

机器学习PAI快速入门与业务实战

- 初级课程（最新课程）
- 实操时长：12课时

开始学习

机器学习PAI平台的前世今生

PAI：一站式云原生AI平台

- 初级课程（最新课程）

正在限时免费邀测，欢迎尝鲜

重磅发布：PAI-DSW探索者版

- 初级课程

实操演示课程

如何上传训练数据

- 初级课程

3.准备工作

3.1. 开通并创建默认工作空间

工作空间是PAI的顶层概念，为团队提供统一的计算资源管理及人员权限管理能力，为AI开发者提供支持团队协作的全流程开发工具及AI资产管理能力。因此使用PAI功能之前，您需要开通并创建默认工作空间。

操作步骤

1. 登录PAI控制台。
2. 进入欢迎页面后，单击**免费开通并创建默认工作空间**。

3. 进入默认工作空间配置页面后，查看并了解默认工作空间的地域、计费方式。

   PAI自动设置地域为登录进入欢迎页面的地域，如果需要修改地域可在页面左上角切换地域后重新进入默认工作空间配置页面。

4. 配置其他产品开通和授权等操作。

i. 选择是否同时免费开通训练过程中必要的存储及训练资源，包括OSS按量付费、MaxCompute按量付费、DataWorks基础版。开通过程免费。

   如果您此前已开通过OSS、MaxCompute、DataWorks，则此处默认选中，无需修改，后续操作一致。

ii. 单击**角色授权**下面的**去授权**，根据界面提示为PAI需要的关联角色进行授权。

iii. 阅读了解服务协议后，勾选已阅读并同意服务协议。

iv. 单击**确认开通并创建默认工作空间**，完成默认工作空间的创建。

执行结果

开通并创建默认工作空间后，系统会完成以下事宜：

- 开通当前地域的PAI产品及选中同步开通的OSS、MaxCompute、DataWorks。
  您可以分别登录对应云产品的控制台，在开通地域下查看确认云产品是否成功开通。
- 自动创建一个默认工作空间。
  由于PAI的工作空间与DataWorks的工作空间在底层是打通的，因此您可以分别登录PAI控制台和DataWorks管理控制台后查看默认工作空间是否成功创建。
- 为默认工作空间关联开通好的MaxCompute后付费资源组和PAI-DLC公共资源组（按量付费），作为默认计算资源。
- 将当前阿里云账号下的全部RAM用户添加为拥有算法开发角色的工作空间成员。该配置为默认行为，建议创建完成后，进入默认工作空间对成员进行详细的角色配置，详情请参见修改成员角色。

创建完成默认工作空间后，如果您根据业务需求规划了多个工作空间，也可以再创建多个其他工作空间，创建操作请参见创建工作空间。

3.2. PAI-EAS服务关联角色

PAI-EAS通过服务关联角色AliyunServiceRoleForPaiEas获取其他云服务的访问权限，首次使用PAI-EAS时，阿里云账号需要为该角色授权。本文介绍AliyunServiceRoleForPaiEas角色拥有的访问权限及如何删除该角色。

背景信息

PAI-EAS服务关联角色AliyunServiceRoleForPaiEas是PAI-EAS在某些情况下，为了完成自身的某个功能，需要获取其他云服务的访问权限而提供的RAM角色。更多关于服务关联角色的信息请参见服务关联角色。

当PAI-EAS部分功能需要访问对象存储OSS、日志服务SLS、云服务器ECS及专有网络VPC云服务的资源时，您可以通过PAI-EAS服务关联角色AliyunServiceRoleForPaiEas获取访问权限。

AliyunServiceRoleForPaiEas权限说明

AliyunServiceRoleForPaiEas拥有以下云服务的访问权限：

- 对象存储OSS的访问权限

```
{
  "Action": [
    "oss:GetObject",
    "oss:PutObject",
    "oss:DeleteObject",
    "oss:ListParts",
    "oss:AbortMultipartUpload",
    "oss:ListObjects",
    "oss:ListBuckets",
    "oss:PutBucketCors",
    "oss:GetBucketCors",
    "oss:DeleteBucketCors"
  ],
  "Resource": "*",
  "Effect": "Allow"
}
```

- 日志服务SLS的访问权限

```
{
  "Action": [
    "log:CreateConfig",
    "log:GetConfig",
    "log:UpdateConfig",
    "log:DeleteConfig",
    "log:CreateMachineGroup",
    "log:GetMachineGroup",
    "log:DeleteMachineGroup",
    "log:ApplyConfigToGroup",
    "log:ListProject",
    "log:ListLogStores"
  ],
  "Resource": "*",
  "Effect": "Allow"
}
```

- 云服务器ECS的访问权限

```
{
  "Action": [
    "ecs:CreateNetworkInterface",
    "ecs:DeleteNetworkInterface",
    "ecs:DescribeNetworkInterfaces",
    "ecs:CreateNetworkInterfacePermission",
    "ecs:DescribeNetworkInterfacePermissions",
    "ecs:DeleteNetworkInterfacePermission",
    "ecs:DescribeSecurityGroups"
  ],
  "Resource": "*",
  "Effect": "Allow"
}
```

- 专有网络VPC的访问权限

```
{
  "Action": [
    "vpc:DescribeVSwitchAttributes",
    "vpc:DescribeVpcs",
    "vpc:DescribeVSwitches",
    "vpc:DescribeVpcAttribute"
  ],
  "Resource": "*",
  "Effect": "Allow"
}
```

删除AliyunServiceRoleForPaiEas

如果您已部署了PAI-EAS服务，出于安全或其他方面的考虑想要删除服务关联角色AliyunServiceRoleForPaiEas，则需要明确删除该角色的影响。删除AliyunServiceRoleForPaiEas后，您可能无法再更新服务版本、创建VPC高速通道或投递服务日志到SLS。

删除AliyunServiceRoleForPaiEas的操作步骤如下：

1. 登录RAM控制台。
2. 在左侧导航栏，选择**身份管理 > 角色**。
3. 在**角色**页面的搜索框中，输入AliyunServiceRoleForPaiEas，系统会自动搜索到名称为AliyunServiceRoleForPaiEas的RAM角色。
4. 单击目标角色**操作**列下的删除。

5. 在删除RAM角色的确认对话框，单击**确定**。

如果RAM用户无法删除AliyunServiceRoleForPaiEas，请参见为什么RAM用户无法自动创建或删除PAI-EAS服务关联角色AliyunServiceRoleForPaiEas？。

3.3. 开通并授权依赖的阿里云产品

您如果需要使用PAI的所有功能，则需要阿里云账号预先开通依赖的阿里云其他产品，并为RAM用户进行授权。本文介绍如何开通并授权依赖的阿里云产品。

操作步骤

1. 进入全部产品依赖页面
   i. 登录PAI控制台。
   ii. 在左侧导航栏，选择**资源管理 > 全部云产品依赖**。
2. 在**全部产品依赖**页面，您可以查看各子产品模块依赖的其它阿里云产品的开通和授权状态。PAI依赖的所有阿里云产品列表请参见下文的PAI依赖的其他阿里云产品。

   对于依赖的阿里云产品，您可以对其进行统一管理：
   - 对于未开通的产品，单击目标产品**操作**列下的**去开通**，即可跳转至该产品控制台，您可以根据提示完成开通。
   - 对于需要为PAI进行授权才可以使用的产品，单击目标产品**操作**列下的**一键授权**，即可根据提示完成授权。

PAI依赖的其他阿里云产品

从PAI的各子产品角度，汇总依赖的所有阿里云产品：

- Designer

云产品名称	依赖说明（使用场景）
MaxCompute	Designer中提供上百种基于MaxCompute框架实现的自研算法。
OSS	使用深度学习算法组件时，依赖OSS数据源。

- iTAG
  标注数据集的输入和输出依赖于OSS数据源。
- DSW
  文件持久化存储需要依赖NAS。
- DLC

云产品名称	依赖说明（使用场景）
NAS	文件持久化存储依赖NAS。
OSS	数据存储依赖OSS。

- EAS

云产品名称	依赖说明（使用场景）
API网关	通过API网关的公网调用。
OSS	读取OSS的模型文件。
SLS	配置日志写入到SLS。

云产品名称	依赖说明（使用场景）
VPC	VPC高速直连。
云监控	服务监控报警。

3.4. 授权

本文汇总为PAI各模块授权的方法。

为各模块授权

您可以分别参考以下链接，授予PAI各模块所需的权限：

- 为PAI-Studio授权
- 为PAI-DSW授权
- 为PAI-DLC授权
- 为PAI-EAS授权

3.5. 资源管理

推荐您在使用PAI功能之前开通并购买用于训练的PAI-DLC和MaxCompute计算资源，以及用于推理的PAI-EAS资源组。本文介绍三种资源详情。

您可以在**资源管理 > 资源仪表**中查看您对PAI-DLC、MaxCompute及PAI-EAS资源组的持有情况。以下详细介绍三种资源：

- PAI-DLC计算资源
  PAI-DLC（Deep Learning Containers）是基于阿里巴巴容器服务ACK（Alibaba Cloud Container Service for Kubernetes）的深度学习训练平台，为您提供灵活、稳定、易用和极致性能的深度学习训练环境。您可以在**资源仪表**页面购买并管理PAI-DLC资源组，详情请参见概述。
  开通PAI以后，系统默认会为您创建PAI-DLC公共资源组，提供多种型号的异构资源，您可以直接将该资源组关联至工作空间进行使用，仅当执行具体训练任务时才开始计费。
- MaxCompute计算资源
  MaxCompute是适用于数据分析场景的企业级SaaS（Software as a Service）模式云数据仓库，以Serverless架构提供快速、全托管的在线数据仓库服务，消除了传统数据平台在资源扩展性和弹性方面的限制，最小化运维投入，使您可以经济并高效地分析处理海量数据。
  您可以先购买并创建不同类型的MaxCompute资源组，然后从PAI进行模型开发，再提交至MaxCompute进行大规模分布式训练。推荐您使用按量付费资源组。关于MaxCompute的资源组类型及计费标准请参见规格概述。

  > 注意 MaxCompute开发版不支持PAI任务类型。

- PAI-EAS推理资源
  PAI-EAS推理资源暂未接入工作空间，您可以在PAI-EAS模块直接购买资源组并使用它进行模型服务部署。关于PAI-EAS资源组的详细信息请参见PAI-EAS资源组概述。

3.6. AI工作空间（旧版）

AI工作空间可以统一管理PAI相关的计算资源和人员权限。本文介绍如何创建并配置AI工作空间，包括如何管理成员、为成员添加角色权限及添加计算资源。

背景信息

AI工作空间是PAI产品体系的统一资源和权限的管理模块，您可以在该模块对PAI相关资源（例如DataWorks和MaxCompute）进行购买、升级或降配操作。此外，您也可以对不同的RAM用户设置不同的角色权限。

创建AI工作空间

1. 进入**AI工作空间列表**页面。
   i. 登录PAI控制台。
   ii. 在左侧导航栏，单击**AI工作空间**。
2. 在**AI工作空间列表**页面，单击**新建AI工作空间**。
3. 在**创建AI工作空间**页面，配置如下参数。

区域	参数	描述
基本信息	**工作空间名称**	3~23个字符，以字母开头，只能包含字母、下划线和数字。
	描述	建议结合实际业务进行描述，以区分不同的工作空间。
AI工作空间资源选择	**AI工作空间资源选择**	根据实际需要选择相应的计算资源，系统会自动进行是否已购买的相关校验。
实例配置	**实例配置**	单击**添加资源**，在**新建实例**对话框，配置如下参数，并单击**确定**： ○ **实例显示名称**：3~27个字符，以字母开头，只能包含字母、下划线和数字。 ○ **CPU**：仅支持**按量付费**。 ○ **GPU**：您可以不使用GPU资源，或通过按量付费使用GPU资源。 ○ **项目名称**：该名称需要在工作空间列表中唯一，建议命名相对复杂。

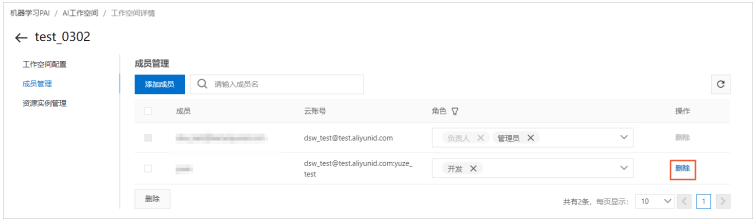
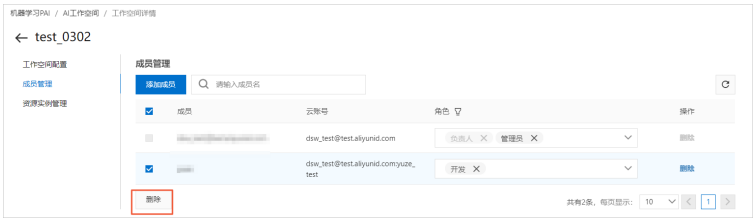
4. 单击**创建**。

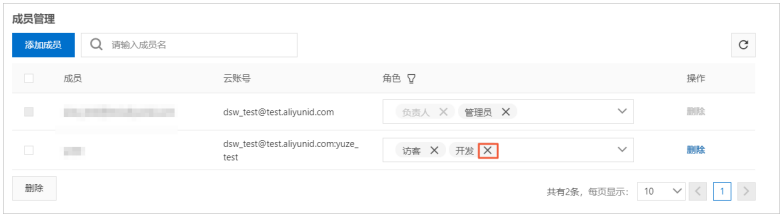
配置工作空间

您可以对工作空间中的成员和资源实例进行管理，包括添加成员、删除成员、修改成员角色、添加实例及查看实例详情。

1. 进入**AI工作空间列表**页面。
   i. 登录PAI控制台。
   ii. 在左侧导航栏，单击**AI工作空间**。
2. 在**AI工作空间列表**页面，单击目标工作空间**操作**列下的**工作空间配置**。
3. 在**工作空间详情**页面，您可以预览工作空间的资源和成员权限。
4. 单击**成员管理**，您可以进行以下操作。

操作	具体方法

操作	具体方法
添加成员	i. 在**成员管理**页面，单击**添加成员**。 ii. 在**添加成员**对话框的左侧**账号**区域，选中目标账号前面的复选框。 iii. 单击 图标，选中的目标账号即可显示在右侧的**已选子账号**区域。 iv. 选择为该成员赋予的**角色**（支持选择多个角色）。 不同**角色**的权限如下： ■ **负责人**：负责任命管理员。 ■ **管理员**：负责所有的增、删、改、查操作。 ■ **开发**：不能进行删除操作，只能进行修改和查看操作。如果该成员仅使用PAI-Studio，则建议为其配置**开发**角色。 ■ **访客**：只能进行查看操作。 v. 单击**确定**。
删除成员	您可以每次删除单个成员，或批量删除成员： **说明** 无法删除角色为**负责人**的成员。 ○ 删除单个成员： a. 在**成员管理**页面，单击目标成员**操作**列下的**删除**  b. 在**删除**对话框，单击**确定**。 ○ 批量删除成员： a. 在**成员管理**页面，单击目标成员前面的复选框。 b. 单击页面底部的**删除**。  c. 在**删除**对话框，单击**确定**。

操作	具体方法
修改角色	i. 在**成员管理**页面，单击目标成员**角色**列下的列表。 ii. 在角色列表中，单击某角色，如果该成员已经拥有了这个角色权限，则会为成员删除该角色。如果该成员没有这个角色权限，则会为成员添加该角色。 您也可以直接单击角色名后的×图标，即可为成员删除该角色。  **说明** ■ 每个成员至少要拥有一个角色，即您不能删除一个成员的所有角色。 ■ 不能删除**负责人**角色。

5. 单击**资源实例管理**，您可以进行以下操作。

操作	具体方法

操作	具体方法
添加实例	您可以通过**添加实例**添加MaxCompute或PAI-DLC计算资源： ○ 每个AI工作空间可以挂载多个MaxCompute计算资源。添加MaxCompute计算资源的方法如下： a. 在**资源实例管理**页面的**MaxCompute**页签，单击**添加实例**。 b. 在**添加实例**对话框，配置以下参数。 ■ **实例显示名称**：3~27个字符，以字母开头，只能包含字母、下划线（_）和数字。 ■ **CPU**：仅支持**按量付费**。 ■ **GPU**：您可以不使用GPU资源，或通过按量付费使用GPU资源。 ■ **项目名称**：该名称需要在工作空间列表中唯一，建议命名相对复杂。 c. 单击**确定**。 ○ 添加PAI-DLC计算资源的方法如下： **说明** AI工作空间目前只支持PAI-DLC全托管后付费实例，全托管后付费实例只能创建1项并且无法解绑。 a. 在**资源实例管理**页面的**DLC**页签，单击**添加实例**。 b. 在**添加实例**对话框，配置以下参数。 ■ **集群类型**：仅支持全托管的集群。 ■ **实例显示名称**：3~27个字符，以字母开头，只能包含字母、下划线（_）和数字。 c. 单击**确定**。
查看实例详情	○ 对于MaxCompute计算资源，您可以在**MaxCompute**页签，单击目标实例**操作**列下的**进入资源实例**，即可查看实例的详细信息。 ○ 对于PAI-DLC计算资源，您可以在**DLC**页签，单击实例操作列下的**集群控制台**，即可跳转至PAI-DLC集群控制台。
设置默认资源	当存在多个MaxCompute计算资源时，您可以在**MaxCompute**页签，单击目标实例**操作**列下的**设为默认**，将其中一个资源设置为默认的计算资源。

3.7. AI工作空间（新版）

3.7.1. 创建工作空间

工作空间是PAI的顶层概念，为企业和团队提供统一的计算资源管理及人员权限管理能力，为AI开发者提供支持团队协作的全流程开发工具以及AI资产管理能力。本文介绍如何创建工作空间。

前提条件

- 已开通PAI，详情请参见开通并创建默认工作空间。

- 已为创建工作空间的阿里云账号或RAM用户完成了AliyunDataWorksFullAccess授权。
  由于PAI的工作空间与DataWorks的工作空间在底层是打通的，因此操作创建PAI工作空间的账号需拥有AliyunDataWorksFullAccess才能成功创建工作空间，授权操作详情请参见云产品依赖与授权：Designer。

操作步骤

1. 登录PAI控制台。
2. 进入**新建工作空间**页面。

   您可以通过如下任意一种方式进入**新建工作空间**页面：
   - 在**概览**页面，单击**新建工作空间**。
   - a. 在左侧导航栏，单击**工作空间列表**。
     b. 在**工作空间列表**页面，单击**新建AI工作空间**。
3. 在**新建工作空间**的**基础信息**配置向导页面，配置各项参数，单击**下一步**。

参数	描述
工作空间名称	工作空间名称的长度需要在3~23个字符，以字母开头，且只能包含字母、下划线（_）和数字。 **说明** 如果界面提示工作空间名称已存在，但是在PAI的工作空间列表中没有同名工作空间，则可能是因为在DataWorks的工作空间中存在同名工作空间。由于PAI和DataWorks的工作空间在底层是打通的，因此您需要修改名称，以保障不会有同名工作空间。
工作空间描述	对创建的工作空间进行简单描述，以区分不同的工作空间。
工作空间成员	单击**添加成员**，您可以为该工作空间添加相关成员及配置角色，详情请参见配置添加成员参数。 左侧导航栏，您可以单击**角色与权限列表**，查看各角色与权限点的映射关系，从而为成员添加合适的角色。

4. 在**新建工作空间**的**关联资源**配置向导页面，配置各项参数，单击**创建**。

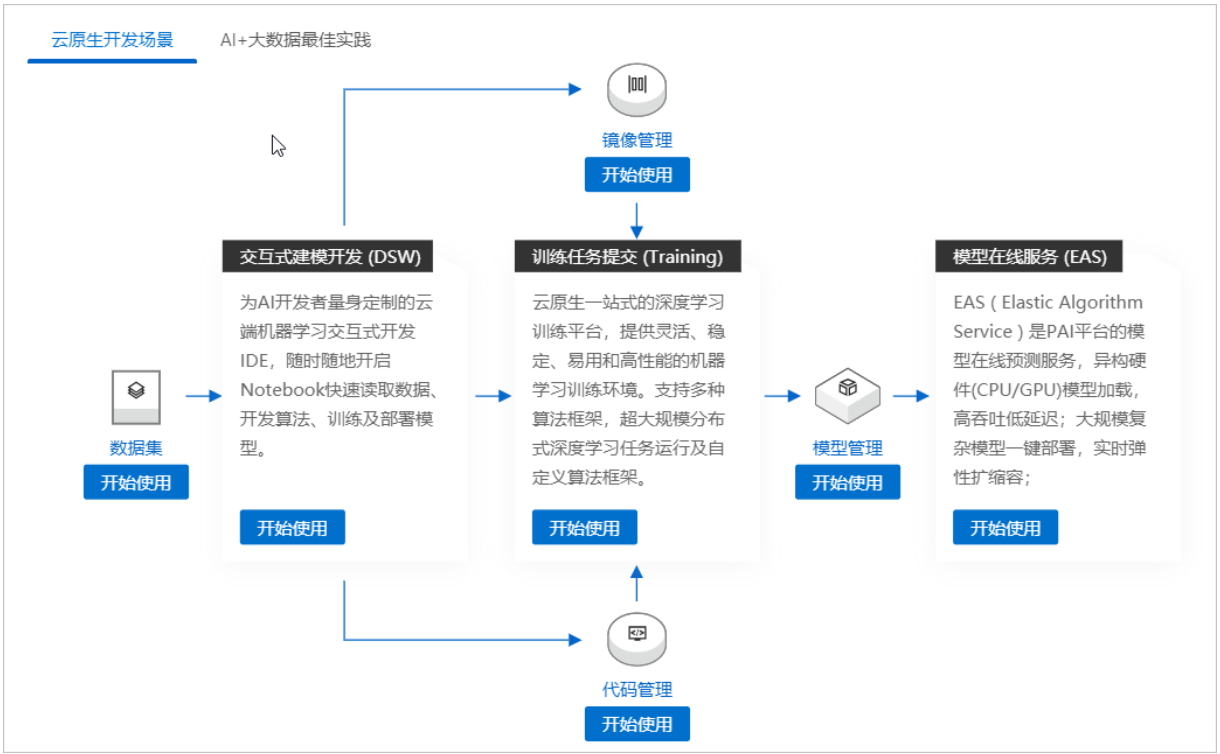
参数	描述
关联资源组	选择为工作空间关联的计算资源，目前支持MaxCompute资源组（按量付费或预付费）和PAI-DLC资源组。 您可以在**资源管理 > 资源仪表**页中，查看可关联的资源组。 **说明** 如果需要使用的资源组类型在资源仪表中未列出，请联系资源管理员进行添加。 当您选择MaxCompute资源组时，还需要配置已选资源组的MaxCompute项目名，同时可以选择是否需要GPU资源（目前支持不使用GPU或通过按量付费的方式使用GPU资源）。
相关模块介绍	在该区域，您可以了解资源组可使用的PAI相关模块功能。 MaxCompute资源组和PAI-DLC资源组均可在PAI-Designer中使用，不同的算法组件支持不同的计算资源，您可以在PAI-Designer中查看组件介绍了解相关信息。 此外，您也可以在训练任务提交模块中，直接提交分布式训练任务代码和镜像到PAI-DLC资源组中。对于PAI-DLC计算资源，系统仅支持关联按量付费公共资源组到工作空间。

5. 在**新建工作空间**的**确认信息**配置向导页面，确认已配置的工作空间信息，单击**进入工作空间**即可进入工作空间。

   在**确认信息**配置向导页面，您可以查看前面的步骤是否出错。如果存在失败操作，您可以返回并重新操作。

后续步骤

进入工作空间后，在左侧导航栏可以看到PAI全部子产品模块，您需要根据实际开发场景进行机器学习全生命周期的开发及管理。例如，在云原生开发场景下，您可以按照如下开发流程引导，使用子产品模块。



3.7.2. 管理成员

同一个AI工作空间可以添加多个成员，且为其授予特定的角色，从而拥有不同的执行权限。本文介绍如何将RAM用户添加为工作空间的成员并为其配置角色权限。

背景信息

当多个成员共同使用一个工作空间时，如果成员权限过大，使用不当会影响数据安全。反义，如果权限过小，则可能无法使用所需的功能。为了解决该问题，PAI的工作空间为您提供了多种角色身份，您可以根据不同RAM用户对工作空间的使用需求授予其相应的角色。左侧导航栏，单击**角色与权限列表**，您可以查看各角色与权限点的映射关系。

本文介绍与工作成员相关的如下操作：

- 添加成员
- 修改成员角色
- 删除成员

使用限制

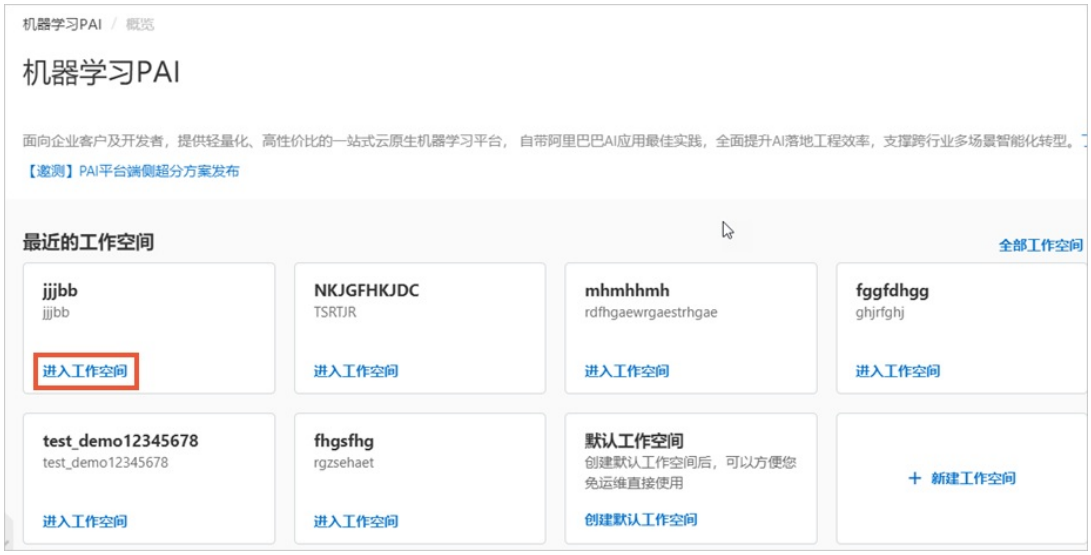
仅工作空间管理员和负责人能够进行工作空间成员管理。

进入成员管理面板

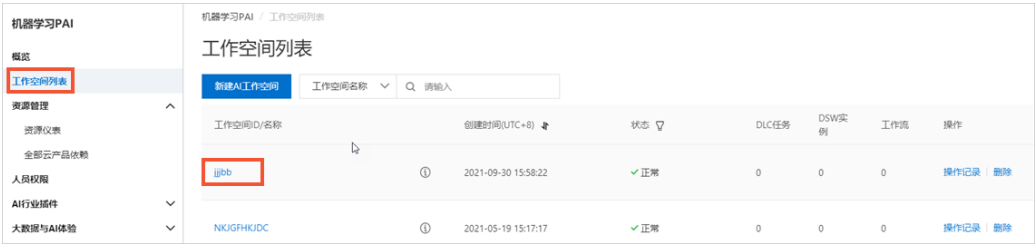
1. 进入工作空间。
    i. 登录PAI控制台。

ii. 您可以通过以下任意一种方式进入工作空间。

- 在**概览**页面的**最近的工作空间**区域，单击目标工作空间名称下的**进入工作空间**。



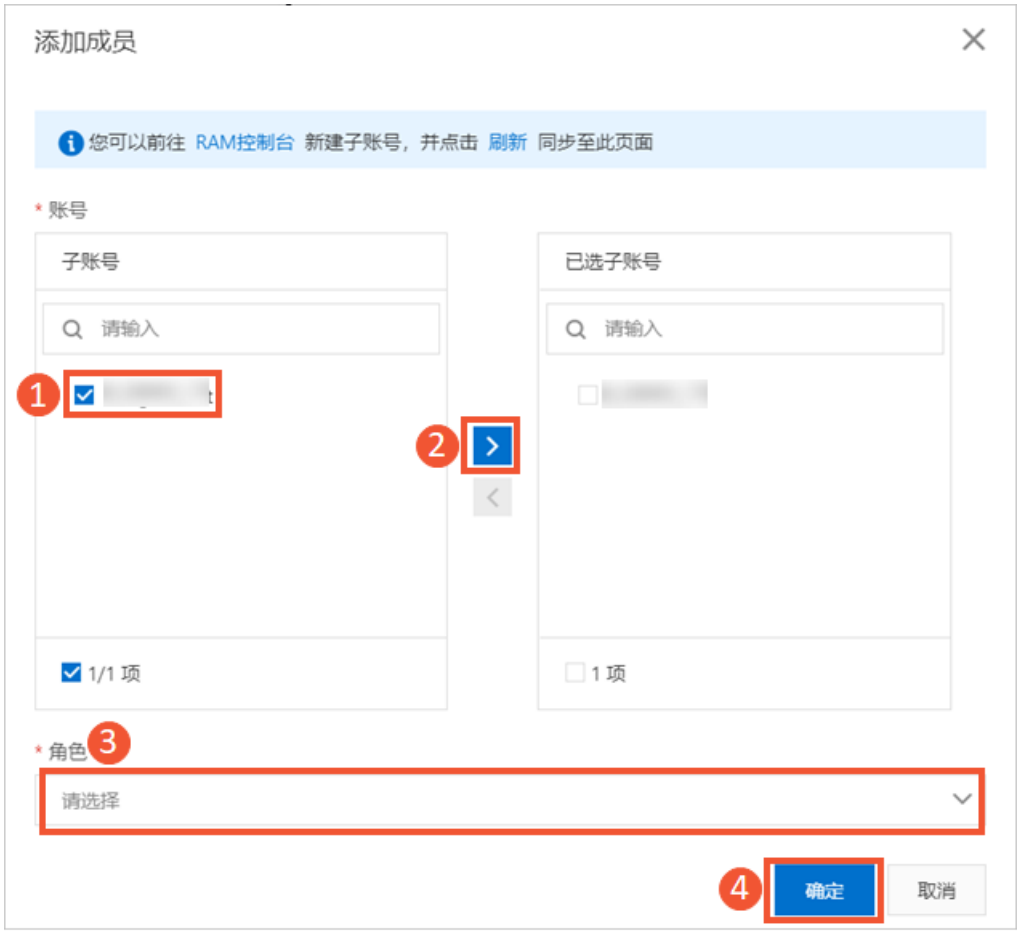
- a. 在左侧导航栏，单击**工作空间列表**。
  b. 在**工作空间列表**页面，单击目标工作空间的名称。



2. 在工作空间详情页面，单击工作空间成员后面的**编辑**，即可进入**成员管理**面板。

添加成员

1. 进入成员管理面板。
2. 在**成员管理**面板，单击**添加成员**。
3. 在**添加成员**对话框，配置如下参数。



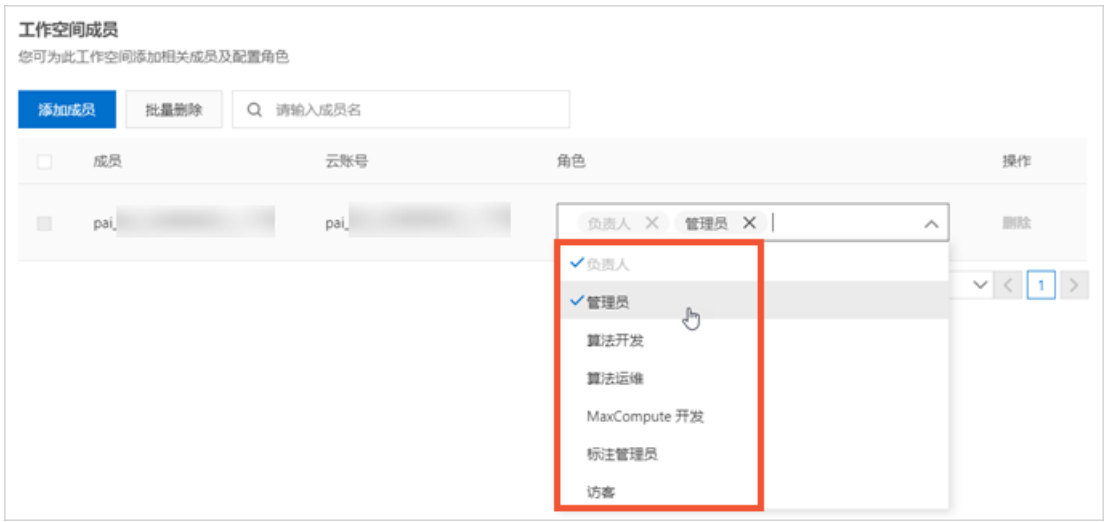
参数	描述
账号	左侧**子账号**区域显示可添加的所有RAM用户。选中某RAM用户前面的复选框，再单击>图标，该RAM用户就会显示在右侧的**已选子账号**区域。 如果您需要移除已选择的RAM用户，则在右侧的**已选子账号**区域选中RAM用户前面的复选框，再单击<图标，该RAM用户即会从右侧的**已选子账号**区域移除，显示在左侧**子账号**区域。

参数	描述
角色	系统支持以下角色，您需要根据实际需要进行选择： ○ **管理员**：拥有编辑工作空间成员、管理资源组及管理工作空间内全部资产的权限。 ○ **算法开发**：拥有在所属工作空间中进行开发和模型训练的权限。 ○ **算法运维**：拥有任务优先级管理、模型发布及线上服务监控等权限。 ○ **标注管理员**：拥有智能标注的操作权限。 ○ **MaxCompute开发**：DataWorks中的开发角色，拥有MaxCompute数据开发相关权限。您可以为从PAI提交任务至MaxCompute执行的RAM用户添加该角色。 ○ **访客**：拥有工作空间中各种资产的只读权限。 **说明** 您可以为同一个RAM用户添加多个角色，以满足实际需求。 您可以左侧导航栏，单击**角色与权限列表**，查看各角色与权限点的映射关系。

4. 单击**确定**。

修改成员角色

1. 进入成员管理面板。
2. 在**成员管理**面板，单击目标成员**角色**列下的列表。
3. 在角色列表中，单击某角色，如果该成员已经拥有了这个角色权限，则会为成员删除该角色。如果该成员没有这个角色权限，则会为成员添加该角色。



您也可以直接单击角色名后的×图标，即可为成员删除该角色。

说明

- 每个成员至少要拥有一个角色，即您不能删除一个成员的所有角色。
- 不能删除**负责人**角色。创建工作空间的阿里云账号或RAM用户自动成为该工作空间的**负责人**，他拥有为编辑工作空间成员、引用和管理资源组、管理工作空间内全部资产的权限。

删除成员

您可以分别通过如下方式每次删除单个或多个成员：

- 删除单个成员
- 批量删除成员

> **说明** 无法删除角色为**负责人**的成员。

您可以通过如下方法删除单个成员。

1. 进入成员管理面板。
2. 在**成员管理**面板，单击目标成员**操作**列下的**删除**。
3. 在**删除**对话框，单击**确定**。

您可以通过如下方法批量删除成员。

1. 进入成员管理面板。
2. 在**成员管理**面板，单击目标成员前面的复选框。
3. 单击**批量删除**。
4. 在删除确认对话框，单击**确定**。

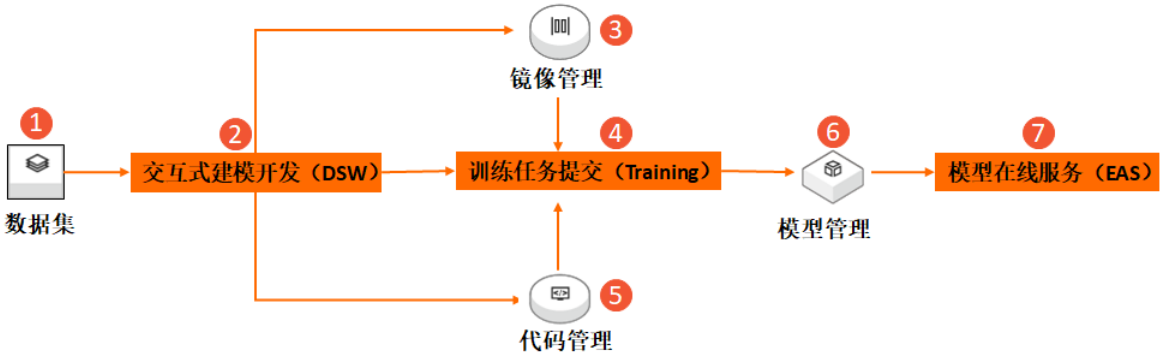
4.AI开发

4.1. 开发流程

进入工作空间后，您可以使用PAI子产品进行AI开发。本文介绍常用的AI开发流程，您可以结合实际情况选择。

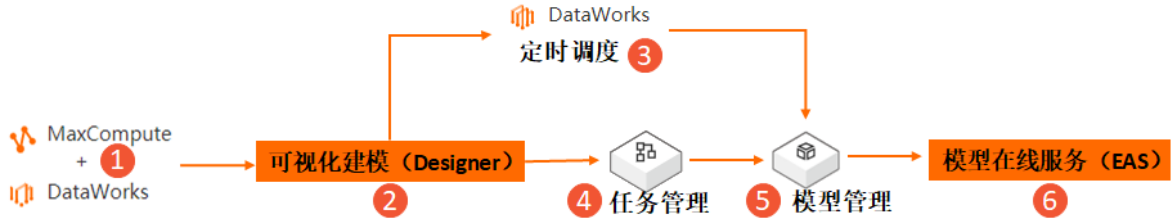
进入工作空间后，在左侧导航栏可以看到PAI全部子产品模块，您需要根据实际开发场景进行机器学习全生命周期的开发及管理。常见的使用场景如下所示，您可以按照流程引导，使用子产品模块：

- 云原生开发场景



区域	描述	相关文档
①	高质量的数据集是高精度模型的基础，也是数据准备的核心目标。您可以通过数据集管理模块将本地数据、阿里云存储中的数据及公共数据集进行注册，也可以通过扫描OSS文件夹生成索引数据集，从而统一管理PAI的相关数据，为数据标注及模型训练环节做准备。	创建数据集
②	交互式建模（DSW）是为AI开发者量身定制的云端机器学习交互式开发IDE，随时随地开启Notebook快速读取数据、开发算法、训练及部署模型。	概述
③	镜像管理模块为您展示PAI提供的官方公开镜像以及自定义镜像关联功能，从而在PAI中统一管理应用的镜像。	查看并添加镜像
④	云原生一站式的深度学习训练平台，提供灵活、稳定、易用和高性能的机器学习训练环境。支持多种算法框架，超大规模分布式深度学习任务运行及自定义算法框架。	概述
⑤	为了方便您在提交任务时指定所需的数据集和代码仓，PAI支持添加文件系统NAS或对象存储OSS的数据集以及Git代码仓。	数据配置
⑥	模型管理模块统一管理训练完成的模型，直接对接模型在线服务（EAS），您可以直接将模型部署为在线服务。	无
⑦	PAI平台的模型在线预测服务，支持异构硬件（CPU和GPU）模型加载，高吞吐低延迟，大规模复杂模型一键部署及实时弹性扩缩容。	模型在线服务（EAS）

- AI加大数据最佳实践



区域	描述	相关文档
①	使用MaxCompute存储数据，首先在DataWorks中对数据进行预处理，然后PAI引用MaxCompute表作为数据源进行模型训练。	○ 创建表 ○ 导入数据 ○ 数据开发流程
②	可视化建模（Designer）支持大规模分布式的传统机器学习、深度学习、强化学习训练，支持流批一体训练，该子模块封装了上百种机器学习算法，您可以拖拽式建模、自动调参，从而无编程玩转人工智能。	概述
③	DataWorks根据配置的调度和时间属性等参数，周期调度任务。	○ 配置调度参数 ○ 时间属性配置说明
④	任务管理支持将Designer的实验数据或自定义任务的执行信息记录到PAI提供的任务管理服务中，从而便于进行任务间的实验比较。	实验对照组
⑤	模型管理模块统一管理训练完成的模型，直接对接模型在线服务（EAS），您可以直接将模型部署为在线服务。	无
⑥	PAI平台的模型在线预测服务，支持异构硬件（CPU和GPU）模型加载，高吞吐低延迟，大规模复杂模型一键部署及实时弹性扩缩容。	模型在线服务（EAS）

4.2. 智能标注（iTAG）

4.2.1. 概述

智能标注（iTAG）是一款智能化数据标注平台，支持图像、文本、视频、音频等多种数据类型的标注以及多模态的混合标注。智能标注（iTAG）提供了丰富的标注内容组件和题目组件，您可以直接使用平台预置的标注模板，也可以根据自己的场景自定义模板进行数据标注。

任务流程

在智能标注（iTAG）中，完整的数据标注流程包括以下步骤：

1. 创建数据集：用于数据标注
   通过数据集管理模块将待标注的原始数据创建为数据集，从而生成.manifest索引文件。
2. 创建标注任务

对于已创建的数据集，通过智能标注（iTAG）提供的通用模板或自定义模板创建标注任务并分发任务。任务分发流程分为打标、质检及验收三个环节，其中打标为必选环节，质检和验收为可选环节。因此创建标注任务时，您可以选择以下四种流程：打标、打标-检测、打标-验收、打标-检测-验收。任务分发各环节的主要任务如下：

- 打标：标注员登录标注平台，在标注任务页面，先领取标注任务包，再完成标注，最后提交相应的标注任务。
- 检查：标注环节完成后，标注员登录标注平台，在质检任务页面，领取已经标注完成的任务包后，对其进行检查、修改或驳回。
- 验收：需求方将标注完成或质检完成的任务进行最后一步的验收。需求方登录标注平台，在验收任务页面，领取相应任务包后，对其进行验收、修改或驳回。

3. 处理标注任务
   按照任务流程为任务包进行打标、质检或验收，从而获得标注好的数据。
4. 导出标注结果数据
   将标注结果导出至指定的OSS目录，从而将其用于模型训练。

数据格式

- 智能标注的输入数据格式
  在创建标注任务过程中需要选择已创建的数据集，该数据集需要满足.manifest格式。您可通过PAI数据集管理模块创建数据集，系统会为该数据集自动生成标注所需要的.manifest数据格式。具体的数据格式请参见创建数据集：通用模板数据标注场景和创建数据集：自定义模板数据标注场景。
- 导出的标注结果的数据格式
  智能标注（iTAG）的标注结果为.manifest格式，标注流程中，打标、质检、验收环节的数据格式请参见导出标注结果数据。

人员管理及权限

在智能标注（iTAG）中，人员管理包括管理员、标注组长及标注员三种角色，各角色的权限如下表所示。

角色	描述	设置方法	权限
管理员	标注任务的需求方和管理者创建数据集和标注任务，并分发给标注组长或标注员进行打标和检查，最终管理员对任务进行抽样验收，不合格的任务可以驳回重新打标。	1. 通过AI工作空间详情页的成员配置，配置该账号为标注管理员。 2. 进入iTAG人员管理页面，将该RAM用户账号设置为管理员。	• 能够登录管理员控制台 • 能够登录标注员控制台 • 能够进行人员管理和分配 • 能够参与标注的每个环节
标注组长	标注任务的负责人和标注员的管理人。标注组长可以对标注任务进行人员分配。也可对分配给自己的标注或质检任务进行标注或检查。	1. 通过AI工作空间详情页的成员配置，将该RAM用户加入指定的AI工作空间。 2. 进入iTAG人员管理页面，将该RAM用户设置为标注组长。	• 不能登录管理员控制台 • 能够登录标注员控制台 • 能够进行人员管理和分配 • 能够参与标注的每个环节

角色	描述	设置方法	权限
标注员	标注任务的具体执行人。标注员对分配给自己的标注或质检任务进行标注或检查。	1. 通过AI工作空间详情页的成员配置，将该RAM用户加入指定的AI工作空间。 2. 进入iTAG人员管理页面，将该RAM用户设置为标注员。	• 不能登录管理员控制台 • 能够登录标注员控制台 • 不能进行人员管理和分配 • 能够参与标注的每个环节

4.2.2. 创建数据集：用于数据标注

在创建标注任务过程中需要选择已创建的数据集，该数据集需要满足用于数据标注的格式要求，不同数据标注方式对数据集内的文件格式要求不一致。本文为您介绍用于数据标注的数据集的格式要求和创建操作指导。

背景信息

使用智能标注（iTAG）进行数据标注时，您需要先将待标注的文件上传至阿里云OSS，创建为待数据标注使用的数据集后，再在智能标注中创建标注任务，开始进行数据标注。

PAI智能标注支持通过**通用模板**和**自定义模板**两种方式创建数据标注任务，不同方式的标注任务需要的数据准备及数据集创建方式不一致。

- **通用模板**
  通用模板可进一步细分为文本、图像、视频、音频这四类，各个细分类别的创建数据集操作步骤和数据集格式要求请参见下文的创建数据集：通用模板数据标注场景。
- **自定义模板**
  自定义模板支持更灵活的数据标注场景，例如支持在同一个标注任务中同时标注图片、文本等多种类型的样本，自定义模板应用场景下的创建数据集操作步骤和数据集格式要求请参见下文的创建数据集：自定义模板数据标注场景。

前提条件

已开通阿里云OSS。

创建数据集：通用模板数据标注场景

使用通用模板创建标注任务时，系统会自动解析内容。通用模板的应用可以细分为文本、图像、视频和音频这几种文件标注场景，各场景下待标注文件的准备和.manifest文件格式要求如下。

对比项	通用模板：文本	通用模板：图像/视频/音频
创建数据集操作指导	1. 根据下文的后缀和格式要求，在本地创建好.manifest或TXT文件。 2. 将本地文件上传至阿里云OSS，上传文件至OSS请参见上传文件。 3. 以**阿里云存储**的方式创建数据集，操作详情请参见创建数据集：从阿里云存储。	1. 将图片、视频或音频文件上传至阿里云OSS，生成OSS存储路径URL，上传文件至OSS请参见上传文件。 2. 通过**扫描文件**的方式创建数据集，自动生成.manifest文件，操作详情请参见创建数据集：扫描文件夹。
文件后缀类型要求	.manifest或TXT文件。	.manifest文件。

对比项	通用模板：文本	通用模板：图像/视频/音频

文件内容格式要求	文件内容格式为：
	`{"data":{"source":"text sample 1"}}`
	`{"data":{"source":"text sample 2"}}`
	`{"data":{"source":"text sample 3"}}`
	其中**source**表示需要标注的样本内容，您需要将source的取值替换为对应需要标注的文本内容。
		文件内容格式为：
		`{"data":{"source":"oss://****.oss-cn-hangzhou.aliyuncs.com/iTAG/pic/1.jpg"}}`
		`{"data":{"source":"oss://****.oss-cn-hangzhou.aliyuncs.com/iTAG/pic/10.jpg"}}`
		`{"data":{"source":"oss://****.oss-cn-hangzhou.aliyuncs.com/iTAG/pic/11.jpg"}}`
		其中**source**表示需要标注的样本内容，source的取值为对应样本的OSS存储路径URL。

文件demo参考	textDemo.manifest	• 图片：pictureDemo.manifest • 视频：videoDemo.manifest • 音频：audioDemo.manifest

创建数据集：自定义模板数据标注场景

通过自定义模板创建标注任务之前，您需要生成.manifest文件的格式如下，需要标注的样本内容的键值可以自己定义，在创建标注任务时可以自行选择标注内容的对应字段。

对比项	自定义模板
创建数据集操作指导	1. 根据下文的后缀和格式要求，在本地创建好.manifest或TXT文件。 2. 将本地文件上传至阿里云OSS，上传文件至OSS请参见上传文件。 3. 以**阿里云存储**的方式创建数据集，操作详情请参见创建数据集：从阿里云存储。
文件后缀类型要求	.manifest或TXT文件。

对比项	自定义模板

文件内容格式要求	文件内容格式为：
	`{"data":{"picture_url":"oss://****.oss-cn-hangzhou.aliyuncs.com/iTAG/pic/1.jpg","text":"马云带领下的18位创始人在杭州的公寓中正式成立了阿里巴巴集团，集团的首个网站是英文全球批发贸易市场阿里巴巴"}}`
	`{"data":{"picture_url":"oss://****.oss-cn-hangzhou.aliyuncs.com/iTAG/pic/10.jpg","text":"阿里巴巴集团举办首届西湖论剑，汇聚互联网界的商业和意见领袖讨论业界重要议题"}}`
	`{"data":{"picture_url":"oss://****.oss-cn-hangzhou.aliyuncs.com/iTAG/pic/11.jpg","text":"阿里巴巴集团从数家一线投资机构融资8200万美元，成为当时中国互联网届最大规模的私募融资"}}`
	
	其中，每一行 `"data"` 均表示一个标注任务，每一行 `"data"` 中可以同时包含多种类型的样本检测，每个检测样本之间通过逗号分隔。
	例如：
	`{"data":{"picture_url":"oss://****.oss url 01","text":"text sample1"}}`
	
	这个标注任务会同时检测标注图片（样本图片的存储路径是 *oss://****.oss url 01* ）和文本（文本样本的取值是 *text sample1* ）。

文件demo参考	multiModal.manifest

后续步骤

您可以使用已经注册的数据集创建标注任务，详情请参见创建标注任务。

4.2.3. 创建标注任务

创建完成用于数据标注的数据集后，您可以使用智能标注（iTAG）开展标注任务。PAI为您提供通用模板用于创建标注任务，如果通用模板无法满足需求，您也可以根据实际场景，通过拼接内容组件和题目组件自定义模板。本文为您介绍如何通过预置的通用模板创建标注任务。

前提条件

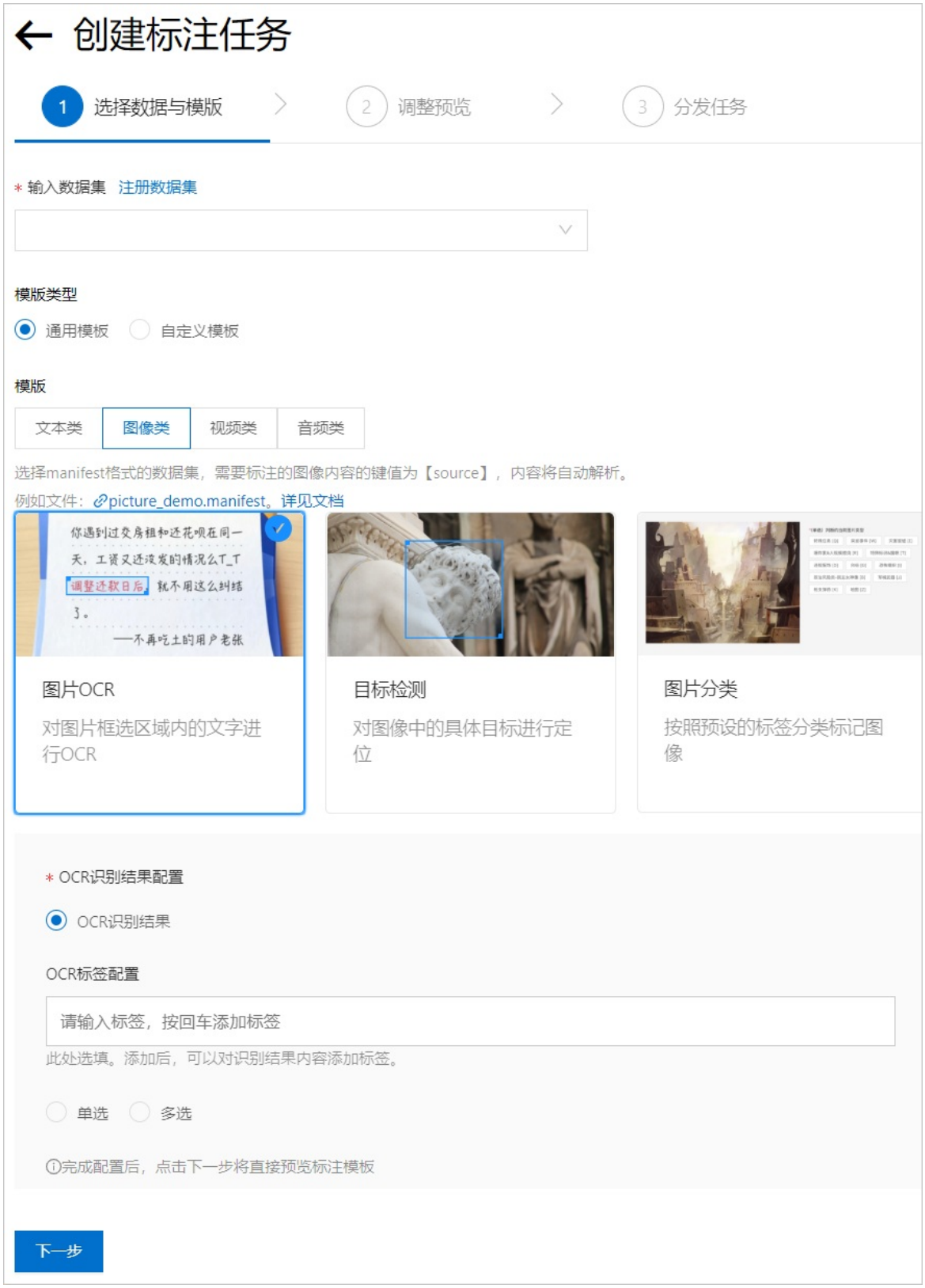
- 已开通PAI并创建好工作空间。
  您可以使用默认工作空间，或根据业务规划新建其他工作空间。默认空间的创建请参见开通并创建默认工作空间，其他工作空间的创建请参见创建工作空间。
- 已开通阿里云OSS，将待标注的数据文件上传至OSS并创建为数据集。操作详情请参见创建数据集：用于数据标注。

使用限制

仅管理员或标注管理员可以进行标注相关操作。如果您的账号没有权限，请联系管理员为您的账号授予标注管理员权限，操作详情请参见管理成员。

操作步骤

1. 进入智能标注（iTAG）。
   i. 登录PAI控制台。
   ii. 在左侧导航栏单击**工作空间列表**，在工作空间列表页面中单击待操作的工作空间名称，进入对应工作空间内。
   iii. 在工作空间页面的左侧导航栏选择**数据准备 > 智能标注（iTAG）**，进入智能标注页面。
2. 在**智能标注（iTAG）**页面的**任务中心**页签，单击**创建任务**。
3. 在**创建标注任务**页面的**选择数据与模板**配置向导页面，配置如下参数，并单击**下一步**。



参数	描述
输入数据集	选择在PAI数据集管理中已经创建的数据集。

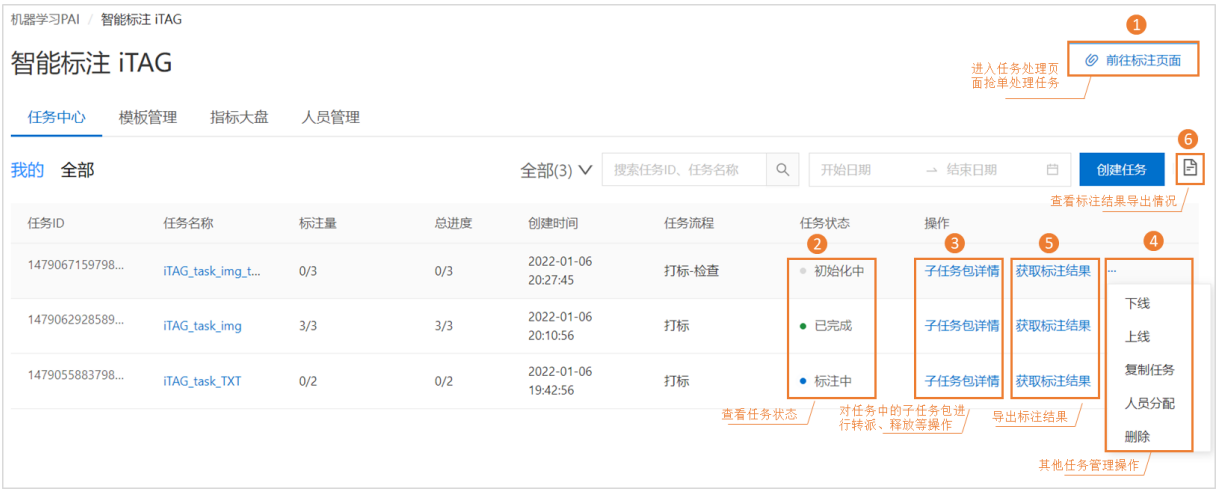
参数	描述
模板类型	支持以下类型的模型： ○ **通用模板**：平台预置的常用模板。 ○ **自定义模板**：根据自己特定的场景，参考页面提示拼接**内容组件**和**题目组件**，从而通过自定义模板创建标注任务。 自定义模板适用于自定义的场景，模板的输入输出数据格式详情请参见自定义模板。
模板	**模板类型**为**通用模板**时，支持选择细分的通用模板类型，当前支持： ○ **文本类**包含以下三种细分类型： ■ **实体识别**：建立文本实体之间的关系。 ■ **文本分类**：对文本按照预设标签进行分类标记，支持单标签和多标签分类。 ■ **实体关系**：文本实体之间的关系，用于知识图谱场景。 模板的详细应用场景，及此类模板的输入输出数据格式详情请参见文本类。 ○ **图像类** ■ **图片OCR**：对图片框选区域内的文字进行OCR。 ■ **目标检测**：对图像中的具体目标进行定位。 ■ **图像分类**：按照预设的标签分类标记图像。 模板的详细应用场景，及此类模板的输入输出数据格式详情请参见图像类。 ○ **视频类** **视频分类**：对视频按照预设标签进行分类标记，支持单标签和多标签分类。 模板的详细应用场景，及此类模板的输入输出数据格式详情请参见视频类。 ○ **音频类** ■ **音频分类**：对音频按照预设标签进行分类标记，支持单标签和多标签分类。 ■ **音频分割**：对音频数据集的内容进行分割并分段添加标签。 ■ **音频识别**：将音频内容的文字进行识别。 模板的详细应用场景，及此类模板的输入输出数据格式详情请参见语音类。
标签配置	依次输入本标注任务中，后续打标人员需要进行识别框选并打标的标签名称，并按回车键完成添加标签。 例如，对图片中的猫进行识别时，您可以添加标签：“猫”、“美短”、“英短”等，便于后续打标人员对样本进行打标。 您可以同时设置本标注任务中，打标人员对一个样本的一次框选打标时，可以对框选的对象打一个标签或多个标签。 ○ 一次框选打标只能打一个标签时，可设置标签为**单选**。 ○ 一次框选打标可以打多个标签时，可设置标签为**多选**。 例如，对图片中的猫进行识别时，如果标签设置为**多选**，标注人员框选出样本中的猫后，可以同时打上标签“猫”、“美短”。 **说明** 此处的**单选**与**多选**指的是对一个样本的一次框选打标过程中，支持添加一个还是多个标签，而非对一个样本进行多次框选打标。

4. 在**调整预览**配置向导页面，预览标注任务，并单击**下一步**。
5. 在**分发任务**配置向导页面，配置如下参数，并单击**创建**。

参数	描述
任务名称	长度为1~100个字符，以小写字母、大写字母、数字或中文开头，可以包含下划线（_）或短划线（-）。
任务说明	标注任务的简要说明，以区分不同的任务。
子任务包分配	iTAG会将所有数据集中所有标注任务按照指定规则汇总分配为一个个子任务包，后续在标注任务处理过程中，处理人员来抢单获取子任务包，进行子任务包内的所有标注任务处理。 当前支持通过以下规则来分配子任务包： ○ **固定大小**：每个子任务包中包含的标注任务数量为固定值。 设置为固定大小时，数据集的数据量与子任务包中的任务数范围有如下对应关系： ■ 当数据集为0~2万条数据，子任务包大小范围是1~200。 ■ 当数据集为2万~10万条数据，子任务包大小范围是5~200。 ■ 当数据集为10万~50万条数据，子任务包大小范围是25~200。 ■ 当数据集为50万~100万条数据，子任务包大小范围是50~200。 ○ **按导入字段**：按照数据集中所选字段来划分子任务包，该字段数值相同的数据会放在同一个子任务包里面。
任务流程	您可以结合实际情况设置本次创建的任务需要包含的任务流程。 ○ **打标**：本次任务仅需仅需打标这一个流程，打标员完成打标提交后，本次任务即全部完成。 ○ **打标-检查**：本次任务需经过打标、检查两个流程，打标人员完成打标提交后，需检查员完成检查提交，本次任务才全部完成。 ○ **打标-验收**：本次任务需经过打标、验收两个流程，打标人员完成打标提交后，需验收员完成验收提交，本次任务才全部完成。 ○ **打标-检查-验收**：本次任务需经过打标、检查、验收三个流程，打标人员完成打标提交后，需检查员完成检查提交、验收员完成验收提交，本次任务才全部完成。
检查比例	如果您选择的**任务流程**包括检查环节，例如**打标-检查**或**打标-检查-验收**，则需要配置在检查过程中，对标注任务抽查的比例，默认为100%。
人员配置	根据选择的**任务流程**，配置标注人员、检查人员、验收人员或任务管理员，您可以与该AI工作空间下的多个成员，协作完成标注任务。智能标注（iTAG）中的角色权限请参见人员管理及权限。

查看任务列表

创建完成打标、检查、验收任务后，您可以在智能标注的**任务中心**页面看到当前所有任务列表，可查看任务的状态，并通过操作列的操作入口了解子任务包详情和获取标注结果等操作。



- 处理标注任务：
  您可以单击页面右上角的**前往标注页面**进入iTAG标注页面，对待处理的打标、检查、验收任务进行抢单处理，操作详情请参见处理标注任务。
- 转派、释放子任务包：
  您可以在**任务中心**页面查看所有任务的状态，对于未完成的任务，您可以单击**子任务包详情**，进一步查看子任务包的完成情况，对于未完成的子任务包，您可以单击**转派**，将子任务包直接转交给其他人员处理；或单击**释放**，释放后的子任务包可以被其他人员继续抢单进行处理。
- 导出并查看标注结果：

  对已完成的任务，您可以单击**获取标注结果**，根据界面提示将标注结果导出，并可以单击右上方的 

  按钮查看导出的进度和结果。操作详情请参见导出标注结果数据。
- 其它任务管理操作：
  您可以单击任务操作列后的更多操作，对任务进行下线、上线等管理操作。

后续步骤

您可以抢单来认领标注任务并进行标注处理，详情请参见处理标注任务。

4.2.4. 处理标注任务

创建标注任务后，您需要按照任务流程为任务包进行打标、检查或验收，从而获得标注好的数据，为模型训练做准备。本文以处理图像类标注任务为例，为您介绍如何处理标注任务，其他类型的标注任务操作类似。

前提条件

管理员已分配标注任务，详情请参见创建标注任务。

使用限制

仅管理员或标注管理员可以进行标注相关操作。如果您的账号没有权限，请联系管理员为您的账号授予标注管理员权限，操作详情请参见管理成员。

操作步骤

1. 进入智能标注（iTAG）。
   i. 登录PAI控制台。

ii. 在左侧导航栏单击**工作空间列表**，在工作空间列表页面中单击待操作的工作空间名称，进入对应工作空间内。

iii. 在工作空间页面的左侧导航栏选择**数据准备 > 智能标注（iTAG）**，进入智能标注页面。

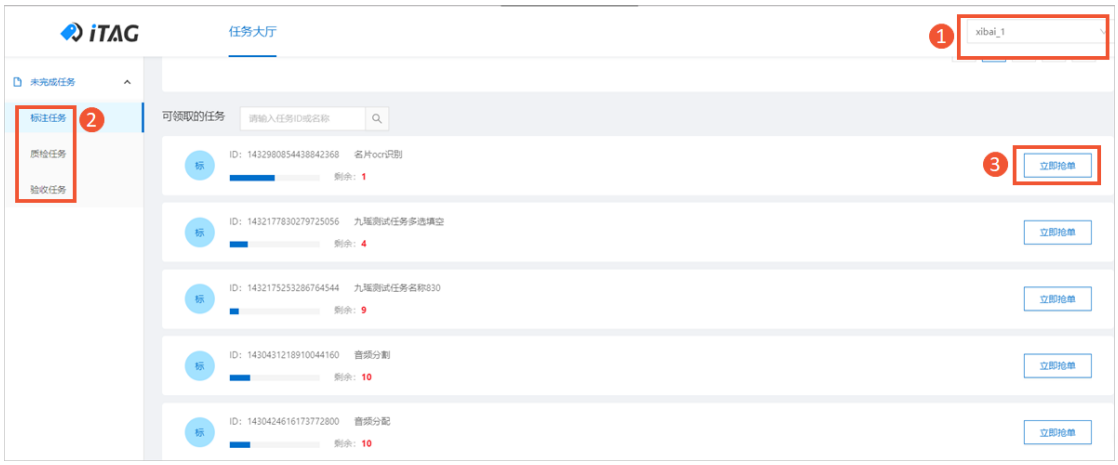
2. 在**智能标注iTAG**页面，单击右上方的**前往标注页面**，进入iTAG平台。

3. 领取标注任务。

i. 在iTAG平台的右上方，选择AI工作空间。

ii. 在左侧导航栏，您可以单击**标注任务**、**质检任务**或**验收任务**，进入相应类型的任务列表页面。

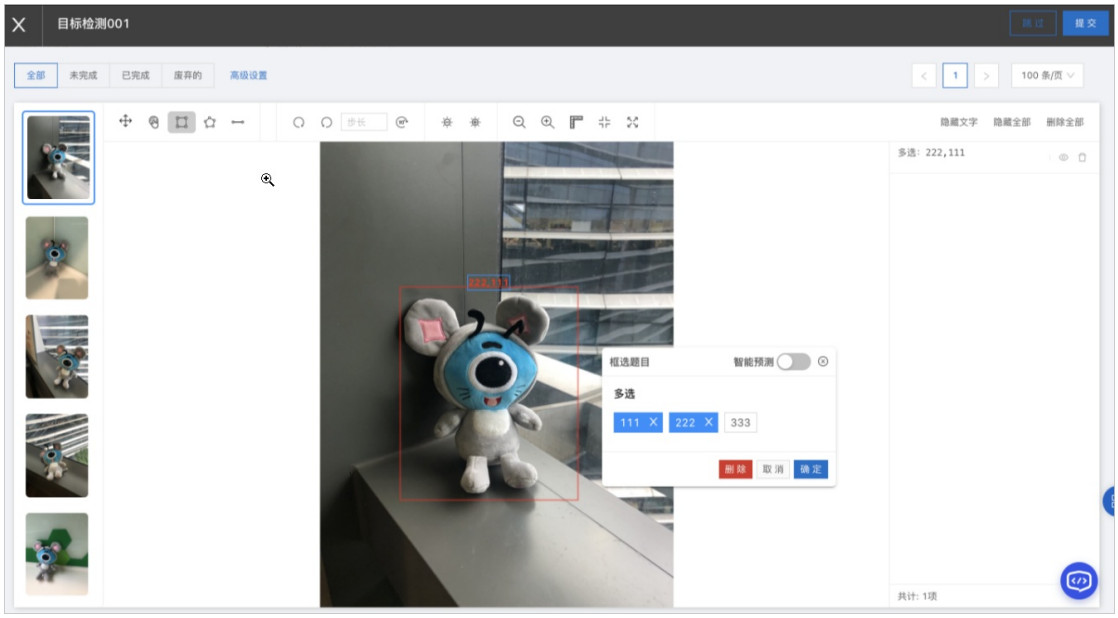
iii. 在**可领取的任务**区域，单击目标任务后的**立即抢单**，即可进入标注页面。



4. 打标。

i. 在标注页面，选择合适的标注工具，对目标任务进行打标。

您可以单击左侧的任务列表缩略图，切换任务包中的任务。



ii. 完成所有的标注任务后，单击右上方的**提交**。

5. 质检。

i. 在iTAG平台的左侧导航栏，单击**质检任务**。

ii. 按照页面提示信息，对领取的已经标注完成的任务包进行检查、修改或驳回。

6. 验收。
   i. 在iTAG平台的左侧导航栏，单击**验收任务**。
   ii. 按照页面提示信息，对领取验收任务包进行验收、修改或驳回。

后续步骤

对于标注完成的任务，您可以将标注结果导出，用于后续的模型训练。关于如何导出标注结果，请参见导出标注结果数据。

4.2.5. 导出标注结果数据

处理完成标注任务后，您可以将标注结果导出至指定的OSS目录，也可以将标注结果同步创建为一个数据集，便于后续将其直接用于模型训练。本文介绍如何导出标注结果，及如何查看导出的进度与结果。

前提条件

已处理完成数据标注，详情请参见处理标注任务。

使用限制

仅管理员或标注管理员可以进行标注相关操作。如果您的账号没有权限，请联系管理员为您的账号授予标注管理员权限，操作详情请参见管理成员。

导出标注结果

1. 进入智能标注（iTAG）。
   i. 登录PAI控制台。
   ii. 在左侧导航栏单击**工作空间列表**，在工作空间列表页面中单击待操作的工作空间名称，进入对应工作空间内。
   iii. 在工作空间页面的左侧导航栏选择**数据准备 > 智能标注（iTAG）**，进入智能标注页面。
2. 在**智能标注（iTAG）**页面的**任务中心**页签，单击目标标注任务**操作**列下的**获取标注结果**。
3. 在**获取标注结果**对话框，配置如下参数，并单击**确认**。

参数	描述
导出至数据集	选择是否需要将标注结果直接创建为一个数据集，便于后续模型训练时，直接使用此标注结果数据。 如果标注的结果需要继续使用PAI进行模型训练，建议此处选择**是**。
OSS输出位置	设置标注结果导出后存储的OSS路径。如果选择同步导出至数据集，则此OSS路径即也是数据集的存储路径。

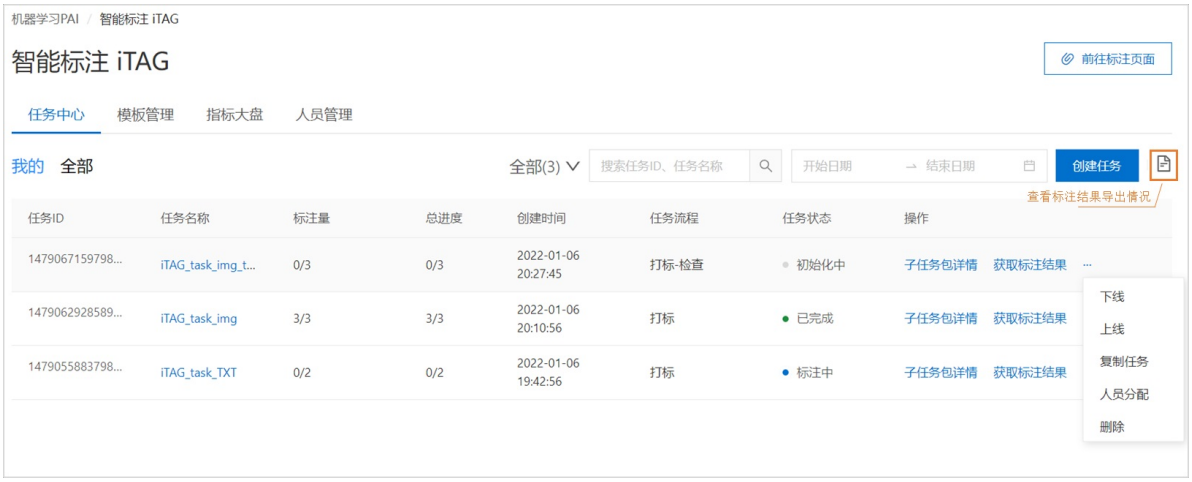
4. 当界面弹出已导出完成的提示后，您即可前往查看导出结果文件。

   导出的标注结果文件是一个后缀为.manifest格式的文件，文件内容格式请参见标注数据格式概述。

查看导出进度与结果

- 您可以单击智能标注页面右上方的 按钮，在弹出的页面中查看导出的进度。

  对于导出成功的任务可直接单击**跳转**，进入存储标注结果数据文件的OSS Bucket页面，查看导出文件，导出文件是一个后缀为.manifest格式的文件，文件内容格式请参见标注数据格式概述。



- 如果您同时将导出结果创建为一个数据集，您可以进入数据集管理页面，查看同步创建的数据集结果。

后续步骤

您可以使用导出的标注结果数据集进行模型训练。

4.2.6. 附录：标注数据格式

4.2.6.1. 标注数据格式概述

智能标注（iTAG）的标注结果为.manifest格式，标注流程中，打标、质检、验收环节的数据格式如下。

```
{
    "data": {
        "source": "oss://********/pics/fruit/apple-1.jpg"
    },
    "label-xxxxx(任务id)": {
        "results": [{...}, {...}, {...}]
          // 按照标注环节的题目顺序列出问题，其中每个元素可能的数据结构，详见下文。
    },
    "label-xxxxx(任务id)-check": {
        "results": [{...}, {...}, {...}]
          // 按照质检环节的题目顺序列出问题，其中每个元素可能的数据结构，详见下文。
    },
     "label-xxxxx(任务id)-verify": {
        "results": [{...}, {...}, {...}]
          // 按照验收环节的题目顺序列出问题，其中每个元素可能的数据结构，详见下文。

    }
}
```

上述格式中的字段含义如下：

- data：标注的目标对象。
- label-xxxxx(任务id)：标注环节的结果数据。
- label-xxxxx(任务id) -check：质检环节的结果数据。
- label-xxxxx(任务id) -verify：验收环节的结果数据。
- results：各环节具体存放的标注结果。每种模板的标注结果内容请参见如下链接：
  - 文本类
  - 图像类
  - 视频类
  - 语音类
  - 自定义模板

4.2.6.2. 文本类

智能标注（iTAG）提供了实体识别、文本分类、实体关系的文本类标注模板，创建标注任务时，您需要根据应用场景选择标注模板。本文为您介绍文本类标注模板的应用场景及数据结构。

背景信息

本文介绍以下文本类标注模板的数据结构：

- 实体识别
- 文本分类
- 实体关系

实体识别

实体识别NER标注任务是对文本中的具体内容进行框选，并添加标签。

- 应用场景
  商品主体词识别、新闻主体词识别等。
- 数据结构
  - 输入数据
    manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

    ```
    {"data":{"source":"阿里巴巴收购两家服务美国小企业的电子商务解决方案供应商Vendio及Auctiva。同月，手机淘宝客户端推出。"}}
    ...
    ```

  - 输出数据
    manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "阿里巴巴收购两家服务美国小企业的电子商务解决方案供应商Vendio及Auctiva。同月，手机淘宝客户端推出。"
    },
    "label-1430082002522152960": {
        "results": [
            {
                "objects": [
                    {
                        "result": {
                            "文本内容": [
                                "标签1"
                            ]
                        },
                        "color": null,
                        "id": null,
                        "text": "ocr本文的识别内容1",
                        "start": 49,
                        "end": 51
                    },
                    {
                        "result": {
                            "文本内容": [
                                "标签2",
                                "标签3"
                            ]
                        },
                        "color": null,
                        "id": null,
                        "text": "ocr本文的识别内容2",
                        "start": 34,
                        "end": 40
                    },
                ],
                "empty": false
            }
        ]
    }
}
```


文本分类

文本分类（Text Classification）是指在一组固定的分类标签集合中，找到与输入文本内容相匹配的一个或多个分类标签，并将其分配给该输入文本。该分类模板支持单标签和多标签。

- 应用场景
  新闻推荐、知识管理及垃圾信息过滤等。
- 数据结构

- 输入数据
  manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

```
{"data":{"source":"阿里巴巴更改其中国交易市场的名称为“1688”。同月，淘宝网推出团购网站聚划算。"}
}
...
```

- 输出数据
  manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "阿里巴巴更改其中国交易市场的名称为“1688”。同月，淘宝网推出团购网站聚划算。"
    },
    "label-1432989439570944000": {
        "results": [
            {
                "questionId": "2",
                "data": [
                    "标签2",
                    "标签1"
                ],
                "markTitle": "多选",
                "type": "survey/multivalue"
            }
        ]
    }
}
```

实体关系

实体关系（三元组和Knowledge Graph）标注任务，主要是针对知识图谱的场景，对实体词之前的关系添加标签。

- 应用场景
  知识图谱等。
- 数据结构
  - 输入数据
    manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

```
{"data":{"source":"阿里巴巴更改其中国交易市场的名称为“1688”。同月，淘宝网推出团购网站聚划算。"}
}
...
```

  - 输出数据
    manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "阿里巴巴更改其中国交易市场的名称为“1688”。同月，淘宝网推出团购网站聚划算。"
    },
    "label-1435488346167255040": {
        "results": [
            {
                "objects": [
```

```
                {
                    "result": {
                        "多选": [
                            "标签3"
                        ]
                    },
                    "color": null,
                    "id": null,
                    "text": "团购网站",
                    "start": 32,
                    "end": 35
                },
                {
                    "result": {
                        "多选": [
                            "标签2"
                        ]
                    },
                    "color": null,
                    "id": null,
                    "text": "1688",
                    "start": 18,
                    "end": 21
                },
                {
                    "result": {
                        "多选": [
                            "标签1"
                        ]
                    },
                    "color": null,
                    "id": null,
                    "text": "交易市场",
                    "start": 9,
                    "end": 12
                }
            ],
            "empty": false
        },
        [
            {
                "result": {
                    "单选": "标签4"
                },
                "from": {
                    "x": -225,
                    "y": -126,
                    "start": 9,
                    "end": 12,
                    "text": "交易市场"
                },
                "to": {
                    "x": -233,
                    "y": 75,
                    "start": 18,
```

```
                        "end": 21,
                        "text": "1688"
                    }
                },
                {
                    "result": {
                        "单选": "标签6"
                    },
                    "from": {
                        "x": -225,
                        "y": -126,
                        "start": 9,
                        "end": 12,
                        "text": "交易市场"
                    },
                    "to": {
                        "x": 24,
                        "y": -93,
                        "start": 32,
                        "end": 35,
                        "text": "团购网站"
                    }
                },
                {
                    "result": {
                        "单选": "标签4"
                    },
                    "from": {
                        "x": -233,
                        "y": 75,
                        "start": 18,
                        "end": 21,
                        "text": "1688"
                    },
                    "to": {
                        "x": 24,
                        "y": -93,
                        "start": 32,
                        "end": 35,
                        "text": "团购网站"
                    }
                }
            ]
        ]
    }
}
```


4.2.6.3. 图像类

智能标注（iTAG）提供了图片OCR、目标检测、图像分类的图像类标注模板，创建标注任务时，您需要根据应用场景选择标注模板。本文为您介绍图像类标注模板的应用场景及数据结构。

背景信息

本文介绍以下图像类标注模板的数据结构：

- 图片OCR
- 目标检测
- 图像分类

图片OCR

图片OCR（Optical Character Recognition）任务首先将输入图像中的文字转换为文本格式，再根据文字信息类别对输入图像进行分组。

- 应用场景
  证件识别、票据识别、车牌识别及银行卡识别等。
- 数据结构
  - 输入数据
    manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

    ```
    {"data":{"source":"oss://****.oss-cn-hangzhou.aliyuncs.com/demo_test/ocr_pic/img6.jpeg"
    }}
    ...
    ```

  - 输出数据
    manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "oss://****.oss-cn-hangzhou.aliyuncs.com/demo_test/ocr_pic/img6.jpeg"
    },
    "label-144863699223676****": {
        "results": [
            {
                "questionId": "1",
                "data": [
                    {
                        "id": "ecdb7552-2a4e-4d0e-8abb-0f1a2dc0****",
                        "type": "image/polygon",
                        "value": [
                            [
                                368.1112214498511,
                                71.72740814299901
                            ],
                            [
                                444.34359483614696,
                                71.72740814299901
                            ],
                            [
                                444.34359483614696,
                                106.26762661370405
                            ],
                            [
                                368.1112214498511,
                                106.26762661370405
                            ]
                        ],
                        "labels": {
                            "OCR识别结果": "理财顾问",
                            "单选": "标签1"
                        }
                    }
                ],
                "rotation": 0,
                "markTitle": "OCR标签配置",
                "width": 1024,
                "type": "image",
                "height": 1024
            }
        ]
    }
}
```


目标检测

目标检测（Object Detection）标注任务是对图像中的具体目标进行定位，常用矩形框工具。

- 应用场景
  车辆检测、行人检测及图片搜索等。
- 数据结构

- 输入数据
  manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

```
{"data":{"source":"oss://****.oss-cn-hangzhou.aliyuncs.com/pic_ocr/img17.jpeg"}}
...
```

- 输出数据
  manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "oss://****.oss-cn-hangzhou.aliyuncs.com/pic_ocr/img17.jpeg"
    },
    "label-144853549785619****": {
        "results": [
            {
                "questionId": "1",
                "data": [
                    {
                        "id": "e02a574b-9fd9-45e9-8c8a-9682567b****",
                        "type": "image/polygon",
                        "value": [
                            [
                                499.93454545454546,
                                255.0981818181818
                            ],
                            [
                                911.0109090909091,
                                255.0981818181818
                            ],
                            [
                                911.0109090909091,
                                338.6836363636363
                            ],
                            [
                                499.93454545454546,
                                338.6836363636363
                            ]
                        ],
                        "labels": {
                            "单选": "标签1"
                        }
                    }
                ],
                "rotation": 0,
                "markTitle": "目标检测标签配置",
                "width": 1024,
                "type": "image",
                "height": 1024
            }
        ]
    }
}
```

图像分类

图像分类（Image Classification）是指从一组固定的分类标签集合中，找到与输入图像内容相匹配的一个或多个分类标签，并将其分配给该输入图像。该模板支持单标签和多标签图像分类。

- 应用场景
  图片分类、拍照识图、图片搜索及内容推荐等。
- 数据结构
  - 输入数据
    manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

```
{"data":{"source":"oss://****.oss-cn-hangzhou.aliyuncs.com/iTAG/pic/1.jpg"}}
...
```

  - 输出数据
    manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "oss://****.oss-cn-hangzhou.aliyuncs.com/pic/3.jpg"
    },
    "label-143082452899667****": {
        "results": [
            {
                "questionId": "2",
                "data": [
                    "标签1",
                    "标签2"
                ],
                "markTitle": "多选",
                "type": "survey/multivalue"
            }
        ]
    }
}
```

4.2.6.4. 视频类

智能标注（iTAG）提供了视频分类标注模板，创建标注任务时，您需要根据应用场景选择标注模板。本文为您介绍视频类标注模板的应用场景及数据结构。

视频分类

视频分类（Video Classification）是指在一组固定的分类标签集合中，找出与输入视频内容相匹配的一个或多个分类标签，并将其分配给该输入视频。该模板支持单标签和多标签分类。

- 应用场景
  视频监控、直播推荐及短视频推荐等。
- 数据结构

- 输入数据
  manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

```
{"data":{"source":"oss://tongxin-lly.oss-cn-hangzhou.aliyuncs.com/iTAG/video/video1.mp4
"}}
...
```

- 输出数据
  manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "oss://itag.oss-cn-hangzhou.aliyuncs.com/tongxin_video/video3.mp4"
    },
    "label-1433000711494508544": {
        "results": [
            {
                "questionId": "2",
                "data": [
                    "标签2",
                    "标签1"
                ],
                "markTitle": "多选",
                "type": "survey/multivalue"
            }
        ]
    }
}
```

4.2.6.5. 语音类

智能标注（iTAG）提供了音频分类、音频分割、音频识别的语音类标注模板，创建标注任务时，您需要根据应用场景选择标注模板。本文为您介绍语音类模板的应用场景及数据结构。

背景信息

本文介绍以下语音类标注模板的数据结构：

- 音频分类
- 音频分割
- 音频识别

音频分类

音频分类（Audio Classification）是指从一组固定的分类标签集合中，找到与输入音频内容相匹配的一个或多个分类标签，并将其分配给该输入音频。该模板支持单标签和多标签音频分类。

- 应用场景
  场景声音分类等。
- 数据结构
  - 输入数据
    manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

```
{"data":{"source":"oss://tongxin-lly.oss-cn-hangzhou.aliyuncs.com/iTAG/audio/1.wav"}}
...
```

- 输出数据
  manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "oss://itag.oss-cn-hangzhou.aliyuncs.com/tongxin_audio/6.wav"
    },
    "label-1432993193909231616": {
        "results": [
            {
                "questionId": "1",
                "data": "标签1",
                "markTitle": "单选",
                "type": "survey/value"
            }
        ]
    }
}
```

音频分割

音频分割（Audio segmentation）是指将一段音频通过识别后，利用波形图将音频分割成多段，并匹配上不同的标签内容。

- 应用场景
  对话内容分析等。
- 数据结构
  - 输入数据
    manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

```
{"data":{"source":"oss://tongxin-lly.oss-cn-hangzhou.aliyuncs.com/iTAG/audio/1.wav"}}
...
```

  - 输出数据
    manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "oss://itag.oss-cn-hangzhou.aliyuncs.com/tongxin_audio/21.wav"
    },
    "label-1435480301706092544": {
        "results": [
            {
                "duration": 0,
                "objects": [
                    {
                        "result": {
                            "音频识别结果": "通过音频识别得到的结果内容1。",
                            "单选": "标签1"
                        },
                        "color": null,
                        "id": "wavesurfer_ei0aet9uvp8",
                        "start": 2.3886218302094817,
                        "end": 4.635545755237045
                    },
                    {
                        "result": {
                            "音频识别结果": "通过音频识别得到的结果内容2。",
                            "单选": "标签2"
                        },
                        "color": null,
                        "id": "wavesurfer_kl39gnlb2k",
                        "start": 5.698280044101433,
                        "end": 7.348048511576626
                    }
                ],
                "empty": false
            }
        ]
    }
}
```

音频识别

音频识别ASR是指将一段音频识别为文本内容，同时可以进行相应标签的匹配。

- 应用场景
  方言识别等。
- 数据结构
  - 输入数据
    manifest文件的每行数据是一道题目，且每行数据必须包含source字段。

```
{"data":{"source":"oss://tongxin-lly.oss-cn-hangzhou.aliyuncs.com/iTAG/audio/1.wav"}}
...
```

- 输出数据
  manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "source": "oss://itag.oss-cn-hangzhou.aliyuncs.com/tongxin_audio/14.wav"
    },
    "label-1435448359497441280": {
        "results": [
            {
                "questionId": "1",
                "data": "通过音频识别得到的结果内容。",
                "markTitle": "音频识别结果",
                "type": "survey/value"
            },
            {
                "questionId": "3",
                "data": [
                    "标签1",
                    "标签2"
                ],
                "markTitle": "多选",
                "type": "survey/multivalue"
            }
        ]
    }
}
```

4.2.6.6. 自定义模板

如果智能标注（iTAG）提供的标注模板无法满足您的需求，创建标注任务时，您也可以选择自定义模板。本文介绍自定义模板的数据结构。

- 应用场景
  您特定的标注场景。
- 数据结构
  - 输入数据
    manifest文件的每行数据是一道题目，且每行数据的字段名不限制，在创建任务时，您可以自由选择字段。

```
{"data":{"picture_url":"oss://tongxin-lly.oss-cn-hangzhou.aliyuncs.com/iTAG/pic/13.jpg"
,"text":"阿里巴巴集团庆祝创立十周年，同时成立阿里云计算"}}
...
```

  - 输出数据
    manifest文件的每行数据由题目和标注结果一起生成。每行数据的JSON结构如下。

```
{
    "data": {
        "picture_url": "oss://itag.oss-cn-hangzhou.aliyuncs.com/tongxin_pic/13.jpg",
        "text": "阿里巴巴集团庆祝创立十周年，同时成立阿里云计算"
    },
    "label-1435504604476547072": {
        "results": [
```

```
    {
        "questionId": "1",
        "data": [
            {
                "id": "e0755602-b90f-4d15-a11d-8951e001****",
                "type": "image/polygon",
                "value": [
                    [
                        281.0181818181821,
                        451.49090909090916
                    ],
                    [
                        281.0181818181821,
                        1116.7636363636366
                    ],
                    [
                        829.927272727273,
                        1116.7636363636366
                    ],
                    [
                        829.927272727273,
                        451.49090909090916
                    ]
                ],
                "labels": {
                    "boundingbox": "标签2"
                }
            }
        ],
        "rotation": 0,
        "markTitle": "框选题-IMG-3",
        "width": 960,
        "type": "image",
        "height": 960
    },
    {
        "objects": [
            {
                "result": {
                    "NER": [
                        "标签7"
                    ]
                },
                "color": null,
                "id": null,
                "text": "集团庆祝",
                "start": 4,
                "end": 7
            }
        ],
        "empty": false
    },
    {
        "questionId": "3",
        "data": "这里是填空题的答案"
```

```
                "data": "这里是填空题的答案",
                "markTitle": "文本内容",
                "type": "survey/value"
            },
            {
                "questionId": "4",
                "data": "标签222",
                "markTitle": "单选题",
                "type": "survey/value"
            }
        ]
    }
}
```

5.开发参考与工具

5.1. API参考

5.2. SDK参考

5.2.1. 图像视频分析

5.2.1.1. PAI-EasyVision简介

PAI-EasyVision（视觉智能增强算法包）提供多种模型的训练及预测功能，旨在帮助计算机视觉应用开发者方便快捷地构建视觉模型并应用于生产。
随着深度学习技术的快速发展，计算视觉技术已经跨入大规模商业化应用阶段。对于视觉AI应用开发者而言，熟练地运用深度学习CV建模技术存在较高门槛，主要体现在以下几个方面：

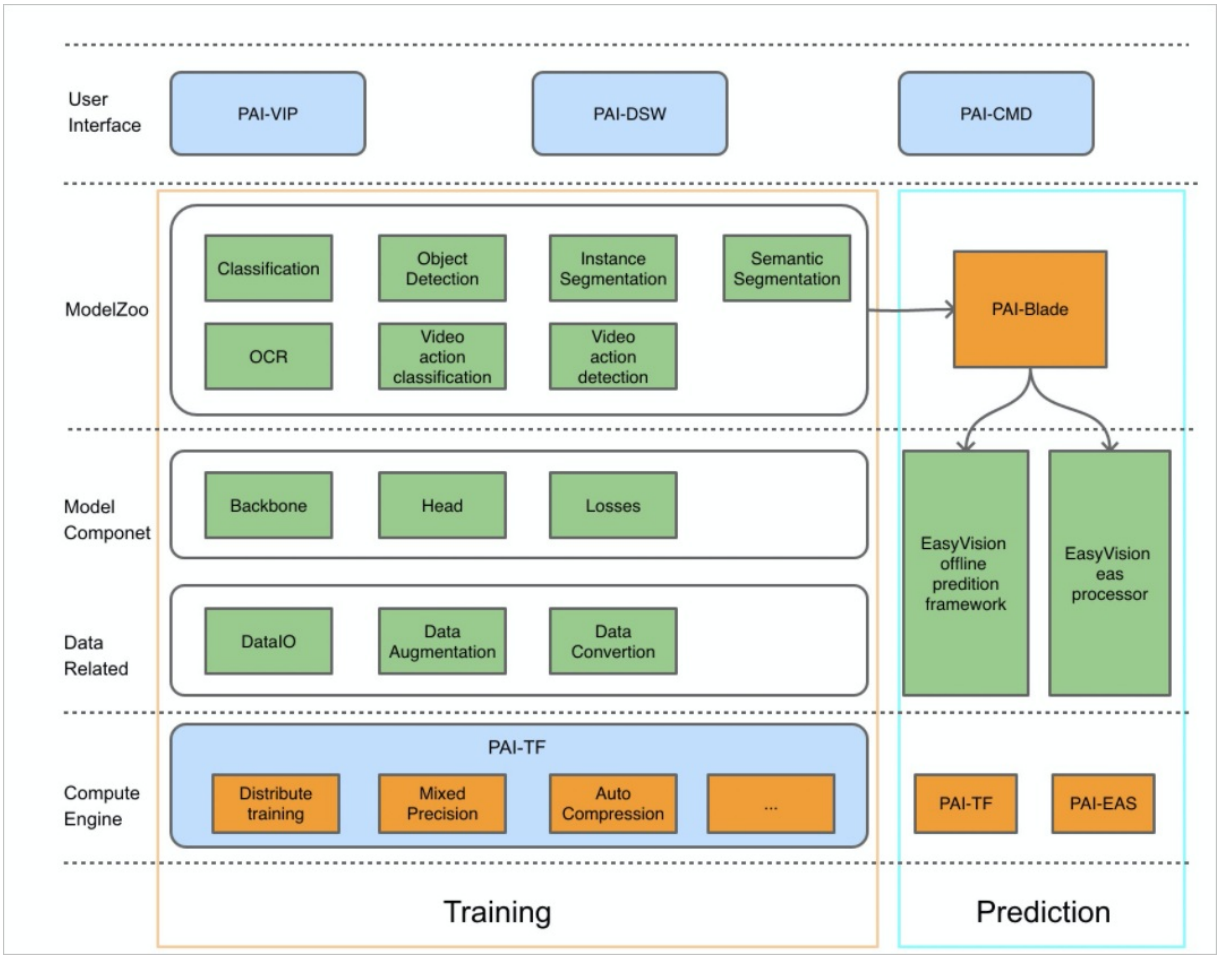
- 深度学习算法代码开发成本高，对大量细节进行Debug的代价很高。
- 模型更新迭代快，理解其原理和细节需要花费大量时间。
- 算法训练和推理性能优化都需要专业的系统知识。
- 数据标注成本太高。
- 在PAI上直接使用开源算法存在一定的学习和改造成本。

为此，PAI推出了一套方便且易用的CV建模框架PAI-EasyVision，旨在帮助CV应用开发者快捷地构建视觉模型并应用于生产。PAI-EasyVision核心能力体现在框架易用性、性能及模型丰富度方面：

- 易用性方面
  针对视觉任务的多样性，PAI-EasyVision支持面向多任务、模块化及可插拔的原子化功能接口，其功能涵盖了数据IO、预处理、训练及离线预测的完整建模流程。同时，您可以在PAI-Studio或PAI-DSW等多种环境中使用PAI-EasyVision。
- 性能方面
  算法封装了PAI-TF的多种优化引擎，包括分布式训练、编译优化及混合精度等，您通过简单的配置文件即可在PAI中享受极致的性能体验。同时，兼容在开源TF中使用PAI-EasyVision。
- 模型丰富度方面
  提供了大量在开源数据集上训练完成的模型，且集成了PAI中优秀的模型（例如OCR模型），进而降低开发和训练成本。

架构

PAI-EasyVision在Model Zoo基础上进行了大量的模型扩充，提供多种模型的训练预测能力，且支持PAI-VIP、PAI命令及PAI-DSW多种灵活调用方式，以满足各层次用户的建模需求。PAI-EasyVision灵活高可用的分布式流水线离线预测架构，支持上亿级别数据的快速离线处理。 同时，基于PAI的系统优化和模型优化功能，使得训练模型更小、更快地在PAI-EAS上进行预测。 此外，PAI-EasyVision支持自定义训练预测接口，以便复用已有的功能和优化工作。PAI-EasyVision的具体架构如下所示。



特性

- 易用性
  考虑到用户分层，有些用户希望通过简单的交互操作完成模型训练，有的用户希望定时调度模型训练和预测任务，有的用户希望复用PAI-EasyVision已有模块，在此基础上进行模型结构调整，再重新训练。因此，PAI-EasyVision支持通过PAI-VIP、PAI命令或PAI-DSW方式调用。
- 性能优化
  依托PAI-TF团队进行了分布式训练性能优化，支持高性能的单机多卡、多机多卡分布式运行方式。同时支持对模型进行Inference阶段优化，包括图优化及模型压缩等方式。
- 对接PAI标记平台
  PAI-EasyVision对接PAI标记平台，您可以通过提供的转换工具，方便地将PAI标记格式文件转换为TFRecord，从而进行相关任务训练。此外，PAI-EasyVision提供了丰富的数据增强模块，用于在训练时动态扩充训练数据。
- 高效的离线预测
  PAI-EasyVision提供多机流水线的预测系统，便于将PAI-EasyVision训练的模型进行离线数据处理。每个处理过程支持多机多线程加速，且各个过程异步流水线处理，极大提高了处理效率。此外，离线预测支持用户自定义各个处理过程。

- 对接在线服务平台PAI-EAS
  训练过程会产出SaveModel，用户可以自行接入原有的在线预测业务系统。 同时，PAI-EAS提供了强大的在线预测服务能力，实现了PAI-EasyVision EAS Python Processor，用户只需要在配置文件中配置模型地址及模型类别信息，即可进行实时数据处理。

5.2.1.2. 标注文件格式说明

5.2.1.2.1. 标注文件格式说明

如果您需要根据已有数据生成TFRecord，则可以先将已有文件转换为PAI标注格式，再生成TFRecord。本文为您介绍标注文件格式。
CSV数据格式如下。

字段	数据类型	描述
数据ID	INT	数据标识
原始数据	JSON	包含图片URL
融合答案	JSON	标注结果

图像分类

```
#数据ID,原始数据,融合答案
1,"{""url"":""http://a.jpg""}","{""option"":""护照""}"
2,"{""url"":""http://b.jpg""}","{""option"":""护照""}"
```

融合答案字段说明如下。

```
{
    "option":"护照"     # 图片类别。
}
```

图像多标签分类

```
#数据ID,原始数据,融合答案
1,"{""url"":""http://a.jpg""}","{[""option"":""护照"", ""option"":""身份证""]}"
2,"{""url"":""http://b.jpg""}","{[""option"":""护照"", ""option"":""港澳通行证""]}"
```

融合答案字段说明如下。

```
{
    "option":["护照", "身份证"]     # 图片标签。
}
```

物体检测

```
#数据ID,原始数据,融合答案
1,"{""url"": ""http://b.jpg""}","[{""text"": ""{\""class*\"": \""类别1\""}"", ""coord"": [""306.73"", ""517.59"", ""324.42"", ""282.07"", ""347.69"", ""282.07"", ""333.73"", ""519.45""]}, {""text"": ""{\""class*\"": \""类别2\""}"", ""coord"": [""342.11"", ""723.32"", ""349.56"", ""608.81"", ""366.31"", ""606.95"", ""360.73"", ""730.76""]}]"
2,"{""url"": ""http://a.jpg""}","[{""text"": ""{\""class*\"": \""类别1\""}"", ""coord"": [""338.35"", ""8.53"", ""700.16"", ""8.53"", ""700.16"", ""50.35"", ""338.35"", ""50.35""]}, {""text"": ""{\""class*\"": \""类别2\""}"", ""coord"": [""26.88"", ""64.00"", ""218.03"", ""64.00"", ""218.03"", ""99.84"", ""26.88"", ""99.84""]}]"
```

融合答案字段说明如下。

```
[         # 物体列表。
    {
        "text":"{\"class*\": \"类别1\"}",     # 物体类别，JSON字符串。
        "coord":[     # 物体包围盒顶点坐标，x1,y1,x2,y2,x3,y3,x4,y4。
            "338.35",
            "8.53",
            "700.16",
            "8.53",
            "700.16",
            "50.35",
            "338.35",
            "50.35"
        ]
    },
    {
        "text":"{\"class*\": \"类别2\"}",
        "coord":[
            "26.88",
            "64.00",
            "218.03",
            "64.00",
            "218.03",
            "99.84",
            "26.88",
            "99.84"
        ]
    }
]
```

图像分割

示例下载

```
#数据ID,原始数据,融合答案
1,"{""http://a.jpg""}","{""ossUrl"":""http://ossgw.alicdn.com/a.png""}"
```

融合答案字段说明如下。

```
{
    "ossUrl":"http://ossgw.alicdn.com/a.png"
    # Mask图片地址。Mask图片为PNG格式，第2个通道(0开始)保存类别信息，
    # 取值从0到num_class-1。第一类通常为Background。
}
```

文字识别

```
#数据ID,原始数据,融合答案
1,"{""url"": ""http://b.jpg""}","{""text"": ""文本1""}"
2,"{""url"": ""http://a.jpg""}","{""text"": ""文本2""}"
```

融合答案字段说明如下。

```
{
    "text":"文本1"     # 文字识别内容。
}
```

文字检测

```
#数据ID,原始数据,融合答案
1,"{""url"": ""http://b.jpg""}","[[{""text"": ""{\""direction\"": \""底部朝右\"", \""class*\
"": \""类别1\""}"", ""coord"": [""306.73"", ""517.59"", ""324.42"", ""282.07"", ""347.69"",
""282.07"", ""333.73"", ""519.45""]}, {""text"": ""{\""direction\"": \""底部朝右\"", \""clas
s*\"": \""类别2\""}"", ""coord"": [""342.11"", ""723.32"", ""349.56"", ""608.81"", ""366.31"
", ""606.95"", ""360.73"", ""730.76""]}], {""option"": ""底部朝右""}]"
2,"{""url"": ""http://a.jpg""}","[[{""text"": ""{\""direction\"": \""底部朝下\"", \""class*\
"": \""类别1\""}"", ""coord"": [""338.35"", ""8.53"", ""700.16"", ""8.53"", ""700.16"", ""50
.35"", ""338.35"", ""50.35""]}, {""text"": ""{\""direction\"": \""底部朝下\"", \""class*\"":
\""类别2\""}"", ""coord"": [""26.88"", ""64.00"", ""218.03"", ""64.00"", ""218.03"", ""99.84
"", ""26.88"", ""99.84""]}], {""option"": ""底部朝下""}]"
```

融合答案字段说明如下。

```
[         # 文字行列表。
    [
        {
            "text":"{\"direction\": \"底部朝下\", \"class*\": \"类别1\"}",
                        # 文字行标注，JSON字符串。其中direction表示文字行朝向，class*表示类别。
            "coord":[ # 文字行包围盒顶点坐标，x1,y1,x2,y2,x3,y3,x4,y4。
                "338.35",
                "8.53",
                "700.16",
                "8.53",
                "700.16",
                "50.35",
                "338.35",
                "50.35"
            ]
        },
        {
            "text":"{\"direction\": \"底部朝下\", \"class*\": \"类别2\"}",
            "coord":[
                "26.88",
                "64.00",
                "218.03",
                "64.00",
                "218.03",
                "99.84",
                "26.88",
                "99.84"
            ]
        }
    ],
    {
        "option":"底部朝下"     # 整体图片朝向。
    }
]
```

图片朝向分为底部朝下、底部朝上、底部朝左及底部朝右。

端到端的文字识别

```
#数据ID,原始数据,融合答案
1,"{""url"": ""http://b.jpg""}","[[{""text"": ""{\""text\"": \""文本1\"", \""direction\"": \
""底部朝右\"", \""class*\"": \""类别1\""}"", ""coord"": [""306.73"", ""517.59"", ""324.42"",
""282.07"", ""347.69"", ""282.07"", ""333.73"", ""519.45""]}, {""text"": ""{\""text\"": \""
文本2\"", \""direction\"": \""底部朝右\"", \""class*\"": \""类别2\""}"", ""coord"": [""342.11
"", ""723.32"", ""349.56"", ""608.81"", ""366.31"", ""606.95"", ""360.73"", ""730.76""]}],
{""option"": ""底部朝右""}]"
2,"{""url"": ""http://a.jpg""}","[[{""text"": ""{\""text\"": \""文本3\"", \""direction\"": \
""底部朝下\"", \""class*\"": \""类别1\""}"", ""coord"": [""338.35"", ""8.53"", ""700.16"", ""
8.53"", ""700.16"", ""50.35"", ""338.35"", ""50.35""]}, {""text"": ""{\""text\"": \""文本4\"
", \""direction\"": \""底部朝下\"", \""class*\"": \""类别2\""}"", ""coord"": [""26.88"", ""64
.00"", ""218.03"", ""64.00"", ""218.03"", ""99.84"", ""26.88"", ""99.84""]}], {""option"":
""底部朝下""}]"
```

融合答案字段说明如下。

```
[         # 文字行列表。
    [
        {
            "text":"{\"text\": \"文本3\", "\"direction\": \"底部朝下\", \"class*\": \"类别1\"
}",
                            # 文字行标注，JSON字符串。其中direction表示文字行朝向，class*表示类别。

            "coord":[ # 文字行包围盒顶点坐标，x1,y1,x2,y2,x3,y3,x4,y4。
                "338.35",
                "8.53",
                "700.16",
                "8.53",
                "700.16",
                "50.35",
                "338.35",
                "50.35"
            ]
        },
        {
            "text":"{\"text\": \"文本4\", \"direction\": \"底部朝下\", \"class*\": \"类别2\"}
",
            "coord":[
                "26.88",
                "64.00",
                "218.03",
                "64.00",
                "218.03",
                "99.84",
                "26.88",
                "99.84"
            ]
        }
    ],
    {
        "option":"底部朝下"     # 整体图片朝向。
    }
]
```

图片朝向分为底部朝下、底部朝上、底部朝左及底部朝右。

5.2.1.2.2. TFRecord数据转换

PAI提供多种数据转换功能，可以将图片文件快速转换为TFRecord文件，从而使用训练组件进行模型训练。如果通过PAI智能标注平台进行数据标注，则系统会自动生成标记结果文件，您可以直接调用数据转换组件生成TFRecord文件。 如果通过其他平台进行数据标注，则需要使用PAI命令将标记文件转换为PAI标注文件。

图片分类或图片多标签分类

使用PAI命令将图片分类或图片多标签分类的标记文件转换为TFRecord的示例如下。

```
pai -name easy_vision_ext
      -Dbuckets='oss://{bucket_name}.{oss_host}/{path}/'
      -Darn='acs:ram::*******:role/aliyunodpspaidefaultrole'
      -DossHost='{oss_host}'
      -Dcmd  convert
      -Dlabel_file 'oss://{bucket_name}/path/to/your/{label_file}'
      -Dconvert_param_config '
        --class_list_file oss://{bucket_name}/path/to/your/{class_list_file}
        --max_image_size 600
        --write_parallel_num 8
        --num_samples_per_tfrecord 128
        --test_ratio 0.1
        --model_type CLASSIFICATION
      '
      -Doutput_tfrecord 'oss://{bucket_name}/path/to/output/data_prefix'
      -Dcluster='{
              \"worker\" : {
                \"count\" : 1,
                \"cpu\" : 800
              }
            }'
```

文字检测识别

使用PAI命令将文字检测识别的标记文件转换为TFRecord的示例如下。

```
pai -name easy_vision_ext
      -Dbuckets='oss://{bucket_name}.{oss_host}/{path}/'
      -Darn='acs:ram::*******:role/aliyunodpspaidefaultrole'
      -DossHost='{oss_host}'
      -Dcmd  convert
      -Dlabel_file 'oss://{bucket_name}/path/to/your/{label_file}'
      -Dconvert_param_config '
        --model_type TEXT_END2END
        --default_class text
        --max_image_size 2000
        --char_replace_map_path oss://{bucket_name}/path/to/your_char_replace_map
        --default_char_dict_path oss://{bucket_name}/path/to/your_char_dict
        --test_ratio 0.1
        --write_parallel_num 8
        --num_samples_per_tfrecord 64
      '
      -Doutput_tfrecord 'oss://pai-vision-data-hz/test/convert/recipt_text_end2end/data'
```

命令参数说明

参数	是否必选	描述	参数值格式	默认值
cmd	是	必须配置为convert。	STRING	convert

参数	是否必选	描述	参数值格式	默认值
buckets	否	可以输入多个Bucket，以英文逗号（,）分隔，每个Bucket必须须以正斜线（/）结尾。	“oss://bucket_name/?role_arn=xxx&host=yyy” “oss://bucket_1/?role_arn=xxx&host=yyy,oss://bucket_2/”	空
label_file	是	PAI标注文件的OSS路径，具体格式说明请参见标注文件格式说明。	oss://your_bucket/xxx.csv	无
convert_param_config	否	转换配置参数，详情请参见下方表格。该参数与convert_config参数选择其一即可。	—paramavaluea —parambvalueb	” ”
output_tfrecord	否	输出TFRecord前缀。	oss://your_dir/prefix	” ”
cluster	否	分布式转换参数配置。	JSON格式字符串	“{\” worker\” :{\” count\” :3, \” cpu\” : 800, \” gpu\” :0, \” memory\” : 20000}}”

convert_param_config参数说明如下。

参数	是否必选	描述	参数值格式	默认值

参数	是否必选	描述	参数值格式	默认值
model_type	是	转换数据用于何种模型训练，取值包括： • CLASSIFICATION：图像分类或多标签 • DETECTION：物体检测 • SEGMENTATION：语义分割 • INSTANCE_SEGMENTATION：Instance分割 • TEXT_END2END：端到端OCR • TEXT_RECOGNITION：单行文字识别 • TEXT_DETECTION：文字检测 • VIDEO_CLASSIFICATION：视频分类 • SELF_DEFINED：自定义转换 **说明** model_type取值为TEXT_END2END或TEXT_RECOGNITION时，char_replace_map_path和default_char_dict_path参数生效。model_type取值为VIDEO_CLASSIFICATION时，decode_type sample_fps、reshape_size、decode_batch_size及decode_keep_size参数生效。	STRING	无
class_list_file	否	类别列表文件路径，文件内容每行格式为类别名或类别名:映射类别名。	oss://path/to/your/classlist	,,
test_ratio	否	测试数据分割比例。如果取值为0，则所有数据转换为训练数据。如果取值为0.1，则表示10%的数据作为验证集。	0.1	0.1
max_image_size	否	图片最大边限制。如果配置了该参数，则大图片会被Resize后存入TFRecord，从而节省存储、提高数据读取速度。	INT	None，即不指定该参数
max_test_image_size	否	同max_image_size，用于配置测试数据。	INT	${max_image_size}
default_class	否	默认类别名称，在class_list中未找到的类别均会映射到该名称。	STRING	None
error_class	否	错误类别名称，含有该类别的物体和Box会被过滤，不参与训练。	STRING	None，即不指定该参数
ignore_class	否	忽略类别名称，只用于检测模型，含有该类别的Box在训练中会被忽略。	STRING	None，即不指定该参数

参数	是否必选	描述	参数值格式	默认值
converter_class	否	转换类名称。	STRING	QinceConverter
seperator	否	分隔符，用于标记内容的Split。	STRING	None，即不指定该参数
image_format	否	TFRecord中图片的编码方式。	STRING	jpg
read_parallel_num	否	读取并发数。	INT	10
write_parallel_num	否	写TFRecord并发数。	INT	1
num_samples_per_tfrecord	否	每个TFRecord保存图片数。	INT	256
user_defined_converter_path	否	自定义converter代码路径，支持HTTP或OSS路径。例如http://path/to/your/converter.py。	STRING	无
user_defined_generator_path	否	自定义generator代码路径，支持HTTP或OSS路径。例如http://path/to/your/generator.py。	STRING	无
generator_class	否	自定义generator类名	STRING	无
char_replace_map_path	否	字符映射替换文件。格式为CSV文件，包含如下两列： • original：原字符串。 • replaced：替换字符串，即使用replaced替换original。	STRING	None，即不指定该参数。
default_char_dict_path	否	字符到ID映射文件路径，每一行是一个字符， 第k行的字符ID为k-1	STRING	None，即不指定该参数。
decode_type	否	视频解码方式。取值范围为： • 1：Intra only • 2：Keyframe only • 3：Without bidir • 4：Decode all	INT	4
sample_fps	否	抽帧频率。	FLOAT	5
reshape_size	否	输出帧的大小。	INT	224

参数	是否必选	描述	参数值格式	默认值
decode_batch_size	否	每次Decode步骤中的Batch大小。	INT	10
decode_keep_size	否	不同Batch中Overlap的帧数量。	INT	0

5.2.2. TensorFlow使用指南

5.2.2.1. PAI-TF概述

PAI-TF是阿里云计算平台PAI为了追求更极致的深度学习训练效率，优化原生Tensorflow的内核并开发周边工具，推出的一款产品。PAI-TF拥有服务化、分布式调度、全局计算调度、GPU卡映射及模型在线预测等特点。

背景

Tensorflow是Google最新的开源深度学习计算框架，支持CNN、RNN及LSTM等多种神经网络模型，对语音、图像及文本等领域的模型训练效率极佳。Tensorflow的功能丰富且强大，并拥有高度灵活的API，受到业界的高度关注。

PAI-TF是阿里云计算平台PAI为了追求更极致的深度学习训练效率，优化原生Tensorflow的内核并开发周边工具，推出的一款产品。PAI-TF完全兼容原生Tensorflow的代码，并且在许多工业化生产场景的性能更加优越。目前，PAI-TF已经在阿里云机器学习PAI、阿里云E-MapReduce等产品上线并应用。

产品特点

PAI-TF产品的特点如下：

- 服务化
  MaxCompute是阿里云自主研发的飞天大数据平台，已经支持了数万企业及个人开发者。PAI-TF帮助您直接在MaxCompute中使用TensorFlow的计算框架。PAI-TF使用的API与开源版本一致，您可以直接通过TensorFlow Training Script接口提交作业至MaxCompute的计算集群中执行。
- 分布式调度
  PAI为您提供海量的计算资源，所有的计算资源通过GPU Quota进行管理。PAI-TF的作业都是基于底层的分布式调度系统动态调度至不同机器。当您提交PAI-TF作业时，无需担心是否需要提前申请GPU物理主机，PAI-TF所需要的GPU资源随作业的提交动态分配，随作业的结束动态释放。
- 全局计算调度
  当您在使用MaxCompute计算引擎时，您可以在一个项目中同时提交SQL作业和PAI-TF作业。
  MaxCompute全局计算调度服务能够将PAI-TF作业自动调度至相应的GPU集群，并将基于CPU集群的数据预处理作业和基于GPU集群的模型训练作业连接起来。
- GPU卡映射
  PAI-TF支持将不同算子（Operators）指定至特定的CPU或GPU上。基于GPU卡映射，您无需感知宿主机的GPU卡物理结构，PAI-TF会将您作业中申请的GPU卡自动映射至作业进程空间，则您感知到的GPU卡为gpu:0、gpu:1....等。
- 模型在线预测
  PAI为您提供了在线预测服务PAI-EAS。您可以将PAI-TF中训练生成的模型一键部署至在线预测服务。在线预测服务支持模型的动态扩容、滚动更新、A/B测试、高吞吐及低延时等特性。

支持的Python三方库

PAI-TF已经安装了Numpy及Six等常见的Python三方库，您可以在TensorFlow作业中直接导入相关的库。

5.2.2.2. PAI-TF调用方式

PAI-TF是深度学习计算框架，支持多种模型训练。您可以使用PAI-Studio、MaxCompute Console以及DataWorks的开发节点调用PAI-TF。

使用限制

目前仅北京和上海两个地域支持调用PAI-TF。

使用PAI-Studio调用PAI-TF

1. 登录PAI Console，在左侧导航栏，单击**模型开发和训练 > Studio-可视化建模**，进入**PAI可视化建模**页面。

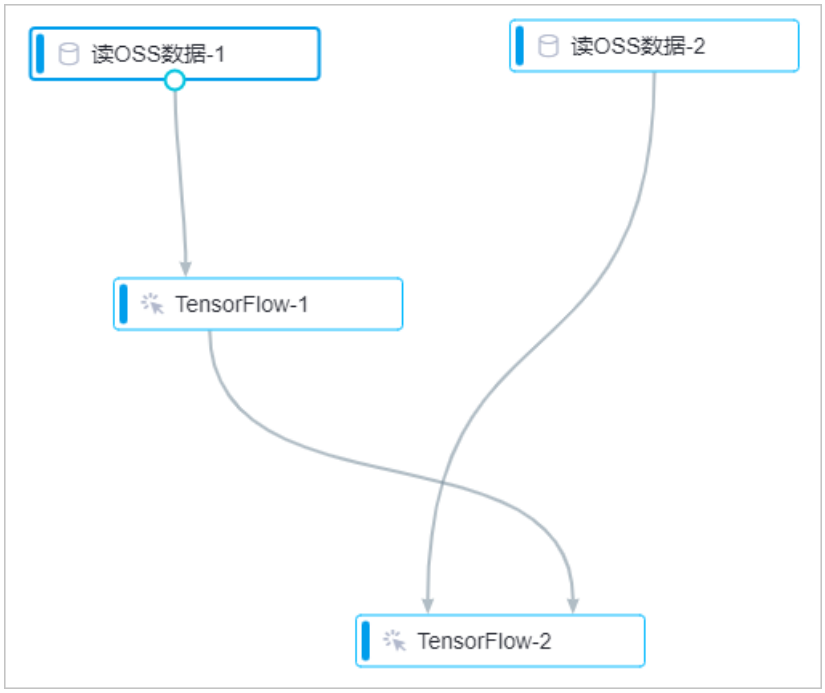
   创建项目时，建议您使用按量付费模式（后付费），并开启GPU，PAI-TF任务只能在GPU资源中运行。

2. 单击目标项目**操作**列的**进入机器学习**。
3. 在**机器学习PAI**页面的左侧导航栏，单击**首页**。
4. 单击**模板列表 > Tensorflow图片分类**的**从模板创建**。

   模板已经内置了训练代码和数据，您可以通过模板快速了解PAI-TF的用法。
   模板案例是基于开源图像数据cifar10的图像分类案例，您可以从CIFAR 10案例中获取数据和代码。

5. 配置**新建试验**对话框的**名称**、**描述**以及**位置**参数。
6. 单击**确定**，进入实验页面。

   实验流程如下图所示。

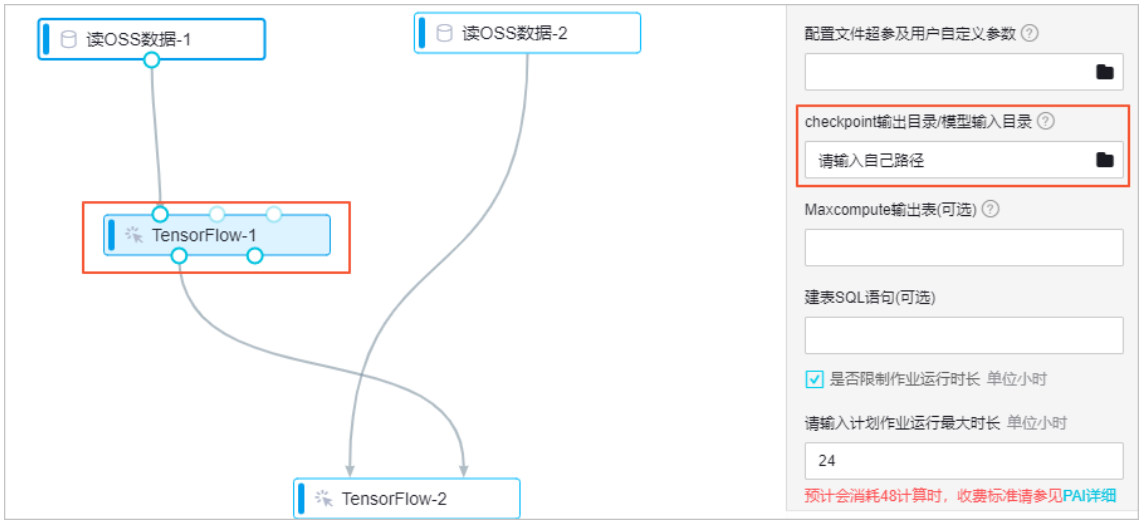


说明如下：

- 您可以使用OSS或MaxCompute表作为数据源，详情请参见PAI-TF数据IO方式介绍。
- 执行任务前，您需要使用主账号授予PAI读取OSS数据的权限。步骤如下：
  a. 在**机器学习PAI**页面左侧导航栏，单击**设置**。
  b. 在**基本设置**区域，勾选**授权机器学习读取我的OSS中的数据**。

需要主账号登录，并在设置页面进行OSS授权

- 如果您使用的是实验模板，您需要修改TensorFlow的checkpoint为自己OSS的checkpoint即可，如下图所示。



训练组件参数如下表所示。

页签	参数	描述
参数设置	Tensorflow版本	您可以根据代码选择合适的Tensorflow版本。
	Python代码文件	您需要将执行的代码放至OSS路径下。如果是工程文件，则需要使用tar.gz格式的压缩包。 **说明** OSS需要与当前项目位于同一区域。
	Python主文件	如果代码文件使用的tar.gz压缩包，则需要指定入口Python文件。
	OSS数据源	输入的OSS地址。
	配置文件超参及用户自定义参数	您可以单击文件夹，在OSS Bucket上选择或上传文件。
	checkpoint输出目录/模型输入目录	选择自己的OSS路径用来存放模型。
	MaxCompute输出表	MaxCompute输出表必须是已经创建的表，并且输出的表名称需要与代码中的输出表名称一致。
	建表SQL语句	如果代码中的输出表不存在，则您可以通过输入框输入建表语句创建表。 建表语句会在TF脚本执行前执行，示例建表语句为 `create table iris_output(f1 DOUBLE,f2 DOUBLE,f3 DOUBLE,f4 DOUBLE,f5 STRING);` 。

页签	参数	描述
	请输入计划作业运行最大时长	运行作业的最大时长。

分布式参数如下表所示。

页签	参数	描述
执行调优	**单机或分布式**	计算的机器数量。
	指定worker GPU卡数	每个worker的GPU卡数。 例如，worker个数为3，指定worker GPU卡数为2，则总卡数为 `3*2=6` 。
	指定worker个数	分布式计算的机器数量。
	指定ps个数	参数服务器的个数，通常不超过worker个数的50%。

使用MaxCompute Console调用PAI-TF

MaxCompute是阿里云自研的大数据计算平台，PAI是基于MaxCompute建设的人工智能平台，PAI-TF可以在MaxCompute上计算运行，本小节为您介绍如何基于MaxCompute Console工具运行PAI任务。

您需要购买PAI的后付费服务，才可以使用MaxCompute Console方式运行PAI-TF任务。

MaxComupute Console是一个可以被封装的命令行工具，您可以下载并安装，详情请参见MaxCompute客户端（odpscmd）。配置好环境后，您可以进入命令行工具，输入PAI-TF命令运行脚本，详情请参见PAI-TF任务参考文档。

使用DataWorks开发节点调用PAI-TF

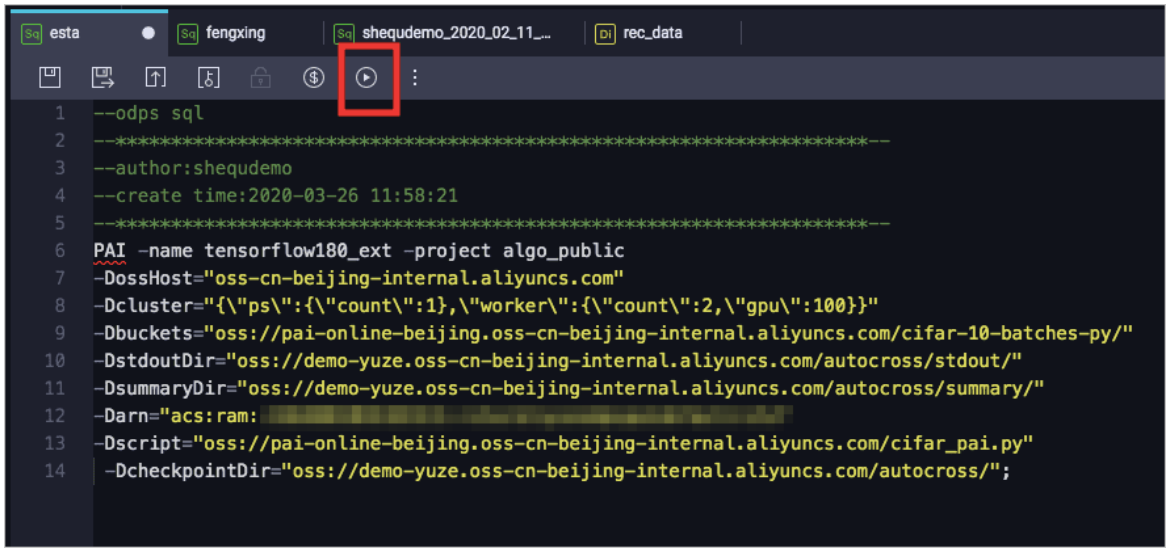
1. 登录DataWorks管理控制台。
2. 单击目标工作空间**操作**列的**进入数据开发**。

   DataWorks与PAI-Studio公用项目，创建项目时需要开通底层MaxCompute引擎，并选择后付费模式。
3. 在**数据开发**页面，单击图标，选择**MaxCompute > ODPS SQL**。
4. 配置**新建节点**对话框的**节点类型**、**节点名称**以及**目标文件夹**参数。

新建节点之前，您需要先创建业务流程，详情请参见创建业务流程。

5. 复制需要执行的PAI命令至SQL编辑框，并单击图标运行。

   示例如下。



5.2.2.3. PAI-TF任务参数介绍

PAI-TF是深度学习计算框架，支持多种模型训练。本文为您介绍PAI-TF任务涉及的命令参数和相关IO参数。

PAI-TF命令参数

您可以在Dataworks SQL节点及MaxCompute的命令行客户端使用PAI命令发起任务，PAI命令及其参数如下。

```
# 实际使用中，需要根据业务需求设置参数。
pai -name tensorflow1120_ext
    -project algo_public
    -Dscript= 'oss://<bucket_name>.<oss_host>.aliyuncs.com/*.tar.gz'
    -DentryFile='entry_file.py'
    -Dbuckets='oss://<bucket_name>.<oss_host>.aliyuncs.com/<path>'
    -Dtables='odps://prj_name/tables/table_name'
    -Doutputs='odps://prj_name/tables/table_name'
    -DcheckpointDir='oss://<bucket_name>.<oss_host>.aliyuncs.com/<path>'
    -Dcluster="{\"ps\":{\"count\":1},\"worker\":{\"count\":2,\"gpu\":100}}"
    -Darn="acs:ram::******:role/aliyunodpspaidefaultrole"
    -DossHost="oss-cn-beijing-internal.aliyuncs.com"
```

参数说明如下表所示。

参数	描述	示例	默认值	是否必选

参数	描述	示例	默认值	是否必选
script	用于指定要执行的TF算法脚本，script的格式为 `file:///path/to/file` 或 `project_name/resources/resource_name` 。`file:///path/to/file` 为绝对路径。 TensorFlow模型文件（python）。文件类型如下： • 本地文件 • 本地的tar包（gzip压缩，扩展名为tar.gz） • python文件 格式为 `oss://..aliyuncs.com/.tar.gz` 或 `oss://..aliyuncs.com/*.py` 。	`oss://demo-yuze.oss-cn-beijing-internal.aliyuncs.com/deepfm/deepfm.tar.gz`	无	是
entryFile	如果script是一个tar包，则需通过该参数指定入口脚本。	`main.py`	如果script是单个脚本，则无需配置该参数。	是
buckets	输入bucket。 如果使用多个bucket，则需要以逗号隔开，每个bucket须以 `/` 结尾。	`oss://..aliyuncs.com/`	无	否
tables	输入表，如果使用多个表，则需要以逗号隔开。	`odps:///tables/`	无	否
outputs	输出表，如果使用多个表，则需要以逗号隔开。	`odps:///tables/\"”odps:///tables/`	无	否

参数	描述	示例	默认值	是否必选
gpuRequired	该参数决定运行script参数所指定的训练脚本的服务器是否需要GPU。 默认为一张卡，200表示申请2张卡。该功能只适用于单机训练任务，多机任务请参考Cluster。如果想运行单机CPU，可以将gpuRequired参数值设置为*0*（该功能只支持TensorFlow1120）。	100	yes **说明** 表示使用PAI命令提交训练任务后，PAI命令中指定的训练脚本将在有GPU的服务器上执行。	否
checkpointDir	TF checkpoint目录	`oss://..aliyuncs.com/`	无	否
cluster	分布式运行信息，详情请参见下面的分布式命令参数表格。	`{\"ps\":{\"count\":1},\"worker\":{\"count\":2,\"gpu\":100}}`	无	否
enableDynamicCluster	指定是否打开单个worker节点failover的功能。 当该参数设置为*true*时，worker节点出现故障时会被重新拉起，训练job不会因此而失败。	• true • false	false	否
jobName	您需要指定实验名称，用于在后期更好的分析历史所有该实验的性能指标。 参数建议是一个有意义的字符串，避免为 `test` 之类的字符串。	jk_wdl_online_job	无	是
maxHungTimeBeforeGCInSeconds	提交作业时新增-DmaxHungTimeBeforeGCInSeconds参数，用于设置自动回收需要观察到GPU*挂起*状态的最大持续时间。 该参数设置为0时，则关闭该功能。	3600	3600	否

分布式PAI-TF支持cluster参数，您可以通过该参数指定PS和Worker数目。cluster的格式为JSON，引号需要转义，示例如下。

```
{
  "ps": {
    "count": 2
  },
  "worker": {
    "count": 4
  }
}
```

其中JSON包含两个key，即ps和worker，每个key又分别包含如下信息。

参数	描述	默认值	是否必选
count	ps或worker的数量。	无	是
gpu	ps或worker申请的GPU数量，取值100表示一张GPU卡。如果worker的gpu配置为0，则系统会将其自动配置为100，以保障调度。	ps对应的gpu默认值为0，worker对应的gpu默认值为100。	否
cpu	ps或worker申请的CPU数量，取值100表示一张CPU卡。	600	否
memory	ps或worker申请的内存，取值100表示100 MB。	30000	否

IO参数

IO相关参数如下表所示。

参数	描述
tables	用于指定需要读取的Table路径。
outputs	用于指定写入的Table路径，如果需要指定多个路径，可以使用逗号分割。 • 非分区表Table路径格式：`odps://<prj_name>/tables/<table_name>`。 • 分区表Table路径格式：`odps://<proj_name>tables/<table_name>/<pt_key1=v1>`。 • 多级分区表Table路径格式：`odps://<prj_name>tables/<table_name>/<pt_key1=v1>/<pt_key2=v2>`。
buckets	用于指定算法将要读取的OSS bucket。 与其他MaxCompute的IO不同，使用OSS需要配置role_arn和host。 您可以登录PAI Console，进入**进入机器学习**，在**机器学习PAI**页面的左侧导航栏，单击**设置**，单击**基本设置**页面的**OSS访问授权**，获取role_arn。
checkpointDir	用于指定算法写入的OSS bucket。

5.2.2.4. PAI-TF数据IO方式介绍

PAI-TensorFlow支持读取OSS对象存储数据和MaxCompute表数据。

读取OSS数据

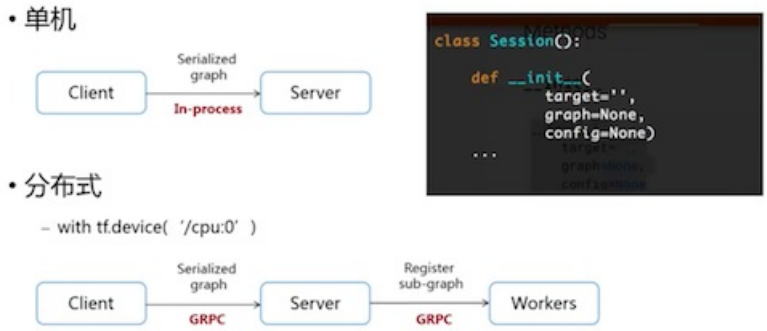
主流程	描述
上传数据至OSS 使用深度学习处理数据时，数据需要先存储到OSS的Bucket中。	1. 创建OSS Bucket。 创建的OSS Bucket的区域需与GPU的计算集群区域相同。数据传输时即可使用阿里经典网络，算法运行时不需要收取流量费用。 注意 不要开通OSS版本控制功能。 2. 创建文件夹、组织数据目录、上传数据。 登录OSS控制台，创建文件夹、组织数据目录和上传数据。
OSS授权	当您在机器学习平台中读写OSS Bucket组件时，需要授予AliyunODPSPAIDefaultRole系统默认角色给数加服务账号，详情请参见云产品依赖与授权：Designer。
RAM授权 RAM授权可以使机器学习平台获得OSS的访问权限	获取机器学习平台访问OSS的权限： 1. 在PAI-Studio项目空间，单击左侧菜单栏的**设置**，选择**基本设置**。 2. 在**基本设置**页面的**OSS访问授权**区域，选中**授权机器学习读取我的OSS中的数据**复选框，其他参数采用默认配置。 3. 在**确认**对话框，单击**点击前往RAM进行授权**。 4. 单击**同意授权**。  5. 单击**读OSS数据**组件字段设置下的**刷新**，查看RAM信息自动录入组件。 
TensorFlow读取OSS数据	连接组件**读OSS数据**和**TensorFlow**

默认角色AliyunODPSPAIDefaultRole包含权限信息如下。

权限名称（Action）	权限说明
oss:PutObject	上传文件或文件夹对象
oss:GetObject	获取文件或文件夹对象
oss:ListObjects	查询文件列表信息
oss:DeleteObjects	删除对象

TensorFlow读取OSS数据方法：

- 低效的IO方式
  本地执行TensorFlow代码和分布式云端执行TensorFlow的区别：
  - 本地读取数据：Server端直接从Client端获取Graph进行计算。
  - 云端服务：Server获得Graph后，还需要将计算下发至各Worker处理。



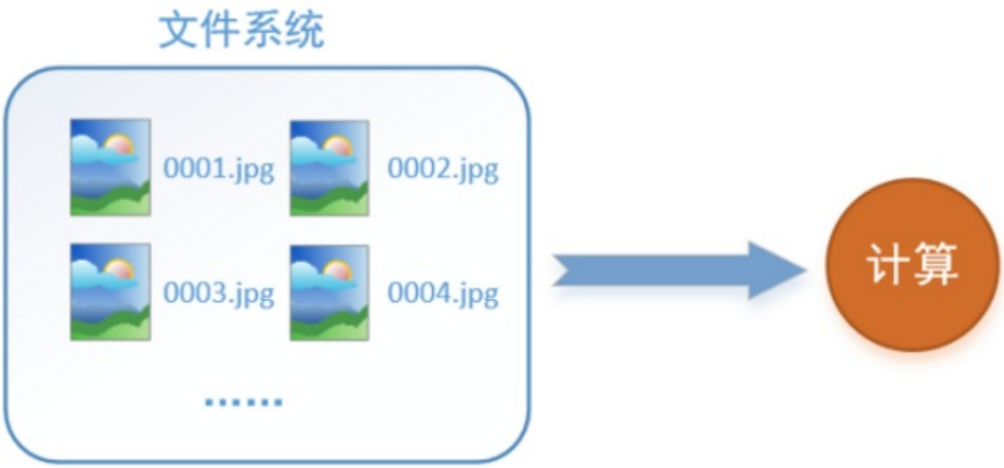
注意事项

- 不建议使用Python本地读取文件的方式。
  机器学习平台支持Python自带IO方式，但需要将数据源和代码打包上传。这种读取方式是将数据写入内存之后再计算，效率比较低，不建议使用。示例代码如下。

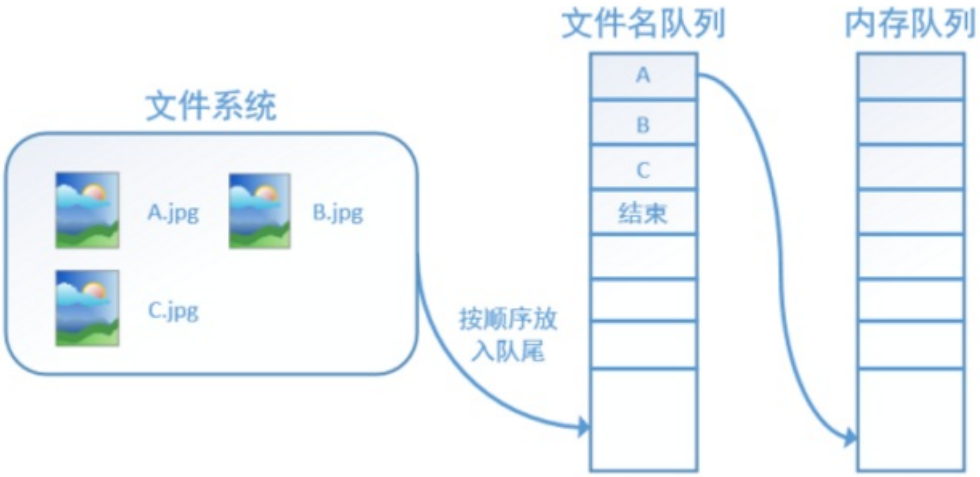
```
import csv
csv_reader=csv.reader(open('csvtest.csv'))
for row in csv_reader:
  print(row)
```

- 不建议使用第三方库读取文件的方式。
  使用第三方库（如TFLearn、Panda）的数据IO方式读取数据。通常，第三方库是通过封装Python的读取方式实现，所以在机器学习平台使用时也会造成效率低下的问题。

- 不建议使用Preload读取文件的方式。
  如果您发现GPU并没有比本地的CPU速度快很多，则有可能是数据IO，导致性能浪费。Preload读取方式先将数据全部都读到内存中，然后再通过Session计算，例如Feed读取方式。这样造成性能浪费，同时内存限制也无法计算大数据。
  例如，硬盘中有图片数据集0001.jpg，0002.jpg，0003.jpg，..... 。先读取数据后再计算，假设读入用时0.1s，计算用时0.9s，这样每过1s，GPU都会有0.1s空闲，降低了运算的效率。



- 高效的IO方式
  TensorFlow读取方式是将数据读取转换成OP，通过Session.run的方式读取数据。读取线程将文件系统中的图片读入到一个内存的队列中。计算是另一个线程，并直接从内存队列中读取进行计算。这样解决了GPU因IO空闲而浪费性能的问题。



在机器学习平台通过OP的方式读取数据的代码如下。

```
import argparse
import tensorflow as tf
import os
FLAGS=None
def main(_):
    dirname = os.path.join(FLAGS.buckets, "csvtest.csv")
    reader=tf.TextLineReader()
    filename_queue=tf.train.string_input_producer([dirname])
    key,value=reader.read(filename_queue)
    record_defaults=[[''],[''],[''],[''],['']]
    d1, d2, d3, d4, d5= tf.decode_csv(value, record_defaults, ',')
    init=tf.initialize_all_variables()
    with tf.Session() as sess:
        sess.run(init)
        coord = tf.train.Coordinator()
        threads = tf.train.start_queue_runners(sess=sess,coord=coord)
        for i in range(4):
            print(sess.run(d2))
        coord.request_stop()
        coord.join(threads)
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--buckets', type=str, default='',
                        help='input data path')
    parser.add_argument('--checkpointDir', type=str, default='',
                        help='output model path')
    FLAGS, _ = parser.parse_known_args()
    tf.app.run(main=main)
```

其中：

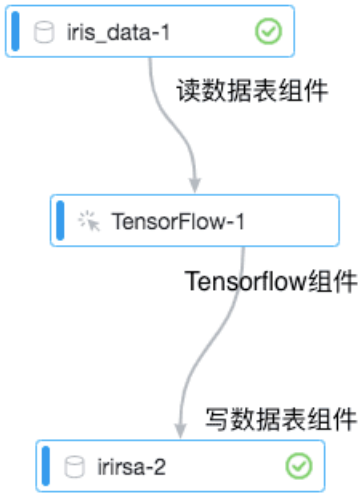
- dirname：OSS文件路径，可以是数组。
- reader：Tensorflow内置各种类型reader API，可以根据需求选用。
- tf.train.string_input_producer：将文件生成队列。
- tf.decode_csv：是一个Splite功能的OP，可以获取每一行的特定参数。
- 通过OP获取数据，在Session中需要tf.train.Coordinator()和tf.train.start_queue_runners(sess=sess,coord=coord)。

读取MaxCompute数据

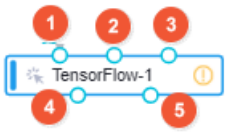
您可以直接使用PAI-Studio的Tensorflow组件读写MaxCompute数据。

下文以iris数据集为例，为您介绍如何读取MaxCompute数据。

主流程	描述

主流程	描述
连接组件	拖动组件并连接。 

<table>
<tr><th>主流程</th><th>描述</th></tr>
<tr><td>配置读数据表组件</td><td>
在读数据表组件中，输入如下代码。
<pre>pai_online_project.iris_data</pre>
即可获取数据。


数据格式如下。
<table>
<tr><th>f1</th><th>f2</th><th>f3</th><th>f4</th><th>type</th></tr>
<tr><td>5.1</td><td>3.5</td><td>1.4</td><td>0.2</td><td>Iris-setosa</td></tr>
<tr><td>4.9</td><td>3</td><td>1.4</td><td>0.2</td><td>Iris-setosa</td></tr>
<tr><td>4.7</td><td>3.2</td><td>1.3</td><td>0.2</td><td>Iris-setosa</td></tr>
<tr><td>4.6</td><td>3.1</td><td>1.5</td><td>0.2</td><td>Iris-setosa</td></tr>
<tr><td>5</td><td>3.6</td><td>1.4</td><td>0.2</td><td>Iris-setosa</td></tr>
<tr><td>5.4</td><td>3.9</td><td>1.7</td><td>0.4</td><td>Iris-setosa</td></tr>
<tr><td>4.6</td><td>3.4</td><td>1.4</td><td>0.3</td><td>Iris-setosa</td></tr>
<tr><td>5</td><td>3.4</td><td>1.5</td><td>0.2</td><td>Iris-setosa</td></tr>
<tr><td>4.4</td><td>2.9</td><td>1.4</td><td>0.2</td><td>Iris-setosa</td></tr>
<tr><td>4.9</td><td>3.1</td><td>1.5</td><td>0.1</td><td>Iris-setosa</td></tr>
</table>
</td></tr>
<tr><td></td><td>


①输入桩连接OSS输入
②输入桩连接MaxCompute输入
③输入桩连接模型输入
④输出桩连接模型输出
⑤输出桩连接MaxCompute输出

如果输入MaxCompute表，输出也是MaxCompute表，则只需要接入②和⑤。
</td></tr>
</table>

主流程	读写MaxCompute表需要配置数据源、代码文件、输出模型路径、建表等操作。 描述
配置Tensorflow组件	 • Python代码文件：将执行代码存放至OSS路径下。 注意 OSS需要与当前项目在同一区域 • Checkpoint输出目录/模型输入目录：选择您自己的OSS路径用来存放模型。 • MaxCompute输出表：写MaxCompute表要求输出表是已经存在的表，并且输出的表名需要与代码中的输出表名一致。在本案例中需要填写iris_output。 • 建表SQL语句：如果代码中的输出表并不存在，可以通过这个输入框输入建表语句自动建表。本案例中建表语句 `create table iris_output(f1 DOUBLE,f2 DOUBLE,f3 DOUBLE,f4 DOUBLE,f5 STRING);` 。 组件API命令 `PAI -name tensorflow180_ext -project algo_public -Doutputs="odps://${当前项目名}/tables/${输出表名}" -DossHost="${OSS的host}" -Dtables="odps://${当前项目名}/tables/${输入表名}" -DgpuRequired="${GPU卡数}" -Darn="${OSS访问RoleARN}" -Dscript="${执行的代码文件}";` 其中，${}类型的参数需要替换成您的真实数据。
读数据表 写数据表	推荐通过TabelRecordDataset读写MaxCompute表，详细的接口使用说明及示例请参见TableRecordDataset。

5.2.2.5. PAI-TF数据转换方法

本文为您介绍PAI-TF数据转换方法。

trans_csv_id2sparse Python接口

将标记有效位置的CSV字符串集合转换成为稀疏矩阵。

```
trans_csv_id2sparse(records, max_id, id_as_value=True, field_delim=",")
```

- 输入以下参数：

参数	是否必选	描述
records	是	类型STRING数组待解析CSV字符串数组（列表），以CSV格式由分隔符分隔。
max_id	是	类型INT64稀疏矩阵的最大列数，用于设定输出中的dense_shape值。如果实际ID大于或等于dense_shape值，则报错。
id_as_value	否	类型BOOL，默认为True，将Index编号作为稀疏矩阵中有效点的值，类型为INT64。无特殊情况不建议更改为False。
field_delim	否	类型STRING，默认为英文逗号（,）。CSV格式数据的分隔符。不支持数字、正负号、字母e和E、小数点（.）和多字节分隔符。当使用空格作为分隔符时，多个连续空格将被视作一个分隔符。

- 输出：由Index CSV序列转换而得的Sparse Tensor，Value类型为INT64。

示例：将一个batch的存有Index数据的STRING转换成一个sparse tensor。

- 输入：

```
["2,10","7","0,8"]
```

- 需求：
  矩阵列宽度为20，有效点填入原Index。
- 代码：

```
outsparse = tf.trans_csv_id2sparse(["2,10","7","0,8"], 20)
```

- 返回结果：

```
SparseTensor(
indices=[[0,2],[0,10],[1,7],[2,0],[2,8]],
values=[2, 10, 7, 0, 8],
dense_shape=[3,20])
```

trans_csv_kv2dense Python接口

将以Key/Value形式标记有效位置和值的CSV字符串集合转换成为稠密矩阵。

```
trans_csv_kv2dense(records, max_id, field_delim=",")
```

- 输入以下参数：

参数	是否必选	描述
records	是	类型STRING数组待解析CSV字符串数组（列表），以CSV格式由分隔符分隔。每一个数据项均为以冒号（:）分隔的Key/Value形式数据，否则报错。
max_id	是	类型INT64输出的稠密矩阵的列数。如果实际ID大于或等于列数，则报错。
field_delim	否	类型STRING，默认为英文逗号（,）。CSV格式数据的分隔符。不支持数字、正负号、字母e和E、小数点（.）和多字节分隔符。当使用空格作为分隔符时，多个连续空格将被视作一个分隔符。

- 输出：由Key/Value形式CSV序列转换而得的稠密矩阵，默认输出类型为FLOAT，空白处以0.0填充。

示例：将一个batch以Key/Value形式 `Index:Value` 存储的STRING转换成为一个稠密矩阵。

- 输入：

```
["1:0.1,2:0.2,4:0.4,10:1.0",
"0:0.22,3:0.33,9:0.99",
"2:0.24,7:0.84,8:0.96" ]
```

- 需求：
  列宽设置为12。
- 代码：

```
outmatrix = tf.trans_csv_kv2dense(
["1:0.1,2:0.2,4:0.4,10:1.0",
 "0:0.22,3:0.33,9:0.99",
 "2:0.24,7:0.84,8:0.96" ] , 12)
```

- 返回结果：

```
[[0.0, 0.1, 0.2, 0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
[0.22, 0.0, 0.0, 0.33, 0.0, 0.0, 0.0, 0.0, 0.0, 0.99, 0.0, 0.0]
[0.0, 0.0, 0.24, 0.0, 0.0, 0.0, 0.0, 0.84, 0.96, 0.0, 0.0, 0.0]]
```

trans_csv_kv2sparse Python接口

将以Key/Value形式标记有效位置和值的CSV字符串集合转换成为稀疏矩阵。

```
trans_csv_kv2sparse(records, max_id, field_delim=",")
```

- 输入以下参数：

参数	是否必选	描述
records	是	类型STRING数组待解析CSV字符串数组（列表），以CSV格式由分隔符分隔。每一个数据项均为以冒号（:）分隔的Key/Value形式数据，否则报错。当使用空格作为分隔符时，多个连续空格将被视作一个分隔符。
max_id	是	类型INT64稀疏矩阵的最大列数，用于设定输出中的dense_shape值。如果实际ID大于或等于dense_shape值，则报错。
field_delim	否	类型STRING，默认为英文逗号（,）。CSV格式数据的分隔符。不支持数字、正负号、字母e和E、小数点（.）和多字节分隔符。当使用空格作为分隔符时，多个连续空格将被视作一个分隔符。

- 输出：由Key/Value形式CSV序列转换而得的稀疏矩阵，默认输出类型为FLOAT。

示例：将一个batch以Key/Value形式 `Index: Value` 存储的STRING转换成为一个稀疏矩阵。

- 输入：

```
["1:0.1,2:0.2,4:0.4,10:1.0",
"0:0.22,3:0.33,9:0.99",
"2:0.24,7:0.84,8:0.96" ]
```

- 需求：
  列宽设置为20，生成稀疏矩阵Tensor。
- 代码：

```
outsparse = tf.trans_csv_kv2sparse(
["1:0.1,2:0.2,4:0.4,10:1.0",
 "0:0.22,3:0.33,9:0.99",
 "2:0.24,7:0.84,8:0.96" ] , 20)
```

- 返回结果：

```
SparseTensor(
indices=[[0,1],[0,2],[0,4],[0,10],[1,0],[1,3],[1,9],[2,0],[2,7],[2,8]],
values=[0.1, 0.2, 0.4, 1.0, 0.22, 0.33, 0.99, 0.24, 0.84, 0.96],
dense_shape=[3,20])
```

trans_csv_id2dense Python接口

将标记有效位置的CSV字符串集合转换成为稠密矩阵。

```
trans_csv_id2dense(records, max_id, id_as_value=False, field_delim=",")
```

- 输入以下参数：

参数	是否必选	描述
records	是	类型STRING数组待解析CSV字符串数组（列表），以CSV格式由分隔符分隔。
max_id	是	类型INT64输出的稠密矩阵的列数。如果实际ID大于或等于列数，则报错。
id_as_value	否	类型BOOL，默认为False，稀疏矩阵中有效点的值将会填入int64(1)。
field_delim	否	类型STRING，默认为英文逗号（,）。CSV格式数据的分隔符。不支持数字、正负号、字母e和E、小数点（.）和多字节分隔符。当使用空格作为分隔符时，多个连续空格将被视作一个分隔符。

- 输出：由Index CSV序列转换而得的稠密矩阵，类型为INT64，空白处以0值填充。

示例：将一个batch的存有Index数据的STRING转换成一个稠密矩阵：

- 输入：

```
["2,10","7","0,8"]
```

- 需求：
  需求：列宽设置为12，有效点填入1。
- 代码：

```
outmatrix = tf.trans_csv_id2dense(
["2,10","7","0,8"], 12)
```

- 返回结果：

```
[[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]]
```

trans_csv_to_dense Python接口

将由数值组成的CSV字符串集合转换成为稠密矩阵。

```
trans_csv_to_dense(records, max_id, field_delim=",")
```

- 输入以下参数：

参数	是否必选	描述
records	是	类型STRING数组待解析CSV字符串数组（列表），以CSV格式由分隔符分隔。

参数	是否必选	描述
max_id	是	类型INT64输出的稠密矩阵的列数。如果实际CSV字符串的列数大于或等于列数，则报错。
field_delim	否	类型STRING，默认为英文逗号（,）。CSV格式数据的分隔符。不支持数字、正负号、字母e和E、小数点（.）和多字节分隔符。当使用空格作为分隔符时，多个连续空格将被视作一个分隔符。

- 输出：由Key/Value形式CSV序列转换而得的稠密矩阵，默认输出类型为FLOAT，空白处以0.0填充。

示例：将一个batch的CSV格式STRING转换成为一个稠密矩阵。

- 输入：

```
["0.1,0.2,0.4,1.0",
"0.22,0.33,0.99",
"0.24,0.84,0.96" ]
```

- 需求：
  列宽设置为6。
- 代码：

```
outmatrix = tf.trans_csv_to_dense(
["0.1,0.2,0.4,1.0",
 "0.22,0.33,0.99",
 "0.24,0.84,0.96" ] , 6)
```

- 返回结果：

```
[[0.1, 0.2, 0.4, 1.0, 0.0, 0.0]
[0.22, 0.33, 0.99, 0.0, 0.0, 0.0]
[0.24, 0.84, 0.96, 0.0, 0.0, 0.0]]
```

代码实例

以下代码通过TensorFlow从存放在ODPS的数据表中读取数据。数据共有6列，第1列为ID，第2列为Key/Value格式的CSV数据，后4列为Index格式的CSV数据。数据读取后调用TransCSV的ODPS，将这5列数据分别转换为1个稠密矩阵和4个稀疏矩阵，用于模型训练。

```
import tensorflow as tf
import numpy as np
def read_table(filename_queue):
    batch_size = 128
    reader = tf.TableRecordReader(csv_delimiter=';', num_threads=8, capacity=8*batch_size)
    key, value = reader.read_up_to(filename_queue, batch_size)
    values = tf.train.batch([value], batch_size=batch_size, capacity=8*capacity, enqueue_ma
ny=True, num_threads=8)
    record_defaults = [[1.0], [""], [""], [""], [""], [""]]
    feature_size = [1322,30185604,43239874,5758226,41900998]
    col1, col2, col3, col4, col5, col6 = tf.decode_csv(values, record_defaults=record_defau
lts, field_delim=';')
    col2 = tf.trans_csv_kv2dense(col2, feature_size[0])
    col3 = tf.trans_csv_id2sparse(col3, feature_size[1])
    col4 = tf.trans_csv_id2sparse(col4, feature_size[2])
    col5 = tf.trans_csv_id2sparse(col5, feature_size[3])
    col6 = tf.trans_csv_id2sparse(col6, feature_size[4])
    return [col1, col2, col3, col4, col5, col6]
if __name__ == '__main__':
    tf.app.flags.DEFINE_string("tables", "", "tables")
    tf.app.flags.DEFINE_integer("num_epochs", 1000, "number of epoches")
    FLAGS = tf.app.flags.FLAGS
    table_pattern = FLAGS.tables
    num_epochs = FLAGS.num_epochs
    filename_queue = tf.train.string_input_producer(table_pattern, num_epochs)
    train_data = read_table(filename_queue)
    init_global = tf.global_variables_initializer()
    init_local = tf.local_variables_initializer()
    with tf.Session() as sess:
      sess.run(init_global)
      sess.run(init_local)
      coord = tf.train.Coordinator()
      threads = tf.train.start_queue_runners(sess=sess, coord=coord)
      for i in range(1000):
        sess.run(train_data)
      coord.request_stop()
      coord.join(threads)
```

5.2.2.6. PAI-TF日志查看方式

PAI-TF日志分为实验运行状态日志和Logview日志，其中PAI-TF的Logview日志蕴含更多的实验信息。如果您的实验在运行过程中报错，则可以通过该操作查看具体报错信息。

无论以何种方式提交PAI-TF任务，都可以得到如下图所示的日志。

序号	描述
①	该蓝色链接日志即为Logview日志，可以在浏览器中打开。
②	运行状态日志。

运行状态日志

运行状态日志主要展示分布式作业的服务器运行情况。通常以参数1/参数2/参数3/格式展示该日志，其中：

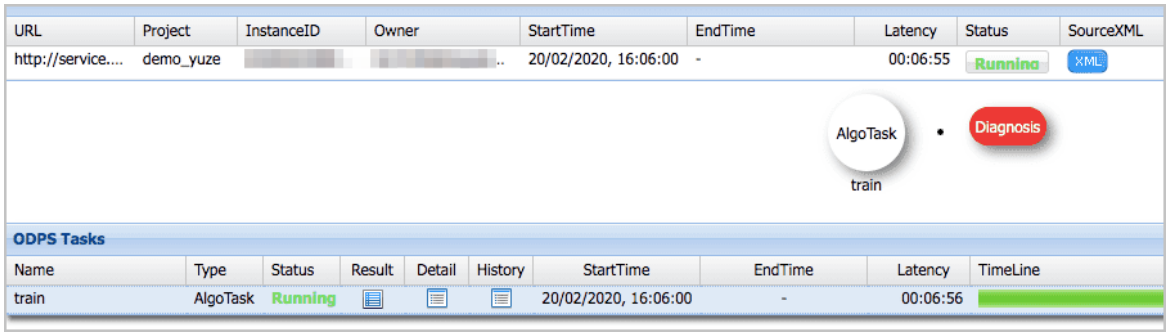
- 参数1：表示正在计算的节点数。
- 参数2：表示已经计算结束的节点数。
- 参数3：表示总的计划节点数。

实验进度100%表示实验已经结束。

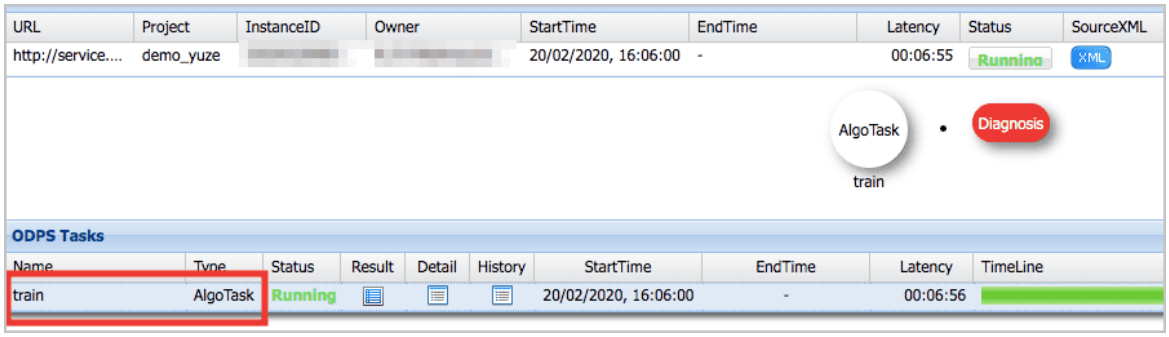
Logview日志

Logview日志包含实验的Debug信息，您可以通过如下方式查看相关日志：

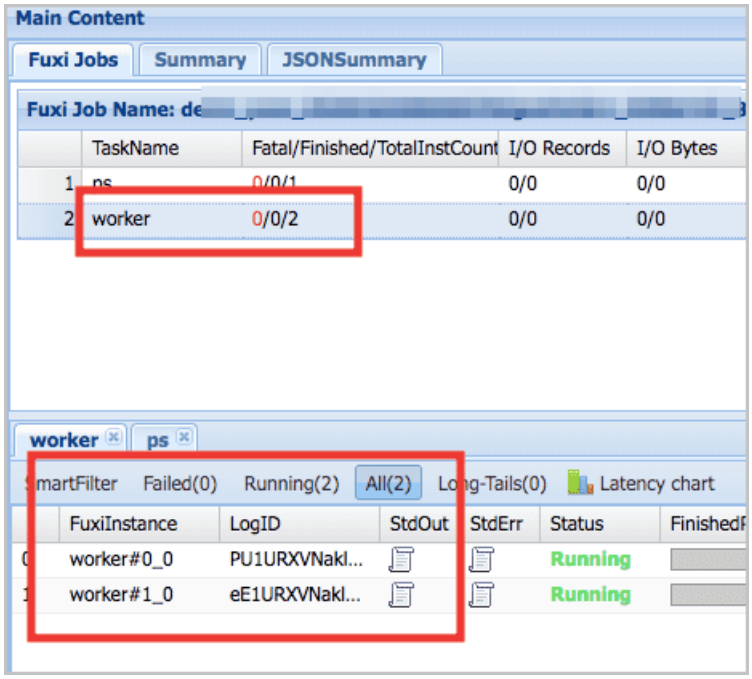
1. 在浏览器中打开上图中的蓝色链接，进入如下页面。



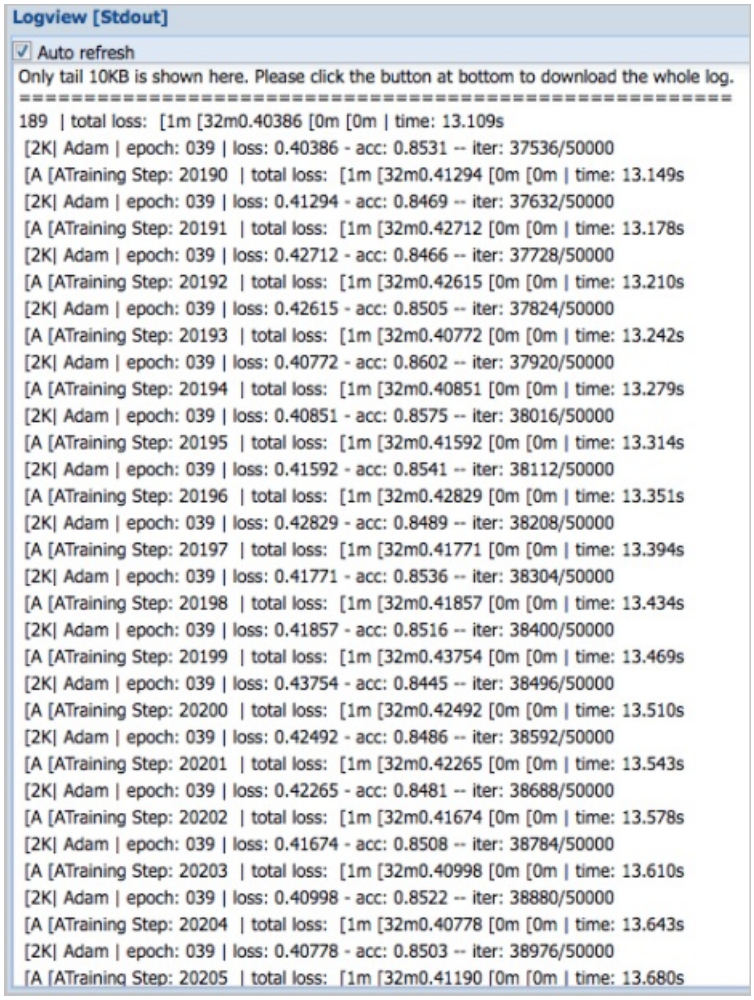
2. 如果实验报错查看报错日志，或需要查看实验运行过程中代码Print信息，则双击ODPS Tasks实例。



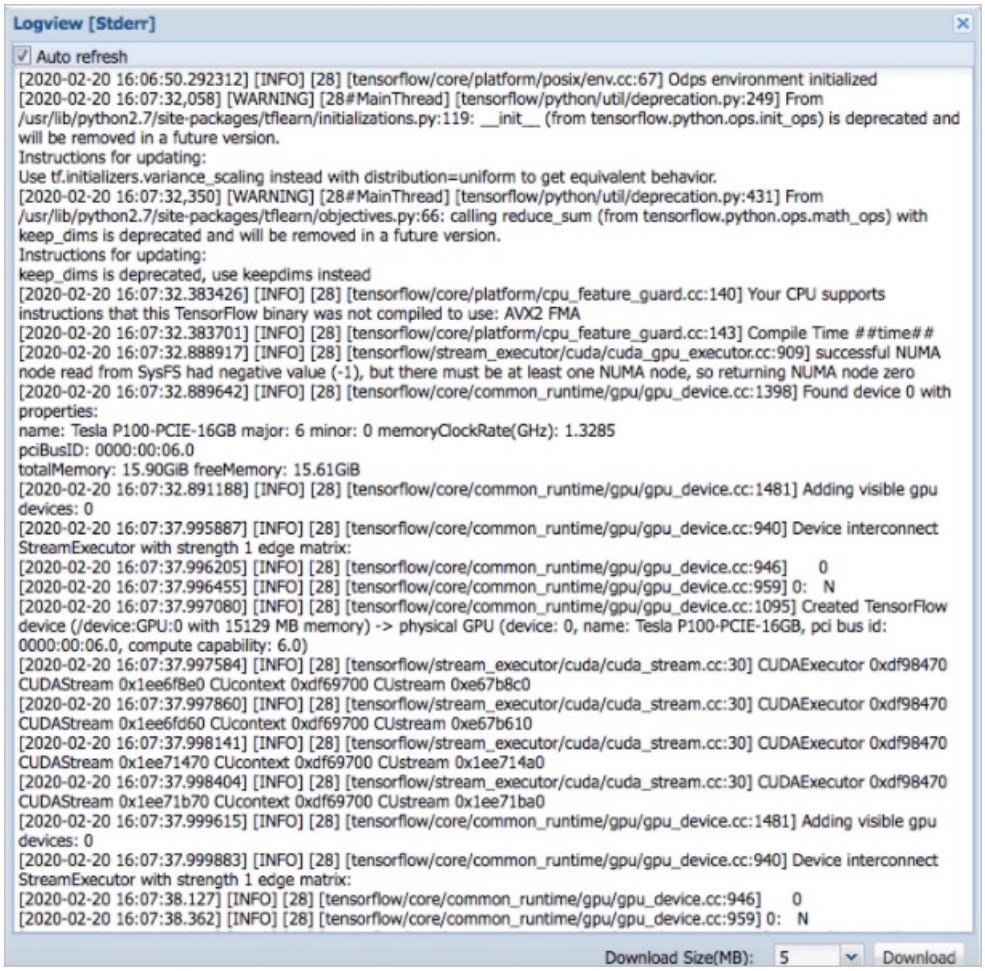
3. 在Worker运行页面，可以根据需要选择查看的Task实例及Worker对应的实例。



4. 如果需要查看代码运行过程中Print函数打印的信息，则单击**Stdout**列下的图标。



5. 如果需要查看产品报错信息Debug，则单击**StdErr**列下的图标。



5.2.2.7. PAI-TF超参支持

PAI-TF支持通过超参TXT文件或Command传入相应的超参配置，从而在模型试验时可以尝试不同的Learning Rate及Batch Size等。

超参文件

您可以通过一个本地文件配置相应的超参信息，格式如下。

```
batch_size=10
learning_rate=0.01
```

TensorFlow Python SDK提供了相应的参数以便获取相应的超参，您可以通过 `tf.app.flags.FLAGS` 读取所需的超参，再将其传入运行脚本中，即可在模型训练文件中读取到相应的超参定义。具体方法如下：

1. 假设上面定义的超参文件存储在*oss://xxx.oss-cn-beijing.aliyuncs.com/tf/hyper_para.txt*，参考如下Python代码读取超参。

   ```
   import tensorflow as tf
   tf.app.flags.DEFINE_string("learning_rate", "", "learning_rate")
   tf.app.flags.DEFINE_string("batch_size", "", "batch size")
   FAGS = tf.app.flags.FLAGS
   print("learning rate:" + FAGS.learning_rate)
   print("batch size:" + FAGS.batch_size)
   ```

2. 通过 `-DhyperParameters` 将超参传入到运行脚本中，示例如下。

```
pai -name tensorflow1120_ext
        -Dscript='oss://xxx.oss-cn-beijing.aliyuncs.com/tf/hello_hyperpara.py'
        -Dbuckets='oss://xxx.oss-cn-beijing.aliyuncs.com/'
        -DhyperParameters='oss://xxx.oss-cn-beijing.aliyuncs.com/tf/hyper_para.txt'
        -Darn='acs:ram::111***:role/***role';
```

字符串形式参数

PAI-TF也支持以字符串形式传入参数，您可以直接将字符串通过 `userDefinedParameters` 传入，示例如下。

```
pai -name tensorflow1120_ext
        -Dscript='oss://xxx.oss-cn-beijing.aliyuncs.com/tf/hello_hyperpara.py'
        -Dbuckets='oss://xxx.oss-cn-beijing.aliyuncs.com/'
        -DuserDefinedParameters="--batch_size=10 --learning_rate=0.01"
        -Darn='acs:ram::111***:role/***role';
```

> **说明** 以字符串传入的参数，使用KV格式，每一个KV前面需要以“--”作为前缀。

5.2.2.8. PAI-TF模型导出和部署说明

本文为您介绍PAI-TF模型导出和部署相关说明。包括导出SaveModel通用模型、保存和恢复检查点以及如何将TF模型部署到EAS。

导出SaveModel通用模型

- SavedModel格式
  SavedModel是目前官方推荐的模型保存的格式（SessionBundle自Tensorflow 1.0以后不再推荐使用），目录结构如下。
  assets/assets.extra/variables/ variables.data-?????-of-????? variables.indexsaved_model.pb
  目录中各个子目录和文件的含义请参见TensorFlow SavedModel官方文档或介绍。
- 导出SavedModel
  代码片段：

```
class Softmax(object):
    def __init__(self):
        self.weights_ = tf.Variable(tf.zeros([FLAGS.image_size, FLAGS.num_classes]),
                name='weights')
        self.biases_ = tf.Variable(tf.zeros([FLAGS.num_classes]),
                name='biases')
    # ...
    def signature_def(self):
        images = tf.placeholder(tf.uint8, [None, FLAGS.image_size],
            name='input')
        normalized_images = tf.scalar_mul(1.0 / FLAGS.image_depth,
            tf.to_float(images))
        scores = self.scores(normalized_images)
        tensor_info_x = tf.saved_model.utils.build_tensor_info(images)
        tensor_info_y = tf.saved_model.utils.build_tensor_info(scores)
        return 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)
    def savedmodel(self, sess, signature, path):
        export_dir = os.path.join(path, str(FLAGS.model_version))
        builder = tf.saved_model.builder.SavedModelBuilder(export_dir)
        builder.add_meta_graph_and_variables(
            sess, [tf.saved_model.tag_constants.SERVING],
            signature_def_map={
                'predict_images':
                    signature,
            },
            clear_devices=True)
        builder.save()
#...
model = Softmax()
signature = model.signature_def()
#...
model.savedmodel(sess, signature, mnist.export_path())
```

代码说明：

- Softmax类封装了机器学习模型，其中weights和biases是其最主要的模型参数。
- signature_def方法描述了预测时，如何从一个placeholder经过数据标准化和前向计算得到输出的逻辑，并分别作为输入和输出构建出一个SignatureDef。

导出SavedModel至OSS：
训练并导出模型的命令如下。

```
PAI -name tensorflow
    -Dscript="file://path/to/mnist_savedmodel_oss.py"
    -Dbuckets="oss://mnistdataset/?host=oss-test.aliyun-inc.com&role_arn=acs:ram::127488*
*****:role/odps"
    -DcheckpointDir="oss://mnistdataset/?host=oss-test.aliyun-inc.com&role_arn=acs:ram::1
27488*********:role/odps";
```

保存和恢复检查点

- Checkpoint存储

非交互式TensorFlow存储模型的示例程序如下。

```
# -*- coding: utf-8 -*-
# usage
# pai -name tensorflow -DcheckpointDir="oss://tftest/examples/?host=oss-test.aliyun-inc.c
om&role_arn=acs:ram::****:role/odps" -Dscript="file:///path/to/save_model.py";
import tensorflow as tf
import json
import os
tf.app.flags.DEFINE_string("checkpointDir", "", "oss info")
FLAGS = tf.app.flags.FLAGS
print("checkpoint dir:" + FLAGS.checkpointDir)
# 定义变量
counter = tf.Variable(1, name="counter")
one = tf.constant(2)
sum = tf.add(counter, one)
new_counter = tf.assign(counter, sum)
saver = tf.train.Saver()
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)
    ret = sess.run(new_counter)
    print("Counter:%d" % ret)
    ckp_path = os.path.join(FLAGS.checkpointDir, "model.ckpt")
    save_path = saver.save(sess, ckp_path)
    print("Model saved in file: %s" % save_path)
    coord.request_stop()
    coord.join(threads)
```

tf.app.flags.DEFINE_string()和tf.app.flags.FLAGS可以获取PAI命令中的checkpointDir参数，checkpointDir指定了模型将要存储到OSS上。
以下代码完成了new_counter的计算，并将名称为counter的变量存储到模型中（值为3），save_path = saver.save(sess, ckp_path)将模型写到OSS指定路径。

```
ret = sess.run(new_counter)
print("Counter:%d" % ret)
ckp_path = os.path.join(FLAGS.checkpointDir, "model.ckpt")
save_path = saver.save(sess, ckp_path)
print("Model saved in file: %s" % save_path)
```

- Checkpoint恢复
  TensorFlow的Saver类也可以用于模型的恢复，TensorFlow恢复模型的示例如下。

```
# -*- coding: utf-8 -*-
# usage
# pai -name tensorflow -Dbuckets="oss://tftest/examples/?host=oss-test.aliyun-inc.com&rol
e_arn=acs:ram::***:role/odps" -Dscript="file:///path/to/restore_model.py";
import tensorflow as tf
import json
import os
tf.app.flags.DEFINE_string("buckets", "", "oss info")
FLAGS = tf.app.flags.FLAGS
print("buckets:" + FLAGS.buckets)
# 定义变量
counter = tf.Variable(1, name="counter")
saver = tf.train.Saver()
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init_op)
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)
    ret = sess.run(counter)
    print("Before restore counter:%d" % ret)
    print("Model restore from file")
    ckp_path = os.path.join(FLAGS.buckets, "model.ckpt")
    saver.restore(sess, ckp_path)
    ret = sess.run(counter)
    print("After restore counter:%d" % ret)
    coord.request_stop()
    coord.join(threads)
```

tf.app.flags.DEFINE_string()和tf.app.flags.FLAGS可以获取PAI命令中的buckets参数，buckets指定了模型将要从OSS上恢复模型。
以下代码中，首先定义了名称counter的变量，初始值为1。调用saver.restore(sess, ckp_path)，根据给定的OSS路径恢复已存储的模型，最后执行ret = sess.run(counter)得到恢复后的变量的值也是3。

```
ret = sess.run(counter)
print("Before restore counter:%d" % ret)
print("Model restore from file")
ckp_path = os.path.join(FLAGS.buckets, "model.ckpt")
saver.restore(sess, ckp_path)
ret = sess.run(counter)
print("After restore counter:%d" % ret)
```

TF模型部署到EAS

EAS是PAI平台自研的模型部署工具，支持深度学习框架生成的模型，特别是部署TensorFlow SavedModel函数生成的模型。EAS有两种模型部署方式，一种是通过PAI-EAS的线上服务进行部署，另一种是通过EAS CMD进行部署：

- 线上服务部署方式
  i. 将模型存储于OSS中。
  ii. 登录PAI控制台。
  iii. 在左侧导航栏，单击**EAS-模型在线服务**。
  iv. 在顶部菜单栏处，选择地域。

v. 在**PAI EAS 模型在线服务**页面，单击**模型上传部署**。

vi. 在右侧的配置页面，设置**Processor 种类**为**TensorFlow1.12**或**TensorFlow1.14**，并选择您已上传的OSS中的模型文件。

vii. 单击**下一步**，配置模型部署信息并确认，然后单击**部署**。
系统会把SavedModel格式的TensorFlow模型打包上传，完成模型服务的部署。

- EASCMD部署方式
  详情请参见命令使用说明。

5.2.2.9. 分布式训练框架StarServer

本文为您介绍如何使用分布式训练框架StarServer进行分布式训练。
StarServer不仅将原生TensorFlow中的Send/Recv语义修改为Pull/Push语义，而且实现了图执行过程中的Lock-Free，大幅度提高了并发执行子图效率。PAI-TF支持更大的训练规模和训练性能，针对典型业务场景，其训练性能比原生TensorFlow提升了数倍。其中，最大测试规模为3000 Worker时，PAI-TF达到近似线性扩展性。

开启StarServer分布式训练

使用StarServer进行分布式训练，需要在tf.train.Server中添加 `protocol="star_server"` 。

```
cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})
server = tf.train.Server(cluster,
                         job_name=FLAGS.job_name,
                         task_index=FLAGS.task_index,
                         protocol="star_server")
```

5.2.2.10. TableWriter API

您可以使用TableWriter API对MaxCompute表进行读写。

读写MaxCompute表的功能由*tensorflow.python_io.TableWriter*提供，可以在TensorFlow的Graph执行逻辑以外，直接对MaxCompute表进行操作。

> **说明** PAI-TF作业执行过程中，写入MaxCompute表的数据，必须在作业正常结束以后，才能通过TableWriter API访问。如果作业正在运行或异常退出，则无法访问。

创建Writer并打开表

初始化TableWriter将打开一个MaxCompute表并返回Writer对象。接口定义如下。

```
writer = tf.python_io.TableWriter(table, slice_id=0)
```

- table：string类型，表示需要打开的MaxCompute表名。该参数需要与PAI命令-Doutput中的输出表名一致，否则系统报`table xxx not predefined`错误。
- slice_id：int类型，表示在表的不同分区进行写入，避免写冲突。单机场景下，使用默认值0即可。分布式场景下，如果多个worker（包括PS）同时使用相同的slice_id写入数据，则会导致写入失败。

> **注意** 每次打开一张表，相当于打开一张空表，即原表中的数据被清空。

写入记录

将数据写入已打开表的对应列，该写入数据必须在关闭表之后才能读取。接口定义如下。

```
writer.write(values, indices)
```

- values：表示写入的单行或多行数据。
  - 如果写入单行数据，则传入一个由标量组成的Tuple、List或1D-Ndarray。其中List和1D-Ndarray表示写入的各列数据类型相同。
  - 如果写入N（N>=1）行数据，则传入一个List或1D-Ndarray，参数中的每个元素对应一个单行数据（Tuple、List或以Structure为元素的Ndarray）。
- indices：表示数据写入的列，支持由整形index组成的Tuple、List或1D-Ndarray类型。indices中的每个数对应表的每一列（列数从0开始计算）。

关闭表

接口定义如下。

```
writer.close()
```

在`with`语句块中，无需显示调用`close()`关闭表。

> **注意** 关闭表之后，如果使用`open`命令重新打开表，则表中原数据被清空。

通过with语句使用TableWriter

TableWriter支持通过`with`语句管理上下文，示例代码如下。

```
with tf.python_io.TableWriter(table) as writer:
    # Prepare values for writing.
    writer.write(values, incides)
    # Table would be closed automatically outside this section,
```

示例

1. 在project中新建一张含有4列元素的MaxCompute表test_write，表的列名及数据类型如下。

ColumnName	ColumnType
uid	bigint
name	string
price	double
virtual	bool

2. 使用Python命令，并配置-Doutputs为odps://project/tables/test_write，将如下表格中的数据写入test_write表。

uid	name	price	virtual
25	"Apple"	5.0	False
38	"Pear"	4.5	False
17	"Watermelon"	2.2	False

```
# table_writer_test.py文件。
import tensorflow as tf
# Prepare data，准备数据。
values = [(25, "Apple", 5.0, False),
          (38, "Pear", 4.5, False),
          (17, "Watermelon", 2.2, False)]
# Open a table，打开一个表并返回writer对象。
writer = tf.python_io.TableWriter("odps://project/tables/test")
# Write records to the 0~3 columns of the table，将数据写入表的第0~3列。
records = writer.write(values, indices=[0, 1, 2, 3])
# Close the table，关闭表和Writer。
writer.close()
```

3. 提交任务至PAI-TF，并执行任务。

```
$ odpscmd -e "pai -name tensorflow140 -Dscript=<absolute_path_of_script>/table_writer_test.py -Doutputs=odps://project/tables/test_write ;"
```

5.2.2.11. 分布式通信框架gRPC++

使用PAI-DLC进行深度学习训练时，您可以使用gRPC++分布式训练，从而加速模型训练。本文介绍如何开启分布式通信框架gRPC++。

gRPC++通过Sharing Nothing架构、BusyPolling机制、用户态零拷贝及Send/Recv融合等多种优化技术，降低了E2E的通信延时，提高了Server的吞吐能力，从而可以支持更大的训练规模和训练性能。针对典型业务场景，其训练性能比原生TensorFlow提升了数倍。

开启gRPC++分布式训练

使用gRPC++进行分布式训练，需要在*tf.train.Server*中添加 `protocol="grpc++"` 。

```
cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})
server = tf.train.Server(cluster,
                         job_name=FLAGS.job_name,
                         task_index=FLAGS.task_index,
                         protocol="grpc++")
```

5.2.2.12. EmbeddingVariable

使用EmbeddingVariable进行超大规模训练，不仅可以保证模型特征无损，而且可以节约内存资源。

背景信息

Embedding已成为深度学习领域处理Word及ID类特征的有效途径。作为一种“函数映射”，Embedding通常将高维稀疏特征映射为低维稠密向量，再进行模型端到端训练。在TensorFlow中，使用Variable定义模型或节点状态，其实现依赖于数据结构Tensor。Tensor是TensorFlow的抽象数据类型，包括标量、向量、矩阵及更高维的数据结构。Tensor作为数据载体在各算子间流通，任何支持Tensor作为输入和输出的算子，都可以加入Graph的计算过程中。因为Tensor采用连续存储，所以定义Variable时，必须指定类型（Type）和空间大小（Shape），且该空间大小不支持修改。

TensorFlow通过Variable的方式实现Embedding机制，用于存储Embedding的Variable空间大小为[vocabulary_size, embedding_dimension]。在大规模稀疏特征场景下，存在以下弊端：

- vocabulary_size通常由ID空间决定，随着在线学习场景的ID不断增加，会导致vocabulary_size难以估计。
- ID通常为字符串类型且规模庞大，进行Embedding之前，需要将其Hash到vocabulary_size范围内：
  - 如果vocabulary_size过小，则导致Hash冲突率增加，不同特征可能查找到相同的Embedding，即特征减少。
  - 如果vocabulary_size过大，则导致Variable存储永远不会被查找的Embedding，即内存冗余。
- Embedding变量过大是导致模型增大的主要原因，即使通过正则手段降低某些特征的Embedding对整个模型效果的影响，也无法从模型中去除该Emebdding。

为解决上述问题，PAI-TF推出动态Embedding语义的EmbeddingVariable，在特征无损训练的条件下，以经济的方式使用内存资源，从而实现超大规模特征的离线训练和模型上线。PAI-TF提供EmbeddingVariable（3.1）及Feature_Column（3.3）API，推荐使用Feature_Column API，它可以自动增加字符串的Feature ID化流程。

EmbeddingVariable特性

- 动态Embedding。
  无需指定Vocabulary规模，只需要指定Embedding Dim，PAI-TF就可以根据训练，动态地扩展或缩减词典大小。适用于在线学习场景，同时省略了TensorFlow模型的数据预处理流程。
- Group Lasso正则。
  通常经过深度学习的Embedding变量规模超大，如果将其部署为在线服务，则会造成服务器压力。使用Group Lasso正则处理Embedding，可以减少模型部署的压力。
- 支持将原始特征值传入Embedding Lookup，省略了Hash等ID化操作，从而实现特征的无损训练。
- 支持Graph的Inference、Back Propagation及变量的导入导出，模型训练中通过Optimizer自动更新

EmbeddingVariable。

tf.get_embedding_variable接口说明

tf.get_embedding_variable接口返回一个已有的EmbeddingVariable变量或新建的EmbeddingVariable变量，接口定义如下。

```
get_embedding_variable(
    name,
    embedding_dim,
    key_dtype=dtypes.int64,
    value_dtype=None,
    initializer=None,
    regularizer=None,
    trainable=True,
    collections=None,
    caching_device=None,
    partitioner=None,
    validate_shape=True,
    custom_getter=None,
    constraint=None,
    steps_to_live=None
)
```

- name：Embedding Variable的名称。
- embedding_dim：Embedding的维度。例如8或64。
- key_dtype：Lookup时key的类型，默认值为int64。
- value_dtype： Embedding Vector的类型，仅支持float类型。
- initializer：Embedding Vector的初始值。
- trainable：是否被添加到GraphKeys.TRAINABLE_VARIABLES的Collection。
- partitioner：分区函数。
- steps_to_live：全局步长，用于自动淘汰过期特征。系统会删除全局步数超过该参数值的未更新特征。

EmbeddingVariable

EmbeddingVariable的结构如下。

```
class EmbeddingVariable(ResourceVariable)
  def total_count():
    # 返回Embedding当前的total_count，[rowCount，EmbeddingDim]。
  def read_value():
    raise NotImplementedError("...")
  def assign():
    raise NotImplementedError("...")
  def assign_add():
    raise NotImplementedError("...")
  def assign_sub():
    raise NotImplementedError("...")
```

- 支持读取稀疏数据的 `sparse_read()` 方法。如果查询的key不存在，则返回EmbeddingVariable初始化时，该key对应的initializer。

- 支持查询EmbeddingVariable词表总数的 `total_count()` 方法，该方法返回EmbeddingVariable的动态Shape值。
- 不支持全量读取EmbeddingVariable的 `read_value()` 方法。
- 不支持EmbeddingVariable的赋值方法，包括 `assign()` 、 `assign_add()` 及 `assign_sub()` 。

使用feature_column接口构建EmbeddingVariable

```
def tf.contrib.layers.sparse_column_with_embedding(column_name=column_name,
                                                   dtype=tf.string,
                                                   partition_num=None,
                                                   steps_to_live=None,
                                                   # 1120版本不支持以下两个参数，仅140lite版本
支持。
                                                   steps_to_live_l2reg=None,
                                                   l2reg_theta=None)
  # column_name: column name
  # dtype: type, default is tf.string
```

示例

- 使用底层tf.get_embedding_variable接口构建包含EmbeddingVariable的TensorFlow Graph。

```
#!/usr/bin/python
import tensorflow as tf
var = tf.get_embedding_variable("var_0",
                                embedding_dim=3,
                                initializer=tf.ones_initializer(tf.float32),
                                partitioner=tf.fixed_size_partitioner(num_shards=4))
shape = [var1.total_count() for var1 in var]
emb = tf.nn.embedding_lookup(var, tf.cast([0,1,2,5,6,7], tf.int64))
fun = tf.multiply(emb, 2.0, name='multiply')
loss = tf.reduce_sum(fun, name='reduce_sum')
opt = tf.train.FtrlOptimizer(0.1,
                             l1_regularization_strength=2.0,
                             l2_regularization_strength=0.00001)
g_v = opt.compute_gradients(loss)
train_op = opt.apply_gradients(g_v)
init = tf.global_variables_initializer()
sess_config = tf.ConfigProto(allow_soft_placement=True, log_device_placement=False)
with tf.Session(config=sess_config) as sess:
  sess.run([init])
  print(sess.run([emb, train_op, loss]))
  print(sess.run([emb, train_op, loss]))
  print(sess.run([emb, train_op, loss]))
  print(sess.run([shape]))
```

- 将EmbeddingVariable保存为Checkpoint。

```
#!/usr/bin/python
import tensorflow as tf
var = tf.get_embedding_variable("var_0",
                                embedding_dim=3,
                                initializer=tf.ones_initializer(tf.float32),
                                partitioner=tf.fixed_size_partitioner(num_shards=4))
emb = tf.nn.embedding_lookup(var, tf.cast([0,1,2,5,6,7], tf.int64))
init = tf.global_variables_initializer()
saver = tf.train.Saver(sharded=True)
print("GLOBAL_VARIABLES: ", tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES))
print("SAVEABLE_OBJECTS: ", tf.get_collection(tf.GraphKeys.SAVEABLE_OBJECTS))
checkpointDir = "/tmp/model_dir"
sess_config = tf.ConfigProto(allow_soft_placement=True, log_device_placement=False)
with tf.Session(config=sess_config) as sess:
  sess.run([init])
  print(sess.run([emb]))
  save_path = saver.save(sess, checkpointDir + "/model.ckpt", global_step=666)
  tf.train.write_graph(sess.graph_def, checkpointDir, 'train.pbtxt')
  print("save_path", save_path)
  print("list_variables", tf.contrib.framework.list_variables(checkpointDir))
```

- 从Checkpoint中恢复EmbeddingVariable变量。

```
#!/usr/bin/python
import tensorflow as tf
var = tf.get_embedding_variable("var_0",
                                embedding_dim=3,
                                initializer=tf.ones_initializer(tf.float32),
                                partitioner=tf.fixed_size_partitioner(num_shards=4))
emb = tf.nn.embedding_lookup(var, tf.cast([0,1,2,5,6,7], tf.int64))
init = tf.global_variables_initializer()
saver = tf.train.Saver(sharded=True)
print("GLOBAL_VARIABLES: ", tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES))
print("SAVEABLE_OBJECTS: ", tf.get_collection(tf.GraphKeys.SAVEABLE_OBJECTS))
checkpointDir = "/tmp/model_dir"
sess_config = tf.ConfigProto(allow_soft_placement=True, log_device_placement=False)
with tf.Session(config=sess_config) as sess:
  print("list_variables", tf.contrib.framework.list_variables(checkpointDir))
  saver.restore(sess, checkpointDir + "/model.ckpt-666")
  print(sess.run([emb]))
```

- 使用feature_colume接口构建包含EmbeddingVariable的TensorFlow Graph。

```
import tensorflow as tf
import os
columns_list=[]
columns_list.append(tf.contrib.layers.sparse_column_with_embedding(column_name="col_emb",
dtype=tf.string))
W = tf.contrib.layers.shared_embedding_columns(sparse_id_columns=columns_list,
        dimension=3,
        initializer=tf.ones_initializer(tf.float32),
        shared_embedding_name="xxxxx_shared")
ids={}
ids["col_emb"] = tf.SparseTensor(indices=[[0,0],[1,0],[2,0],[3,0],[4,0]], values=["aaaa",
"bbbbb","ccc","4nn","5b"], dense_shape=[5, 5])
emb = tf.contrib.layers.input_from_feature_columns(columns_to_tensors=ids, feature_column
s=W)
fun = tf.multiply(emb, 2.0, name='multiply')
loss = tf.reduce_sum(fun, name='reduce_sum')
opt = tf.train.FtrlOptimizer(0.1, l1_regularization_strength=2.0, l2_regularization_stren
gth=0.00001)
g_v = opt.compute_gradients(loss)
train_op = opt.apply_gradients(g_v)
init = tf.global_variables_initializer()
init_local = tf.local_variables_initializer()
sess_config = tf.ConfigProto(allow_soft_placement=True, log_device_placement=False)
with tf.Session(config=sess_config) as sess:
  sess.run(init)
  print("init global done")
  sess.run(init_local)
  print("init local done")
  print(sess.run([emb, train_op,loss]))
  print(sess.run([emb, train_op,loss]))
  print(sess.run([emb, train_op,loss]))
  print(sess.run([emb]))
```

> **说明** EmbeddingVariable仅支持Ftrl Optimizer、Adagrad Optimizer、Adam Optimizer及AdagradDecay Optimizer。

5.2.2.13. AdagradDecay Optimizer

本文为您介绍如何使用AdagradDecay Optimizer进行超大规模训练。

背景信息

超大规模模型的训练样本通常在10亿规模以上，且持续增量训练时间在一个月以上。为解决该问题，PAI-TF推出AdagradDecay优化器。

开启AdagradDecay Optimizer优化器

使用AdagradDecay Optimizer优化器进行超大规模训练，需要定义 `tf.train.AdagradDecayOptimizer` 。AdagradDecay Optimizer的使用方法与TensorFlow原生Optimizer的使用方法相同，具体定义如下。

```
class AdagradDecayOptimizer(optimizer.Optimizer):
  """Optimizer that implements the Adagrad algorithm with accumulator decay.
  Different from the original Adagrad algorithm, AdagradDecay performs decay
  at given step with given rate. So that the accumulator will not be infinity.
  """
  def __init__(self,
               learning_rate,
               global_step,
               initial_accumulator_value=0.1,
               accumulator_decay_step=100000,
               accumulator_decay_rate=0.9,
               use_locking=False,
               name="AdagradDecay"):
    """Construct a new AdagradDecay optimizer.
    Args:
      learning_rate: A `Tensor` or a floating point value.  The learning rate.
      global_step: global step variable, used for calculating t%T .
      initial_accumulator_value: A floating point value. Starting and baseline
        value for the accumulators, must be positive. The accumulators will not
        be less than it.
      accumulator_decay_step: When global_step reaches times of
        accumulator_decay_step, accumulator will be decayed with
        accumulator_decay_rate. accumulator *= accumulator_decay_rate
      accumulator_decay_rate: Decay rate as above described.
      use_locking: If `True` use locks for update operations.
      name: Optional name prefix for the operations created when applying
        gradients.  Defaults to "AdagradDecay".
    Raises:
      ValueError: If the `initial_accumulator_value`, `accumulator_decay_step`
        or `accumulator_decay_rate` is invalid.
    """
```

5.2.2.14. TableRecordDataset

您可以使用TableRecordDataset接口按照行读取MaxCompute表数据并构建数据流。
TensorFlow社区推荐在1.2及以上版本，使用Dataset接口代替线程和队列构建数据流。通过多个Dataset接口的组合变换生成计算数据，可以简化数据输入代码。

接口说明

PAI-TF提供的TableRecordDataset与原生TensorFlow RecordDataset相似，可以为数据变换（Transformation）的Dataset接口提供数据源。TableRecordDataset的接口定义如下。

```
class TableRecordDataset(Dataset):
  def __init__(self,
               filenames,
               record_defaults,
               selected_cols=None,
               excluded_cols=None,
                slice_id=0,
                slice_count=1,
               num_threads=0,
               capacity=0):
```

参数	描述
filenames	待读取的表名集合（列表），同一张表可以重复读取。
record_defaults	待读取列的数据类型或列为空时的默认数据类型。如果该类型与实际读取的列类型不符，或数据类型无法自动转换，则执行过程中系统会抛出异常。系统支持的数据类型包括FLOAT32、FLOAT64、INT32、INT64、BOOL及STRING。
selected_cols	选取的列，格式为英文逗号（,）分隔的字符串。
excluded_cols	排除的列，格式为英文逗号（,）分隔的字符串。不能同时使用excluded_cols和selected_cols。
slice_id	当前分区的编号。分布式读取时，系统根据slice_count将表平均分为多个分区，读取slice_id对应的分区。
slice_count	分布式读取时，总的分区数量，通常为Worker数量。
num_threads	预取数据时，每个访问表的内置Reader启用的线程（独立于计算线程）数量。取值范围为1~64。如果num_threads取值为0，则系统自动将新建的预取线程数配置为计算线程池线程数的1/4。 **说明** 因为I/O对每个模型的整体计算影响不同，所以提高预取线程数，不一定可以提升整体模型的训练速度。
capacity	读取表的总预取量，单位为行数。如果num_threads大于1，则每个线程的预取量为capacity/num_threads行（向上取整）。如果capacity为0，则内置Reader根据所读表的前N行（系统默认N=256）平均值自动配置总预取量，使得每个线程的预取数据约占空间64 MB。 **说明** 如果手动配置预取量，当单线程的预取量大于1 GB，系统仅输出告警信息以提示您检查配置，而不会中断程序运行。

说明 如果MaxCompute表字段为DOUBLE类型，则TensorFlow中需要使用np.float格式与其对应。

返回值

TableRecordDataset返回一个新的Dataset对象，可以作为Pipeline工作流构建的输入。

```
### other definition codes was ignored here.
# Suppose an odps table named 'sample_table' was built in
# 'test' project, which includes 5 columns:
#   (itemid bigint, name string, price double,
#    virtual bool, tags string)
# Table name would be passed from run commands.
tables = ["odps://test/tables/sample_table"]
# Firstly, we define a new TableRecordDataset to read itemid and price.
dataset = tf.data.TableRecordDataset(tables,
                                     record_defaults = (0, 0.0),
                                     selected_cols = "itemid, price")
# Get a batch of 128
dataset = dataset.batch(128)
# Set epoch as 10
dataset = dataset.repeat(10)
# At last we got a batch of ids and prices.
[ids, prices] = dataset.make_one_shot_iterator().get_next()
### Then we do other graph construction and finally run the graph.
```

执行Session时调用 `get_next()` 方法，从表中读取128行数据，并根据record_defaults指定的类型将每列数据解析为对应类型的Tensor。其中 `get_next()` 返回的output_types需要与record_defaults的参数类型相同，output_shapes的Tensor Shape需要与record_defaults的元素数量一致。

Console参数

- 如果将表作为输入，提交任务时，需要使用-Dtables配置待访问的表名。

  ```
  pai -name tensorflow1120_cpu_ext -Dtables=odps://algo_platform_dev/tables/sample;
  ```

- 如果读取2张以上的表，则需要使用英文逗号（,）分隔多个表名。

  ```
  pai -name tensorflow1120_cpu_ext -Dtables=odps://algo_platform_dev/tables/sample,odps://algo_platform_dev/tables/sample2
  ```

- 如果访问分区表，则需要在表名后添加分区。

  ```
  pai -name tensorflow1120_cpu_ext -Dtables=odps://algo_platform_dev/tables/sample/pt=1;
  ```

示例

以逻辑回归（Logisteristic Regression）为例，介绍如何使用TableRecordDataset读取表数据并进行模型训练。

1. 数据准备。
   TableRecordReader是将整行数据做为一个字符串导入MaxCompute表，读取之后再进行解析。而使用TableRecordDataset时，建议MaxCompute数据表按照列存放相应的数据，Dataset接口会将表中的数据以指定类型的Tensor返回。

i. 创建表。
使用MaxCompute创建一个包含四列数据的表。

```
odps@ algo_platform_dev>create table sample (col1 double, col2 double, col3 double,
col4 double);
Data Health Manager:Your health synthesize score is 5, so, your job priority is 7
ID = 201803050245351****6Vgsxo2
OK
odps@ algo_platform_dev>read sample;
+------------+------------+------------+------------+
| col1       | col2       | col3       | col4       |
+------------+------------+------------+------------+
+------------+------------+------------+------------+
```

ii. 导入数据。
下载测试数据，并使用MaxCompute Console Tunnel命令将其导入MaxCompute表。

```
#查看下载的测试数据。
$head -n 3 sample.csv
0,0,0.017179100152531324,1
0,1,0.823381420409002,1
0,2,1.6488850495540865,1
```

```
#将数据导入MaxCompute表。
odps@ algo_platform_dev>tunnel upload /tmp/data/sample.csv sample -fd=,;
Upload session: 20180305135640c8cc650a0000****
Start upload:sample.csv
Using \n to split records
Upload in strict schema mode: true
Total bytes:260093   Split input to 1 blocks
2018-03-05 13:56:40 scan block: '1'
2018-03-05 13:56:40 scan block complete, blockid=1
2018-03-05 13:56:40 upload block: '1'
2018-03-05 13:56:41 upload block complete, blockid=1
upload complete, average speed is 254 KB/s
OK
odps@ algo_platform_dev>read sample 3;
+------------+------------+------------+------------+
| col1       | col2       | col3       | col4       |
+------------+------------+------------+------------+
| 0.0        | 0.0        | 0.017179100152531324 | 1.0        |
| 0.0        | 1.0        | 0.823381420409002 | 1.0        |
| 0.0        | 2.0        | 1.6488850495540865 | 1.0        |
+------------+------------+------------+------------+
```

> **说明** 因为该测试数据的每行内容使用英文逗号（,）分隔，所以使用 `-fd=,` 配置分隔符为英文逗号（,）才能将每行数据分为四列导入至相应的MaxCompute表。

2. 构建输入数据和模型。
构建输入数据的示例代码如下。除无需定义tf.train.Coordinator和运行start_queue_runners以外，其余代码与使用TableRecordReader的代码相同。

```
#define the input
def input_fn():
    dataset = tf.data.TableRecordDataset([FLAGS.tables], record_defaults=[1.0]*4).repea
t().batch(128)
    v1, v2, v3, v4 = dataset.make_one_shot_iterator().get_next()
    labels = tf.reshape(tf.cast(v4, tf.int32), [128])
    features = tf.stack([v1, v2, v3], axis=1)
    return features, labels
```

完整的示例代码*lr_dataset.py*如下。

```
import tensorflow as tf
tf.app.flags.DEFINE_string("tables", "", "tables info")
FLAGS = tf.app.flags.FLAGS
#define the input
def input_fn():
    dataset = tf.data.TableRecordDataset([FLAGS.tables], record_defaults=[1.0]*4).repea
t().batch(128)
    v1, v2, v3, v4 = dataset.make_one_shot_iterator().get_next()
    labels = tf.reshape(tf.cast(v4, tf.int32), [128])
    features = tf.stack([v1, v2, v3], axis=1)
    return features, labels
#construct the model
def model_fn(features, labels):
    W = tf.Variable(tf.zeros([3, 2]))
    b = tf.Variable(tf.zeros([2]))
    pred = tf.matmul(features, W) + b
    loss =  tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=pred,l
abels=labels))
    # Gradient Descent
    optimizer = tf.train.GradientDescentOptimizer(0.05).minimize(loss)
    return loss, optimizer
features, labels = input_fn()
loss, optimizer = model_fn(features, labels)
init = tf.global_variables_initializer()
local_init = tf.local_variables_initializer()
sess = tf.Session()
sess.run(init)
sess.run(local_init)
for step in range(10000):
    _, c = sess.run([optimizer, loss])
    if step % 2000 == 0:
        print("loss," , c)
```

3. 提交任务。

```
odps@ algo_platform_dev>pai -name tensorflow1120_cpu_ext -Dtables=odps://algo_platform_
dev/tables/sample -Dscript=file:///tmp/lr_dataset.py;
```

4. 查看执行结果。
   单击提交任务返回的Logview链接，查看执行结果。

```
start launching tensorflow job
('loss,', 0.6931472)
('loss,', 0.007929571)
('loss,', 0.0016527221)
('loss,', 0.0023481336)
('loss,', 0.0011788738)
```

5.2.2.15. WorkQueue

在大规模分布式异步训练中，您可以使用WorkQueue进行弹性数据切分，以缓解长尾效应，从而降低模型训练所需的时间。本文介绍WorkQueue的调用格式、参数及其提供的方法。同时，以文件数据源和MaxCompute表数据源为例，介绍实现数据切分的经典示例。

背景信息

在大规模分布式异步训练中，如果每个Worker读取相同数量的样本，则慢节点的训练时长会远大于其他节点，造成长尾效应。并且随着训练规模扩大，长尾效应会越来越严重，导致训练的整体数据吞吐降低，进而增加训练时间。

为解决该问题，PAI提供了`pai.data.WorkQueue`类，支持对多种数据源进行弹性数据切分，让慢节点获取较少的训练数据，快节点获取更多的训练数据，以缓解长尾效应，从而降低模型训练所需的时间。

版本配套关系

- Python版本：Python 2.7
- PAI-TensorFlow版本：PAI-TensorFlow 1.12

pai.data.WorkQueue

- 功能
  工作项队列类，用于统一管理所有Worker上的工作项。每个Worker的当前剩余工作项被消费完后，会从同一个WorkQueue获得新的工作项，并将其作为数据源进行训练，从而使得训练快的Worker获得更多的工作项进行训练，以减少长尾效应。
- 格式

```
class pai.data.WorkQueue(works, num_epochs=1, shuffle=True, seed=None,
                         prefix=None, num_slices=None, name='work_queue')
```

- 参数

参数名	描述	类型	是否必选	默认值
works	文件名或表名列表。	LIST of STRING	是	无
num_epochs	读取全部数据的次数。	INT	否	1
shuffle	是否每个Epoch都随机重洗数据，取值如下： ○ True：每个Epoch都随机重洗数据。 ○ False：不进行数据重洗。	BOOL	否	True

参数名	描述	类型	是否必选	默认值
seed	重洗数据的随机种子。取值为None时，表示系统自动选取随机种子。	INT	否	None
prefix	工作项（文件名或表名）的前缀。取值为None时，表示无前缀。	STRING	否	None
num_slices	工作项的总数量。集群越不稳定，需要将工作项总数量配置的越大，通常为Worker数量的10倍以上。取值为None时，表示不分片。	INT	否	None
num_clients	工作队列支持的最大工作抢占并发数。	INT	否	1
name	工作队列的名称。	STRING	否	work_queue

- 返回值
  返回WorkQueue对象，您可以使用该对象调用 `pai.data.WorkQueue` 类提供的方法。

pai.data.WorkQueue提供的方法

`pai.data.WorkQueue` 类提供以下方法：

- `take`
  - 功能
    从全局工作队列获取一个工作项，并下载至本地。
  - 格式
    ```
    WorkQueue.take()
    ```
  - 参数
    无
  - 返回值
    返回值类型为 `tensorflow.Tensor` 。
- `input_dataset`
  - 功能
    返回一个Dataset，其每个元素为一个工作项。
  - 格式
    ```
    WorkQueue.input_dataset()
    ```
  - 参数
    无
  - 返回值
    返回值类型为 `tensorflow.data.Dataset` 。
- `input_producer`
  - 功能
    返回全局工作队列在本地的代理队列，为Reader类Op使用。

- 格式

```
WorkQueue.input_producer()
```

- 参数
  无
- 返回值
  返回值类型为 `tensorflow.FIFOQueue` 。

- `add_summary`
  - 功能
    在Tensorboard中显示WorkQueue的资源水位信息。
  - 格式

```
WorkQueue.add_summary()
```

  - 参数
    无
  - 返回值
    无

典型示例

`pai.data.WorkQueue` 类支持对多种数据源进行弹性数据切分，以下分别以文件数据源和MaxCompute表数据源为例，介绍如何使用 `pai.data.WorkQueue` 类实现弹性数据切分（仅提供核心代码片段）：

- 文件数据源

```
import pai
# ...
# path1、path2及path3表示需要读取的文件列表。
# shuffle取值为True，表示每个Epoch都随机化打散文件路径。
work_queue = pai.data.WorkQueue([path1, path2, path3], shuffle=True)
# 让WorkQueue支持TensorBoard。
work_queue.add_summary()
# 创建文件读取器。
reader = tf.TextLineReader()
# 从文件列表中读取2条记录。
keys, values = reader.read_up_to(work_queue.input_producer(), num_records=2)
with tf.train.MonitoredTrainingSession() as sess:
  sess.run(...)
```

- MaxCompute表数据源

- TableRecordDataset数据源

```
import pai
#...
# odps_path1、odps_path2及odps_path3表示需要读取的MaxCompute表列表。
# shuffle取值为True，表示每个Epoch都随机化打散表路径。
# num_slices为工作项总数量。
# FLAGS.num_workers为训练中的Worker数量。
work_queue = pai.data.WorkQueue([odps_path1, odps_path2, odps_path3],shuffle=True,
                    num_slices=FLAGS.num_workers * 10)
# 创建文件名Dataset。
filenames_dataset = work_queue.input_dataset()
# 将dataset作为文件名传入TableRecordDataset。
dataset = tf.data.TableRecordDataset(filenames_dataset, record_defaults=...)
```

关于 `tf.data.TableRecordDataset` 接口的调用，请参见TableRecordDataset。

- TableRecordReader数据源

```
import pai
# ...
# odps_path1、odps_path2及odps_path3表示需要读取的MaxCompute表列表。
# shuffle取值为True，表示每个Epoch都随机化打散表路径。
# num_slices为工作项总数量。
# FLAGS.num_workers为训练中的Worker数量。
work_queue = pai.data.WorkQueue(
  [odps_path1, odps_path2, odps_path3], shuffle=True, num_slices=FLAGS.num_workers * 10
)
# 创建表读取器。
reader = tf.TableRecordReader()
# 从表中读取2条记录。
keys, values = reader.read_up_to(work_queue.input_producer(), num_records=2)
```

5.2.2.16. 使用案例

5.2.2.16.1. 使用TensorFlow实现分布式DeepFM算法

本文为您介绍如何使用TensorFlow实现分布式DeepFM算法。

前提条件

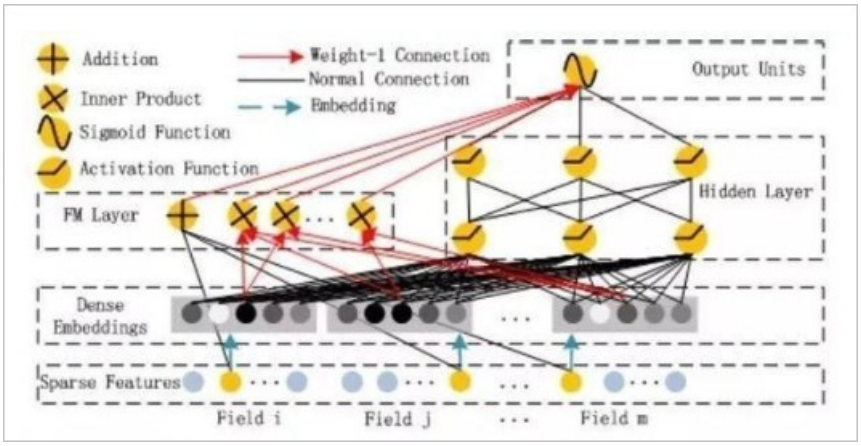
- 开通OSS，并创建Bucket，详情请参见开通OSS服务和创建存储空间。

  > 注意 创建Bucket时，不要开通版本控制，否则同名文件无法覆盖。

- 完成OSS访问授权，详情请参见云产品依赖与授权：Designer。

背景信息

DeepFM算法对应Wide&Deep部分，且将LR替换为FM，从而避免人工特征工程。



训练数据源为pai_online_project.dwd_avazu_ctr_deepmodel_train，测试数据源为pai_online_project.dwd_avazu_ctr_deepmodel_test，都是公开数据源，您可以直接使用。

操作步骤

1. 下载模型文件。
2. 修改模型配置代码。
   i. 修改特征参数，每个特征需要配置embedding_dim、hash_buckets及default_value。

   ```
   self.fields_config_dict['hour'] = {'field_name': 'field1', 'embedding_dim': self.em
   bedding_dim, 'hash_bucket': 50, 'default_value': '0'}
   self.fields_config_dict['c1'] = {'field_name': 'field2', 'embedding_dim': self.embe
   dding_dim, 'hash_bucket': 10, 'default_value': '0'}
   ```

   DeepFM模型中，需要将所有特征的embedding_dim配置为相同值，而Wide&Deep模型无此限制。对于user_id和itemid，建议将hash_buckets配置为较大值，而其他取值较少的特征建议将hash_buckets配置为较小值。

   ii. 配置模型，推荐使用DeepFM（deepfm）和Wide&Deep（wdl）。

   ```
   tf.app.flags.DEFINE_string("model", 'deepfm', "model {'wdl', 'deepfm'}")
   ```

   iii. 配置分布式参数。

   ```
   tf.app.flags.DEFINE_string("job_name", "", "job name")
   tf.app.flags.DEFINE_integer("task_index", None, "Worker or server index")
   tf.app.flags.DEFINE_string("ps_hosts", "", "ps hosts")
   tf.app.flags.DEFINE_string("worker_hosts", "", "worker hosts")
   ```

   提交训练任务时，只需要配置cluster（详情请参见下述提交训练任务步骤），系统自动生成分布式参数。

iv. 配置输入数据。

```
  def _parse_batch_for_tabledataset(self, *args):
    label = tf.reshape(args[0], [-1])
    fields = [tf.reshape(v, [-1]) for v in args[1:]]
    return dict(zip(self.feas_name, fields)), label
  def train_input_fn_from_odps(self, data_path, epoch=10, batch_size=1024, slice_id
=0, slice_count=1):
    with tf.device('/cpu:0'):
      dataset = tf.data.TableRecordDataset([data_path], record_defaults=self.record
_defaults,
                           slice_count=slice_count, slice_id=slice_id)
      dataset = dataset.batch(batch_size).repeat(epoch)
      dataset = dataset.map(self._parse_batch_for_tabledataset, num_parallel_calls=
8).prefetch(100)
      return dataset
  def val_input_fn_from_odps(self, data_path, epoch=1, batch_size=1024, slice_id=0,
slice_count=1):
    with tf.device('/cpu:0'):
      dataset = tf.data.TableRecordDataset([data_path], record_defaults=self.record
_defaults,
                           slice_count=slice_count, slice_id=slice_id)
      dataset = dataset.batch(batch_size).repeat(epoch)
      dataset = dataset.map(self._parse_batch_for_tabledataset, num_parallel_calls=
8).prefetch(100)
      return dataset
```

如果需要进行特征变换，建议在模型外通过MaxCompute实现，从而节约训练开销。如果在模型中进行特征变换，建议在_parse_batch_for_tabledataset函数中实现。

3. 上传已修改的模型文件至OSS。
4. （可选）提交训练任务。

> **说明** 如果没有训练完成的模型文件，则必须执行该步骤。反之，可以跳过该步骤，直接提交离线推理任务。

根据配置的模型类型，选择提交命令：

- DeepFM

```
pai -name tensorflow1120_ext
-project algo_public
-Dbuckets='oss://bucket_name.oss-cn-region-internal.aliyuncs.com/'
-Darn=''
-Dscript='oss://bucket_name.oss-cn-region-internal.aliyuncs.com/demo/deepfm_pai_ctr.p
y'
-Dtables='odps://pai_online_project/tables/dwd_avazu_ctr_predict_deepmodel_train,odps
://pai_online_project/tables/dwd_avazu_ctr_predict_deepmodel_test'
-DuserDefinedParameters="--task_type='train' --model='deepfm' --checkpoint_dir='oss:/
/bucket_name/path/' --output_dir='oss://bucket_name/path/'"
-Dcluster='{\"ps\":{\"count\":2,\"cpu\":1200,\"memory\":10000},\"worker\":{\"count\":
8,\"cpu\":1200,\"gpu\":100,\"memory\":30000}}';
```

需要根据实际情况修改project_name、bucket_name、账号arn（请参见角色ARN）及地域。

- Wide&Deep

```
pai -name tensorflow1120_ext
-project algo_public
-Dbuckets='oss://bucket_name.oss-cn-region-internal.aliyuncs.com/'
-Darn=''
-Dscript='oss://bucket_name.oss-cn-region-internal.aliyuncs.com/demo/deepfm_pai_ctr.py'
-Dtables='odps://pai_online_project/tables/dwd_avazu_ctr_predict_deepmodel_train,odps://pai_online_project/tables/dwd_avazu_ctr_predict_deepmodel_test'
-DuserDefinedParameters="--task_type='train' --model='wdl' --checkpoint_dir='oss://bucket_name/path/' --output_dir='oss://bucket_name/path/'"
-Dcluster='{\"ps\":{\"count\":2,\"cpu\":1200,\"memory\":10000},\"worker\":{\"count\":8,\"cpu\":1200,\"gpu\":100,\"memory\":30000}}';
```

需要根据实际情况修改project_name、bucket_name、账号arn（请参见角色ARN）及地域。

因为该版本为PS分布式，所以需要配置cluster。示例中的cluster参数表示申请两个PS节点和8个Worker节点。同时，每个PS节点拥有12个CPU和10 GB内存，每个Worker节点拥有1个GPU、12个CPU及30 GB内存。各参数的具体介绍请参见PAI-TF任务参数介绍。

5. 提交离线推理任务。

```
pai -name tensorflow1120_ext
-project algo_public
-Dbuckets='oss://bucket_name.oss-cn-region-internal.aliyuncs.com/'
-Darn=''
-Dscript='oss://bucket_name.oss-cn-region-internal.aliyuncs.com/demo/deepfm_pai_ctr.py'
-Dtables='odps://pai_online_project/tables/dwd_avazu_ctr_predict_deepmodel_train,odps://pai_online_project/tables/dwd_avazu_ctr_predict_deepmodel_test'
-DuserDefinedParameters="--task_type='predict' --model='deepfm' --checkpoint_dir='oss://bucket_name/path/' --output_dir='oss://bucket_name/path/'"
-Dcluster='{\"worker\":{\"count\":8,\"cpu\":1200,\"gpu\":100,\"memory\":30000}}'
-Doutputs='odps://project_name/tables/output_table_name';
```

需要根据实际情况修改project_name、bucket_name、账号arn（请参见角色ARN）及地域。
离线推理需要一张已创建的outputs表，每次执行推理任务的结果会覆盖该表。创建表的示例命令如下。

```
drop table project_name.output_table_name;
create table project_name.output_table_name
(
   probabilities STRING
   ,logits STRING
)STORED AS ALIORC;
```

6. 导出训练完成的模型文件。

```
pai -name tensorflow1120_ext
-project algo_public
-Dbuckets='oss://bucket_name.oss-cn-region-internal.aliyuncs.com/'
-Darn=''
-Dscript='oss://bucket_name.oss-cn-region-internal.aliyuncs.com/demo/deepfm_pai_ctr.py'
-Dtables='odps://pai_online_project/tables/dwd_avazu_ctr_predict_deepmodel_train,odps://
/pai_online_project/tables/dwd_avazu_ctr_predict_deepmodel_test'
-DuserDefinedParameters="--task_type='savemodel' --model='deepfm' --checkpoint_dir='oss
://bucket_name/path/' --output_dir='oss://bucket_name/path/'"
-Dcluster='{\"ps\":{\"count\":2,\"cpu\":1200,\"memory\":10000},\"worker\":{\"count\":8,
\"cpu\":1200,\"gpu\":100,\"memory\":30000}}';
```

需要根据实际情况修改project_name、bucket_name、账号arn（请参见角色ARN）及地域。系统实际使用单Worker执行导出模型任务。

5.2.2.16.2. 使用TensorFlow实现图片分类

本文为您介绍如何使用深度学习框架TensorFlow，快速搭架图像识别的预测模型。

前提条件

- 已创建OSS Bucket，并完成了OSS授权，详情请参见创建存储空间和PAI访问云产品授权：OSS。

  > 注意 创建Bucket时，不要开通版本控制，否则可能导致训练失败。

- 已开启GPU，详情请参见云产品依赖与授权：Designer。

背景信息

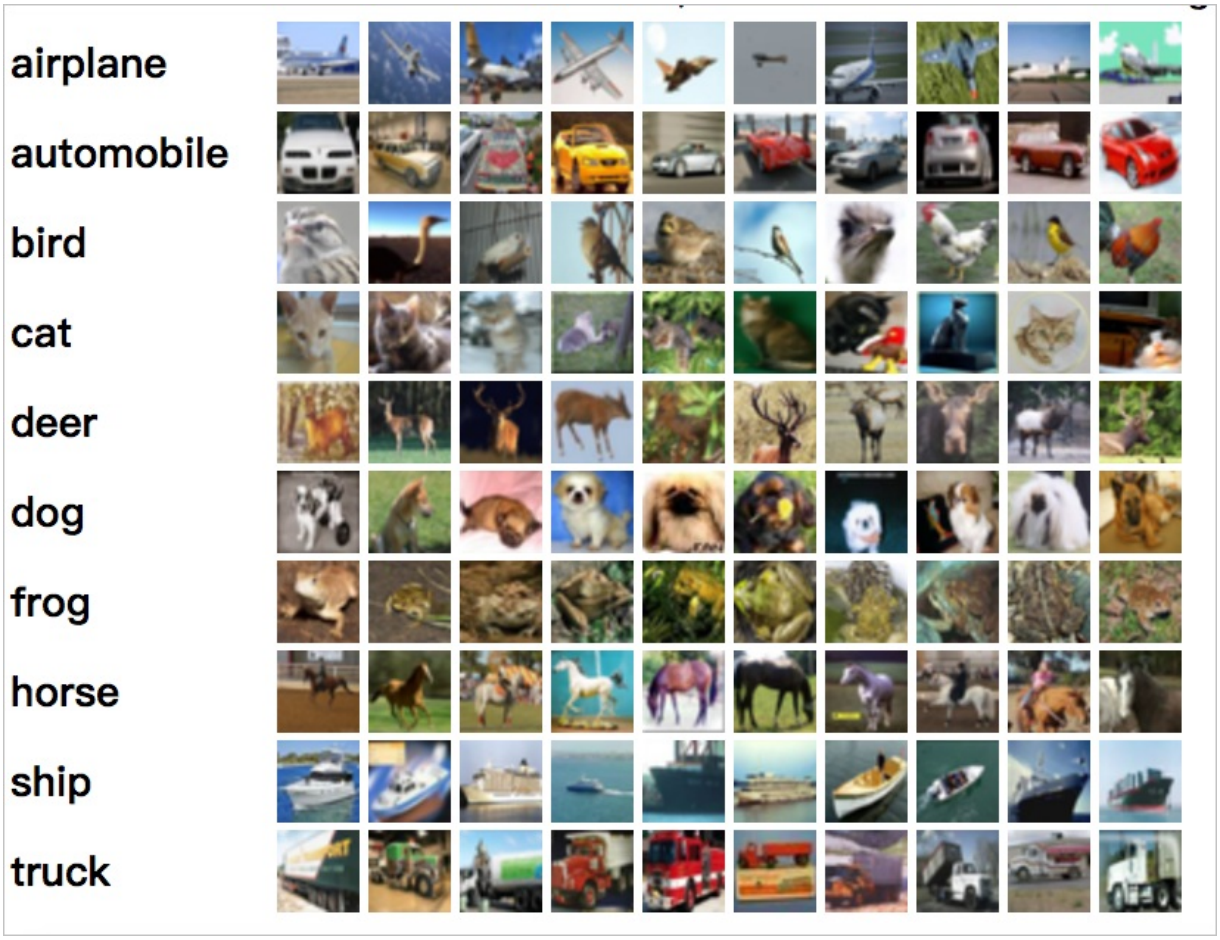
随着互联网发展，产生了大量图片及语音数据。如何有效利用这些非结构化数据，一直是困扰数据挖掘工程师的一道难题。主要原因包括：

- 通常需要使用深度学习算法，上手门槛高。
- 通常需要依赖GPU计算引擎，计算资源费用高。

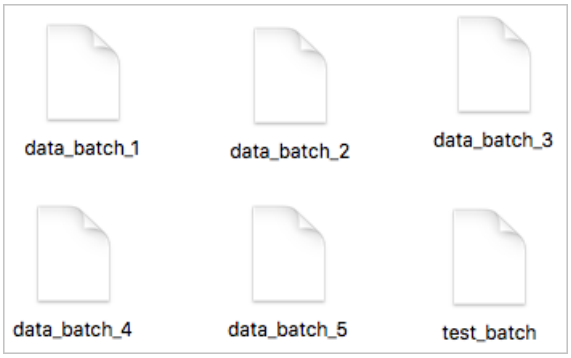
PAI-Studio已经预置了使用深度学习框架实现图片分类的模板，您可以直接从模板创建实验，并将其复用到图片鉴黄、人脸识别及物体检测等领域。

数据集

本实验使用CIFAR-10数据集，该数据集包含6万张像素为32*32的彩色图片，共10个类别，分别为飞机、汽车、鸟、毛、鹿、狗、青蛙、马、船及卡车，如下图所示。您可以下载该数据集及相关代码，详情请参见CIFAR 10案例。

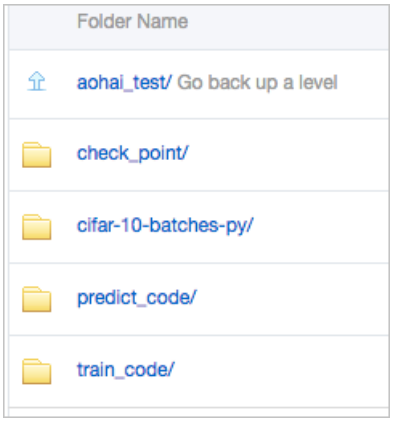


使用过程中将该数据集拆分为训练数据集（5万张图片）和预测数据集（1万张图片）。其中5万张图片的训练数据集又被拆分为5个data_batch，1万张图片的预测数据集组成test_batch，如下图所示。



数据准备

将本实验的数据集和相关代码上传至OSS的Bucket路径。例如，在OSS的Bucket下创建aohai_test文件夹及四个子文件夹，如下图所示。



每个文件夹的作用如下：

- check_point：存储实验生成的模型。

  > 说明 从PAI-Studio模板创建实验后，必须手动将TensorFlow组件的**checkpoint输出目录/模型输入目录**参数配置为已有的OSS文件夹路径，整个实验才能运行。本实验中，将**checkpoint输出目录/模型输入目录**配置为check_point文件夹路径。

- cifar-10-batches-py：存储训练数据集和预测数据集对应的数据源文件cifar-10-batcher-py和预测集文件bird_mount_bluebird.jpg。
- train_code：存储训练数据，即cifar_pai.py。
- predict_code：存储cifar_predict_pai.py。

使用TensorFlow实现图片分类

1. 进入PAI-Studio项目空间。
   i. 登录PAI控制台。
   ii. 在左侧导航栏，选择**模型开发和训练 > 可视化建模（Studio）**。
   iii. 在页面左上方，选择使用服务的地域。
   iv. （可选）在**PAI可视化建模**页面的搜索框，输入项目名称，搜索项目。
   v. 单击待打开的项目**操作**列下的**进入机器学习**。
2. 构建实验。
   i. 在PAI-Studio项目空间的左侧导航栏，单击**首页**。
   ii. 在**模板列表**，单击**Tensorflow图片分类**下的**从模板创建**。

iii. 在**新建实验**对话框，配置参数（可以全部使用默认参数）。

参数	描述
名称	输入TensorFlow图片分类。
项目	不支持修改。
描述	输入使用TensorFlow实现图片分类。
位置	选择我的实验。

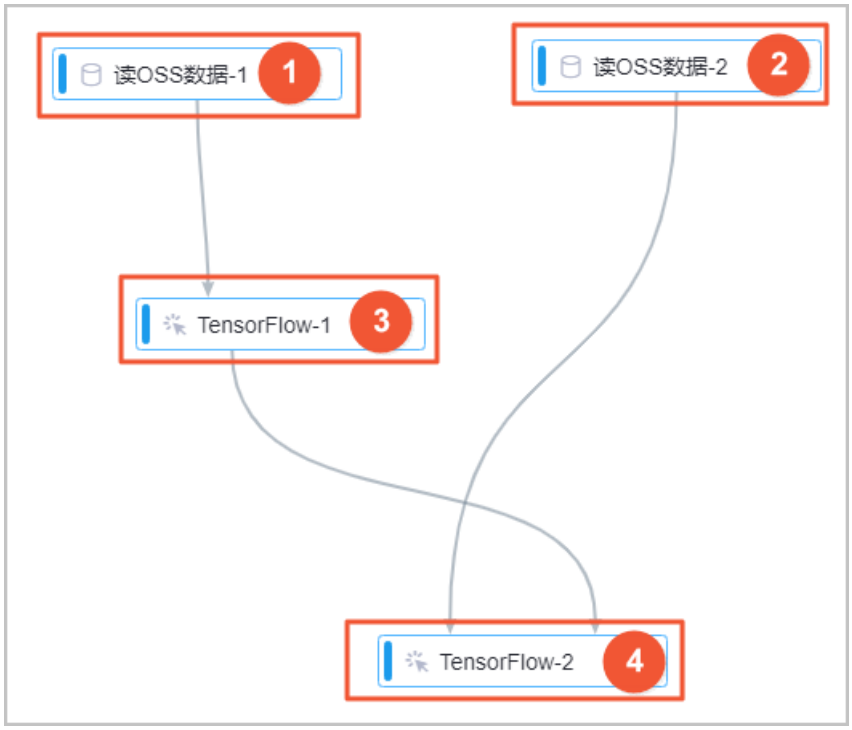
iv. 单击**确定**。

v. （可选）等待大约十秒钟，在PAI-Studio控制台的左侧导航栏，单击**实验**。

vi. （可选）在我的实验下，单击TensorFlow图片分类_XX。

其中我的实验为已配置的实验**位置**，TensorFlow图片分类_XX为已配置的实验**名称**（_XX为系统自动添加的实验序号）。

vii. 系统根据预置的模板，自动构建实验，如下图所示。



区域	描述
①	训练数据，系统已自动配置了本实验的训练数据集路径。如果使用其他数据集，只需单击画布中的**读OSS数据-1**，在右侧**IO/字段设置**面板，将**OSS数据路径**配置为存储训练数据的OSS路径。
②	预测数据，系统已自动配置了本实验的预测数据集路径。如果使用其他数据集，只需单击画布中的**读OSS数据-2**，在右侧**IO/字段设置**面板，将**OSS数据路径**配置为存储预测数据的OSS路径。

区域	描述
③	使用TensorFlow训练模型，只需要手动配置**checkpoint输出目录/模型输入目录**，其他参数使用默认值即可。该组件的参数与OSS路径下的文件对应关系如下： ■ **Python代码文件**：配置为OSS路径中的cifar_pai.py。 ■ **OSS数据源目录**：配置为OSS路径中的cifar-10-batches-py文件夹，系统会自动从上游**读OSS数据-1**节点同步数据。 ■ **checkpoint输出目录/模型输入目录**：配置为OSS路径中的check_point文件夹，用于存储输出模型。
④	生成预测结果，只需要手动配置**checkpoint输出目录/模型输入目录**，其他参数使用默认值即可。该组件的参数与OSS路径下的文件对应关系如下： ■ **Python代码文件**：配置为OSS路径中的cifar_predict_pai.py。 ■ **OSS数据源目录**：配置为OSS路径中的cifar-10-batches-py文件夹，系统会自动从上游**读OSS数据-2**节点同步数据。 ■ **checkpoint输出目录/模型输入目录**：需要与TensorFlow训练组件的模型输出目录保持一致，即配置为OSS路径中的check_point文件夹。

3. 运行实验并查看输出结果。
   i. 单击画布上方的**运行**。
   ii. 实验运行结束后，您可以在配置的OSS路径（**checkpoint输出目录/模型输入目录**）下查看预测结果。

训练代码解析

针对cifar_pai.py文件中的关键代码进行解析：

- 构建CNN图片训练模型

```
network = input_data(shape=[None, 32, 32, 3],
                     data_preprocessing=img_prep,
                     data_augmentation=img_aug)
    network = conv_2d(network, 32, 3, activation='relu')
    network = max_pool_2d(network, 2)
    network = conv_2d(network, 64, 3, activation='relu')
    network = conv_2d(network, 64, 3, activation='relu')
    network = max_pool_2d(network, 2)
    network = fully_connected(network, 512, activation='relu')
    network = dropout(network, 0.5)
    network = fully_connected(network, 10, activation='softmax')
    network = regression(network, optimizer='adam',
                         loss='categorical_crossentropy',
                         learning_rate=0.001)
```

- 训练生成模型model.tfl

```
    model = tflearn.DNN(network, tensorboard_verbose=0)
    model.fit(X, Y, n_epoch=100, shuffle=True, validation_set=(X_test, Y_test),
              show_metric=True, batch_size=96, run_id='cifar10_cnn')
    model_path = os.path.join(FLAGS.checkpointDir, "model.tfl")
    print(model_path)
    model.save(model_path)
```

预测代码解析

针对cifar_predict_pai.py文件中的关键代码进行解析。首先读入图片bird_bullocks_oriole.jpg，将其调整为32*32像素大小。然后传入model.predict预测函数评分，返回这张图片对应的十种分类['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']的权重。最后将权重最高的一个分类作为预测结果返回。

```
    predict_pic = os.path.join(FLAGS.buckets, "bird_bullocks_oriole.jpg")
    img_obj = file_io.read_file_to_string(predict_pic)
    file_io.write_string_to_file("bird_bullocks_oriole.jpg", img_obj)
    img = scipy.ndimage.imread("bird_bullocks_oriole.jpg", mode="RGB")
    # Scale it to 32x32
    img = scipy.misc.imresize(img, (32, 32), interp="bicubic").astype(np.float32, casting='
unsafe')
    # Predict
    prediction = model.predict([img])
    print (prediction[0])
    print (prediction[0])
    #print (prediction[0].index(max(prediction[0])))
    num=['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']
    print ("This is a %s"%(num[prediction[0].index(max(prediction[0]))]))
```

5.2.2.16.3. 模型仓库（FastNN）

PAI模型仓库FastNN（Fast Neural Networks）是一个基于PAISoar的分布式神经网络仓库。目前FastNN已经支持了Inception、Resnet、VGG等经典算法，后续会逐步开放更多的先进模型。目前FastNN已经内置于PAI-Studio平台中，并且可以直接在该平台中使用。

准备数据源

为了方便在PAI控制台上试用FastNN，cifar10、mnist、flowers数据已下载并转换为tfrecord后存储在公开OSS上，可通过PAI的**读数据表**或**OSS数据同步**组件访问。存储OSS的路径如下。

数据集	分类数	训练集	测试集	存储路径
mnist	10	3320	350	• 北京：oss://pai-online-beijing.oss-cn-beijing-internal.aliyuncs.com/fastnn-data/mnist/ • 上海：oss://pai-online.oss-cn-shanghai-internal.aliyuncs.com/fastnn-data/mnist/

数据集	分类数	训练集	测试集	存储路径
cifar10	10	50000	10000	• 北京：oss://pai-online-beijing.oss-cn-beijing-internal.aliyuncs.com/fastnn-data/cifar10/ • 上海：oss://pai-online.oss-cn-shanghai-internal.aliyuncs.com/fastnn-data/cifar10/
flowers	5	60000	10000	• 北京：oss://pai-online-beijing.oss-cn-beijing-internal.aliyuncs.com/fastnn-data/flowers/ • 上海：oss://pai-online.oss-cn-shanghai-internal.aliyuncs.com/fastnn-data/flowers/

FastNN库已支持读取tfrecord格式的数据，并基于TFRecordDataset接口实现dataset pipeline以供模型训练试用，几乎可掩盖数据预处理时间。另外，由于目前FastNN库在数据分片方面不够精细，建议您在准备数据时，尽量保证数据能平均分配到每台机器，即：

- 每个tfreocrd文件的样本数量基本一致。
- 每个worker处理的tfrecord文件数量基本一致。

如果数据格式同为tfrecord，可参考datasets目录下的cifar10、mnist和flowers等各文件实现dataset pipeline。以cifar10数据为例，实现方法如下。

假设cifar10数据的key_to_features格式为如下。

```
features={
        'image/encoded': tf.FixedLenFeature((), tf.string, default_value=''),
        'image/format': tf.FixedLenFeature((), tf.string, default_value='png'),
        'image/class/label': tf.FixedLenFeature(
          [], tf.int64, default_value=tf.zeros([], dtype=tf.int64)),
}
```

1. 在datasets目录下创建数据解析文件*cifar10.py*，并编辑内容。

```
"""Provides data for the Cifar10 dataset.
The dataset scripts used to create the dataset can be found at:
datasets/download_and_covert_data/download_and_convert_cifar10.py
"""
from __future__ import division
from __future__ import print_function
import tensorflow as tf
"""Expect func_name is 'parse_fn'
"""
def parse_fn(example):
  with tf.device("/cpu:0"):
    features = tf.parse_single_example(
      example,
      features={
        'image/encoded': tf.FixedLenFeature((), tf.string, default_value=''),
        'image/format': tf.FixedLenFeature((), tf.string, default_value='png'),
        'image/class/label': tf.FixedLenFeature(
          [], tf.int64, default_value=tf.zeros([], dtype=tf.int64)),
      }
    )
    image = tf.image.decode_jpeg(features['image/encoded'], channels=3)
    label = features['image/class/label']
    return image, label
```

2. 在*datasets/dataset_factory.py*中补足dataset_map。

```
from datasets import cifar10
datasets_map = {
    'cifar10': cifar10,
}
```

3. 执行任务脚本时，指定参数dataset_name=cifar10和train_files=cifar10_train.tfrecord，即可使用cifar10数据进行模型训练。

> **说明** 如果您需要读取其他的格式数据，需自行实现dataset pipeline构建逻辑（参考*utils/dataset_utils.py*）。

超参文件说明

PAI-FastNN支持以下类型的超参：

- 数据集参数：确定训练集的基本属性的参数，例如训练集存储路径dataset_dir。
- 数据预处理参数：数据预处理函数及dataset pipeline相关参数。
- 模型参数：模型训练基本参数，包括model_name、batch_size等。
- 学习率参数：学习率及其相关调优参数。
- 优化器参数：优化器及其相关参数。
- 日志参数：关于输出日志的参数。
- 性能调优参数：混合精度等其他调优参数。

超参文件的格式如下。

```
enable_paisora=True
batch_size=128
use_fp16=True
dataset_name=flowers
dataset_dir=oss://pai-online-beijing.oss-cn-beijing-internal.aliyuncs.com/astnn-data/flowers/
model_name=inception_resnet_v2
optimizer=sgd
num_classes=5
job_name=worker
```

- 数据集参数

名称	类型	描述
dataset_name	string	指定输入数据解析文件的名称。取值包括：mock、cifar10、mnist、flowers，取值说明请参见*images/datasets*目录下所有的数据解析文件。默认使用模拟数据mock。
dataset_dir	string	指定输入数据集的绝对路径，默认为None。
num_sample_per_epoch	integer	指定数据集总样本数，一般用来配合学习率的衰减。
num_classes	integer	指定样本分类数，默认为100。
train_files	string	指定所有训练数据的文件名，文件间分隔符为逗号，例如0.tfrecord,1.tfrecord。

- 数据预处理参数

名称	类型	描述
preprocessing_name	string	和model_name共同指定数据预处理的方法名，取值范围请参见*images/preprocessing*目录下的*preprocessing_factory*文件。默认设置为None，表示不进行数据预处理。
shuffle_buffer_size	integer	在生成数据流水线时，以样本为粒度进行shuffle的缓存池大小，默认为1024。
num_parallel_batches	integer	与batch_size乘积为map_and_batch的并行线程数，协助指定解析样本的并行粒度，默认为8。
prefetch_buffer_size	integer	指定数据流水线预取数据的批数，默认为32。
num_preprocessing_threads	integer	指定数据流水线进行并行数据预取的线程数，默认为16。
datasets_use_caching	bool	是否打开以内存为开销，进行输入数据的压缩缓存。默认为False，表示不打开。

- 模型参数

名称	类型	描述
task_type	string	任务类型，取值包括： ○ pretrain：模型预训练，默认。 ○ finetune：模型调优
model_name	string	指定进行训练的模型，取值包括*images/models*下的所有模型。您可以参考*images/models/model_factory*文件中所有定义的模型设置model_name，默认为inception_resnet_v2。
num_epochs	integer	训练集训练轮数，默认为100。
weight_decay	float	模型训练时权重的衰减系数，默认为0.00004。
max_gradient_norm	float	是否根据全局归一化值进行梯度裁剪。默认为None，表示不进行梯度裁剪。
batch_size	integer	单卡一次迭代处理的数据量，默认为32。
model_dir	string	重载checkpoint的路径。默认为None，表示不进行模型调优。
ckpt_file_name	string	重载checkpoint的文件名，默认为None。

- 学习率参数

名称	类型	描述
warmup_steps	integer	逆衰减学习率的迭代数，默认为0。
warmup_scheme	string	学习率逆衰减的方式。取值为t2t（Tensor2Tensor），表示初始化为指定学习率的1/100，然后exponentiate逆衰减到指定学习率为止。
decay_scheme	string	学习率衰减的方式。可选值： ○ luong234：在2/3的总迭代数之后，开始4次衰减，衰减系数为1/2。 ○ luong5：在1/2的总迭代数之后，开始5次衰减，衰减系数为1/2。 ○ luong10：在1/2的总迭代数之后，开始10次衰减，衰减系数为1/2。
learning_rate_decay_factor	float	指定学习率衰减系数，默认为0.94。
learning_rate_decay_type	string	指定学习率衰减类型，可选值：fixed、exponential（默认）和polynomial。

名称	类型	描述
learning_rate	float	指定学习率初始值，默认为0.01。
end_learning_rate	float	指定衰减时学习率值的下限，默认为0.0001。

- 优化器参数

名称	类型	描述
optimizer	string	指定优化器名称。可选值：adadelta、adagrad、adam、ftrl、momentum、sgd、rmsprop、adamweightdecay， 默认为rmsprop。
adadelta_rho	float	adadelta的衰减系数，默认为0.95。
adagrad_initial_accumulator_value	float	AdaGrad积累器的起始值，默认为0.1。是Adagrada优化器专用参数。
adam_beta1	float	一次动量预测的指数衰减率，默认为0.9。是Adam优化器专用参数。
adam_beta2	float	二次动量预测的指数衰减率，默认为0.999。是Adam优化器专用参数。
opt_epsilon	float	优化器偏置值，默认为1.0。是Adam优化器专用参数。
ftrl_learning_rate_power	float	学习率参数的幂参数，默认为-0.5。是Ftrl优化器专用参数。
ftrl_initial_accumulator_value	float	FTRL积累器的起始，默认为0.1，是Ftrl优化器专用参数。
ftrl_l1	float	FTRL l1正则项，默认为0.0，是Ftrl优化器专用参数。
ftrl_l2	float	FTRL l2正则项，默认为0.0，是Ftrl优化器专用参数。
momentum	float	MomentumOptimizer的动量参数，默认为0.9，是Momentum优化器专用参数。
rmsprop_momentum	float	RMSPropOptimizer的动量参数，默认为0.9。
rmsprop_decay	float	RMSProp的衰减系数，默认为0.9。

- 日志参数

名称	类型	描述
stop_at_step	integer	训练总迭代数，默认为100。
log_loss_every_n_iters	integer	打印loss信息的迭代频率，默认为10。
profile_every_n_iters	integer	打印timeline的迭代频率，默认为0。

名称	类型	描述
profile_at_task	integer	输出timeline的机器对应索引，默认为0，对应chief worker。
log_device_placement	bool	是否输出device placement信息，默认为False。
print_model_statistics	bool	是否输出可训练变量信息，默认为false。
hooks	string	训练hooks，默认为StopAtStepHook,ProfilerHook,LoggingTensorHook,CheckpointSaverHook。

- 性能调优参数

名称	类型	描述
use_fp16	bool	是否进行半精度训练，默认为True。
loss_scale	float	训练中loss值scale的系数，默认为1.0。
enable_paisoar	bool	是否使用paisoar框架，默认True。
protocol	string	默认grpc.rdma集群可以使用grpc+verbs，提升数据存取效率。

开发主文件

如果已有模型无法满足您的需求，您可以通过继承dataset、models和preprocessing接口进一步开发。在此之前需要了解FastNN库的基本流程（以images为例，代码入口文件为*train_image_classifiers.py*），整体代码架构流程如下。

```
# 根据model_name初始化models中对应模型得到network_fn，并可能返回输入参数train_image_size。
    network_fn = nets_factory.get_network_fn(
            FLAGS.model_name,
            num_classes=FLAGS.num_classes,
            weight_decay=FLAGS.weight_decay,
            is_training=(FLAGS.task_type in ['pretrain', 'finetune']))
# 根据model_name或preprocessing_name初始化相应数据预处理函数得到preprocess_fn。
    preprocessing_fn = preprocessing_factory.get_preprocessing(
                FLAGS.model_name or FLAGS.preprocessing_name,
                is_training=(FLAGS.task_type in ['pretrain', 'finetune']))
# 根据dataset_name，选择正确的tfrecord格式，同步调用preprocess_fn解析数据集得到数据dataset_iterat
or。
    dataset_iterator = dataset_factory.get_dataset_iterator(FLAGS.dataset_name,
                                                            train_image_size,
                                                            preprocessing_fn,
                                                            data_sources,
# 调用network_fn、dataset_iterator，定义计算loss的函数loss_fn。
    def loss_fn():
      with tf.device('/cpu:0'):
          images, labels = dataset_iterator.get_next()
        logits, end_points = network_fn(images)
        loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=tf.cast(logits,
tf.float32), weights=1.0)
        if 'AuxLogits' in end_points:
          loss += tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=tf.cast(end_
points['AuxLogits'], tf.float32), weights=0.4)
        return loss
# 调用PAI-Soar API封装loss_fn、tf原生optimizer。
    opt = paisoar.ReplicatedVarsOptimizer(optimizer, clip_norm=FLAGS.max_gradient_norm)
    loss = optimizer.compute_loss(loss_fn, loss_scale=FLAGS.loss_scale)
# 根据opt和loss形式化定义training tensor。
    train_op = opt.minimize(loss, global_step=global_step)
```

5.2.3. Whale分布式深度学习框架

5.2.3.1. 快速开始

Whale可以帮助您便捷地进行分布式并行训练，支持各种并行策略混合训练，同时提供多种通信优化方法。本文介绍在Whale中实现分布式并行策略的详细步骤，包括初始化、资源划分、模型划分及硬件资源到逻辑资源的映射，并提供了完整示例。同时，介绍如何开启Whale的通信优化功能。

背景信息

Whale是灵活、易用、高效、统一的分布式训练框架，提供简单且易用的接口以实现数据并行、模型并行、流水并行、算子拆分及多种并行方式组合的混合并行。Whale基于TensorFlow开发，完全兼容TensorFlow接口，您只需要在已有的TensorFlow模型上增加几行分布式并行策略描述代码，即可进行分布式混合并行训练。

版本配套关系

Whale支持的PAI-TensorFlow及Python版本如下：

- PAI-TensorFlow版本：PAI-TensorFlow 1.12、PAI-TensorFlow 1.15

- Python版本：Python 2.7、Python 3.6

使用Whale添加分布式策略

1. 初始化。

   使用 `init()` 接口进行Whale初始化，详情请参见whale.init。示例如下。

   ```
   import whale as wh
   wh.init()
   ```

2. 划分资源（Cluster）。

   Whale中通过Cluster抽象和定义资源，完成硬件资源到逻辑资源的映射。同时，Whale通过不同策略将资源进行分组，详情请参见whale.cluster。

   ```
   cluster = wh.cluster()
   ```

3. 划分模型（Scope）。

   Whale通过Scope指定分布式训练方式，且支持多种Scope组合成各种混合并行方式，详情请参见whale.scopes。Whale支持的Scopes类型包括：

   - replica：数据并行。
   - stage：模型并行。
   - pipeline：流水并行。
   - split：算子拆分
   - 以上Scope的组合。

   以下列举一些分布式训练在Whale中的实现方式，以供参考：

   - 数据并行

     ```
     with wh.replica():
         model_fn()
     ```

   - 模型并行
     假设模型有20个Layers，前10个Layers划分为一个Stage，后10个Layers划分为另一个Stage，示例如下。

     ```
     with wh.stage():
         # layers 1~10 of model
         model_fn1()
     with wh.stage():
         # layers 11~20 of model
         model_fn2()
     ```

   - 数据并行、模型并行及流水并行
     在模型并行基础上进行数据并行和流水并行，您只需要组合使用replica和pipeline的Scope，示例如下。

```
with wh.replica():
    with wh.pipeline(num_micro_batch=5):
        with wh.stage():
            # layers 1~10 of model
            model_fn1()
        with wh.stage():
            # layers 11~20 of model
            model_fn2()
```

- 数据并行和算子拆分
  假设对一个模型前10个Layers进行数据并行，对后面的Layers进行算子拆分，示例如下。

```
with wh.replica():
    # layers 1~10 of model
    model_fn1()
with wh.split():
    # layers 11~12 of model
    model_fn2()
```

完整示例

1. 对模型进行分布式改写。

   使用Whale将一个单机模型改造为数据并行，此处仅展示主体代码改写过程，示例如下。完整代码请参见simple_data_parallel.py，更多分布式并行策略的使用方法请参见Whale分布式范式剖析。

```
import whale as wh
def main(_):
  wh.init()  # init whale
  with wh.cluster(): # cluster
    with wh.replica(): # data parallelism
       your_model_definition # construct your model here.
```

   主体代码的改写过程主要包括三个调用：

   i. Whale初始化：`wh.init()`

   ii. Cluster定义及硬件资源到逻辑资源的映射：`with wh.cluster()`

   iii. 将并行方式指定为数据并行：`with wh.replica()`

   将单机模型改造为数据并行方式进行训练后，Whale会自动感知任务资源，并根据数据并行方式划分资源。

2. 切分训练数据。

   上一步中读取的Mock数据。如果使用真实数据进行分布式训练，则需要对训练数据进行切分，从而确保每个Worker获取不同的数据。
   Whale会自动对训练数据进行切分，训练数据切分策略及配置方法请参见训练数据分片。

3. 提交作业。

   您可以通过以下任意一种方式提交作业：

   - PAI-Studio
     支持算法组件TensorFlow 1120，提交作业的方法请参见PAI-TF调用方式。
   - PAI-DLC

- 如果登录服务器，手动执行训练脚本，则执行如下命令。

  ```
  python simple_data_parallel.py
  ```

  其中 `simple_data_parallel.py` 表示分布式训练任务的代码文件，需要根据实际情况替换。
- 如果通过Arena提交任务，请参见使用Arena管理任务（推荐）。
- 如果是使用PAI-DLC Dashboard提交任务，请参见使用PAI-DLC Dashboard管理任务。

Whale其他功能

Whale支持多种通信优化，您可以根据需求开启各种优化功能，配置方法请参见通信参数。

5.2.3.2. 经典案例

5.2.3.2.1. 大规模分类的分布式训练（算子拆分）

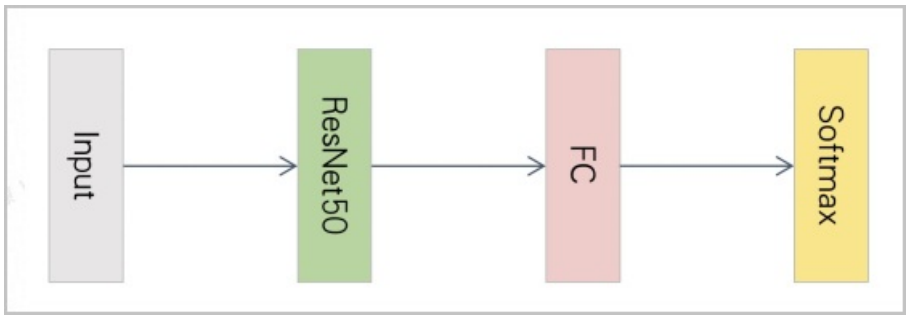
本文针对大规模分类场景存在的问题，介绍Whale的并行化设计和方案。通过结合算子拆分和数据并行，优化通信拓扑结构，以解决大规模分类任务无法单机训练或分布式训练性能较差的问题。在Whale中，您可以通过模型划分、资源划分及映射三个步骤，实现大规模分类任务的Fusion模式。

背景信息

分类问题是机器学习、数据挖掘的基础问题。针对大数据场景，分类数据通常达到数百万、千万或亿级别。随着分类规模增长，模型参数和模型规模本身呈超线性增长，超大规模分类是业务上普遍存在的问题，例如人脸识别分类数达到亿级别。

问题

以ResNet50分类任务为例，其模型结构如下图所示。



ResNet50分类的原始模型代码如下。

```
features = ResNet50(inputs)
logits = FC(features)
predictions = Softmax(logits)
```

本文选择可以使用数据并行的10 W分类数，以便与算子拆分进行性能对比。分类数为10 W时，ResNet50部分的模型权重大小89.6 MB，全连接层FC（Fully Connected Layer）部分的模型权重大小为781.6 MB，是前者的8.7倍。当采用数据并行进行分布式训练时，在后向阶段（Back Propagation过程），FC部分计算得到梯度后，立刻通过AllReduce进行梯度同步，同时ResNet50部分的梯度会继续进行后向计算。然而由于FC部分同步的梯度过大，当ResNet50部分的梯度计算完成时，FC部分的梯度通常还在通信过程中。因此，FC部分的梯度同步无法良好地与ResNet50的后向计算Overlap，导致整轮迭代过程中的通信占比十分高，性能低下。

当分类数更大时，分布式计算模式对显存有更高的要求。以Nvidia 16 GB V100为例，假设隐层数为2048，仅能支持存储190 W的FC权重。再加上中间输出、计算得到的梯度及Optimizer的中间变量，16 GB的V100卡能够支持的分类数约为20 W（未考虑卷积层的显存占用）。

如上所述，即使将分类层（FC部分）单独放到一张GPU卡上也无法支持百万规模的分类任务，现有数据并行方法无法支持超大规模分类。针对超大规模分类任务，必须对分类层进行OP内的算子拆分，将单个OP拆分为多分片，在不同设备中进行分片计算。

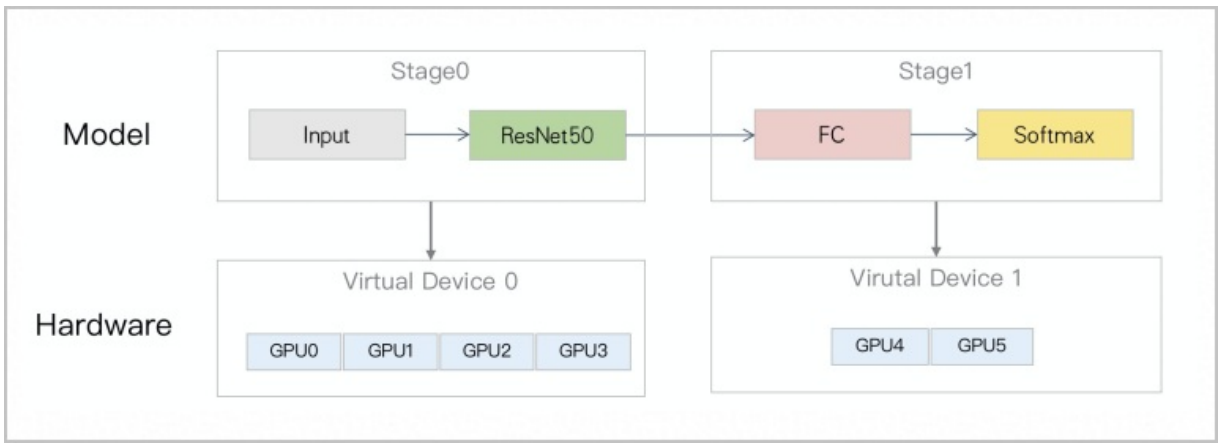
在此场景中，需要考虑分类规模和数据规模，超大的分类数必然伴随着大规模的训练数据。算子拆分可以解决模型本身很大的问题，例如上述的大规模分类层或BertLarge模型。对于卷积层（基于TF-Slim库实现的Resnet50模型Backbone，是通过卷积函数实现的），其模型本身较小，需要通过数据并行解决数据规模大的问题。因此，目前需要解决的问题是如何结合算子拆分和数据并行，进行分布式训练，同时提高通信性能。

实现方案

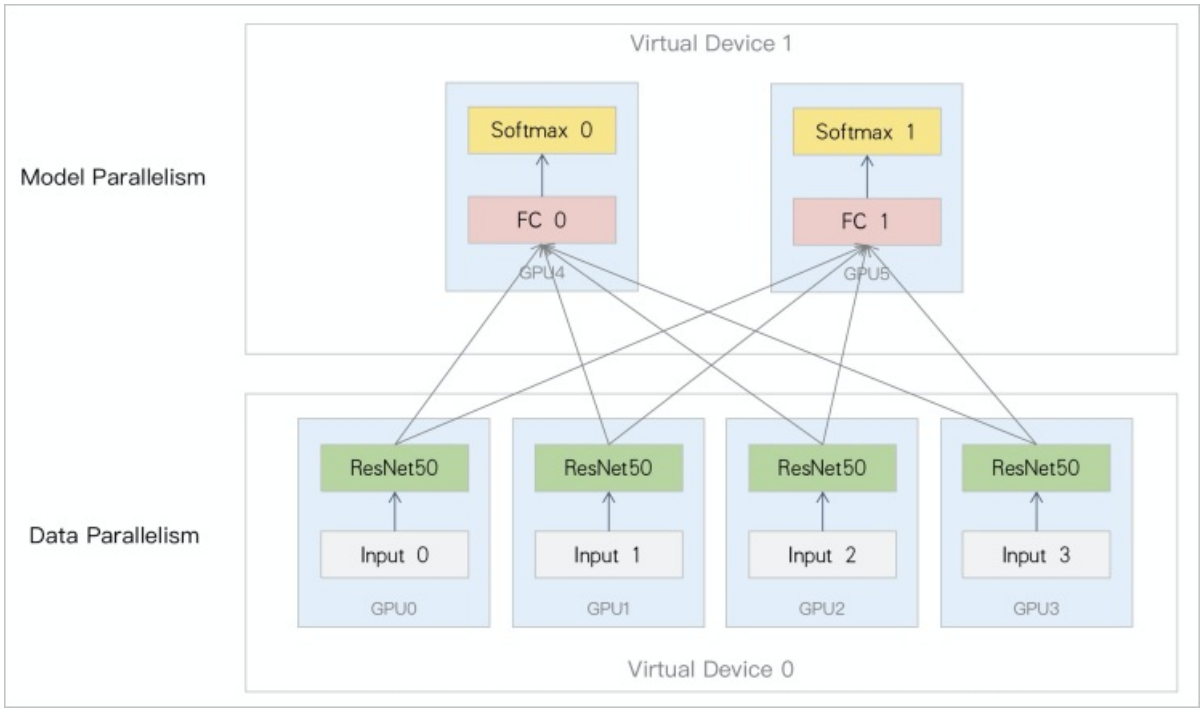
针对模型不同部分，采用不同的并行化策略加速分布式训练，以提高分布式性能：

- FC部分
  权重大，使得梯度同步开销大。为避免数据并行同步过大引起性能问题，也为解决单卡无法存放超大规模分类的问题，先将FC部分的Weight权重按照列进行拆分，再使用多卡存储分片。因此，针对FC和Softmax计算，每张卡只负责不同部分的计算。
- ResNet50部分
  权重小，计算耗时长，该部分梯度同步开销小。对ResNet50部分进行数据并行处理，每张卡读取不同的数据分片，以加速数据处理速度。

综上所述，针对ResNet50的大规模分类任务，将模型分为两个Stage。将Stage 0的ResNet50部分通过数据并行策略复制N份至不同的卡中，进行数据并行。将Stage 1的FC和Softmax部分通过算子拆分策略分片至不同卡中，进行模型并行。假设共六张卡（GPU0、GPU1、GPU2、GPU3、GPU4及GPU5），将GPU分为两组，分别进行数据并行和算子拆分，如下图所示。

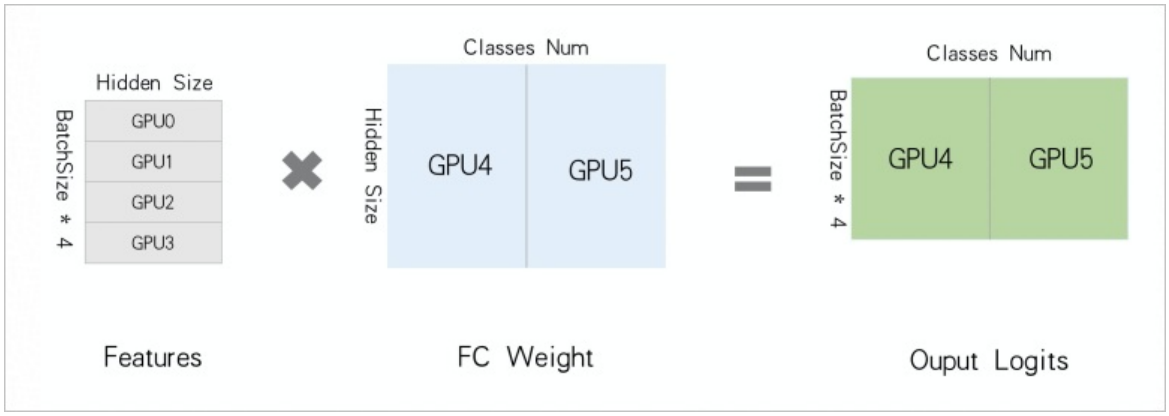


上图中，将Stage 0部分分别拷贝至[GPU0, GPU1, GPU2, GPU3]四张卡中进行数据并行，每张卡读取不同的数据分片进行训练。将Stage 1部分的计算拆分为两片放至[GPU4, GPU5]中进行算子拆分，并行化的计算图如下所示。



上图中，包括以下处理过程：

1. [GPU0, GPU1, GPU2, GPU3]完成ResNet50的前向过程后，[GPU4, GPU5]收到[GPU0, GPU1, GPU2, GPU3]输出的Feature特性。
2. [GPU4, GPU5]先将Feature特性按照行进行Concat，再进入FC部分。FC详细的计算图如下所示。



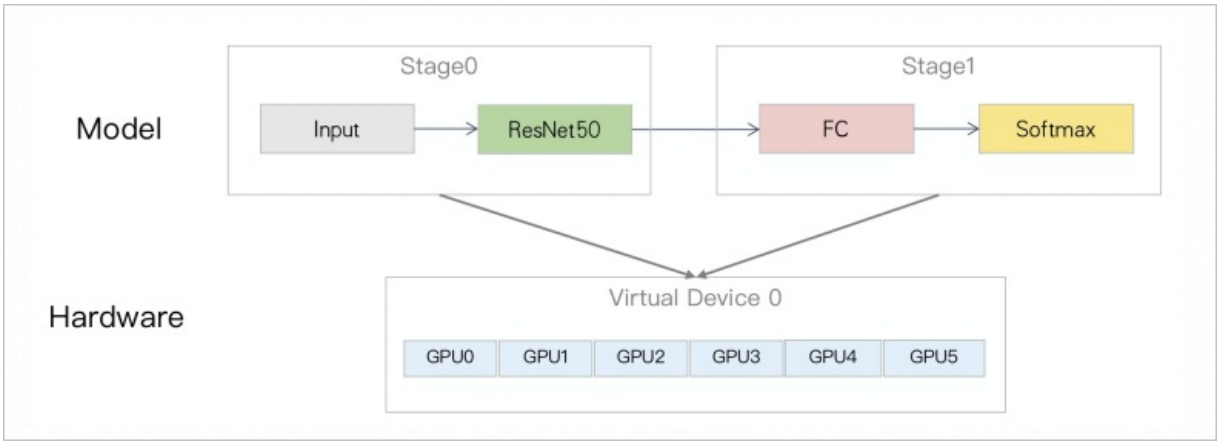
3. [GPU4, GPU5]分别得到输出的Logits后，分别计算Softmax部分（该过程需要进行跨卡通信，以获得全局最大Logits信息和每行的Sum值）。

通过上述方案虽然可以完成大规模分类任务，但是运行过程中任然存在以下问题：

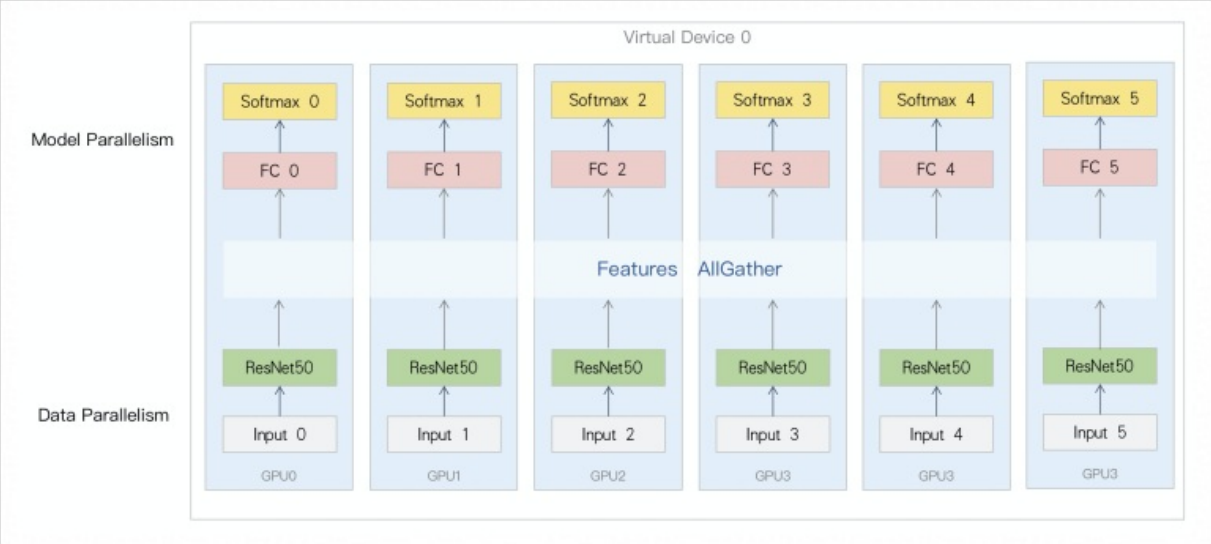
- 数据并行部分需要点对点发送Feature至算子拆分部分，存在热点问题。
- 迭代过程中存在GPU空闲。当数据并行计算时，算子拆分GPU空闲等待。当算子拆分部分计算时，数据并行GPU空闲等待。

该问题严重影响性能，尤其是热点问题会导致分布式规模无法良好的扩展。因此，Whale对此进行了进一步优化，即采用Fusion模式计算。

Whale将Stage 0的ResNet50部分映射至所有卡中进行数据并行，将Stage 1的FC和Softmax部分映射至所有卡中进行分片计算。模型与硬件资源的映射如下图所示。



Stage 0的输出从原来的点对点发送优化为去中心的AllGather通信，避免了热点问题。同时，每张卡中Stage 0计算结束，进行AllGather通信后，立刻计算Stage1部分，不会出现GPU空闲等待。具体的计算图如下所示。



Whale实现多分类任务

在Whale中实现大规模分类任务Fusion模式需要三个步骤（标准的Whale分布式编程范式）：模型划分、资源划分及映射。

1. 模型划分。

   将模型各部分按照不同并行化需求划分为不同部分，即Scopes划分，详情请参见whale.scopes。上述的实现方案中将模型划分为两部分：

   - Stage 0：包括Resnet50部分。
   - Stage 1：包括FC和Softmax部分。

   针对ResNet50模型的原始代码，在Whale中通过两行Scopes代码（`whale.replica`和`whale.split`）即可实现模型划分，示例代码如下。

```
with whale.replica():    # Stage 0
    features = ResNet50(inputs)
with whale.split():      # Stage 1
    logits = FC(features)
    predictions = Softmax(logits)
```

2. 资源划分。

   对硬件资源进行分组，将其划分为一个Virtual Devices（包含 [GPU0, GPU1, GPU2, GPU3, GPU4, GPU5]），即Cluster划分，详情请参见whale.cluster。
   Whale提供的Cluster工具可以对申请的Worker进行划分，`whale.cluster` 默认即可将所有Device虚拟为一个Virtual Device，示例代码如下。

   ```
   cluster = whale.cluster()
   ```

3. 映射。

   将模型各部分分别映射至Virtual Device：

   - 对Resnet50部分采用数据并行，放至Virtual Device，即[GPU0, GPU1, GPU2, GPU3, GPU4, GPU5]每张卡都有一个Resnet50的模型副本。
   - FC和Softmax部分采用算子拆分，放至Virtual Device，即[GPU0, GPU1, GPU2, GPU3, GPU4, GPU5]每张卡先分别计算FC内的Matmul的一个分片，再分片进行Softmax计算。

   对于映射部分，在Whale中只需要对已生成的Cluster执行 `with` 语法，即可轻松完成模型到硬件资源的映射。通过Whale完成并行化的模型核心代码如下（包含算子拆分的大规模分类任务可执行代码请参见large_scale_classification.py）。

   ```
   cluster = whale.cluster() # 资源划分。
   with cluster:             # 映射。
       with whale.replica(): # Stage 0。
           features = ResNet50(inputs)
       with whale.split():   # Stage 1。
           logits = FC(features)
           predictions = Softmax(logits)
   ```

性能

以Whale的数据并行性能数据为Baseline进行对比，测试环境如下。

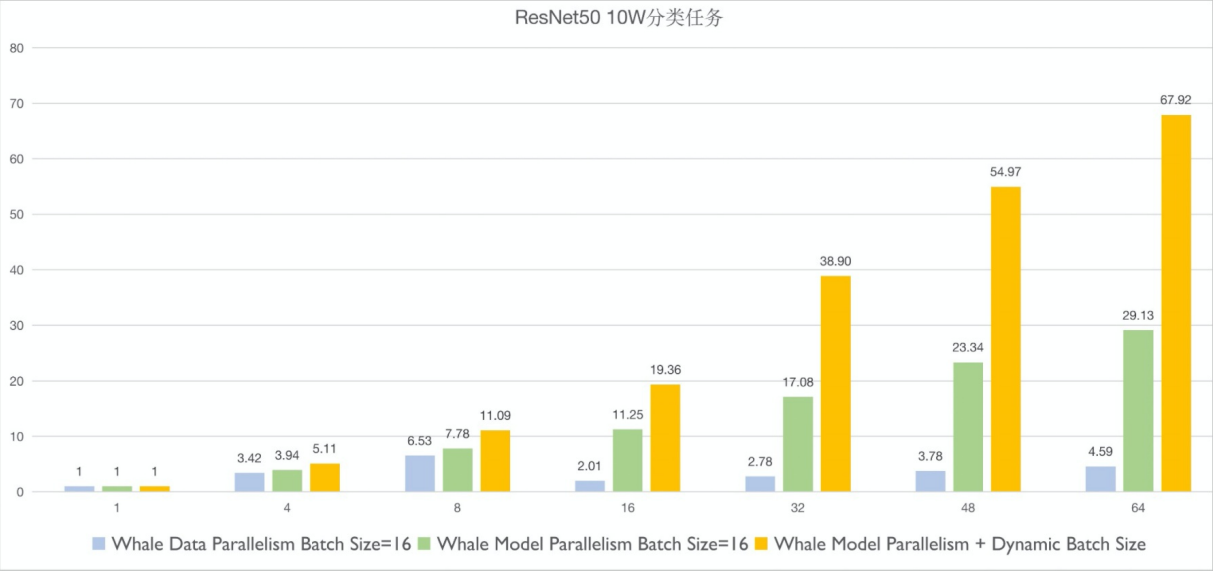
测试环境项	描述
GPU型号	ecs.gn6v-c10g1.20xlarge（V100 * 8）
网络	VPC-35 GB
NCCL_MAX_NRINGS	NVIDIA官方参数，测试时取值为4。
NCCL_MIN_NRINGS	NVIDIA官方参数，测试时取值为4。

在算子拆分的性能测试过程中，采用10 W分类的ResNet50模型。将FC部分通过算子拆分放至分布式服务器上进行分片计算。另外，卷积部分和分片的FC部分采用Fusion模式复用Worker。由于进行了算子拆分，FC部分的显存会分摊至其他卡，因此在算子拆分场景下数据并行的最大BatchSize可以变得更大。本文中的算子拆分对比测试场景分别为：

- 数据并行：Whale Data Parallelism Batch Size=16

- 固定Batch Size的算子拆分：Whale Model Parallelism Batch Size=16
- 动态Batch Size的算子拆分，即Whale Model Parallelism + Dynamic Batch Size

对比结果如下图所示。



说明 横坐标表示GPU卡数，纵坐标表示加速比。

从上图中，可以得到以下结论：

- 对于单机多卡或跨机任务，算子拆分的性能远优于数据并行任务。其原因主要是减少了FC部分的梯度同步通信量。
- 根据GPU卡数动态调整Batch Size的算子拆分性能，相比数据并行的最大Batch Size性能更优。在64卡时，Batch Size可以调整到160，是数据并行的10倍。
- 性能加速结果呈超线性加速。其原因主要因为动态调大Batch Size，使Apply阶段（参数更新）的计算量和通信量均减少了90%。

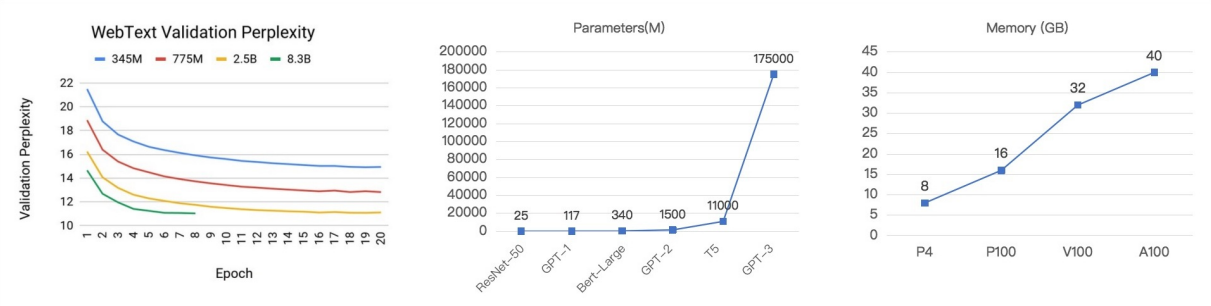
5.2.3.2.2. BertLarge分布式训练（流水并行）

本文针对BertLarge分布式并行训练所存在的问题，介绍Whale的并行化设计和方案。通过为模型并行和数据并行，搭配流水并行辅助并行策略，优化通信拓扑结构，以解决BertLarge分布式训练性能较差的问题。在Whale中，您可以通过模型划分、资源划分及映射三个步骤，实现大规模数据及模型的分布式训练。

背景信息

受益于深度神经网络的发展，NLP和CV领域的模型效果得到大幅度提升。同时，模型参数量也大幅度增加。以Imagenet分类任务为例，优胜算法从2014年的GoogleNet到2018年的Squeeze-and-Excitation Networks，参数量增长约36倍（从4百万增长至1.458亿参数）。

NLP领域很多模型相较于图像领域的模型规模更大。如下图所示，近几年NLP领域的训练数据和模型规模都非常大，例如GPT-3的参数达到了1750亿。由于模型参数的增长速度远超GPU显存的增长速度（从P4的8 GB到A100的40 GB），导致训练支持的Batch Size变小，进而使训练过程中的通信占比变高，影响模型分布式扩展。

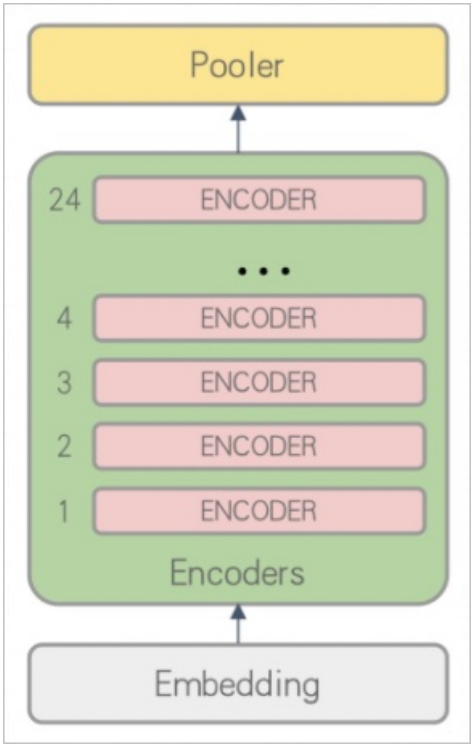


以Bert Large模型为例，它在很多NLP场景里取得了非常好的结果。BERT的Idea与ELMo和GPT都非常接近，但是BERT可以在很大的非监督语料里进行预训练，速度远超ELMo，效果优于GPT，在真实业务场景中广泛应用。

Bert Large只有3.4亿参数规模，与T5和GPT-3相比，模型参数非常小。但是在Nvidia V100 16G GPU上，训练Batch Size仅达到2~8（具体值和Embedding大小、Sequence Length等有关）。模型参数规模大导致通信梯度大，Batch Size小又导致通信占比高，因此使用传统数据并行进行训练的加速效果极差。

问题描述

以Bert Large模型为例，其模型结构如下图所示。



Bert Large的原始模型代码如下所示。

```
embedding_output = embedding(inputs)
encoder_output = encoder_layer_1_24(embedding_output)
pooler_output = pooler(encoder_output)
```

在数据并行场景，以max_seq_len等于384为例，使用V100单卡（16 GB显存），模型最大Batch Size仅达到6。每轮迭代同步1.245 GB的梯度，在50 GB网络环境下，通信耗时为 `2*1.245 GB/50 GB=398.4 ms` （读取训练数据也需要消耗网络带宽，因此实际通信耗时大于该值）。因此，大规模训练主要存在两个问题：

- Batch size过小，导致模型波动较大，从而使得收敛效果差。
- 梯度通信量大，小Batch Size加重训练通信占比，导致分布式扩展效果差。

为解决上述问题，Whale提出模型并行、流水并行及数据并行的混合并行策略。同时支持高性能通信Backend及调度优化，从而为您提供高性能、简洁且易用的分布式训练框架。模型并行、流水并行及数据并行的混合并行策略如下：

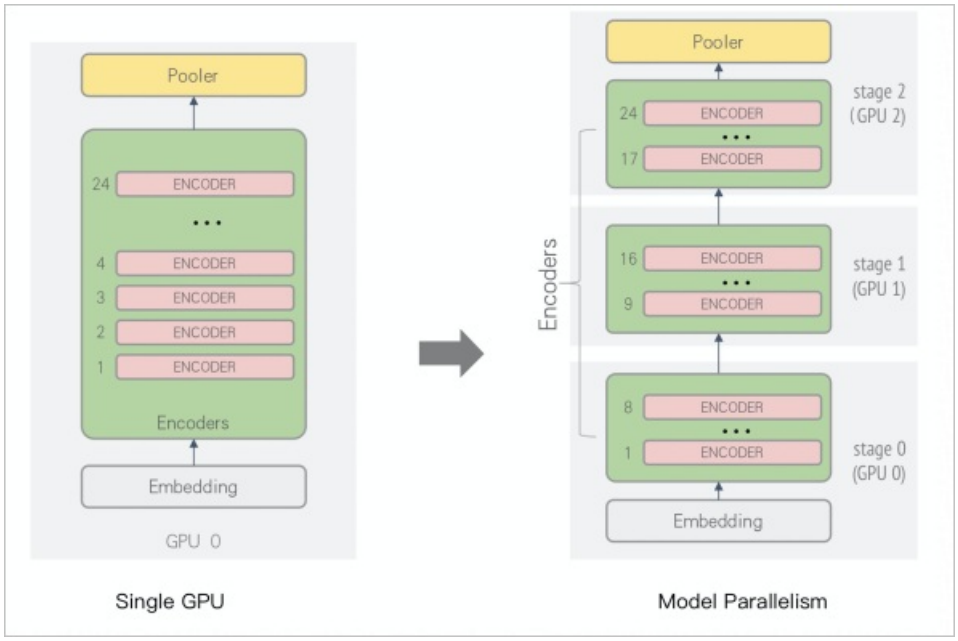
- 模型并行
  - 将模型按照Layer粒度切分为不同的Stage，并将其分配至不同的GPU中执行，以降低每张GPU卡中的模型显存占用，从而提高Batch Size。
  - Stage之间使用Activation通信代替梯度同步，大幅度降低卡间通信量。例如在Bert Llarge模型中，每轮迭代仅需要传输27 MB的Activation数据。
  - 增大Batch Size，使训练更加稳定，从而收敛效果更好。
- 流水并行
  如果仅采用模型并行，则GPU卡间任务执行有依赖，因此同一时间只有一个GPU执行，其他GPU空闲，导致GPU利用率低。Whale流水并行中实现一个Mini-Batch对应一个Micro-Batch，同时可以训练多个Micro-Batch，不同GPU中可以同时执行流水的不同Stage，从而提高GPU利用率。
- 数据并行
  受限于模型Layer总数和流水并行效率，模型并行和流水并行的混合并行策略不能无限地进行分布式扩展。在超大训练数据规模场景，还需要结合数据并行进行分布式扩展。

实现方案

通过模型并行、流水并行及数据并行的混合并行策略，优化Bert Large模型的分布式性能。详细方案如下：

1. 模型并行
   Bert Large模型的Batch Size通常仅达到2~8（具体值与Embedding大小、Sequence Length等有关），导致模型波动大，进而使得收敛效果差。此时，可以将模型以Layer为单位拆分为多份，并将其放至不同GPU卡中进行分布式训练，即模型并行。

无论在何种带宽、硬件拓扑下，模型分片可能受Load Balance（包括显存均衡度、算力均衡度）影响。BertLarge模型的每一层Activation、显存及Flops计算几乎都一致，因此从均衡各个模型部分显存占用、算力需求的角度出发，将BertLarge中的Encoder Layer 1~8层、Encoder Layer 9~16层，Encoder Layer 17~24层分别放至不同的GPU卡中进行训练，并行化后的计算图如下图所示。

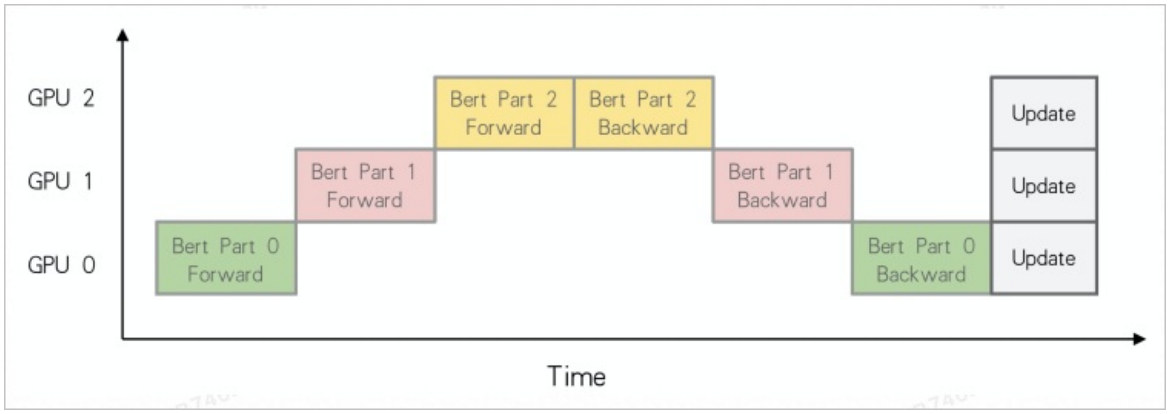


如上图所示，将Embedding Layer和Encoder Layer 1~8层放在GPU 0中进行运算，将Encoder Layer 9~16层放在GPU 1中进行运算，Encoder Layer 17~24层和Pooler层放在GPU 2中进行运算。该处理方式具有以下优势：

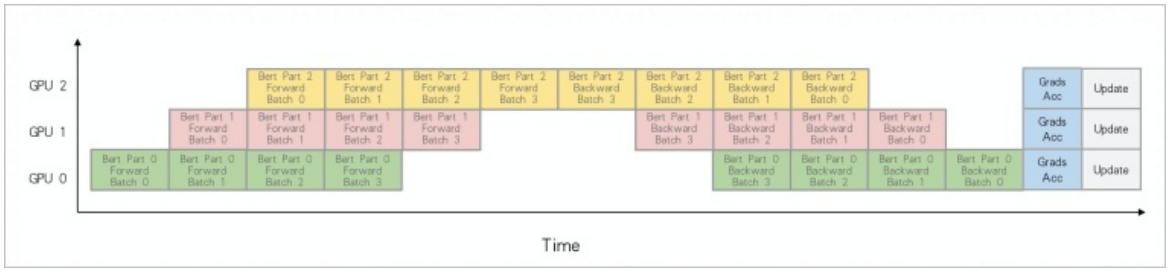
- 每张GPU卡的模型大幅度减小，因此可以增大Batch Size，以提升收敛加速。
- 只需要在GPU之间通信Activation（每轮迭代约27 MB的Activation数据），省掉了梯度通信过程。
- 对于模型过大导致单卡显存无法存放的情况，通过Layer拆分的模型并行方式，使模型训练成为可能。

2. 流水并行

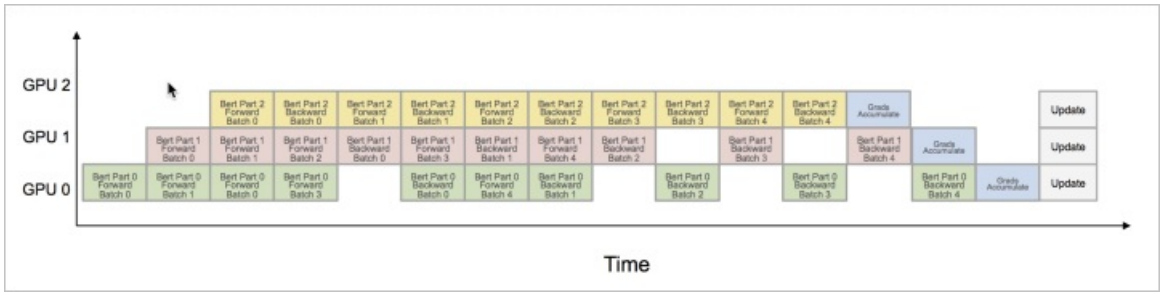
仅采用模型并行进行分布式训练，其中一张GPU卡计算时，其他GPU卡空闲，导致GPU资源利用率低。如下图所示。



上图中，在时间维度，每一个时间点上只有一张GPU卡在执行Forward或Backward，其他GPU卡都处于空闲等待状态。Whale设计了流水并行，以提高模型并行场景GPU卡的利用率，在每张卡训练完一个Micro-Batch数据后，将Activation传给下一个GPU，然后立刻训练下一个Micro-Batch数据。按照该方法，在多个GPU之间形成流水，执行逻辑如下图所示。

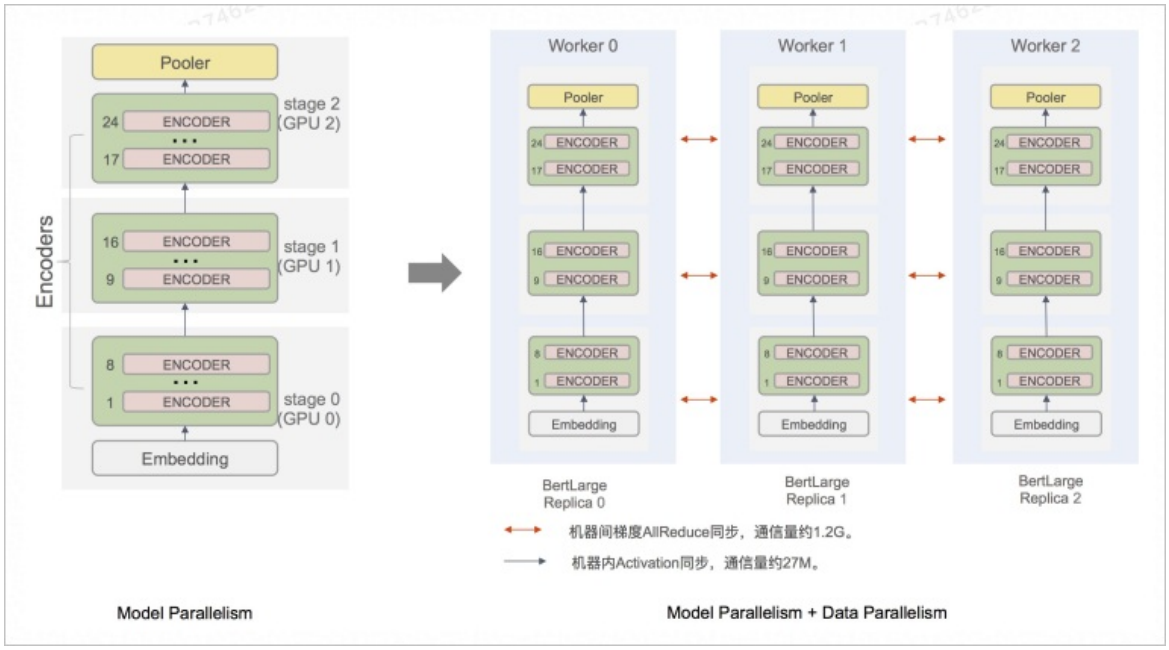


上图的示例中Micro-Batch数量为4，流水并行优化后的时间轴显示，同一时间点上多张GPU卡可以并行计算。当4个Micro-Batch结束后，每张GPU卡将每个Micro-Batch的梯度进行本地累计（即上图中的Grads Acc）之后，进行Update权重操作。与模型并行相比，提升了GPU利用率。然而，上图中依然存在大量空白，即某些时间点上依然存在某些GPU处于空闲状态。针对该情况，可以增大流水并行的Micro-Batch数量，以降低空闲时间。此外，Whale采用Backward-Prefered（调度每个Micro-Batch间的前后向计算顺序，使GPU卡尽量一直处于计算状态）调度优化策略提升流水并行性能，从而降低GPU空闲时间，如下图所示（Micro-Batch配置为5）。



3. 混合并行

因为模型并行不可能将模型拆分为无限份，且切分数量过大会影响性能，所以仅采用模型并行和流水并行依然不能满足性能要求。针对业务场景产生的海量训练数据，需要提高大量计算资源并行处理，因此还需要采用数据并行解决大规模数据问题。在流水并行的基础上增加数据并行后的计算图如下所示。

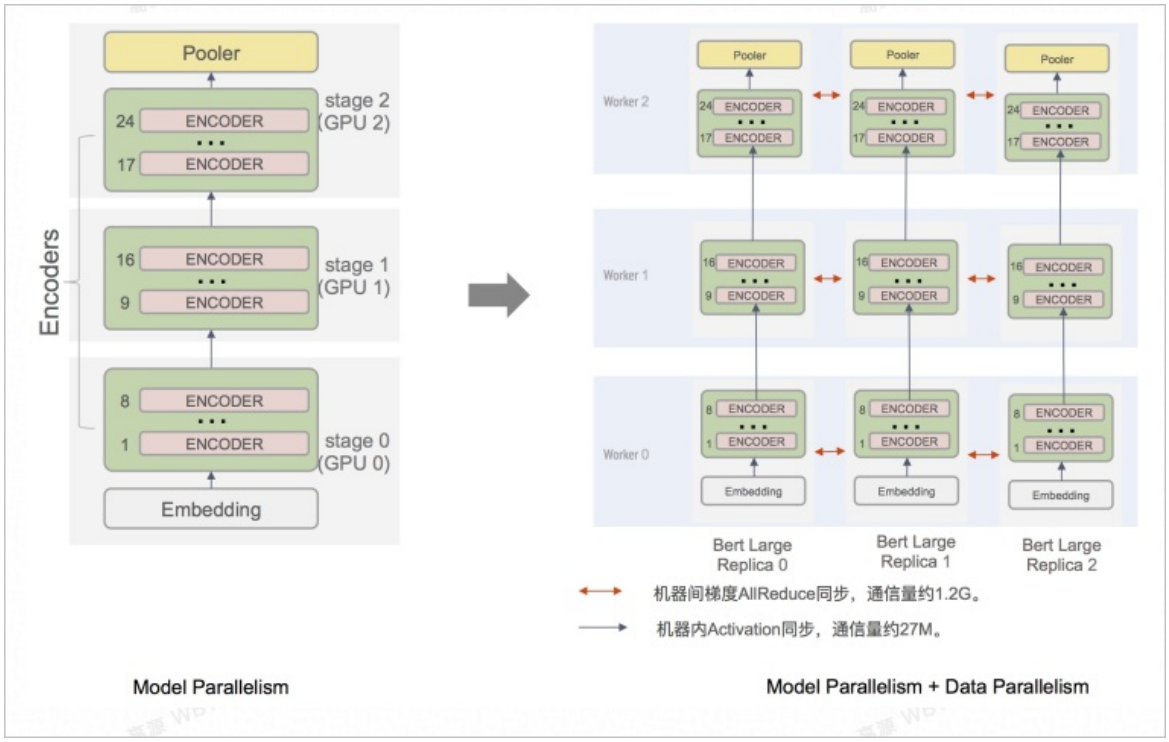


上图中以3个Worker为例，每个Worker拥有3个GPU。Whale将每个Worke内的模型部分分片放至本Worker内的多张GPU卡中进行流水并行计算。每个Worker完成Micro-Batch数量个Batch Size的流水训练后，先累计本地的梯度，再在Worker之间进行梯度AllReduce同步。

为了更高效利用服务器内GPU间的NVLink（如果没有，则通过PCI-e）带宽，降低梯度同步的开销，Whale进行了如下优化：

- 将模型并行拆分放至不同服务器中进行计算。
- 将数据并行部分的梯度同步（通信量约为1.2 GB）放至服务器内进行。
- 将Layer间切分产生的Activation通信（约27 MB）通过TCP网络通信。

该方式可以提高通信效率，从而降低每轮迭代时间。在Whale中实现该优化后的并行化方案，无需修改模型定义部分代码，只需要修改硬件资源分组逻辑，即将参数layout从Column模式改成Row模式，详情请参见whale.cluster。该方案的计算图如下所示。



Whale实现流水并行

在Whale中实现流水并行需要三个步骤（标准的Whale分布式编程范式）：模型划分、资源划分及映射。

1. 模型划分。

   将模型各部分按照显存占用和算力需求划分为不同部分，即Scopes划分，详情请参见whale.scopes。
   上述的实现方案中将模型划分为以下三部分：

   - Stage 0：包括Embedding和Encoder Layer 1~8层
   - Stage 1：包括Encoder Layer 9~16层
   - Stage 2：包括Encoder Layer17~24层和Pooler层

   针对Bert Large模型的原始代码，在Whale中您可以通过如下步骤实现流水并行。

   i. 通过 `whale.stage` 实现模型并行，代码示例如下。

```
import whale as wh
with wh.stage():
    embedding_output = embedding(inputs)
    encoder_output = encoder_layer_1_8(embedding_output)
with wh.stage():
    encoder_output = encoder_layer_9_16(embedding_output)
with wh.stage():
    encoder_output = encoder_layer_17_24(embedding_output)
    pooler_output = pooler(encoder_output)
```

ii. 在模型并行的代码基础上，通过 `whale.pipeline` 实现流水并行，示例代码如下。

```
import whale as wh
with wh.pipeline(num_micro_batch=5): # 按顺序依次处理5个Mini-Batch大小的数据，进行流水并行。每轮迭代每个副本训练5*Mini-Batch条样本数据。
    with wh.stage():
        embedding_output = embedding(inputs)
        encoder_output = encoder_layer_1_8(embedding_output)
    with wh.stage():
        encoder_output = encoder_layer_9_16(embedding_output)
    with wh.stage():
        encoder_output = encoder_layer_17_24(embedding_output)
        pooler_output = pooler(encoder_output)
```

iii. 在流水并行的基础上，通过 `whale.replica` 实现数据并行，示例代码如下所示。

```
import whale as wh
with wh.replica():  # 数据并行。
    with wh.pipeline(num_micro_batch=5):  # 按顺序依次处理num_micro_batch个Mini-Batch大小的数据，进行流水并行。每轮迭代每个副本训练num_micro_batch*Mini-Batch条样本数据。
        with wh.stage():
            embedding_output = embedding(inputs)
            encoder_output = encoder_layer_1_8(embedding_output)
        with wh.stage():
            encoder_output = encoder_layer_9_16(embedding_output)
        with wh.stage():
            encoder_output = encoder_layer_17_24(embedding_output)
            pooler_output = pooler(encoder_output)
```

2. 资源划分。

   由于将模型划分为三个部分，因此需要将硬件资源进划分为三个Virtual Devices。假设申请 `3*3` 的服务器规模，划分策略为 `layout="row"` （详情请参见whale.cluster），则资源分为如下三组：

   - Virtual Devices 0：[[/worker:0/gpu:0], [/worker:0/gpu:1], [/worker:0/gpu:2]]
   - Virtual Devices 1：[[/worker:1/gpu:0], [/worker:1/gpu:1], [/worker:1/gpu:2]]
   - Virtual Devices 2：[[/worker:2/gpu:0], [/worker:2/gpu:1], [/worker:2/gpu:2]]

   Whale提供的Cluster工具可以对申请的Worker进行划分，示例代码如下。

   ```
   cluster = wh.cluster(layout={"row":3})
   ```

3. 映射。

   根据模型划分和资源划分结果，将模型各部分分别映射至Virtual Devices：

   - Stage 0部分采用数据并行放至Virtual Device 0，即[[/worker:0/gpu:0], [/worker:0/gpu:1], [/worker:0/gpu:2]]。每张GPU卡都有一个Stage 0的模型副本。
   - Stage 1部分采用数据并行放至Virtual Device 1，即[[/worker:1/gpu:0], [/worker:1/gpu:1], [/worker:1/gpu:2]]。每张GPU卡都有一个Stage 1的模型副本。
   - Stage 2部分采用数据并行放至Virtual Device 2，即[[/worker:2/gpu:0], [/worker:2/gpu:1], [/worker:2/gpu:2]]。每张GPU卡都有一个Stage 2的模型副本。

   对于映射部分，在Whale中只需要对已生成的Cluster执行 `with` 语法，即可轻松完成模型到硬件资源的映射。通过Whale完成并行化的模型核心代码如下（包含流水并行的可执行代码请参见train_bert_model.py）。

```
import whale as wh
cluster = wh.cluster(layout={"row":3}) #资源划分。
with cluster:                          # 映射。
    with wh.replica():
        with wh.pipeline(num_micro_batch=5):
            with wh.stage():
                embedding_output = embedding(inputs)
                encoder_output = encoder_layer_1_8(embedding_output)
            with wh.stage():
                encoder_output = encoder_layer_9_16(embedding_output)
            with wh.stage():
                encoder_output = encoder_layer_17_24(embedding_output)
                pooler_output = pooler(encoder_output)
```

性能

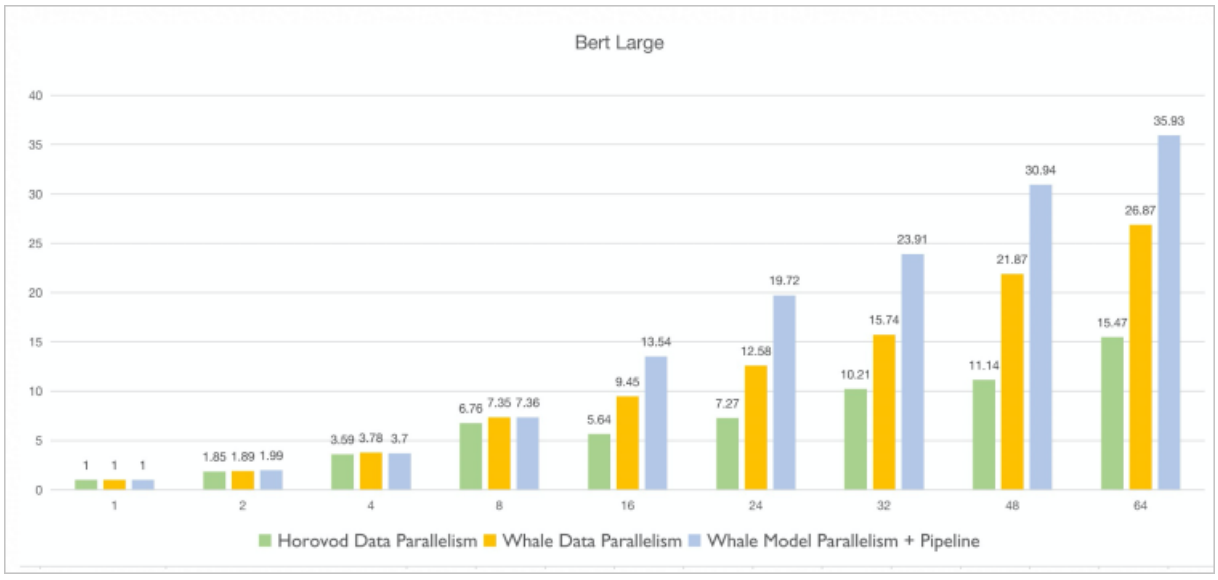
以Horovod的数据并行性能数据为Baseline进行对比，测试环境如下。

测试环境项	描述
GPU型号	ecs.gn6v-c10g1.20xlarge（V100 * 8）
网络	VPC-35 GB
NCCL_MAX_NRINGS	NVIDIA官方参数，测试时取值为4。
NCCL_MIN_NRINGS	NVIDIA官方参数，测试时取值为4。

本文中的流水并行对比测试场景分别为：

- Horovod数据并行
- Whale数据并行
- 结合Whale模型并行和流水并行

对比结果如下图所示。



> **说明** 横坐标表示GPU卡数，纵坐标表示加速比。

从上图中，可以得到以下结论：

- Whale在数据并行场景的性能优于Horovod。尤其在GPU 64卡环境下，Whale的加速比是Horovod的1.74倍。
- 在GPU小于8卡的场景，Whale数据并行与结合Whale模型并行和流水并行的性能相差不大。因为数据并行的效果主要得益于服务器内GPU通过高速NVLink相连，通信效率很高。当GPU卡数逐渐增多，出现跨服务器通信时，流水并行的优势才会显现。
- 在GPU 64卡场景，结合Whale模型并行和流水并行的吞吐性能是Whale数据并行的1.34倍。

5.2.3.3. API

5.2.3.3.1. whale.init

在使用Whale进行资源划分（Cluster）和模型划分（Scopes）之前，需要先使用whale.init接口初始化Whale环境。本文介绍whale.init接口的调用格式、功能说明及调用示例。

- 格式

```
init()
```

- 功能
  在模型定义之前调用该接口，用于初始化Whale环境。此时，Cluster和Scopes功能生效，系统根据指定的模式进行并行化。如果未调用该接口，则通过Scopes对模型的划分无法生效，代码以单机单卡模式运行。
- 参数
  无
- 返回值
  无
- 示例

```
import whale as wh
wh.init()
```

5.2.3.3.2. whale.cluster

Cluster是硬件资源到逻辑资源的映射，支持不同分组策略将硬件资源划分为多组Virtual Devices。本文介绍Cluster支持的分组策略、调用格式、参数说明、调用示例及Worker角色概念，以指导您进行资源划分和映射。

背景信息

分布式框架的工作是将用户模型的各部分通过不同并行化方法放至分布式计算硬件上，并进行计算。该过程在Whale中总结为模型划分、资源划分及映射。Whale提供的分布式计算资源划分工具Cluster，适用于资源划分和映射部分。

该部分涉及的重要概念包括：

- cluster
  cluster是对整个硬件资源到逻辑资源的划分映射管理类，支持不同分组策略将硬件资源划分为多组Virtual Devices。
- layout

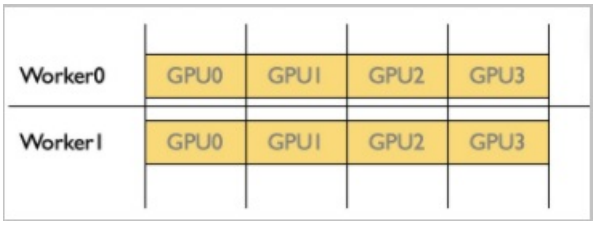
layout决定了如何从硬件资源到逻辑资源划分映射，其中layout是cluster的一个参数。

格式

```
cluster(worker_hosts=None,
        ps_hosts=None,
        job_name="worker",
        rank=0,
        layout="all")
```

参数

- worker_hosts：Worker Hosts信息，STRING类型。默认值为None，表示从TF_CONFIG环境变量中获取Hosts、Rank等信息。
- ps_hosts：Parameter Server Hosts信息，STRING类型。
- job_name：当前Worker Job Name，STRING类型，默认值为"worker"。
- rank：当前Worker Rank，INTEGER类型。
- layout：Cluster的分组策略，DICT类型，默认值为all。Whale的Cluster提供的资源分组策略有以下几种：
  - all
    将所有GPU放置在一个分组中。
    以下图的资源为例，介绍如何实现该分组策略及实现效果。

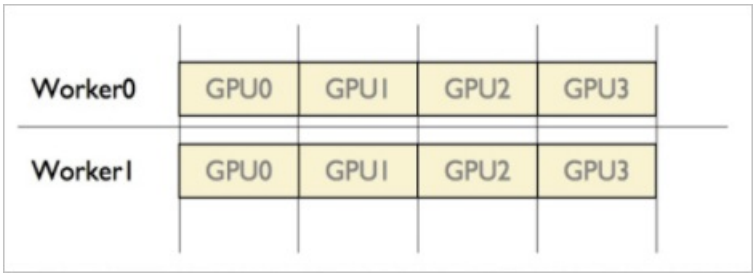
    您可以使用如下代码实现该分组策略。

    ```
    import whale as wh
    cluster = wh.cluster(layout="all")
    ```

    返回的 `cluster.slices` 内容如下。

    ```
    [
        [
            [Worker0:GPU0],
            [Worker0:GPU1],
            [Worker0:GPU2],
            [Worker0:GPU3],
            [Worker1:GPU0],
            [Worker1:GPU1],
            [Worker1:GPU2],
            [Worker1:GPU3]
        ]
    ]
    ```

- average
  Cluster内的所有资源平均划分，每份N张卡。
  将Worker内的GPU按行排布，如下图所示。



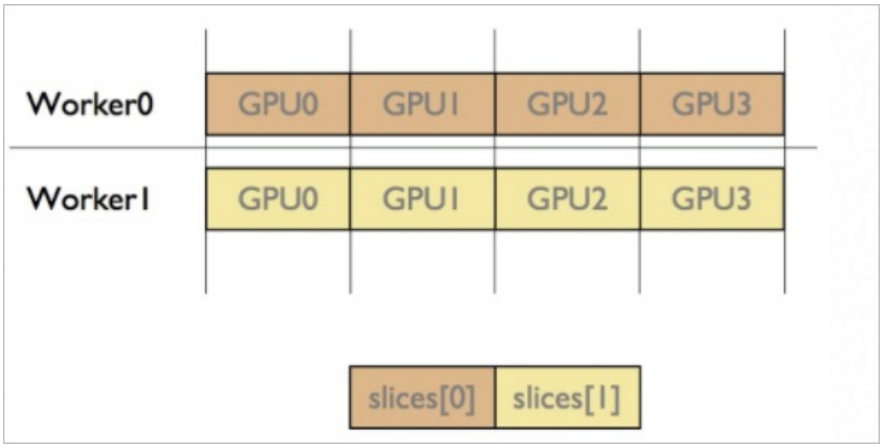
您可以使用如下代码实现该分组策略。

```
import whale as wh
cluster = wh.cluster(layout = {'average' : 4 } )
```

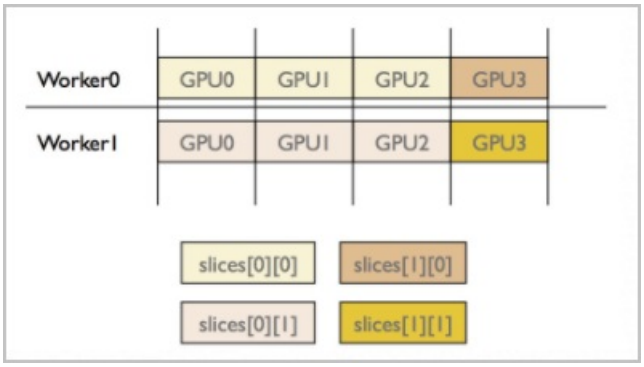
返回的 `cluster.slices` 内容如下。

```
[
    [
        [Worker0:GPU0],
        [Worker0:GPU1],
        [Worker0:GPU2],
        [Worker0:GPU3]
    ],
    [
        [Worker1:GPU0],
        [Worker1:GPU1],
        [Worker1:GPU2],
        [Worker1:GPU3]
    ],
]
```

Cluster的划分结果如下图所示。



- block
  将Cluster按照GPU Block分组。
  下图中的资源划分为两个组，分别包含6张卡和2张卡。每个分组又细分为两个子分组，通过Cluster的slices属性可以获取各分组信息。



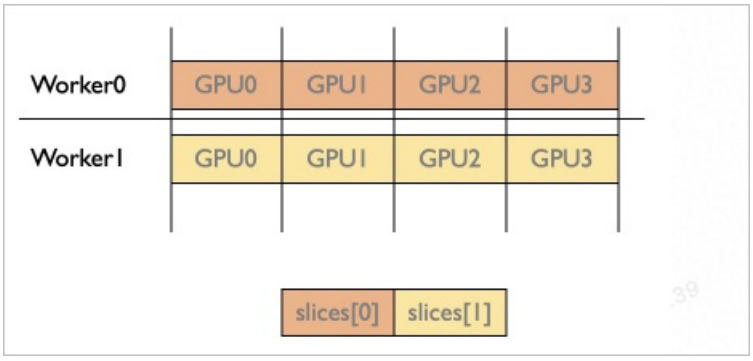
  您可以使用如下代码实现该分组策略。

```
import whale as wh
cluster = wh.cluster(layout={"block" : "3,1"})
```

  返回的 `cluster.slices` 内容如下。

```
[
    [
        [
            Worker0:GPU0,
            Worker0:GPU1,
            Worker0:GPU2
        ],
        [
            Worker1:GPU0,
            Worker1:GPU1,
            Worker1:GPU2
        ]
    ],
    [
        [Worker0:GPU3],
        [Worker1:GPU3]
    ]
]
```

- row
  将Worker内的GPU按行排布。
  如下图所示，row的划分则是水平机器内划分。



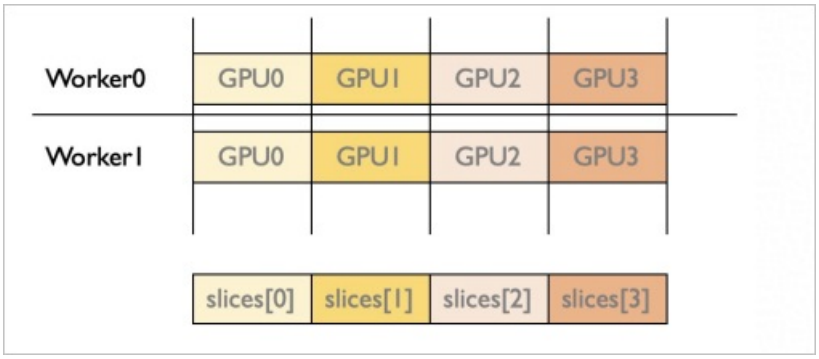
  您可以使用如下代码实现该分组策略。

```
import whale as wh
cluster = wh.cluster(layout = {"row" : 4 })
```

  返回的 `cluster.slices` 内容如下。

```
[
    [
        [Worker0:GPU0],
        [Worker0:GPU1],
        [Worker0:GPU2],
        [Worker0:GPU3]
    ],
    [
        [Worker1:GPU0],
        [Worker1:GPU1],
        [Worker1:GPU2],
        [Worker1:GPU3]
    ]
]
```

- column
  将Worker内的GPU按行排布，Column的划分按照垂直跨机划分。
  Column的划分如下图所示。



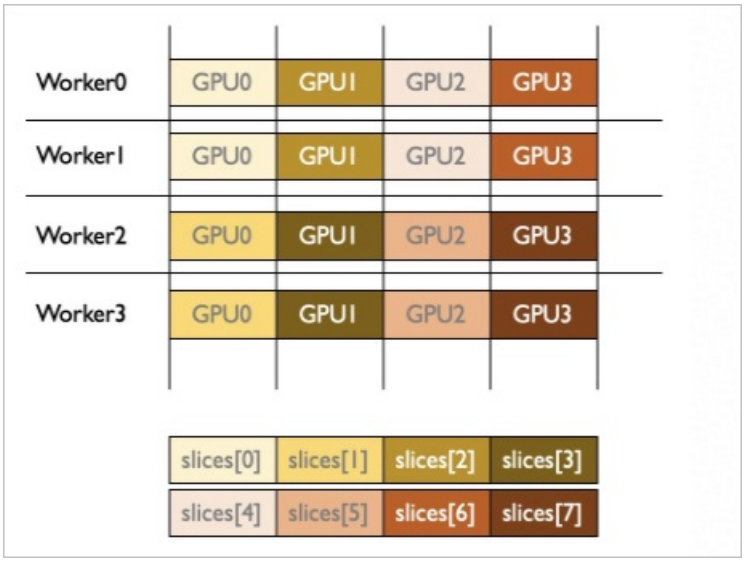
  您可以使用如下代码实现该分组策略。

```
import whale as wh
cluster = wh.cluster(layout = {"column" : 2 })
```

返回的 `cluster.slices` 内容如下。

```
[
    [
        [Worker0:GPU0],
        [Worker1:GPU0]
    ],
    [
        [Worker0:GPU1],
        [Worker1:GPU1]
    ],
    [
        [Worker0:GPU2],
        [Worker1:GPU2]
    ],
    [
        [Worker0:GPU3],
        [Worker1:GPU3]
    ]
]
```

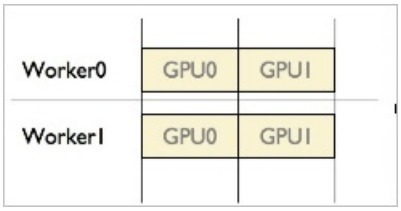
当存在更多Worker时，Cluster按column划分，如果使用 `layout={"column" : 2 }` 策略，则Cluster划分结果如下图所示。



返回的 `cluster.slices` 内容如下。

```
[
    [
        [Worker0:GPU0],
        [Worker1:GPU0]
    ],
    [
        [Worker2:GPU0],
        [Worker3:GPU0]
    ],
    [
        [Worker0:GPU1],
        [Worker1:GPU1]
    ],
    [
        [Worker2:GPU1],
        [Worker3:GPU1]
    ],
    [
        [Worker0:GPU2],
        [Worker1:GPU2]
    ],
    [
        [Worker2:GPU2],
        [Worker3:GPU2]
    ],
    [
        [Worker0:GPU3],
        [Worker1:GPU3]
    ],
    [
        [Worker2:GPU3],
        [Worker3:GPU3],
    ]
]
```

- specific
  通过配置具体的Device信息进行cluster划分。
  将Worker内的GPU按行排布，如下图所示。



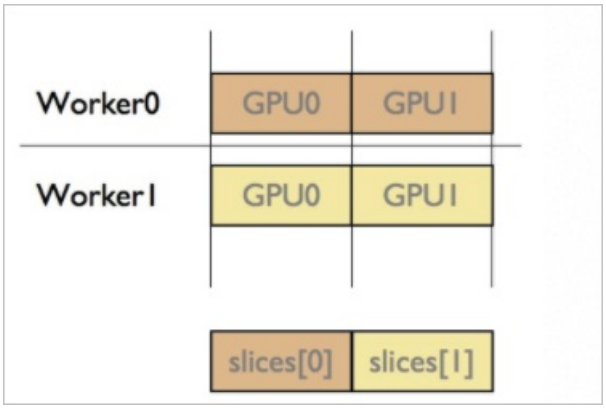
  您可以使用如下代码实现该分组策略。

```
import whale as wh
cluster = wh.cluster(
    layout = {'specific' : [ [['worker:0/gpu:0'], ['worker:0/gpu:1']], [['worker:1/gpu:
0'], ['worker:1/gpu:1']] ] } )
```

  返回的 `cluster.slices` 内容如下。

```
[
    [
        [Worker0:GPU0],
        [Worker0:GPU1]
    ],
    [
        [Worker1:GPU0],
        [Worker1:GPU1]
    ],
]
```

  Cluster划分结果如下图所示。



返回值

返回指定layout分组策略后的 `whale.Cluster` 对象，具有以下属性：

- rank
  调用方式如下。

```
cluster.rank
```

  返回当前的Worker Rank。
- worker_num

调用方式如下。

```
cluster.worker_num
```

返回Worker数量。

- gpu_num_per_worker
  调用方式如下。

```
cluster.gpu_num_per_worker
```

返回每个Worker的GPU数量。

- slices
  调用方式如下。

```
cluster.slices
```

返回Cluster通过layout划分后的资源分组，返回类型为嵌套的LIST。

- total_gpu_num
  调用方式如下。

```
cluster.total_gpu_num
```

返回Cluster中所有GPU数量。

示例

将模型映射到Cluster划分好的Virtual Devices slices组，有两种实现方法，示例分别如下：

- 使用 `with` 自动映射
  在模型定义最前面，通过 `with` 语法，Whale框架可以自动将模型映射到Virtual Devices，示例代码如下。

```
import whale as wh
with wh.cluster(layout = {"row" : 4 }):
    # 假设将Cluster划分为如下2个Slices。
    # [
    #    [ [Worker0:GPU0], [Worker0:GPU1], [Worker0:GPU2], [Worker0:GPU3] ],
    #    [ [Worker1:GPU0], [Worker1:GPU1], [Worker1:GPU2], [Worker1:GPU3] ]
    # ]
    with wh.stage():
        model_part_1()
    with wh.stage():
        model_part_2()
```

- 使用Slices手动映射
  Whale的Cluster通过layout划分之后，可以通过Slices方法获取划分后的分组。Whale Scopes支持直接设置Devices信息，因此可以使用Slices手动映射。示例代码如下。

```
import whale as wh
cluster = wh.cluster(layout = {"row" : 4 })
# 假设将cluster划分成如下2个slices。
# [
#    [ [Worker0:GPU0], [Worker0:GPU1], [Worker0:GPU2], [Worker0:GPU3] ],
#    [ [Worker1:GPU0], [Worker1:GPU1], [Worker1:GPU2], [Worker1:GPU3] ]
# ]
with wh.stage(cluster.slices[0]):
    model_part_1()
with wh.stage(cluster.slices[1]):
    model_part_2()
```

手动设置的方法可以更灵活地为每个Scope设置Device信息，例如多个Scopes复用相同Slices。

> **说明** 不用混用 `with` 自动用法和手动用法。

Whale中Worker角色概念

根据 `whale.cluster` 定义，申请到的所有Worker会通过layout策略进行分组，生成Virtual Devices Slices，该Slices是一个嵌套LIST。Whale框架将 `slices[0]` 中所有Device的对应Worker称之为Master Worker，负责原始图构造，并完成并行化图的改写。其余Worker不进行构图，直接调用 `server.join` 等待Master Worker分配任务，这些Worker称之为Slave Workers。

例如申请了3台服务器，每台服务器2张GPU卡。那么通过 `layout=wh.cluster(layout={"block" : "3,3"})` 生成对应的 `cluster.slices` 如下。

```
[
    [
        ["/job:worker/replica:0/worker:0/device:GPU:0"],
        ["/job:worker/replica:0/worker:0/device:GPU:1"],
        ["/job:worker/replica:0/worker:1/device:GPU:0"]
    ],
    [
        ["/job:worker/replica:0/worker:1/device:GPU:1"],
        ["/job:worker/replica:0/worker:2/device:GPU:0"],
        ["/job:worker/replica:0/worker:2/device:GPU:1"]
    ]
]
```

由上可知 `cluster.slices[0]` 取值如下。

```
[
  ["/job:worker/replica:0/worker:0/device:GPU:0"],
  ["/job:worker/replica:0/worker:0/device:GPU:1"],
  ["/job:worker/replica:0/worker:1/device:GPU:0"]
]
```

所以task_index为0和1的Worker即为Master Workers，task_index为2的Worker即为Slave Workers。

5.2.3.3.3. whale.scopes

Whale提供的Scopes工具，适用于模型划分阶段。本文介绍Whale支持的不同Scopes类型的接口格式、参数描述及调用示例，以指导您进行并行化操作。

分布式框架的工作是将用户模型的各部分通过不同并行化方法放至分布式计算硬件中，并进行计算。该过程在Whale中总结为：模型划分、资源划分及映射。Whale提供的Scopes工具，适用于模型划分阶段。

在Whale中，需要将模型计算图划分为不同的计算子图（Subgraph），这些计算子图就是通过 `whale scopes` 进行划分。Whale提供多种不同的Scopes类型，分别表示对不同Subgraph进行不同类型的并行化操作，即Whale以SubGraph的粒度进行并行化。Whale提供的Scopes类型包括：

> **说明** 所有的Scopes均通过 `with` 语法进行调用。每种并行化的原理与机制请参见Whale分布式范式剖析。

- `whale.replica`
  - 格式
    ```
    replica(devices=None, name=None)
    ```
  - 功能
    Replica Scope下的Subgraph被拷贝到对应的Device上，即数据并行。
  - 参数
    - devices：Replica Scope下的Subgraph执行的GPU Devices，STRING LIST类型，默认值为None。
    - name：当前Scope的名称，STRING类型，默认值为None。
  - 返回值
    返回Replica对象。
  - 示例
    ```
    import whale as wh
    with wh.replica():
        Your_Model_Definition()
    ```
- whale.split
  - 格式
    ```
    Split(devices=None, name=None)
    ```
  - 功能
    Split Scope下的Subgraph被拆分为多份，放置到多个Device上，即算子拆分的模型并行。
    Whale支持的可拆分算子列表如下：
    - `tf.layers.dense`
    - `tf.losses.sparse_softmax_cross_entropy`
    - `tf.arg_max`
    - `tf.equal`
    - `tf.math.equal`
    - `tf.metrics.accuracy`
  - 参数
    - devices：该Split Scope对应的Devices信息，Whale会将Split Scope下的subgrah拆分到这些Devices上进行执行。STRING LIST类型，默认值为None，即从Cluster的Slices中获取。
    - name：当前Scope的名称，STRING类型，默认值为None。

- 返回值
  返回Split对象。
- 示例

```
import whale as wh
with wh.split():
    Your_Model_Definition()
```

- whale.stage
  - 格式

```
stage(devices=None, name=None)
```

  - 功能
    不同的Stage Scope独立生成一个Subgraph，将完整模型拆分到多个Device上执行，即Layer间拆分的模型并行。
  - 参数
    - devices：指定Stage Scope下的Subgraph执行的GPU Devices信息，STRING LIST类型，默认值为None。
    - name：当前Scope的名称，STRING类型，默认值为None。
  - 返回值
    返回Stage对象。
  - 示例

```
import whale as wh
# 场景说明：将模型分成2个Stage，实现模型并行。
with wh.stage():
    Model_Part_1()
with wh.stage():
    Model_Part_2()
```

- whale.pipeline
  - 格式

```
pipeline(num_micro_batch, devices=None, strategy=None, name=None)
```

  - 功能
    Pipeline Scope下的Subgraph会构造流水进行执行，即流水并行。该类型Scope需要与 `whale.stage` 配合使用。
  - 参数
    - num_micro_batch：Pipeline Scope下执行流水并行的micro-batch数量，INTEGER类型。
    - devices：指定Pipeline Scope下的Subgraph执行的GPU devices，STRING LIST类型，默认值为None。
    - strategy：流水并行策略，支持PreferForward、PreferBackward及PreferBackwardOptimizer策略。STRING类型，该参数取值为None时，表示使用PreferBackwardOptimizer策略。
    - name：当前Scope的名称，STRING类型，默认值为None。
  - 返回值
    返回Pipeline对象。

- 示例

```
import whale as wh
# 通常配合whale.stage一起使用。
# 场景说明：将模型分成2个Stage，并应用流水并行策略。
with wh.pipeline(num_micro_batch=5):
    with wh.stage():
        Model_Part_1()
    with wh.stage():
        Model_Part_2()
```

5.2.3.3.4. whale.current_scope_as_default

通过whale.current_scope_as_default接口，可以将当前Scope指定为默认Scope。本文介绍该接口的调用格式、功能说明及调用示例。

- 格式

```
current_scope_as_default()
```

- 功能
  指定当前Scope为默认Scope。如果用户模型定义在Scope之外，则将该算子放置到默认Scope中。
- 参数
  无
- 返回值
  无
- 示例

```
import whale as wh
with wh.replica():  # 当前Scope。
    c1 = tf.constant(1)
    wh.current_scope_as_default()  # 指定当前Scope为默认Scope。
c2 = tf.constant(2)  # C2定义在当前Scope之外，由于已经将当前scope设置为了默认scope，因此最终C2和C1放在同一Scope下。
```

5.2.3.3.5. whale.auto_parallel

通过whale.auto_parallel接口，您可以快速实现模型分布式训练。本文介绍该接口的调用格式、参数说明及调用示例。

背景信息

Whale可以通过资源Cluster划分和模型Scope划分实现模型分布式训练。该实现方式需要理解资源和模型的配置组合，以达到高效地分布式训练性能。为了进一步降低使用成本及简化操作，Whale提供了whale.auto_parallel接口，可以通过一行代码的简易模式自动进行并行化操作。

接口说明

- 格式

```
auto_parallel(modes)
```

- 功能
  通过一行代码的简易模式自动进行并行化操作。例如：

- `whale.auto_parallel(whale.replica)`：自动对整个模型进行数据并行。
- `whale.auto_parallel(whale.split)`：自动对整个模型进行算子拆分。
- `whale.auto_parallel(whale.pipeline)`：自动对整个模型进行模型拆分，再进行流水并行。
- `whale.auto_parallel([whale.pipeline, whale.replica])`：Whale通过Split和Replica两种模式进行自动并行，其中的参数modes可以自由组合多种`whale.scopes`原语。
- `whale.auto_parallel()`：全自动并行化训练模式，Whale自动推断生成分布式策略。

> **说明** 当前仅支持`whale.auto_parallel(whale.replica)`自动数据并行模式。

- 参数
  modes：并行化模式，SCOPES类型。例如数据并行、模型并行、流水并行及组合的并行方式。当前仅支持自动数据并行，即`modes=whale.replica`。
- 返回值
  无
- 示例
  在`import whale as wh`之后，添加`wh.auto_parallel(wh.replica)`即可自动配置Cluster，并实现数据并行。此处仅给出主体代码，完整代码请参见auto_data_parallel.py。

```
import whale as wh
wh.auto_parallel(wh.replica)
# Construct your model here.
model_definition()
```

5.2.3.3.6. whale.GraphKeys

Whale提供的GraphKeys工具，可以区分不同聚合操作的Collection。本文为您提供GraphKeys支持的接口格式、参数说明及调用示例。

接口	功能
add_to_collection	将Tensor对象加入到特定聚合方式对应的Collection。
get_all_collections	返回所有非空Collection的所有Tensor对象组成的列表。
get_collection	返回特定聚合方式对应Collection中的所有Tensor对象组成的列表。

背景信息

为提升模型训练的样本吞吐，经常混合使用不同的分布式并行方式，例如混合使用数据并行、模型并行及流水并行。然而Whale默认不对任何算子的输出Tensor Value进行多副本聚合，为了调试或算法收敛，需要定期查看局部或全局loss、accuracy等指标变化。Whale提供若干以graph_key区分的Collections，您可以使用`whale.add_to_collection`、`whale.get_all_collections`或`whale.get_collection`接口查看或修改对应Collection的Tensor列表。

add_to_collection

- 格式

```
add_to_collection(tensor, graph_key)
```

- 功能
  将某tensor加入到graph_key对应的Collection，以便Whale框架按需自动聚合该tensor的数值。如果未调用该接口，则默认所有Tensor在 `sess.run` 时只返回单个副本的对应数值。
- 参数
  - tensor：待聚合的某算子输出的Tensor对象，TensorFlow Tensor类型。
  - graph_key：Tensor对象的聚合方式，STRING类型，取值请参见GraphKeys取值及场景示例。
- 返回值
  无
- 示例

```
import tensorflow as tf
import whale as wh
# 场景说明：查看所有模型副本的全局loss均值。
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=tf.cast(logits, tf.float32))
wh.add_to_collection(loss, wh.GraphKeys.GLOBAL_MEAN_OBJECTS)
```

get_all_collections

- 格式

```
get_all_collections()
```

- 功能
  返回所有非空Collection的所有Tensor对象组成的列表。
- 参数
  无
- 返回值
  LIST类型。如果所有Collection为空，则返回空LIST。
- 示例

```
import whale as wh
# 场景说明：输出所有Collection元素。
print(wh.get_all_collections())
```

get_collection

- 格式

```
get_collection(graph_key)
```

- 功能
  返回graph_key对应Collection中的所有Tensor对象组成的列表。
- 参数
  graph_key：指定相应Collection中Tensor聚合方式，STRING类型，取值详情请参见GraphKeys取值及场景示例。
- 返回值
  LIST类型。如果相应Collection为空，则返回空LIST。
- 示例

```
import whale as wh
# 场景说明：查看graph key为whale.GraphKeys.GLOBAL_MEAN_OBJECTS时，Collection中的所有Tensor对象。
print(wh.get_collection(wh.GraphKeys.GLOBAL_MEAN_OBJECTS))
```

GraphKeys取值及场景示例

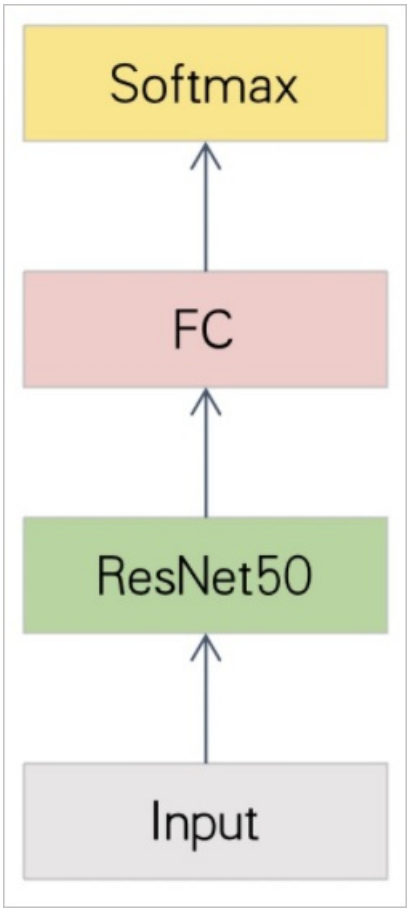
取值	描述	场景示例
GLOBAL_CONCAT_OBJECTS	该Collection内的Tensor将进行数值的全局（针对所有模型副本）拼接，拼接维度 `axis=0` 。	查看一次迭代中所有模型副本消耗的所有数据。
LOCAL_CONCAT_OBJECTS	该Collection内的Tensor将进行数值的局部（针对当前模型副本）拼接，拼接维度 `axis=0` 。	查看一次迭代中当前模型副本消耗的所有数据。例如，流水并行场景下多个micro-batch的聚合。
GLOBAL_MEAN_OBJECTS	该Collection内的Tensor将进行数值的全局（针对所有模型副本）求平均。	查看一次迭代中的所有模型副本全局loss均值。
LOCAL_MEAN_OBJECTS	该Collection内的Tensor将进行数值的局部（针对当前模型副本）求平均。	查看一次迭代中的当前模型副本局部loss均值，例如，流水并行场景下多个micro-batchloss的均值。
GLOBAL_SUM_OBJECTS	该Collection内的Tensor将进行数值的全局（针对所有模型副本）求和。	查看一次迭代中的所有模型副本全局loss之和。
LOCAL_SUM_OBJECTS	该collection内的Tensor将进行数值的局部（针对当前模型副本）求和。	查看一次迭代中的当前模型副本局部loss之和。例如，流水并行场景下多个micro-batchloss之和。

5.2.3.4. Whale分布式范式剖析

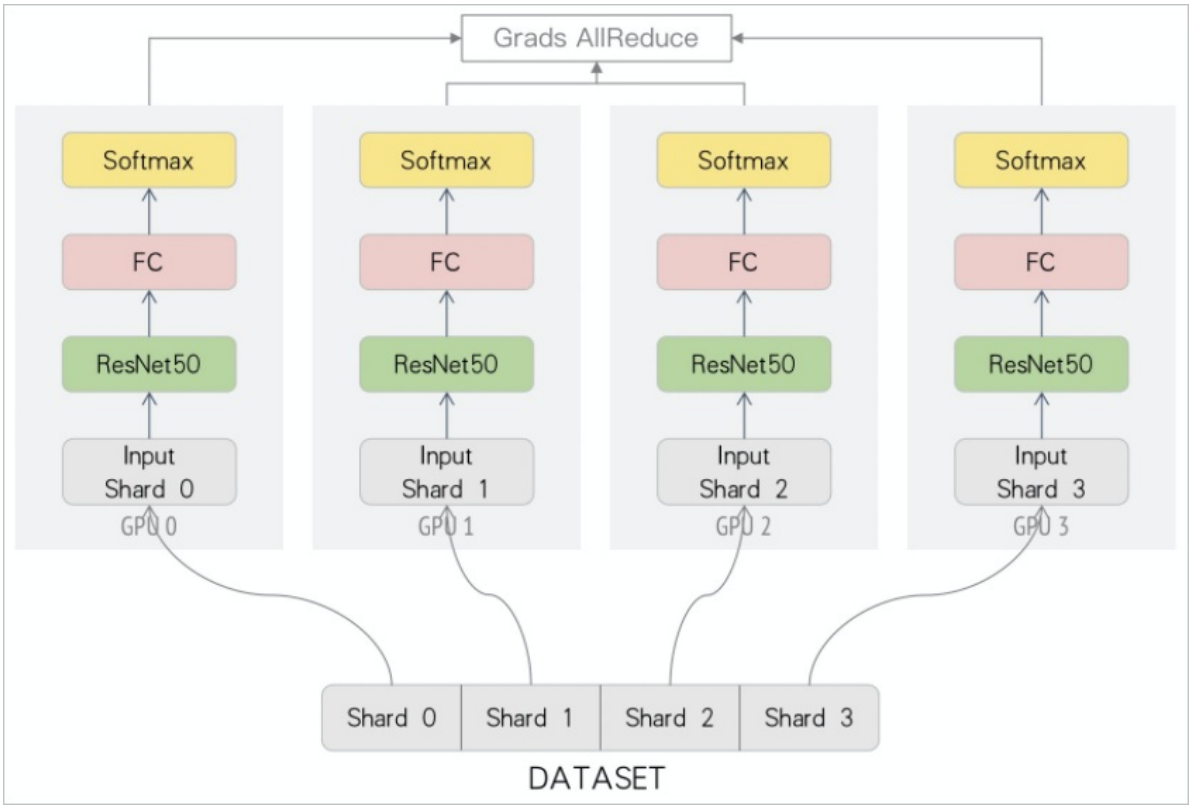
针对Whale支持的几种并行化方式（数据并行、流水并行、Layer间拆分的模型并行、算子拆分的模型并行及各种混合并行策略），本文介绍其实现逻辑及在Whale中的实现方式，以指导您快速使用Whale进行分布式训练。

数据并行

- 背景信息
  对于大规模应用数据，机器学习模型通常需要通过数据并行进行分布式训练加速。数据并行是分布式训练最常见的并行化方式，也是Whale支持的最基本的并行化方式之一。
- 定义
  数据并行是指将训练数据分片放到不同运算节点中，并以相同的运算逻辑进行运算。
- 实现逻辑
  以ResNet50模型为例，介绍数据并行的实现逻辑。该模型如下图所示。



当数据并行时，每张GPU卡都会有一个独立的模型副本，但是读取的训练数据不同，如下图所示。



每轮迭代分为三步：

i. 根据上一轮更新好的模型参数（权重），每个模型副本分别计算loss并得到梯度。

ii. 所有模型副本进行Reduce操作，完成梯度同步（所有Worker梯度ReduceSum在一起，同步后，每个模型副本得到同样的梯度）。

iii. 将梯度更新到模型参数（权重）。

- Whale中实现数据并行
  在Whale中，您可以使用默认的Cluster Layout方式，并将模型代码放置在Replica Scope下，即可实现数据并行，代码示例如下（数据并行完整的可执行代码请参见simple_data_parallel.py）。

```
import whale as wh
wh.init()
with wh.cluster():
    with wh.replica():
          ResNet50_Model_Defination()
```

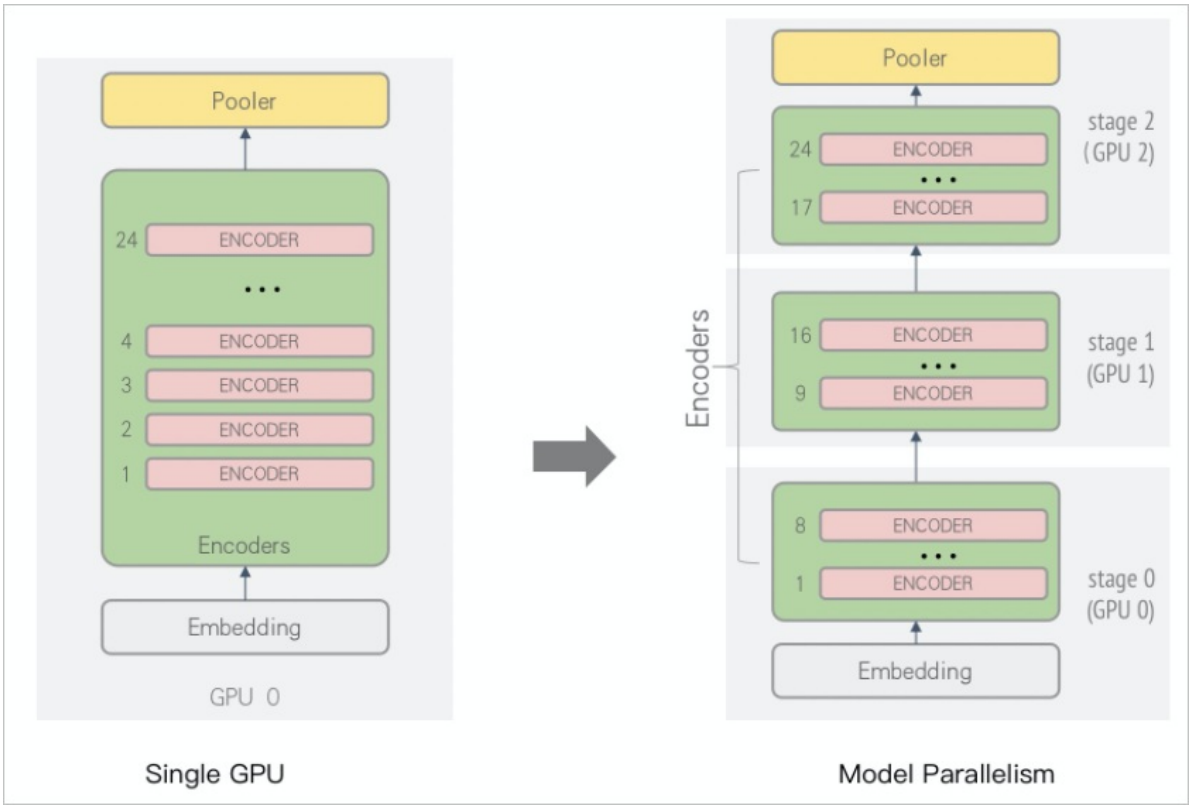
模型并行

- 背景信息
  受益于深度神经网络的发展，NLP及CV等领域的模型效果得到很大提升，但是同时大幅度增加了模型参数量。以Imagenet分类任务为例，优胜算法从2014年GoogleNet到2018年的Squeeze-and-Excitation Networks，参数量增长约36倍（从4百万增长至1.458亿参数）。Turing-NLG模型甚至达到170亿参数规模大小。模型参数飞速增长带来如下挑战：
  - 分布式训练梯度同步通信量增大。此外，受限于GPU显存，训练的Batch Size变小，导致训练的通信占比增加，分布式扩展效率降低。
  - 过小的Batch Size导致模型训练波动变大，影响收敛效果。

  模型并行适用于模型本身或部分规模较大，使用纯数据并行的训练加速效果不好或无法进行纯数据并行的场景。例如以下类似场景：
  - NLP领域的GPT 2和T5等参数规模超大的场景。由于模型的显存消耗过大，单纯的数据并行无法训练。
  - NLP领域的Bert Large等参数规模较大的场景。模型中层与层间存在Feature Map（Activation），通信量远小于Weights 的通信量。
  - 以VGG-16模型为例，各层显存不平衡，单纯数据并行会造成较大的显存浪费。

- 定义
  一种并行策略，不同计算设备（GPU和CPU等）负责网络模型不同层的计算，例如神经网络模型的不同网络层被分配到不同的设备。
- 实现逻辑

以BertLarge模型为例，通过Layer间的拆分策略，将BertLarge分配至不同的GPU卡中进行计算，以降低每张GPU卡的模型显存占用，从而提高BatchSize。同时，使用Activation通信代替梯度同步，大幅度降低卡间通信量。模型拆分如下图所示。



数据并行时，每轮迭代需要通信一个完整的BertLarge模型梯度（约1.2G）。由于模型较大，导致在16G V100环境（还涉及Sequence length、Embedding Size等因素）中每次计算2条样本就需要进行梯度通信。因此，不仅通信占比过高导致性能差，而且由于Batch Size过小导致模型波动较大，收敛效果差。而模型并行时，每轮迭代只需要通信约27M的Activation数据，无需梯度通信过程。同时可以大幅度增加Batch Size，因此训练更加稳定，收敛效果更好。

- Whale中实现模型并行
  您需要配置Cluster Layout，并将模型的不同部分放至Stage Scope下即可在Whale中实现模型并行。此处以BertLarge为例，单卡模型与模型并行的示例代码片段分别如下所示（BertLarge的模型并行可执行代码请参见train_bert_model.py）：
  - 单卡模型

```
embedding_output = embedding(inputs)
encoder_output = encoder(embedding_output)
pooler_output = pooler(encoder_output)
```

- 模型并行

```
import whale as wh
cluster = wh.cluster()
with cluster:
    with wh.stage():
        embedding_output = embedding(inputs)
        encoder_output = encoder_layer_1_8(embedding_output)
    with wh.stage():
        encoder_output = encoder_layer_9_16(embedding_output)
    with wh.stage():
        encoder_output = encoder_layer_17_24(embedding_output)
        pooler_output = pooler(encoder_output)
```

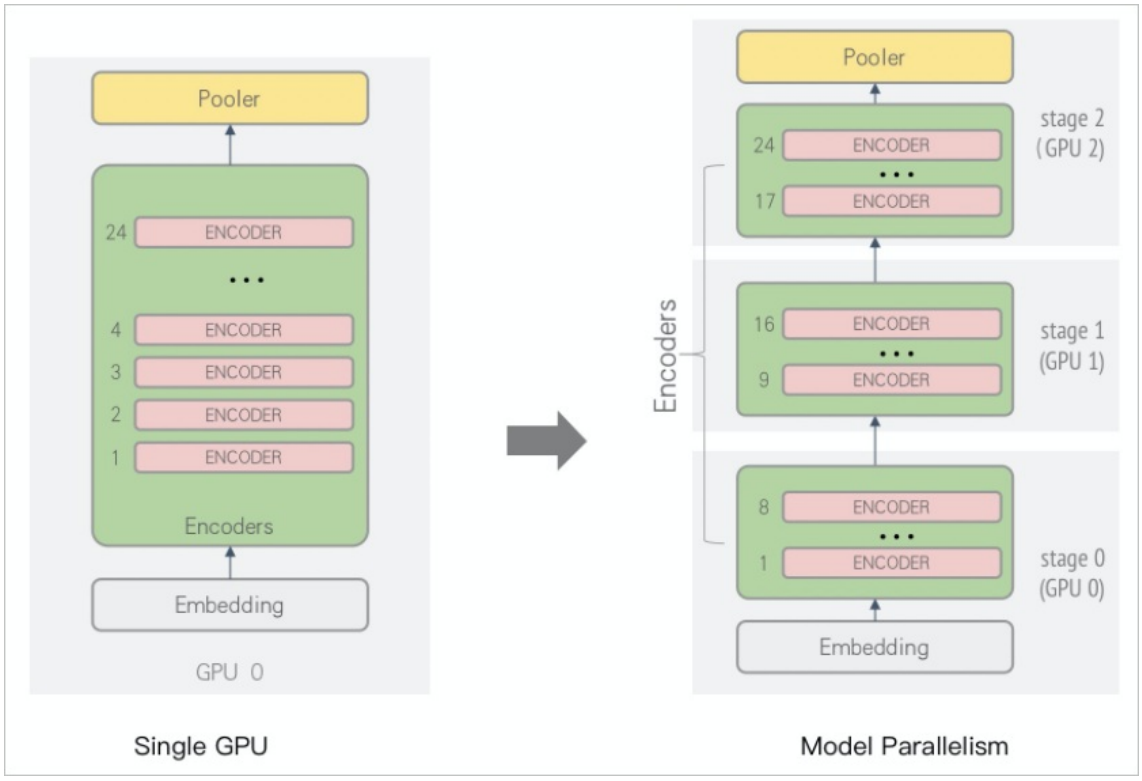
> **说明** 模型并行通常需要搭配流水并行，进行训练加速。

流水并行

- 背景信息
  模型并行时，整个模型被划分为若干Stages，分配到不同的GPU卡中进行计算。由于不同Stage间存在依赖，因此GPU卡间存在明显的交替使用现象，无法充分利用GPU算力。此外，虽然模型划分的粒度有限，但是业务场景产生的海量数据对训练性能的要求越来越高，因此需要增加计算资源并行，以减少模型训练时间，从而提升模型收敛效果。
  针对模型并行存在的问题，Whale提供混合模型并行和流水并行，或混合模型并行、流水并行及数据并行的并行策略，以提高GPU的利用效率，从而解决分布式扩展问题。
- 定义
  一种辅助性并行策略，通常与模型并行混合使用，是指不同设备（GPU和CPU等）接收不同批次数据（包括原始训练数据、Activation及Error）作为输入，同时计算设备对应的网络层部分。
- 实现逻辑
  不同场景的模型可能适合不同的分布式并行策略。以BertLarge模型为例，不管在何种带宽和硬件拓扑下，每一层的Activation、显存及Flops计算都几乎一致，进行模型分片更易受Load Balance（包括显存均衡度和算力均衡度）影响。通过混合模型并行、流水并行及数据并行的混合策略，可以优化分布式BertLarge的性能，具体实现逻辑如下：

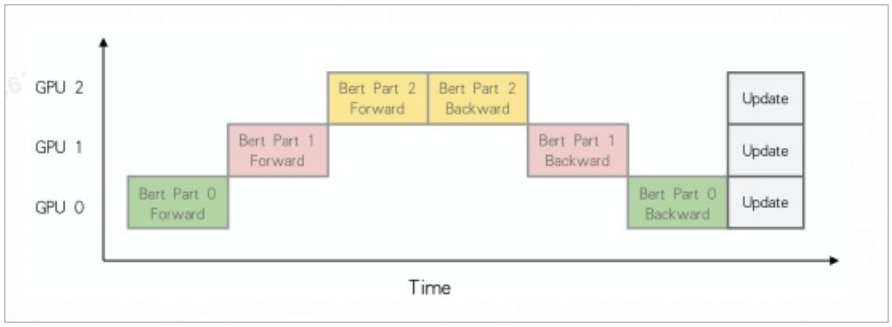
  i. 模型并行

针对BertLarge模型过大的问题，可以使用Layer间的拆分策略将BertLarge分配至不同的GPU卡中进行计算，以降低每张GPU卡上的模型显存占用，从而提高Batch Size。同时使用Activation通信替代梯度同步，大幅度降低卡间通信量。模型拆分如下图所示。

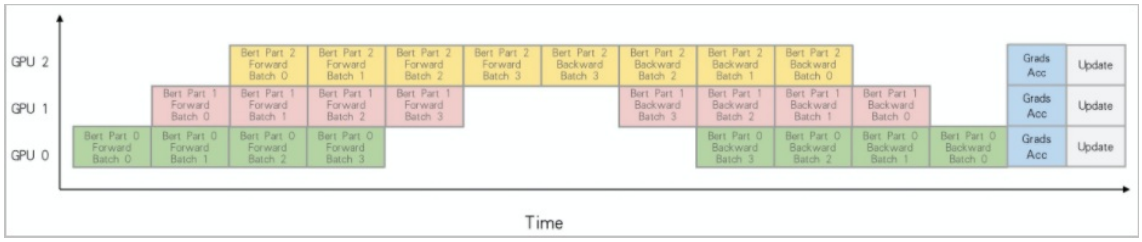


ii. 流水并行

如果仅使用Layer间拆分的模型并行，则当其中一张GPU卡计算时，其他GPU卡会闲置等待，即同一时间段只使用了一张GPU卡，如下图所示。

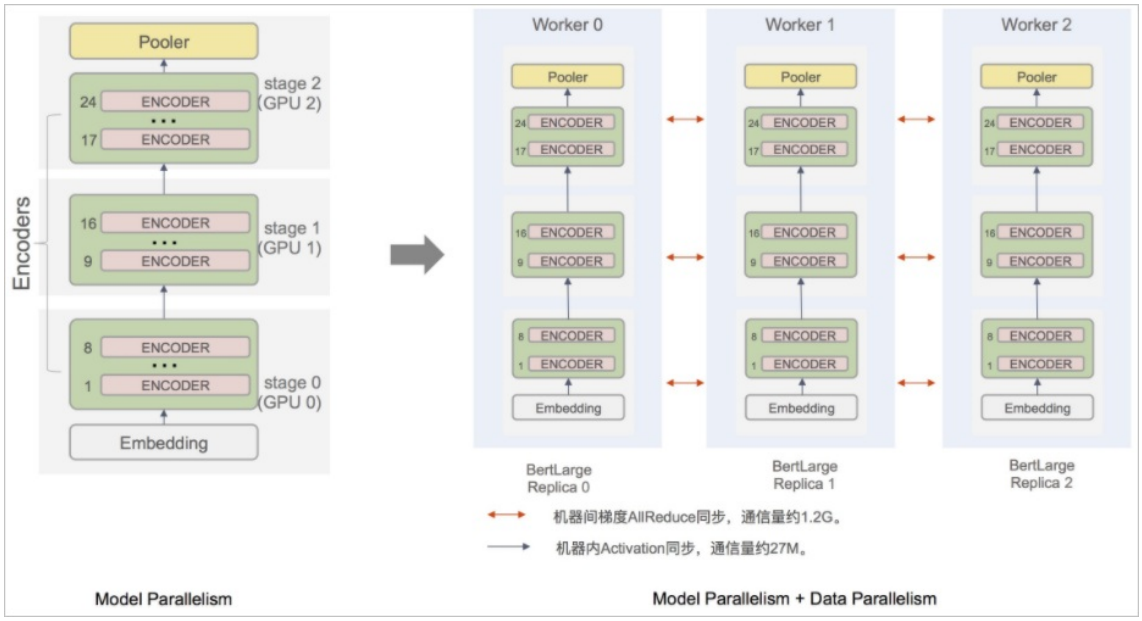


因此，需要增加流水并行以提高其他GPU卡的利用率，其原理是每张GPU卡训练完一个Batch数据的当前部分后，立刻训练下一个Batch数据，执行逻辑如下图（该图为示例图，实际生产中会优化执行调度逻辑）所示。



iii. 数据并行

由于模型并行不可能将模型拆分为无限份，拆分数量过大也会影响性能，因此如何利用更多的GPU卡进行训练加速就成为另外一个问题。您可以引入数据并行解决大数据规模问题，从而利用更大规模的GPU卡数进行训练。为了能够高效利用GPU间的NVLink（如果没有NVLink，则利用PCI-e）带宽，降低梯度同步开销，需要将模型并行的拆分放至不同服务器中进行，数据并行部分的梯度同步（通信量约为1.2 GB）放至服务器内进行，Layer间切分产生的Activation通信（约27 MB）通过TCP网络通信。假设每台服务器有4张GPU卡，将BertLarge进行4个副本的数据并行，模型拆分的逻辑如下图所示。



至此，通过Whale灵活易用的接口，便完成了模型并行、流水并行及数据并行的混合并行策略，以此对BertLarge进行训练加速。

- Whale中实现流水并行
  您需要配置Cluster Layout，并将模型的不同部分放至Stage Scope，即可在Whale中实现模型并行。以BertLarge为例，单卡模型及流水并行的示例代码片段如下（BertLarge的流水并行可执行代码请参见train_bert_model.py）：
  - 单卡模型

```
embedding_output = embedding(inputs)
encoder_output = encoder(embedding_output)
pooler_output = pooler(encoder_output)
```

  - 流水并行

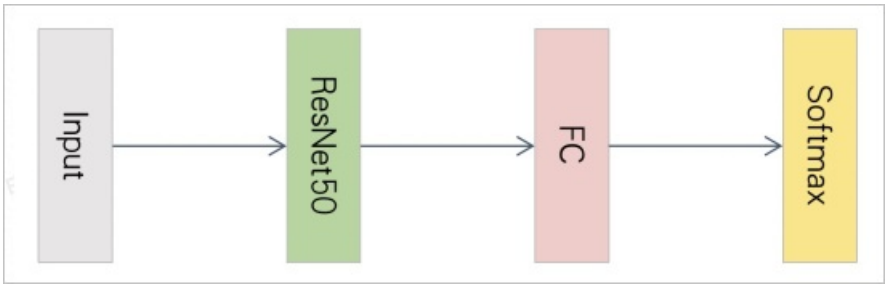
```
import whale as wh
# 假设申请1*3的GPU规模。
cluster = wh.cluster()
with cluster:
    with wh.pipeline(num_micro_batch=5):
        with wh.stage(): # stage 0
            embedding_output = embedding(inputs)
            encoder_output = encoder_layer_1_8(embedding_output)
        with wh.stage(): # stage 1
            encoder_output = encoder_layer_9_16(embedding_output)
        with wh.stage(): # stage 2
            encoder_output = encoder_layer_17_24(embedding_output)
            pooler_output = pooler(encoder_output)
```

- 流水并行搭配数据并行

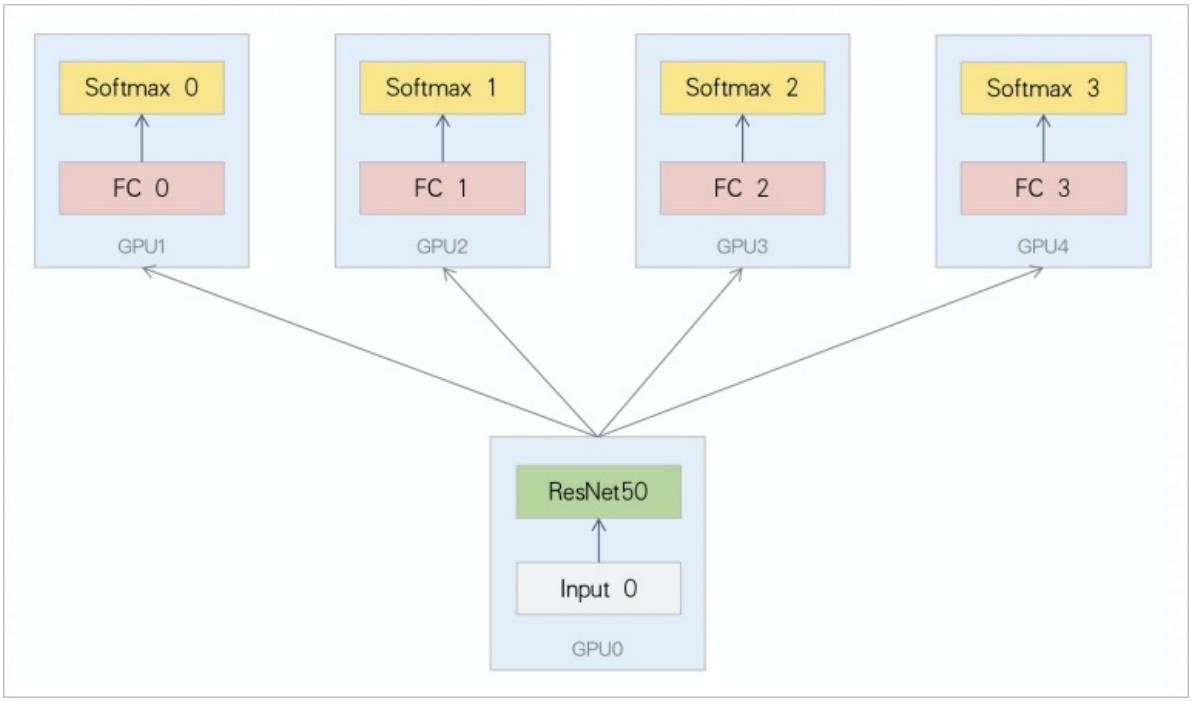
```
import whale as wh
# 假设申请3*4的GPU规模。
# 每个Stage进行4份副本的数据并行。
cluster = wh.cluster(layout={"average": 4}) # 也可使用cluster = wh.cluster(layout="row")
。
with cluster:
    with wh.replica():
        with wh.pipeline(num_micro_batch=5):
            with wh.stage(): # stage 0
                embedding_output = embedding(inputs)
                encoder_output = encoder_layer_1_8(embedding_output)
            with wh.stage(): # stage 1
                encoder_output = encoder_layer_9_16(embedding_output)
            with wh.stage(): # stage 2
                encoder_output = encoder_layer_17_24(embedding_output)
                pooler_output = pooler(encoder_output)
```

算子拆分

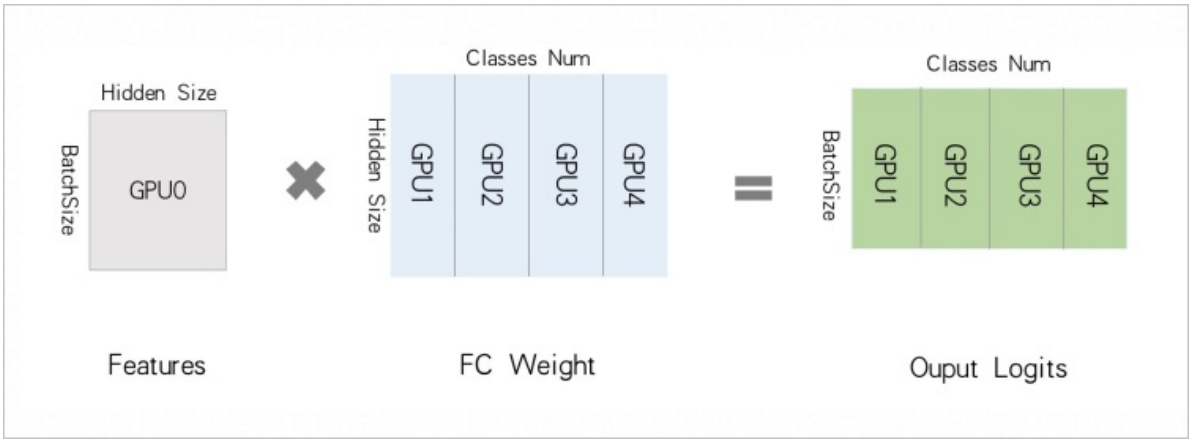
- 背景信息
  针对日益复杂的深度学习模型，通过数据并行进行任务训练仍然存在分布式性能差、单卡显存无法存储等问题。因此需要通过模型算子拆分进行分布式加速优化。算子拆分适用于模型本身或部分规模较大，无法使用数据并行进行训练的场景。例如以下类似场景：
  - 大规模分类任务场景
  - NLP领域的Bert Large、GPT 2及T 5等参数规模超大的场景。
  - 搜索推荐等Sparse模型，需要对稀疏特征进行分片的场景（Variable拆分在Whale中也视为一种算子拆分问题）。
- 定义
  一种并行策略，将算子的存储部分和计算部分拆分到不同计算设备（GPU和CPU等）上进行存储和计算。
- 实现逻辑
  不同算子拆分的实现逻辑各不相同，在此简单介绍ResNet 50分类任务中的FC节点的拆分实现。ResNet 50模型结构如下图所示。



分类任务中，当分类数较大时，Fully Connected（FC）部分权重数据并行的通信性能较差。分类数更多时，GPU显存无法存放过大的模型权重，此时需要通过算子拆分进行分布式优化。分类数通过算子拆之后的计算图如下图所示。



FC部分会将Weight权重按照列进行分割（如下图所示），ResNet50的输出会发送到每个FC分片卡上，每个分片只计算分配到自己的相应部分。得到输出的Logits之后，分别计算Softmax部分，但是需要跨卡通信以获得全局最大Logits信息和每行的Sum值。实现分类任务分布式计算的更多信息请参见“大规模分类分布式训练-算子拆分”。



- Whale中实现算子拆分
  您需要配置Cluster Layout，并将需要进行算子拆分的部分代码放至Split Scope下，即可在Whale中实现算子拆分。以ResNet50为例，算子拆分的代码片段如下（算子拆分的大规模分类任务可执行代码请参见large_scale_classification.py）。

```
import whale as wh
cluster = wh.cluster()
with cluster:
    with wh.replica():
        features = ResNet50(inputs)
    with wh.split():
        logits = FC(features)
        predictions = Softmax(logits)
```

5.2.3.5. 超参配置

5.2.3.5.1. 通信参数

Whale支持配置通信参数，以开启多种优化功能。本文介绍配置通信参数的方法及Whale支持的通信参数。

配置方法

Whale通信支持多种优化，且支持通过环境变量配置参数。您可以根据需求，在训练代码或执行脚本中配置参数，从而开启优化功能：

- 在训练代码中配置
  可以通过 `os.environ` 配置环境变量。例如，您可以使用如下命令，开启Hierarchical AllReduce通信（该通信参数为WHALE_COMMUNICATION_HIERARCHICAL_ALL_REDUCE，详情请参见下面的通信参数列表部分）。

  ```
  import os
  os.environ["WHALE_COMMUNICATION_HIERARCHICAL_ALL_REDUCE"]="True"
  ```

  如果使用PAI-Studio提交任务，在训练代码中未配置通信参数，则可以在训练时通过PAI-TensorFlow超参传入（详情请参见PAI-TF超参支持），再使用上述代码将参数赋值给环境变量。
- 在执行脚本中配置
  如果在PAI-DLC环境或手动开启训练，有Launch脚本，则可以在脚本中，配置环境变量：
  - 在执行脚本最开始，配置环境变量

    ```
    export WHALE_COMMUNICATION_HIERARCHICAL_ALL_REDUCE=True
    ...
    python train.py
    ```

    其中 `train.py` 表示训练文件。
  - 在加载执行命令的语句前，配置环境变量。

    ```
    WHALE_COMMUNICATION_HIERARCHICAL_ALL_REDUCE=True python train.py
    ```

    其中 `train.py` 表示训练文件。

通信参数列表

Whale通信支持的参数如下表所示。

参数	描述	类型	默认值	备注
WHALE_COMMUNICATION_SPARSE_AS_DENSE	梯度同步时，是否将Sparse梯度转成Dense进行通信。取值范围如下： • False：未开启该功能，Sparse梯度通过AllGather进行通信。 • True：将Sparse梯度转成Dense进行通信。	BOOL	False	当Sparse梯度的Dense Shape不大，即转成Dense梯度后，大小不会放大很多，则可以开启该功能。例如，BertLarge模型的Sparse梯度，转成Dense后不大，可以开启该功能。
WHALE_COMMUNICATION_NUM_SPLITS	梯度通信时，梯度融合的分组数。	INT	5	Whale会自动调整分组数，通常无需手动配置。

WHALE_COMMUNICATION_NUM_COMMUNICATORS	梯度通信时，进行并行通信的Communicator数量。取值为None，表示创建与梯度分组数相同数量的Communicator进行并行通信。	INT	None	Communicator数量越多，则并发越多，同时也会越消耗显存。如果模型显存紧张（例如出现Allocated Memory告警），可以配置该参数为2或1。
WHALE_COMMUNICATION_HIERARCHICAL_ALL_REDUCE	对于Dense梯度，是否采用Hierarchical AllReduce通信。取值范围如下： • False：未开启该功能。 • True：开启该功能，即采用Hierarchical AllReduce通信。	BOOL	False	如果服务器内或服务器间带宽差距大（例如服务器内有NVLink，服务器间25 GB网络），且Dense梯度很大，则可以开启该功能，进行通信加速。
WHALE_COMMUNICATION_HIERARCHICAL_ALL_GATHER	对于Sparse梯度，是否采用Hierarchical AllGather通信。取值范围如下： • False：未开启该功能。 • True：开启该功能，即采用Hierarchical AllGather通信。	BOOL	False	如果服务器内或服务器间带宽差距大，且Sparse梯度很大，则可以开启该功能，进行通信加速。
WHALE_COMMUNICATION_DENSE_FP16	对于Dense梯度，是否采用半精度通信。取值范围如下： • False：未开启该功能。 • True：开启该功能，即采用半精度通信。	BOOL	False	开启该功能，会减少一半的Dense梯度通信，但是会影响收敛，可能需要调参。
WHALE_COMMUNICATION_SPARSE_FP16	对于Sparse梯度，是否采用半精度通信。取值范围如下： • False：未开启该功能。 • True：开启该功能，即采用半精度通信。	BOOL	False	开启该功能，会减少一半的Sparse梯度通信，但是会影响收敛，可能需要调参。
WHALE_COMMUNICATION_FP16_SCALE	采用半精度通信时，为防止梯度消失，可以配置梯度Scale系数。取值为None，表示未设置Scale系数。	FLOAT	None	采样半精度通信时，可以结合该参数进行调参。

5.2.3.5.2. 训练数据分片

Whale根据分布式模式和用户配置的资源，自动对训练数据进行分片。如果训练数据无法均分，您可以通过环境变量配置数据分片策略。本文介绍Whale支持的数据分片策略、数据分片策略的参数列表及数据分片策略的配置方法。

数据分片策略

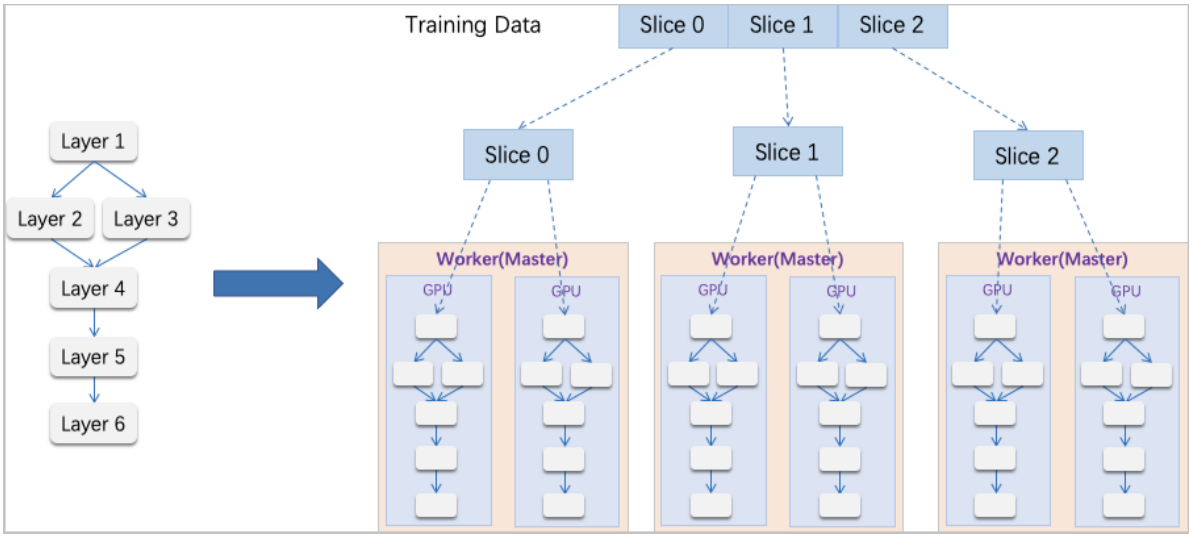
分布式训练时，每个Worker需要获取训练数据的不同分片。如果每个Worker处理的数据完全一样，则训练效果等同于单机，无法发挥分布式加速效果。Whale根据分布式模式和用户配置的资源，自动对训练数据进行分片。每个Worker依据分布式模式分配不同的角色，包括Master和Slave。Master负责构图和驱动执行，Slave被动执行分配到的任务。

Whale的训练数据分片策略如下：

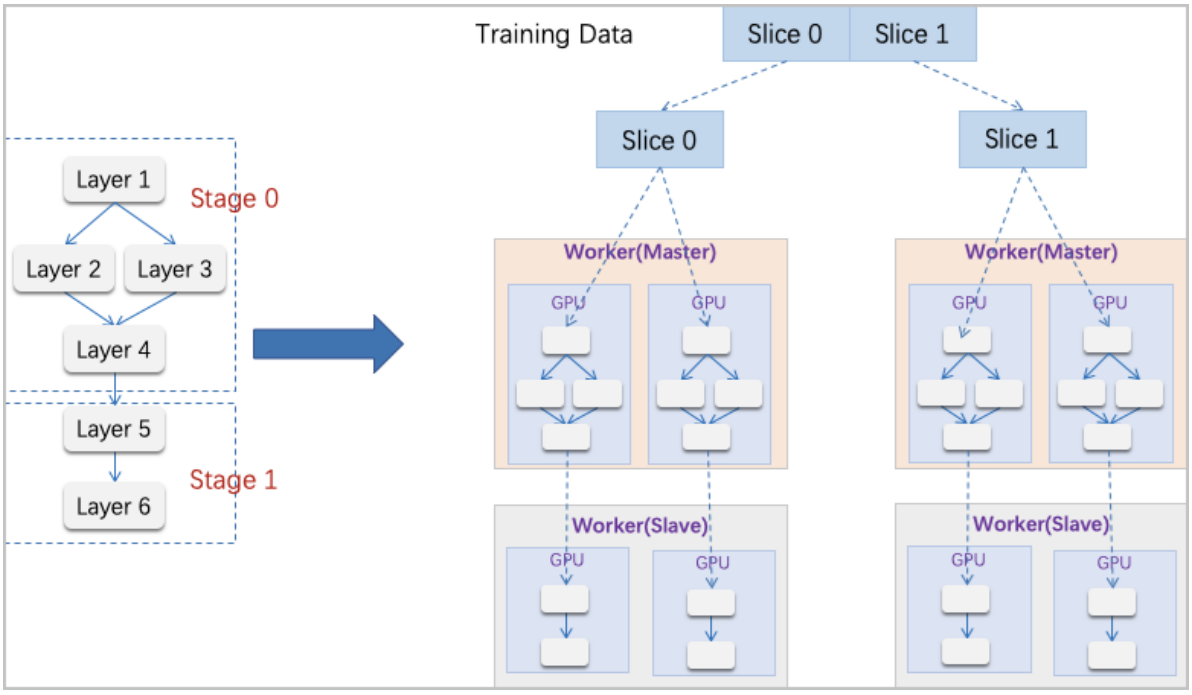
1. 根据分布式模式（例如数据并行、模型并行、流水并行或算子拆分）和资源，确定每个Worker的角色。
2. 根据Master Worker切分训练数据。

数据分片场景实例

- 数据并行
  在数据并行场景中，每个Worker都是Master角色，Whale按照Master Worker数量均匀切分训练数据。如下图所示，共3个Master Worker，Whale会将训练数据的文件列表均分为3份，每个Worker获取一个分片。Worker内的多个GPU按照Batch依次获取数据。



- 混合数据并行和模型并行
  在混合数据并行和模型并行的场景，模型会切分为多个Stage分配至不同Driver。如下图所示，模型分成2个Stages，Master Worker执行Stage 0，Slave Worker执行Stage 1。（4个Worker中有2个Master角色，2个Slave角色。）
  Whale根据Master Worker数量均匀切分训练数据。如下图所示，有2个Master Worker，即将训练数据的文件列表均分为2份，每个MasterWorker获取一个分片，Worker内的多个GPU按照Batch依次获取数据。



- 训练数据无法均分
  当训练数据的文件数目不能整除Master Worker数量时，Whale提供三种处理方式：
  - 不切分，进行随机Shuffle。
    每个Master Worker获取完整的文件列表，不切分训练数据。为保证收敛，对文件列表进行随机Shuffle。

    > **说明**
    > - 训练数据无法均分时，Whale默认采用该处理方式。
    > - 当文件数量小于Master Worker数量时，Whale只支持该处理方式。

  - 不均匀切分，为部分Master Worker多分配一个文件。
    尽可能均匀切分，容许部分Master Worker比其他的多分配一个文件。
  - 均匀切分，多余的文件直接丢弃。
    建议仅在文件数很多，且丢弃的文件不影响收敛时，使用该方式。

数据分片的参数列表

Whale支持的数据分片策略参数如下表所示，如果训练数据的文件数目不能整除Master Worker数量，您可以使用如下参数配置数据分片策略。

参数	描述	类型	默认值	备注

参数	描述	类型	默认值	备注
WHALE_UNBALANCED_IO_SLICING	是否允许切分数据时，每个Worker分配的文件数目不同，即为部分Worker多分配一个文件。取值范围如下： • False：不切分训练数据。 • True：允许非均等切分训练数据，即为部分Worker多分配一个文件。	BOOL	False	由于部分Worker会分到更多的训练数据文件，因此需要注意分布式训练的结束条件。如果以Epoch为结束条件，则训练可能挂起。建议使用StopAtStepHook控制模型训练是否结束。
WHALE_DROP_LAST_FILES	当训练数据的文件数目不能整除Master Worker数量时，是否依然均匀切分训练数据，将多余文件舍弃。该参数的优先级低于WHALE_UNBALANCED_IO_SLICING参数，如果这两个参数同时为True，则执行非均等切分策略。取值范围如下： • False：不切分训练数据。 • True：均匀切分训练数据，将多余文件舍弃。	BOOL	False	建议仅在训练数据文件数量远大于训练规模，且丢弃小部分数据不影响收敛时，才将该参数配置为True。

数据分片策略配置

当训练数据的文件数目不能整除Master Worker数量时，Whale支持通过环境变量配置数据分片策略。您可以根据需求，在训练代码或执行脚本中配置参数：

- 在训练代码中配置
  可以通过 `os.environ` 配置环境变量。例如，您可以使用如下命令，配置非均等切分（该参数为WHALE_UNBALANCED_IO_SLICING，详情请参见上面的数据分片参数列表部分）。

  ```
  import os
  os.environ["WHALE_UNBALANCED_IO_SLICING"]="True"
  ```

- 在执行脚本中配置
  如果在PAI-DLC环境或手动开启训练，有Launch脚本，则可以在脚本中，配置环境变量：
  - 在执行脚本最开始，配置环境变量

    ```
    export WHALE_UNBALANCED_IO_SLICING=True
    ...
    python train.py
    ```

    其中 `train.py` 表示训练文件。

- 在加载执行命令的语句前，配置环境变量。

```
WHALE_UNBALANCED_IO_SLICING=True python train.py
```

其中 `train.py` 表示训练文件。

5.2.4. 训练任务编排

5.2.4.1. 概述

PAI Pipeline Service是PAI提供的机器学习工作流服务，您可以通过构建工作流的方式进行训练任务编排。

PAI Pipeline Service简介

为了获得合适的机器学习模型，将其应用于在线推理或离线推理，通常机器学习工程师需要进行数据提取、数据校验、特征工程、模型训练及模型评估全流程，且以上流程并非只进行一次。为了获得更符合业务需求的模型，机器学习工程师需要尝试不同的特征和参数，或使用增量的数据训练模型，以更好地拟合最新的输入特征。为了复用这些流程，通常的解决方案是先将这些流程以工作流（Pipeline）的形式组织起来，再提交至一个自动化工作流服务中运行。

PAI Pipeline Service是PAI提供的机器学习工作流服务，您可以通过可视化建模（PAI-Studio 2.0）拖拉拽的方式或Python SDK的方式构建一个机器学习工作流，并提交至PAI Pipeline Service运行。关于如何通过Python SDK的方式构建工作流，请参见通过Python SDK调用PAI Pipeline Service。

构建工作流时，您可以使用PAI提供的组件进行数据处理、特征工程、模型训练及离线推理。此外，如果您需要自己定义工作流中的节点功能，则可以通过构建自定义组件实现相关功能。例如，训练完成后调用自己后端API进行消息通知，或使用PyODPS处理MaxCompute上的数据表。关于如何构建自定义组件，请参见通过Python SDK使用自定义组件。

组件运行原理

PAI Pipeline Service根据工作流的定义运行工作流，工作流中展开的节点，对应一个运行容器。对于工作流中的一个节点，PAI Pipeline Service会完成以下工作：

- 搬运上游节点的输出数据（Artifact），准备将其加载到下游节点的运行容器中。
- 使用组件定义的执行命令和镜像，注入节点的运行参数和环境变量等信息，运行容器。
- 读取运行容器的输出文件，保存节点的输出数据。

组件运行依赖于容器，关于容器和镜像相关的知识请参见Docker docs。

5.2.4.2. 通过Python SDK使用自定义组件

PAI Pipeline Service中可以使用PAI提供的组件（Operator）处理数据和训练模型，也可以使用自定义组件处理具体任务。本文介绍如何在PAI Pipeline Service中，通过Python SDK构建并使用自定义组件。

前提条件

- 已获取阿里云账号的鉴权AccessKey ID和AccessKey Secret，详情请参见获取AccessKey。
- 已创建AI工作空间，详情请参见AI工作空间（旧版）。
- 已创建OSS Bucket，详情请参见创建存储空间。

背景信息

在PAI Pipeline Service中，您可以使用PAI提供的组件进行数据处理、特征工程、模型训练及离线推理。如果需要自己定义工作流中的节点功能，则需要通过构建自定义组件的方式实现相关功能。例如，训练完成后通过调用API通知用户的后端服务，或使用PyODPS处理MaxCompute上的数据表。

构建自定义组件，实际上就是定义组件在PAI Pipeline Service中的运行行为，它主要包括如下两部分信息：

- 组件作为节点输入输出参数或数据的定义，用于引导PAI Pipeline Service完成输入输出数据搬运，同时也作为节点在PAI Pipeline Service中拼接的签名信息 。
- 容器镜像运行定义，包括使用的镜像、镜像仓库访问凭证、镜像的环境变量定义及镜像运行命令。PAI Pipeline Service根据这些信息运行对应的镜像。

Python SDK提供了ScriptOperator（推荐）和ContainerOperator两种方式，帮助您快速构建自定义组件，详情请参见构建自定义组件：ScriptOperator（推荐）和构建自定义组件：ContainerOperator。
使用自定义组件构建工作流时，您需要先获取组件运行的上下文信息，再构建工作流：

1. 获取上下文信息（pai_running_utils）
2. 构建工作流

准备工作

保存或运行自定义组件时，SDK需要与PAI的后端服务进行交互，且交互依赖于您初始化的全局会话Session。因此，您需要安装SDK、初始化全局Session和AI工作空间。

1. 安装SDK。

   如果您通过本地的Python开发环境使用SDK，则需要安装SDK。如果您通过PAI-DSW环境使用SDK，该环境中已经预装了SDK，您可以跳过该步骤。通过 `pip` 命令安装SDK的命令如下。

   ```
   pip install https://pai-sdk.oss-cn-shanghai.aliyuncs.com/alipai/dist/alipai-0.3.0-py2.py3-none-any.whl
   ```

   其中https://pai-sdk.oss-cn-shanghai.aliyuncs.com/alipai/dist/alipai-0.3.0-py2.py3-none-any.whl表示SDK的下载地址，您无需修改。

   > **说明** 虽然SDK支持Python 2和Python 3版本，但是Python社区已经停止维护Python 2版本，因此推荐您使用Python 3环境运行SDK。

2. 初始化默认的SDK Session和AI工作空间。

   PAI Pipeline Service SDK依赖于阿里云机器学习PAI提供的服务，SDK的Session负责与PAI的后端服务和依赖的其他阿里云服务进行交互。Session封装了鉴权凭证AccessKey、使用PAI服务的地域及当前使用的AI工作空间。
   您可以通过 `pai.core.session.setup_default_session` 方法初始化一个全局默认的Session对象。当调用的API（例如 `SaveOperator.list` 和 `Workspace.list` ）需要与PAI后端服务进行通讯时，默认使用该Session进行通讯。
   初始化默认的SDK Session和AI工作空间时，您可以通过以下两种方式指定AI工作空间：

   - 方式一：首先在PAI控制台上查看AI工作空间的名称或ID，然后初始化默认Session的同时指定AI工作空间。
     Python SDK提供了ScriptOperator（推荐）和ContainerOperator两种方式构建自定义组件，这两种方式在初始化默认的SDK Session和AI工作空间时，您需要注意以下差异：

- 如果使用ScriptOperator构建自定义组件，则需要使用OSS存储用户代码，因此初始化Session时，需要额外提供OSS信息。代码示例如下。

```
from pai.core.session import setup_default_session
from pai.common import ProviderAlibabaPAI
# 初始化默认的全局session和AI工作空间，同时提供OSS信息。
session = setup_default_session(access_key_id="<your_access_key_id>",
                                access_key_secret="<your_access_key_secret>",
                                region_id="<region_id>",
                                oss_bucket_name="<your_oss_bucket_name>",
                                oss_bucket_endpoint="<your_oss_bucket_endpoint>",
                                # workspace_id="<your_workspace_id>",          # AI工作空间名称和ID二选一。
                                workspace_name="<your_workspace_name>")
```

参数详情请参见下文初始化的相关参数，您需要将参数值替换为实际值。

- 如果使用ContainerOperator构建自定义组件，则无需使用OSS。示例代码如下。

```
from pai.core.session import setup_default_session
from pai.core.workspace import Workspace
setup_default_session(access_key_id="<your_access_key_id>",
                      access_key_secret="<your_access_key_secret>",
                      region_id="<region_id>",
                      # workspace_id="<your_workspace_id>",          # AI工作空间名称和ID二选一。
                      workspace_name="<your_workspace_name>")
```

参数详情请参见下文初始化的相关参数，您需要将参数值替换为实际值。

- 方式二：首先设置默认的Session，然后获取阿里云账号下可访问的AI工作空间列表，再指定使用的AI工作空间。
  Python SDK提供了ScriptOperator（推荐）和ContainerOperator两种方式构建自定义组件，这两种方式在初始化默认的SDK Session和AI工作空间时，您需要注意以下差异：

  - 如果使用ScriptOperator构建自定义组件，则在初始化Session时，需要额外提供OSS信息。代码示例如下。

```
from pai.core.session import setup_default_session
from pai.core.workspace import Workspace
session = setup_default_session(access_key_id="<your_access_key_id>",
                                access_key_secret="<your_access_key_secret>",
                                region_id="<region_id>",
                                oss_bucket_name="<your_oss_bucket_name>",
                                oss_bucket_endpoint="<your_oss_bucket_endpoint>")
for ws in Workspace.list():
    print(ws.name, ws.id)
session.set_workspace(workspace=Workspace.get_by_name("<your_workspace_name>"))
```

参数详情请参见下文初始化的相关参数，您需要将参数值替换为实际值。

- 如果使用ContainerOperator构建自定义组件，则无需使用OSS。示例代码如下。

```
from pai.core.session import setup_default_session
from pai.core.workspace import Workspace
session = setup_default_session(access_key_id="<your_access_key_id>", access_key_se
cret="<your_access_key_secret>",
                                region_id="<region_id>")
for ws in Workspace.list():
    print(ws.name, ws.id)
session.set_workspace(workspace=Workspace.get_by_name("<your_workspace_name>"))
```

参数详情请参见下文初始化的相关参数，您需要将参数值替换为实际值。

初始化的相关参数

参数	描述
<your_access_key_id>	阿里云账号的AccessKey ID。
<your_access_key_secret>	阿里云账号的AccessKey Secret。
<region_id>	PAI Pipeline Service的地域，后续提交的任务将运行在该地域。该参数支持以下取值： ○ cn-shanghai：华东2（上海） ○ cn-hangzhou：华东1（杭州） ○ cn-beijing：华北2（北京） ○ cn-shenzhen：华南1（深圳）
<your_oss_bucket_name>	仅使用ScriptOperator构建自定义组件时提供，表示OSS Bucket的名称。
<your_oss_bucket_endpoint>	仅使用ScriptOperator构建自定义组件时提供，表示OSS Bucket所在的地域，详情请参见访问域名和数据中心。
<your_workspace_name>或<your_workspace_id>	AI工作空间名称或ID。 对于上面两种指定AI工作空间的方式，分别根据以下方法获取该参数值： ○ 如果使用方式一，则<your_workspace_name>与<your_workspace_id>二选一。您可以登录PAI控制台，在**AI工作空间列表**页面查看参数值。您也可以创建新的AI工作空间，详情请参见AI工作空间（旧版）。 **说明** 如果指定AI工作空间时，同时指定了AI工作空间的名称和ID，则接口报错。 ○ 如果使用方式二，则将<your_workspace_name>指定为 `Workspace.list()` 接口返回的一个ws.name。

构建自定义组件：ScriptOperator（推荐）

通过SDK的ScriptOperator，您只需要定义组件的输入输出信息和镜像内执行的Python脚本，既可完成组件的定义，极大简化了自定义一个组件的成本。以下示例使用ScriptOperator构建了一个自定义组件，该组件对应的容器内会运行定义的entry_point。

1. 定义组件的输入和输出。

> **注意** 调用以下代码之前，需要先设置存储代码的OSS Bucket，详情请参见准备工作。

```
import yaml
from pai.operator import ScriptOperator
from pai.pipeline.types import PipelineParameter
op = ScriptOperator(
    entry_point="main.py",
    # script_dir目录下的文件会被一起打包上传至OSS。
    script_dir="scripts",
    inputs=[
        PipelineParameter(name="foo", default=10),
        PipelineParameter(name="bar", default=10),
    ],
    outputs=[],
)
# 直接运行对应的组件。
op.run(
    job_name="exampleScript",
    arguments={
        "foo": "ThisIsFoo",
        "bar": "BAR",
    }
)
# 保存组件(组件的identifier和version不能冲突)。
op.save(identifier="simpleExample", version="v1")
# 查看组件的定义信息。
print(yaml.dump(op.to_dict()))
```

2. 定义镜像内执行的Python脚本entry_point。

   该示例中entry_point的取值为main.py，即组件对应的容器中运行main.py文件。main.py文件的内容如下所示，在容器内通过 `python -m main --foo ThisIsFoo --bar BAR` 命令调用。

```
import argparse
def main():
    parser = argparse.ArgumentParser("ScriptOperator arguments parser")
    parser.add_argument("--foo")
    parser.add_argument("--bar")
    args, _ = parser.parse_known_args()
    print("Arguments foo is ", args.foo)
    print("Arguments bar is ", args.bar)
if __name__ == "__main__":
    main()
```

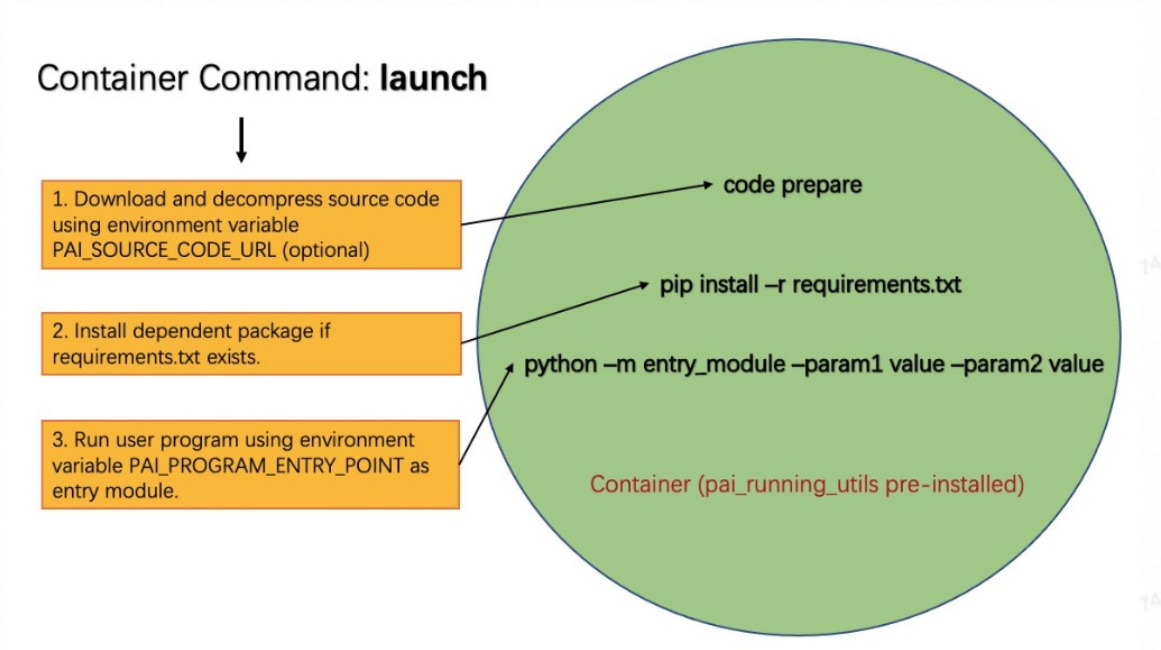
   ScriptOperator将script_dir目录下的文件打包上传到OSS中，将对应的OSS URL和运行脚本entry_point作为对应容器的环境变量，并在组件的描述文件中进行定义，默认使用 `launch` 作为镜像的启动命令。

   容器内的 `launch` 命令是预先安装在默认镜像中的（安装pai_running_utils时默认安装的命令行脚本），它主要完成以下工作（如下图所示）：

   i. 使用环境变量PAI_SOURCE_CODE_URL获取代码，并将其解压到*/work/code*目录。如果相关的脚本已经在对应的镜像内，则无需使用文件下载机制准备相关的脚本，即可以跳过该步骤。

ii. 如果对应的代码包中包括requirements.txt文件，则使用 `pip` 命令安装依赖的三方库。

iii. 根据环境变量PAI_PROGRAM_ENTRY_POINT运行对应的代码。



构建自定义组件：ContainerOperator

使用SDK中的ContainerOperator，您可以构建一个基于容器的自定义组件。以下示例构建了一个自定义组件，该组件实现了打印运行容器环境变量的功能。

1. 构造ContainerOperator对象，并定义组件的输入和输出信息。

```
import yaml
from pai.operator.container import ContainerOperator
from pai.pipeline.types import PipelineParameter
container_op = ContainerOperator(
    image_uri="python:3",
    inputs=[
        PipelineParameter(name="foo"),
        PipelineParameter(name="bar"),
    ],
    outputs=[],
    command=[
        "python",
        "-c",
        "import os; print('\\n'.join(['%s=%s' % (k, v) for k, v in os.environ.items()])
);",
    ],
    env={"CustomEnvKey": "CustomEnvValue"},
)
```

上述代码中ContainerOperator构造函数涉及的参数详情如下：

- inputs：用于定义组件的输入信息。
- outputs：用于定义组件的输出信息。
- image_uri：使用的容器。

- command：运行命令。
- env：注入容器的环境变量信息。

2. 运行组件。

   为构造的ContainerOperator对象指定输入参数，即可运行该组件，示例代码如下。

```
container_op.run(
    job_name="containerTemplExample",
    arguments={
        "foo": "this is foo",
        "bar": "this is bar",
    },
)
```

3. 如果需要将该组件共享给该阿里云账号下的所有RAM用户或使用它构造新工作流的一个节点，则可以通过 `save` 方法将其保存到PAI Pipeline Service后端中。

```
container_op.save(identifier="containerTemplExample", version="v1")
# 打印YAML文件。
print(yaml.dump(container_op.to_dict()))
```

构建工作流

1. 获取上下文信息（pai_running_utils）。

   如果您通过自定义组件构建工作流，则需要获取该组件的输入信息（包括输入的数据类型和表名等）、读取环境变量和本地文件、并理解对应的输入输出格式，该过程比较复杂。为了帮助您快速地获取组件的上下文信息，PAI Pipeline Service提供了一个简单的Runtime SDK pai_running_utils。您可以使用它获取节点的输入参数和输入Artifact，也可以使用它写出当前节点的输出Artifact。关于Artifact的详细信息，请参见附录：工作流中的输入输出数据（Artifact）。
   Python脚本可以通过 `pai_running_utils.context.Context` 实例获取对应的运行环境上下文信息，主要接口如下：

   - 获取运行节点的输入参数input_parameters，示例如下。

```
from pai_running.context import Context
# 构建context实例，获取当前的上下文。
context = Context()
# 可以通过name作为key在input_parameters中查找输入参数的值。例如获取key为featureColNames的输入参数值。
feature_cols = context.input_parameters["featureColNames"]
# 打印所有输入参数的名称和参数值。
for name, value in context.input_parameters.items():
    print("parameters is", name, value)
```

   - 获取输入和输出数据，即input_artifacts和output_artifacts，示例如下。

```
# 遍历组件输入的Artifacts定义。如果有输入信息，则打印Artifact的名称和路径。
for artifact in context.input_artifacts:
    if artifact:
        print(artifact.name, artifact.path)
# 可以通过名称或index定位Artifact。
context.output_artifacts["outputModel"].write_output(
    {
        "location": {
            "bucket": "test",
            "endpoint": "oss-cn-hangzhou.aliyuncs.com",
            "key": "/pipeline/output/lr_model.xml",
        }
    }
)
```

- 获取节点在PAI Pipeline Service中运行的环境信息，示例如下。

```
# 节点对应组件的inputs和outputs定义信息。
print(context.inputs_spec)
print(context.outputs_spec)
# 节点的运行用户信息。
print(context.user_id)
# 节点运行的工作空间信息。
print(context.workspace_id)
```

2. 构建工作流。

   关于如何构建工作流，请参见构建工作流。

示例：使用ScriptOperator构建自定义组件并获取上下文信息

以下示例，使用ScriptOperator定义了一个组件，它可以从MaxCompute输入表odps://pai_online_project/tables/wumai_data（PAI提供的公共数据表，您可以直接使用）中选择部分列，输出到一张新的MaxCompute表中。具体的实现方法如下。

1. 定义组件的输入输出信息。

```
from pai.operator import ScriptOperator
from pai.pipeline.types import PipelineParameter, PipelineArtifact, MetadataBuilder
op = ScriptOperator(
    entry_point="main.py",
    script_dir="scripts",
    inputs=[
        PipelineParameter(name="destTable", desc="输出的目的表"),
        PipelineParameter(name="execution", typ=dict, desc="max_compute config"),
        PipelineParameter(name="selectColNames", desc="输出的目的表"),
        PipelineArtifact(name="inputTable", metadata=MetadataBuilder.maxc_table()),
    ],
    outputs=[
        PipelineArtifact(name="outputTable", metadata=MetadataBuilder.maxc_table()),
    ])
op.run(
    job_name="example",
    arguments={
        "destTable": "sql_script_dest_table",
        "selectColName": "time,hour,pm2,pm10",
        "execution": {
            "project": "{{test_project_name}}",
            "endpoint": "{{max_compute_project_endpoint}}",
        },
        "inputTable":  "odps://pai_online_project/tables/wumai_data",
    }
)
```

上述代码定义的组件共三个输入参数（PipelineParameter）和一个输入的MaxCompute表（PipelineArtifact）。三个输入参数分别为输出的目标MaxCompute表名（destTable）、选择的列信息（selectColNames）及执行任务的MaxCompute引擎配置信息（execution）。

2. 定义镜像内执行的Python脚本。

```
import json
from pai_running.context import Context
from odps import ODPS
def main():
    # 获取当前节点的运行相关信息。
    context = Context()
    # 节点的运行输入参数(parameters)。
    parameters = context.input_parameters
    maxc_config = json.loads(parameters["execution"])
    odps = ODPS(
        access_id=maxc_config["accessKeyId"],
        secret_access_key=maxc_config["accessKeySecret"],
        project=maxc_config["project"],
        endpoint=maxc_config["endpoint"],
    )
    # 节点的输入Artifacts。
    input_table = context.input_artifacts[0].get_table()
    col_names = parameters["selectColNames"]
    output_table = parameters["outputTableName"]
    sql = "create table {0} select {1} from {2}".format(
        output_table,
        col_names,
        input_table,
    )
    run_instance = odps.run_sql(sql)
    run_instance.wait_for_success()
    context.output_artifacts["outputTable"].write_output(
        {
            "location":{
                "project": maxc_config["project"],
                "endpoint": maxc_config["endpoint"],
                "table": "exampleOutputTable",
            }
        }
    )
if __name__ == "__main__":
    main()
```

该脚本首先通过pai_running_utils的 `Context()` 接口获取输入参数和输入数据信息，包括MaxCompute的运行配置、输入表及选择的列等信息，然后调用PyODPS运行构造的SQL。任务运行成功后，输出组件的Artifact信息（MaxComputeLocationArtifact）。关于如何快速获取输入参数和输入数据，请参见上文的获取上下文信息（pai_running_utils）。

附录：工作流中的输入输出数据（Artifact）

Artifact是组件的输入输出数据。组件在工作流中运行时，PAI Pipeline Service会在镜像启动前将上游组件输出的Artifact搬运到当前组件对应的容器文件系统中。默认组件的工作目录示例如下所示。

```
/work/
|── code                          # 下载的依赖代码所在目录。
|── inputs                        # 输入的相关信息所在的文件夹。
|   |── artifacts
|   |     └── inputDataSet
|   |           └── data
|   └── parameters
├── outputs                       # 输出信息所在目录。
|     └── artifacts
|        └── outputDataSet
|              └── data
```

PAI Pipeline Service默认将输入的数据Artifact加载到*/work/inputs/artifacts/<artifactName>/data*路径下，组件运行结束后（容器退出后），从*/work/outputs/artifacts/<artifactName>/data*路径下读取保存组件的输出数据。在上述工作目录示例中，其输入数据<artifactName>为`inputDataSet`，输出数据<artifactName>为`outputDataSet`。
目前PAI Pipeline Service支持的Artifact包括OssArtifact和MaxComputeTableArtifact，格式分别如下：

- OssArtifact
  OssArtifact表示在OSS上存储的数据，格式如下所示。

  ```
  {
    "location": {
      "bucket": "pai-test",
      "endpoint": "oss-cn-shanghai-internal.aliyuncs.com",
      "key": "/paiflow/model_transfer2oss_test/test_health_prediction_by_pipeline_500940.xml"
    },
  }
  ```

  各字段的含义如下：
  - bucket：OSS Bucket名称。
  - endpoint：OSS Endpoint，详情请参见访问域名和数据中心。
  - key：具体的文件路径。

- MaxComputeTableArtifact
  MaxComputeTableArtifact的格式如下所示。

  ```
  {
    "location": {
      "project": "myProject",
      "table": "myTable",
      "partition": "myPartition",
      "endpoint": "myEndpoint",
    }
  }
  ```

  各字段的含义如下：
  - project：MaxCompute的项目名。
  - table：MaxCompute的表名。
  - partition：分区信息，例如name1=value。如果存在多级分区，则使用正斜线（/）分隔，例如name1=value1/name2=value2。

- endpoint：MaxCompute项目的Endpoint，详情请参见Endpoint。

5.2.4.3. 通过Python SDK调用PAI Pipeline Service

PAI不仅支持通过可视化建模（PAI-Studio 2.0）拖拉拽的形式构建数据处理、数据验证及模型训练的机器学习工作流，而且支持使用Python SDK构建相同的工作流，并在PAI Pipeline Service中运行。本文介绍如何使用Python SDK构建一个机器学习工作流，并使其运行在PAI Pipeline Service中。

背景信息

为了获得合适的机器学习模型，将其应用于在线推理或离线推理，通常机器学习工程师需要进行数据提取、数据校验、特征工程、模型训练及模型评估全流程，且以上流程并非只进行一次。为了获得更符合业务需求的模型，机器学习工程师需要尝试不同的特征和参数，或使用增量的数据训练模型，以更好地拟合最新的输入特征。为了复用这些流程，通常的解决方案是先将这些流程以工作流（Pipeline）的形式组织起来，再提交至一个自动化工作流服务中运行。

PAI Pipeline Service是PAI提供的机器学习工作流服务，您可以通过可视化建模（PAI-Studio 2.0）拖拉拽的方式或Python SDK的方式构建一个机器学习工作流，并提交至PAI Pipeline Service运行。

准备工作

1. 安装SDK。

   如果您通过本地的Python开发环境使用SDK，则需要安装SDK。如果您通过PAI-DSW环境使用SDK，该环境中已经预装了SDK，您可以跳过该步骤。通过 `pip` 命令安装SDK的命令如下。

   ```
   pip install https://pai-sdk.oss-cn-shanghai.aliyuncs.com/alipai/dist/alipai-0.3.0-py2.py3-none-any.whl
   ```

   其中https://pai-sdk.oss-cn-shanghai.aliyuncs.com/alipai/dist/alipai-0.3.0-py2.py3-none-any.whl表示SDK的下载地址，您无需修改。

   > **说明** 虽然SDK支持Python 2和Python 3版本，但是Python社区已经停止维护Python 2版本，因此推荐您使用Python 3环境运行SDK。

2. 初始化默认的SDK Session和AI工作空间。

   PAI Pipeline Service SDK依赖于阿里云机器学习PAI提供的服务，SDK的Session负责与PAI的后端服务和依赖的其他阿里云服务进行交互。Session封装了鉴权凭证AccessKey、使用PAI服务的地域及当前使用的AI工作空间。
   您可以通过 `pai.core.session.setup_default_session` 方法初始化一个全局默认的Session对象。当调用的API（例如 `SaveOperator.list` 和 `Workspace.list` ）需要与PAI后端服务进行通讯时，默认使用该Session进行通讯。
   初始化默认的SDK Session和AI工作空间时，您可以通过以下两种方式指定AI工作空间：

   - 方式一：首先在PAI控制台上查看AI工作空间的名称或ID，然后初始化默认Session的同时指定AI工作空间。代码示例如下。

     ```
     from pai.core.session import setup_default_session
     from pai.core.workspace import Workspace
     setup_default_session(access_key_id="<your_access_key_id>",
                           access_key_secret="<your_access_key_secret>",
                           region_id="<your_region>",
                           # workspace_id="<your_workspace_id>",         # AI工作空间名称和ID二选一。
                           workspace_name="<your_workspace_name>")
     ```

您需要将下文初始化的相关参数中的参数值替换为实际值。

- 方式二：首先设置默认的Session，然后获取阿里云账号下可访问的AI工作空间列表，再指定使用的AI工作空间。代码示例如下。

```
from pai.core.session import setup_default_session
from pai.core.workspace import Workspace
session = setup_default_session(access_key_id="<your_access_key_id>", access_key_secr
et="<your_access_key_secret>",
                                region_id="<your_region>")
for ws in Workspace.list():
    print(ws.name, ws.id)
session.set_workspace(workspace=Workspace.get_by_name("<your_workspace_name>"))
```

您需要将下文初始化的相关参数中的参数值替换为实际值。

初始化的相关参数

参数	描述
<your_access_key_id>	阿里云账号的AccessKey ID。
<your_access_key_secret>	阿里云账号的AccessKey Secret。
<your_region>	PAI Pipeline Service的地域，后续提交的任务将运行在该地域。该参数支持以下取值： - cn-shanghai：华东2（上海） - cn-hangzhou：华东1（杭州） - cn-beijing：华北2（北京） - cn-shenzhen：华南1（深圳）
<your_workspace_name>	AI工作空间名称。对于上面两种指定AI工作空间的方式，分别根据以下方法获取该参数值： - 如果使用方式一，则该参数与<your_workspace_id>二选一。您可以登录PAI控制台，在**AI工作空间列表**页面查看该参数值。您也可以创建新的AI工作空间，详情请参见AI工作空间（旧版）。 **说明** 如果指定AI工作空间时，同时指定了AI工作空间的名称和ID，则接口报错。 - 如果使用方式二，则将该参数指定为 `Workspace.list()` 接口返回的一个ws.name。
<your_workspace_id>	AI工作空间ID。如果初始化默认Session的同时指定AI工作空间，则该参数与<your_workspace_name>二选一。您可以登录PAI控制台，在**AI工作空间列表**页面查看该参数值。您也可以创建新的AI工作空间，详情请参见AI工作空间（旧版）。 **说明** 如果指定AI工作空间时，同时指定了AI工作空间的名称和ID，则接口报错。

获取算法组件

Operator是PAI Pipeline Service中定义的算法组件，包含了组件的输入输出信息、运行参数及具体执行的实现方式（执行一个DAG或执行一个单独的镜像）。您可以通过Python SDK从PAI Pipeliner Service获取PAI内置的公共算法组件（SavedOperator），也可以将本地构造的工作流对象保存为一个Operator进行复用（参见下文的构建工作流）。

通过Python SDK从PAI Pipeliner Service获取PAI内置的公共算法组件（SavedOperator）的详细步骤如下。

1. 通过 `SavedOperator.list` 方法获取组件列表。

   在 `SavedOperator.list` 方法中，PAI提供了一些公共的算法组件，通过指定provider为ProviderAlibabaPAI，即可获取PAI提供的算法组件列表。此外，您可以通过inputs和outputs属性，查看对应组件的输入输出信息，代码示例如下所示。

   ```
   from pai.operator import SavedOperator
   from pai.common import ProviderAlibabaPAI
   for op in SavedOperator.list(provider=ProviderAlibabaPAI):
       print(op.pipeline_id, op.identifier, op.provider, op.version)
   op = next(SavedOperator.list(provider=ProviderAlibabaPAI))
   # 查看组件的输入输出信息。
   print(op.inputs)
   print(op.outputs)
   ```

2. 获取指定的算法组件。

   通过上一步的 `SavedOperator.list` 方法，可以获取算法组件的identifier-provider-version和pipeline_id信息，您可以使用二者中的任意一个从PAI Pipeline Service中获取一个唯一的算法组件。这两种方式的区别在于identifier-provider-version是由组件开发者在保存组件时指定的，而pipeline_id是由PAI Pipeline Service生成的组件唯一标识ID。以PAI提供的split组件为例，两种方式的代码示例分别如下：

   - 通过identifier-provider-version获取对应的组件

     ```
     from pai.common.utils import gen_temp_table
     op = SavedOperator.get_by_identifier(identifier="split", provider=ProviderAlibabaPAI,
     version="v1")
     print(op.inputs)
     ```

     目前PAI提供的Operator主要基于MaxCompute算法，底层依赖通过PAI命令实现。这些组件的identifier默认为 PAI命令的name，version为v1。您可以在PAI的常规机器学习组件帮助文档中查看组件的输入参数定义，详情请参见常规机器学习组件。

   - 通过pipeline_id获取对应的组件

     ```
     from pai.common.utils import gen_temp_table
     op_by_id = SavedOperator.get(op.pipeline_id)
     print(op_by_id.identifier, op_by_id.provider, op_by_id.version)
     print(op_by_id.inputs)
     ```

3. 提交运行任务。

   通过指定组件的必须输入参数，您可以提交一个单独的组件运行任务，以下以PAI提供的split算法组件为例，介绍如何使用SDK将输入数据表odps://pai_online_project/tables/wumai_data（该输入表为公开表，您可以直接使用）按照给定比例拆分到两张新表中。

```
from pai.common.utils import gen_temp_table
# split运行在MaxCompute中，需要指定运行的MaxCompute项目和执行环境。
# maxc_execution作为算法组件的一个输入，标识算法组件的执行MaxCompute Project项目信息。
maxc_execution = {
    "endpoint": "",
    "odpsProject": "<yourMaxComputeProject>",
}
#提交运行任务。
pipeline_run = op.run(
    job_name="example-split-job",
    arguments={
        "execution":maxc_execution,
        "inputTable": "odps://pai_online_project/tables/wumai_data",
        "fraction": 0.7,  #split算法组件的输入参数，可以通过op.inputs获取输入信息。
        "output1TableName": gen_temp_table(), #gen_temp_table()接口获取随机表名。
        "output2TableName": gen_temp_table(),
    }
)
print(pipeline_run.get_outputs())
```

上述代码中通过 `run` 接口提交任务后，SDK会在Console中输出任务在PAI控制台中的URL，您也可以直接在PAI控制台的任务管理页面，通过返回的运行任务ID或任务名称查找对应的任务实例。找到该任务后，进入任务详情页面，您可以查看任务执行的DAG、日志及输出，并且可以将模型直接部署到PAI-EAS。
组件可以单独运行，也可以将多个组件拼接为一个工作流运行，详情请参见下文的构建工作流。

构建工作流

PAI Pipeline Service支持将多个算法组件拼接为一个新的工作流，通过提供输入参数可以提交并运行该工作流，或将其保存为一个复合工作流组件。保存的工作流组件可以作为一个普通组件直接运行（参见上文的提交任务）或作为新创建的工作流的一个节点。

创建一个新的工作流主要包括以下流程：

1. 定义工作流的输入信息。
   输入信息包括用户输入参数PipelineParameter或数据输入PipelineArtifact。对于数据输入，目前PAI Pipeline Service支持OSS数据、MaxCompute表数据及MaxCompute OfflineModel。
2. 创建工作流中的Step及其输入。
   Step的输入可能来自于其他Step，也可能来源于当前创建的工作流的输入。
3. 指定工作流的输出信息。
   工作流可以使用引用的方式将Step节点的输出作为新的工作流的输出。

下面的示例代码中，构建了一个简单的工作流，它先使用类型转换组件将输入表的部分字段转换为浮点类型，再通过数据拆分组件将表拆分为两张MaxCompute表。

```
from pai.pipeline.types import PipelineParameter, PipelineArtifact, MetadataBuilder
from pai.pipeline import PipelineStep, Pipeline
from pai.operator import SavedOperator
from pai.common.utils import gen_temp_table
def create_composite_pipeline():
    # 定义当前工作流的输入信息。
    # 由于组件底层的处理是运行在MaxCompute上，execution参数传递执行的MaxCompute Project信息。
    execution_input = PipelineParameter(name="execution", typ=dict)
    cols_to_double_input = PipelineParameter(name="cols_to_double")
    table_input = PipelineArtifact(name="input_table", metadata=MetadataBuilder.maxc_table(
))
    # 构建类型转换Step。
    # 指定identifier-provider-version，使用一个已经保存的组件，作为工作流的一个Step。
    type_transform_step = PipelineStep(
        identifier="type_transform", provider=ProviderAlibabaPAI,
        version="v1", name="typeTransform", inputs={
            "inputTable": table_input, "execution": execution_input,
            "outputTable": gen_temp_table(), "cols_to_double": cols_to_double_input,
        }
    )
    # 构建拆分表Step。
    # SavedOperator也可以作为一个Step构建工作流。
    split_operator = SavedOperator.get_by_identifier(identifier="split",
     provider=ProviderAlibabaPAI, version="v1")
    split_step = split_operator.as_step(inputs={"inputTable": type_transform_step.outputs[0
],
            "execution": execution_input, "output1TableName": gen_temp_table(),
            "fraction": 0.5, "output2TableName": gen_temp_table(),
        })
    # Pipeline构造函数中的steps和inputs信息并不要求完整输入，Pipeline graph时，是通过Pipeline的ou
tputs和steps推导他们的依赖，从而构造对应的执行DAG。
    p = Pipeline(
        steps=[split_step],
        outputs=split_step.outputs[:2],
    )
    return p
p = create_composite_pipeline()
# 输入工作流运行所需参数(arguments)后，提交到PAI Pipeline Service运行。
pipeline_run = p.run(job_name="demo-composite-pipeline-run", arguments={
            "execution": maxc_execution,
            "cols_to_double": "time,hour,pm2,pm10,so2,co,no2",
            "input_table": "odps://pai_online_project/tables/wumai_data",
        }, wait=True)
pipeline_run.get_outputs()
```

对于上述构建好的工作流，您可以为其指定组件名称和版本，将该工作流保存到服务端成为一个可复用组件。保存的组件默认共享给阿里云账号下的所有RAM用户，保存组件的示例代码如下。

```
# 指定identifier和版本，保存工作流。保存的组件的provider默认为对应的阿里云账号的UID。
p = p.save(identifier="demo-composite-pipeline", version="v1")
print(p.pipeline_id, p.identifier, p.version, p.provider)
```

心脏病预测案例

以下示例代码通过Python SDK构建了一个心脏病预测工作流，并提交任务使其运行在PAI Pipeline Service。有关心脏病预测案例的详细信息，请参见心脏病预测。

您可以在本地的Python环境或PAI-DSW中，完成Session初始化（参见准备工作）后，运行以下的示例代码训练心脏病预测模型。为了保存输出的PMML模型文件，工作流在运行时，需要提供您的OSS信息。

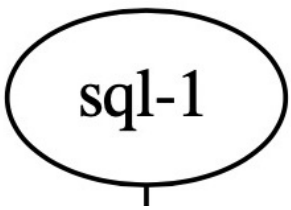
```
from pai.pipeline import Pipeline, PipelineStep, PipelineParameter
from pai.common.utils import gen_run_node_scoped_placeholder
from pai.pipeline.types import PipelineArtifact, MetadataBuilder
def create_pipeline():
    feature_cols = [
                        "sex",
                        "cp",
                        "fbs",
                        "restecg",
                        "exang",
                        "slop",
                        "thal",
                        "age",
                        "trestbps",
                        "chol",
                        "thalach",
                        "oldpeak",
                        "ca",
        ]
    label_col = "ifhealth"
    full_cols = ",".join(feature_cols + [label_col])
    pmml_oss_bucket = PipelineParameter("pmml_oss_bucket")
    pmml_oss_rolearn = PipelineParameter("pmml_oss_rolearn")
    pmml_oss_path = PipelineParameter("pmml_oss_path")
    pmml_oss_endpoint = PipelineParameter("pmml_oss_endpoint")
    execution = PipelineParameter("execution", typ=dict)
    dataset_input = PipelineArtifact(
        "inputTable",
        metadata=MetadataBuilder.maxc_table(),
        required=True,
    )
    sql = (
        "select age, (case sex when 'male' then 1 else 0 end) as sex,(case cp when"
        " 'angina' then 0  when 'notang' then 1 else 2 end) as cp, trestbps, chol,"
        " (case fbs when 'true' then 1 else 0 end) as fbs, (case restecg when 'norm'"
        " then 0  when 'abn' then 1 else 2 end) as restecg, thalach, (case exang when"
        " 'true' then 1 else 0 end) as exang, oldpeak, (case slop when 'up' then 0  "
        "when 'flat' then 1 else 2 end) as slop, ca, (case thal when 'norm' then 0 "
        " when 'fix' then 1 else 2 end) as thal, (case status when 'sick' then 1 else "
        "0 end) as ifHealth from ${t1};"
    )
    sql_step = PipelineStep(
        "sql",
        name="sql-1",
        provider=ProviderAlibabaPAI,
        version="v1",
        inputs={
```

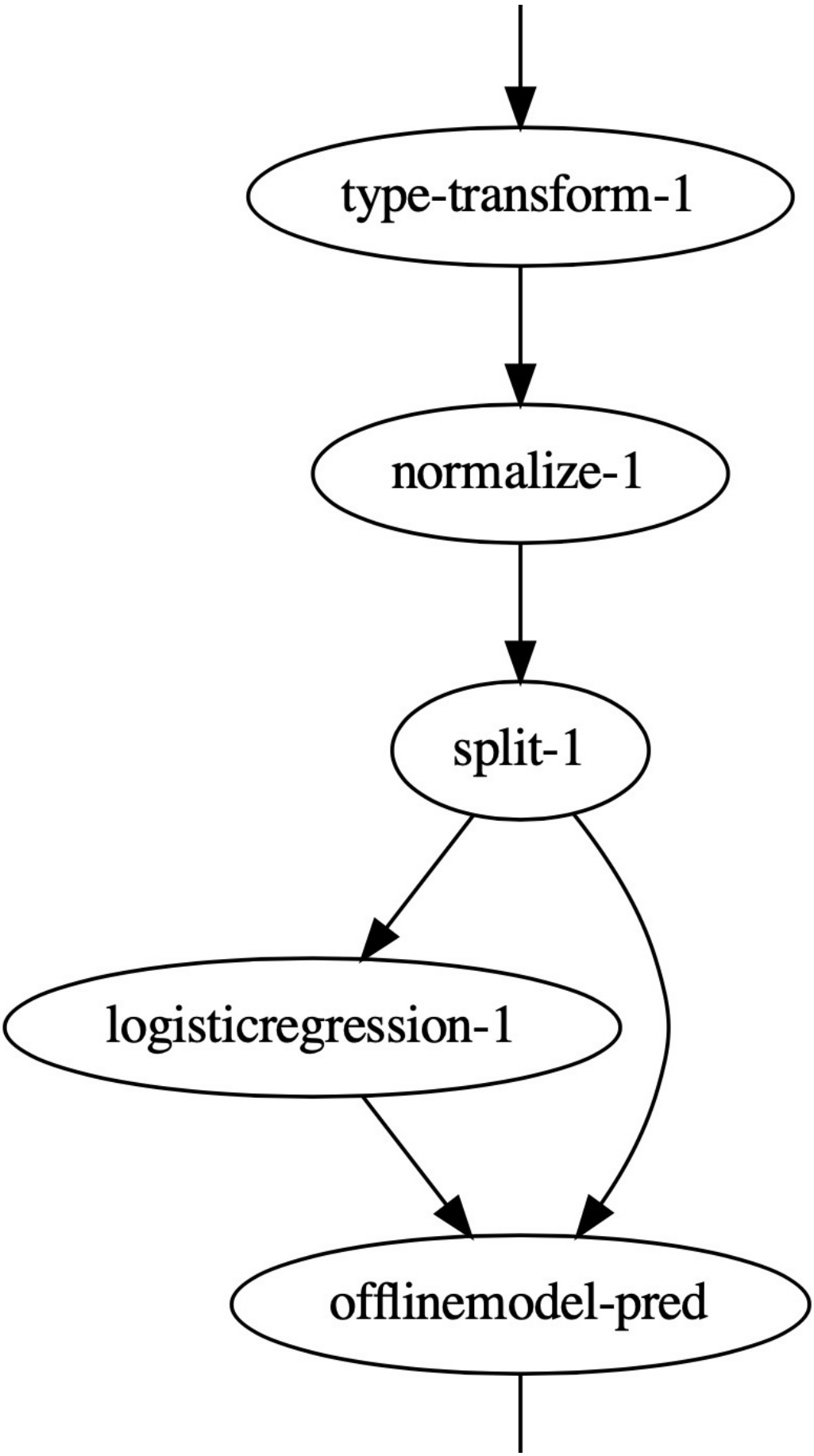
```
            "inputTable1": dataset_input,
            "execution": execution,
            "sql": sql,
            "outputTableName": gen_run_node_scoped_placeholder(
                suffix="outputTable"
            ),
        },
    )
    type_transform_step = PipelineStep(
        "type_transform",
        name="type-transform-1",
        provider=ProviderAlibabaPAI,
        version="v1",
        inputs={
            "execution": execution,
            "inputTable": sql_step.outputs["outputTable"],
            "cols_to_double": full_cols,
            "outputTable": gen_run_node_scoped_placeholder(
                suffix="outputTable"
            ),
        },
    )
    normalize_step = PipelineStep(
        "normalize_1",
        name="normalize-1",
        provider=ProviderAlibabaPAI,
        version="v1",
        inputs={
            "execution": execution,
            "inputTable": type_transform_step.outputs["outputTable"],
            "selectedColNames": full_cols,
            "lifecycle": 1,
            "outputTableName": gen_run_node_scoped_placeholder(
                suffix="outputTable"
            ),
            "outputParaTableName": gen_run_node_scoped_placeholder(
                suffix="outputParaTable"
            ),
        },
    )
    split_step = PipelineStep(
        identifier="split",
        name="split-1",
        provider=ProviderAlibabaPAI,
        version="v1",
        inputs={
            "inputTable": normalize_step.outputs["outputTable"],
            "execution": execution,
            "fraction": 0.8,
            "output1TableName": gen_run_node_scoped_placeholder(
                suffix="output1Table"
            ),
            "output2TableName": gen_run_node_scoped_placeholder(
                suffix="output2Table"
            )
```

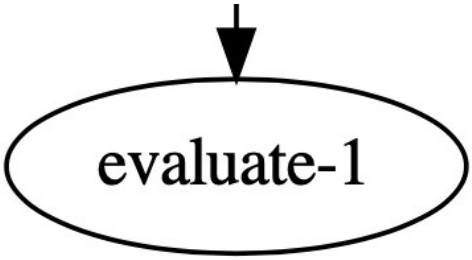
```
            ),
        },
    )
    model_name = "test_health_prediction_by_pipeline_%s" % (
        random.randint(0, 999999)
    )
    lr_step = PipelineStep(
        identifier="logisticregression_binary",
        name="logisticregression-1",
        provider=ProviderAlibabaPAI,
        version="v1",
        inputs={
            "inputTable": split_step.outputs["output1Table"],
            "execution": execution,
            "generatePmml": True,
            "pmmlOssEndpoint": pmml_oss_endpoint,
            "pmmlOssBucket": pmml_oss_bucket,
            "pmmlOssPath": pmml_oss_path,
            "pmmlOverwrite": True,
            "roleArn": pmml_oss_rolearn,
            "regularizedLevel": 1.0,
            "regularizedType": "l2",
            "modelName": model_name,
            "goodValue": 1,
            "featureColNames": ",".join(feature_cols),
            "labelColName": label_col,
        },
    )
    offline_model_pred_step = PipelineStep(
        identifier="Prediction_1",
        name="offlinemodel-pred",
        provider=ProviderAlibabaPAI,
        version="v1",
        inputs={
            "model": lr_step.outputs["model"],
            "inputTable": split_step.outputs["output2Table"],
            "execution": execution,
            "outputTableName": gen_run_node_scoped_placeholder(
                suffix="outputTable"
            ),
            "featureColNames": ",".join(feature_cols),
            "appendColNames": label_col,
        },
    )
    evaluate_step = PipelineStep(
        identifier="evaluate_1",
        name="evaluate-1",
        provider=ProviderAlibabaPAI,
        version="v1",
        inputs={
            "execution": execution,
            "inputTable": offline_model_pred_step.outputs["outputTable"],
            "outputDetailTableName": gen_run_node_scoped_placeholder(
                suffix="outputDetail"
            ),
```

```
            "outputELDetailTableName": gen_run_node_scoped_placeholder(
                suffix="outputELDetail"
            ),
            "outputMetricTableName": gen_run_node_scoped_placeholder(
                suffix="outputMetricDetail"
            ),
            "scoreColName": "prediction_score",
            "labelColName": label_col,
        },
    )
    p = Pipeline(
        steps=[evaluate_step, offline_model_pred_step],
        outputs={
            "pmmlModel": lr_step.outputs["PMMLOutput"],
            "evaluateResult": evaluate_step.outputs["outputMetricTable"],
        },
    )
    return p
p = create_pipeline()
# 输出Pipeline的图片。
p.dot().render()
pmml_oss_endpoint = "{{YourOssBucketEndpoint}}"
pmml_oss_path = "{{PathForThePmmlFile}}"
pmml_oss_bucket = "{{YourOssBucketName}}"
# 如何获取Role Arn，请参见文档https://help.aliyun.com/document_detail/106225.html。
pmml_oss_rolearn = "{{RoleArnForPaiToVisitOssBucket}}"
maxc_execution = {
    "endpoint": "{{MaxComputeProjectEndpoint}}",
    "odpsProject": "{{MaxComputeProjectName}}",
}
run_instance = p.run(
    job_name="test_heart_disease_pred",
    arguments={
        "execution": maxc_execution,
        "pmml_oss_rolearn": pmml_oss_rolearn,
        "pmml_oss_path": pmml_oss_path,
        "pmml_oss_bucket": pmml_oss_bucket,
        "pmml_oss_endpoint": pmml_oss_endpoint,
        "inputTable": "odps://pai_online_project/tables/heart_disease_prediction"
    },
    wait=True,
)
print(run_instance.get_outputs())
```

上述代码构建的工作流，通过 `pipeline.dot().render()` 渲染后，得到的Pipeline DAG如下图所示。




type-transform-1
normalize-1
split-1
logisticregression-1
offlinemodel-pred



5.3. CLI工具

5.3.1. PAI-DLC客户端工具

5.3.1.1. 命令列表

您可以通过客户端工具管理PAI-DLC中的任务和数据。本文介绍客户端工具提供的常用命令。
客户端工具提供了用户认证、创建、删除、提交、停止及查询系列命令，命令列表如下。

类别	命令
准备工作	自动补全（completion）
	用户认证（config）
创建命令	创建代码源（create code_source）
	创建数据源（create data_source）
删除命令	删除代码源（delete code_source）
	删除数据源
提交命令	提交TensorFlow训练任务（submit tfjob）
	提交PyTorch训练任务（submit pytorchjob）
停止命令	停止训练任务（stop）
查询命令	查看任务日志（logs）
	查看仓库中的镜像（get images）
	查看代码源（get code_source）
	查看数据源（get data_source）

5.3.1.2. 准备工作

您可以通过客户端工具管理PAI-DLC中的数据和任务。在使用客户端工具之前，您需要下载客户端并进行用户认证。本文介绍下载客户端及进行用户认证的命令详情。

背景信息

手册使用客户端工具管理数据和任务之前，您需要按照如下流程完成准备工作：

1. 下载客户端工具。
2. 执行自动补全命令，查看客户端工具提供的命令集，详情请参见自动补全（completion）。
3. 用户认证（config）。

下载客户端工具

您可以通过以下链接下载适合的客户端工具：

- 下载Linux 64版本的客户端
- 下载Mac版本的客户端

下载后的客户端工具无需安装，您只需要执行 `chmod +x dlc` 命令，为其添加可执行权限，即可在命令行中调用客户端工具提供的命令。

自动补全（completion）

- 功能
  自动补全功能和社区的Kubectl、Arena命令行的自动补全功能使用方式相同。在命令行中，您可以通过连续按两次Tab键使用PAI-DLC客户端工具的自动补全功能，从而对尚未完成的命令进行提示。
- 格式

```
dlc completion <shelltype>
```

- 参数
  <shelltype>表示待生成自动补全脚本的命令行类型。系统支持的命令行种类包括bash、fish、powershell及zsh。
- 示例
  本文以bash类型为例，介绍自动补全的使用方法：
  i. 执行如下命令，使自动补全命令生效。由于自动补全功能依赖于bash-completion包，因此如果执行过程中报错，则表明您的环境中未安装bash-completion包。

```
source<(dlc completion bash)
```

  ii. 如果没有安装bash-completion包，则使用如下命令安装，并通过 `source` 命令使之生效。然后再执行上一步命令。
  如果已经安装了bash-completion包，即执行上一步命令时没有报错，则跳过该步骤。

```
# Mac系统中使用如下命令。
brew install bash-completion && source /usr/local/etc/bash_completion
# Linux系统中使用如下命令。
yum install bash-completion && source /etc/profile.d/bash_completion.sh
```

  iii. 在如下 `dlc` 后连续按下两次Tab键，即可查看PAI-DLC客户端工具提供的命令。

```
dlc
```

  系统返回的PAI-DLC客户端工具提供的命令如下所示。

```
completion  config      create      delete      get         help        logs        s
top         submit
```

> **说明** 针对其他Shell类型，您可以通过 `dlc completion <zsh | fish | powershell> --help` 命令获取详细的使用方法。

用户认证（config）

- 功能
  首次使用PAI-DLC客户端工具管理数据和任务时，需要先使用阿里云账号的AccessKey ID和AccessKey Secret进行身份认证。一次认证后，再次使用时无需认证。
- 格式

```
dlc config --access_id <yourAccessKeyId> --access_key <yourAccessKeySecret> [--endpoint <yourEndpoint>] [--region <yourRegion>]
```

- 参数

需要替换的参数	是否必选	描述	类型
<yourAccessKeyId>	是	阿里云账号的AccessKey ID。	STRNG
<yourAccessKeySecret>	是	阿里云账号的AccessKey Secret。	STRNG
<yourEndpoint>	否	使用的PAI-DLC服务所在地域的Endpoint，默认为pai-dlc.cn-shanghai.aliyuncs.com。	STRNG
<yourRegion>	否	使用的PAI-DLC服务所在地域，默认为cn-shanghai。	STRNG

- 示例
  进行用户认证后，系统默认会将配置内容保存至家目录的*.dlc/config*文件中。例如，执行以下类似用户命令。

```
dlc config -access_id <yourAccessKeyId> -access_key <yourAccessKeySecret>
```

系统返回如下类似内容。

```
Configuration saved to: ~/.dlc/config
```

5.3.1.3. 创建命令

您可以通过客户端工具创建代码源和数据源。本文介绍创建相关的命令详情，包括调用格式、参数解释及使用示例。

创建代码源（create code_source）

- 功能
  用于创建PAI-DLC训练任务中使用的代码源。
- 格式

```
dlc create code_source --display_name <yourCodeSourceName> [--description <yourCodeSourceDesc>] --code_repo_url <yourCodeRepoUrl> [--code_branch <yourCodeBranch>] [--mount_path <yourMountPath>]
```

- 参数

需要替换的参数	是否必选	描述	类型
<yourCodeSourceName>	是	创建的代码源名称。	STRING
<yourCodeSourceDesc>	否	对代码源的描述，默认值为空。	STRING
<yourCodeRepoUrl>	是	代码源对应的GitHub Repo URL。	STRING
<yourCodeBranch>	否	创建代码源使用的分支，默认值为空。	STRING
<yourMountPath>	否	在服务器上映射的路径，默认为*/root/code/*。	STRING

- 示例
  您可以从GitHub的一个Repo创建代码源，示例如下。

```
dlc create code_source --display_name test_code_source_1 --code_repo_url https://github.com/tensorflow/examples
```

  创建成功后，系统会返回已经创建好的代码源ID和创建代码源的请求ID（如下所示），以便后续答疑使用。

```
+----------------------------------+--------------------------------------+
|           CodeSourceId           |              RequestId               |
+----------------------------------+--------------------------------------+
| code-20210411xxxxxx-3lm541xxxxxx | D4Bxxxxx-xxxx-49FC-A600-xxxxxxxxxxxx |
+----------------------------------+--------------------------------------+
```

创建数据源（create data_source）

- 功能
  用于创建PAI-DLC训练任务中使用的数据源。
- 格式

```
dlc create data_source [--data_source_type <yourDataSourceType>] --display_name <yourDataSourceName> [--description <yourDataSourceDesc>] --file_system_id <yourFileSystemId> [--mount_path <yourMountPath>]
```

- 参数

需要替换的参数	是否必选	描述	类型
<yourDataSourceType>	否	数据源类型，默认值为nas。目前仅支持NAS。	STRING
<yourDataSourceName>	是	创建的数据源名称。	STRING
<yourDataSourceDesc>	否	对数据源的描述，默认值为空。	STRING

需要替换的参数	是否必选	描述	类型
<yourFileSystemId>	是	NAS文件系统的ID。您可以登录NAS控制台，在对应的地域，查看NAS文件系统ID。	STRING
<yourMountPath>	否	在服务器上映射的路径，默认为*/root/data/*。	STRING

- 示例
  以从NAS数据创建数据源为例，需要指定file_system_id参数，命令如下。

```
dlc create data_source --display_name test_data_source_1 --file_system_id XXXXXXXXXXX
```

创建成功后，系统会返回已经创建好的数据源ID和创建数据源的请求ID（如下所示），以便后续答疑使用。

```
+----------------------------------+--------------------------------------+
|           DataSourceId           |              RequestId               |
+----------------------------------+--------------------------------------+
| data-20210411xxxxxx-3lm533xxxxxx | D4Bxxxxx-xxxx-49FC-A600-xxxxxxxxxxxx |
+----------------------------------+--------------------------------------+
```

5.3.1.4. 删除命令

您可以通过客户端工具删除代码源和数据源。本文介绍删除相关的命令详情，包括调用格式、参数解释及使用示例。

删除代码源（delete code_source）

- 功能
  用于删除已经创建的代码源。
- 格式

```
dlc delete code_source <yourCodeSourceId> [--force]
```

- 参数

参数	是否必选	描述	类型
<yourCodeSourceId>	是	待删除的代码源ID。	STRING
[--force]	否	删除代码源时，如果使用了该参数，则无需用户确认，即可删除代码源。反之，需要您确认后才会执行删除命令。	不涉及

- 示例

```
dlc delete code_source code-20210411xxxxxx-3lm541xxxxxx --force
```

执行上述命令，系统会删除ID为code-20210411xxxxxx-3lm541xxxxxx的代码源，且无需您确认。

删除数据源

- 功能
  用于删除已经创建的数据源。

- 格式

```
dlc delete data_source <yourDataSourceId> [--force]
```

- 参数

参数	是否必选	描述	类型
<yourDataSourceId>	是	待删除的数据源ID。	STRING
[--force]	否	删除数据源时，如果使用了该参数，则您无需确认，即可删除数据源。反之，需要您确认后才会执行删除命令。	不涉及

- 示例

```
dlc delete data_source data-20210411xxxxxx-3lm533xxxxxx --force
```

执行上述命令，系统会删除ID为data-20210411xxxxxx-3lm533xxxxxx的数据源，且无需您确认。

5.3.1.5. 提交命令

您可以通过客户端工具提交训练任务。本文介绍提交任务相关的命令详情，包括调用格式、参数解释及使用示例。

提交TensorFlow训练任务（submit tfjob）

- 功能
  用于提交TensorFlow训练任务。
- 格式
  系统支持通过命令行参数或任务参数描述文件的方式提交TensorFlow任务。

```
#方式一：命令行参数的方式。
dlc submit tfjob --name=<yourTaskName> \
  --command=<"yourStartCommand"> \
  [--<parameterName>=<parameterValue>]
#方式二：任务参数描述文件的方式
dlc submit tfjob --job_file=<yourJobFileName>
```

- 参数
  如果通过命令行参数的方式提交TensorFlow任务，则需要将命令中的如下参数替换为实际值。如果通过任务参数描述文件的方式提交TensorFlow任务，则将任务参数描述文件中支持的参数以`<parameterName>=<parameterValue>`的形式写入文件中。

参数	是否必选	描述	类型	任务参数描述文件中是否支持该参数
name=<yourTaskName>	是	任务的名称，多个任务名称可以相同。	STRING	是
command=<"yourStartCommand">	是	各个节点的启动命令。	STRING	是

参数	是否必选	描述	类型	任务参数描述文件中是否支持该参数
<parameterName>=<parameterValue>	否	<parameterName>为提交任务相关的可选参数名称，<parameterValue>为参数值。提交TensorFlow任务的相关可选参数列表如提交TensorFlow任务的相关可选参数所示。	具体参见下表中的每个可选参数	具体参见下表中的每个可选参数

提交TensorFlow任务的相关可选参数

类别	参数名称	描述	类型	任务参数描述文件中是否支持该参数
通用参数	code_source	代码源，只能传入单个代码源，默认值为空。	STRING	是
	data_sources	数据源，默认值为空。如果存在多个数据源，则数据源之间以半角逗号（,）分隔。	STRING	是
	job_file	任务参数描述文件，默认值为空。文件中的内容格式为 `<parameterName>=<parameterValue>` ，<parameterName>与命令行参数名称保持一致。如果指定了该参数，则系统优先使用job_file中的参数。	STRING	否
	thirdparty_libs	Python三方库，默认值为空。如果存在多个三方库，则三方库之间以半角逗号（,）分隔。	STRING	是
	thirdparty_lib_dir	Python三方库安装时使用的*requirements.txt*文件所在的文件夹，默认值为空。	STRING	否
	chief	是否启用TensorFlow Chief节点，该参数取值包括： ○ false：默认值，表示关闭TensorFlow Chief节点。 ○ true：表示开启TensorFlow Chief节点。	BOOL	是
	chief_image	TensorFlow Chief节点的镜像，默认值为空。	STRING	是
	chief_spec	TensorFlow Chief节点使用的服务器型号，默认值为空。	STRING	是

类别	参数名称	描述	类型	任务参数描述文件中是否支持该参数
TensorFlow特有参数	master_image	TensorFlow Master节点的镜像，默认值为空。	STRING	是
	master_spec	TensorFlow Master节点使用的服务器型号。	STRING	是
	masters	TensorFlow Master节点的数量，默认值为0。	INT	是
	ps	TensorFlow Parameter Server节点的数量，默认值为0。	INT	是
	ps_image	TensorFlow Parameter Server节点的镜像，默认值为空。	STRING	是
	ps_spec	TensorFlow Parameter Server节点使用的服务器型号，默认值为空。	STRING	是
	worker_image	TensorFlow Worker节点的镜像，默认值为空。	STRING	是
	worker_spec	TensorFlow Worker节点使用的服务器型号，默认值为空。	STRING	是
	workers	TensorFlow Worker节点的数量，默认值为0。	INT	是

- 示例

- 通过命令行参数提交一个2 Worker+ 1 PS的分布式作业，示例如下。

```
dlc submit tf --name=test_2021 --ps=1 \
  --ps_spec=ecs.g6.8xlarge \
  --ps_image=registry-vpc.cn-beijing.aliyuncs.com/pai-dlc/tensorflow-training:1.12.2PAI
-cpu-py27-ubuntu16.04 \
  --workers=2 \
  --worker_spec=ecs.g6.4xlarge \
  --worker_image=registry-vpc.cn-beijing.aliyuncs.com/pai-dlc/tensorflow-training:1.12.
2PAI-cpu-py27-ubuntu16.04 \
  --command="python /root/data/dist_mnist/code/dist-main.py --max_steps=10000 --data_di
r=/root/data/dist_mnist/data/" \
  --data_sources=data-2021xxxxxxxxxx-xxxxxxxxxxxx
```

系统返回如下类似结果。

```
+----------------------------------+--------------------------------------+
|              JobId               |              RequestId               |
+----------------------------------+--------------------------------------+
| dlc-2021xxxxxxxxxx-xxxxxxxxxxxx | xxxxxxxx-79AF-4EFC-9CE9-xxxxxxxxxxxx |
+----------------------------------+--------------------------------------+
[Info] Job [dlc-2021xxxxxxxxxx-xxxxxxxxxxxx] is [Running]
[Info] Job [dlc-2021xxxxxxxxxx-xxxxxxxxxxxx] is [Running]
[Info] Job [dlc-2021xxxxxxxxxx-xxxxxxxxxxxx] is [Running]
[Info] Job [dlc-2021xxxxxxxxxx-xxxxxxxxxxxx] is [Running]
```

- 通过任务参数描述文件提交一个2 Worker+ 1 PS的分布式作业，示例如下。

```
dlc submit tfjob --job_file=job_file.dist_mnist.1ps2w
```

其中job_file.dist_mnist.1ps2w为任务参数描述文件，采用 `<parameterName>=<parameterValue>` 的格式填写参数。job_file.dist_mnist.1ps2w的内容如下所示。

```
name=test_2021
workers=2
worker_spec=ecs.g6.4xlarge
worker_image=registry-vpc.cn-beijing.aliyuncs.com/pai-dlc/tensorflow-training:1.12.2PAI
-cpu-py27-ubuntu16.04
ps=1
ps_spec=ecs.g6.8xlarge
ps_image=registry-vpc.cn-beijing.aliyuncs.com/pai-dlc/tensorflow-training:1.12.2PAI-cpu
-py27-ubuntu16.04
command=python /root/data/dist_mnist/code/dist-main.py --max_steps=10000 --data_dir=/ro
ot/data/dist_mnist/data/
data_sources=data-2021xxxxxxxxxx-xxxxxxxxxxxx
```

提交PyTorch训练任务（submit pytorchjob）

- 功能
  用于提交PyTorch训练任务。
- 格式
  系统支持通过命令行参数或任务参数描述文件的方式提交TensorFlow任务。

```
#方式一：命令行参数的方式。
dlc submit pytorchjob --name=<yourTaskName> \
  --command=<"yourStartCommand"> \
  [--<parameterName>=<parameterValue>]
#方式二：任务参数描述文件的方式。
dlc submit pytorchjob --job_file=<yourJobFileName>
```

- 参数
  如果通过命令行参数的方式提交PyTorch任务，则需要将命令中的如下参数替换为实际值。如果通过任务参数描述文件的方式提交PyTorch任务，则将任务参数描述文件中支持的参数以`<parameterName>=<parameterValue>`的形式写入文件中。

参数	是否必选	描述	类型	任务参数描述文件中是否支持该参数
name=<yourTaskName>	是	任务的名称，多个任务名称可以相同，默认值为空。	STRING	是
command=<"yourStartCommand">	是	各个节点的启动命令，默认值为空。	STRING	是
<parameterName>=<parameterValue>	否	<parameterName>为提交任务相关的可选参数名称，<parameterValue>为参数值。提交PyTorch任务的相关可选参数列表如提交PyTorch任务的相关可选参数所示。	具体参见下表中的每个可选参数	具体参见下表中的每个可选参数

提交PyTorch任务的相关可选参数

类别	参数名称	描述	类型	任务参数描述文件中是否支持该参数
	code_source	代码源，只能传入单个代码源，默认值为空。	STRING	是
	data_sources	数据源，默认值为空。如果存在多个数据源，则数据源之间以半角逗号（,）分隔。	STRING	是

类别	参数名称	描述	类型	任务参数描述文件中是否支持该参数
通用参数	job_file	任务参数描述文件，默认值为空。文件的内容格式为 `<parameterName>=<parameterValue>`，<parameterName>与命令行参数名称保持一致。 如果指定了该参数，则系统优先使用job_file中的参数。	STRING	否
	thirdparty_libs	Python三方库，默认值为空。如果存在多个三方库，则三方库之间以半角逗号（,）分隔。	STRING	是
	thirdparty_lib_dir	Python三方库安装时使用的*requirements.txt*文件所在的文件夹，默认值为空。	STRING	否
PyTorch特有参数	master_image	PyTorch Master节点的镜像，默认值为空。	STRING	是
	master_spec	PyTorch Master节点使用的服务器型号，默认值为空。	STRING	是
	masters	PyTorch Master节点的数量，默认值为0。	INT	是
	worker_image	PyTorch Worker节点的镜像，默认值为空。	STRING	是
	worker_spec	PyTorch Worker节点使用的服务器型号，默认值为空。	STRING	是
	workers	PyTorch Worker节点的数量，默认值为0。	INT	是

- 示例
  通过命令行参数提交一个GPU的人脸识别模型训练任务，示例如下。

```
dlc submit pytorchjob --name=test_pt_face \
  --workers=1 \
  --worker_spec=ecs.gn6e-c12g1.3xlarge \
  --worker_image=registry-vpc.cn-beijing.aliyuncs.com/pai-dlc/pytorch-training:1.7.1-gpu-py37-cu110-ubuntu18.04 \
  --command="apt-get update; apt-get -y --allow-downgrades install libpcre3=2:8.38-3.1 libpcre3-dev libgl1-mesa-glx libglib2.0-dev; cd /root/data/face; python train.py --num_workers 0 --save_folder outputs" \
  --data_sources=data-2021xxxxxxxxxx-xxxxxxxxxxxx
```

系统返回如下类似结果。

```
+----------------------------------+--------------------------------------+
|              JobId               |              RequestId               |
+----------------------------------+--------------------------------------+
| dlc-2021xxxxxxxxxx-xxxxxxxxxxxx | xxxxxxxx-79AF-4EFC-9CE9-xxxxxxxxxxxx |
+----------------------------------+--------------------------------------+
[Info] Job [dlc-2021xxxxxxxxxx-xxxxxxxxxxxx] is [Running]
[Info] Job [dlc-2021xxxxxxxxxx-xxxxxxxxxxxx] is [Running]
[Info] Job [dlc-2021xxxxxxxxxx-xxxxxxxxxxxx] is [Running]
[Info] Job [dlc-2021xxxxxxxxxx-xxxxxxxxxxxx] is [Running]
```

5.3.1.6. 停止命令

您可以通过客户端工具停止训练任务。本文介绍停止任务相关的命令详情，包括调用格式、参数解释及使用示例。

停止训练任务（stop）

- 功能
  用于停止正在运行或正在创建的任务。如果停止已经处于停止或结束状态的任务，则系统返回告警信息。
- 格式

```
dlc stop <yourJobId> [--force]
```

- 参数

参数	是否必选	描述	类型
<yourJobId>	是	待停止任务的ID。	STRING
[--force]	否	停止任务时，如果使用了该参数，则您无需确认，即可停止任务。反之，需要您确认后才会执行停止命令。	不涉及

- 示例

- 停止一个已经处于停止状态的任务（假设ID为dlc-20210411xxxxxx-xxxxxxxxxxxxx的任务处于停止状态）

  ```
  dlc stop dlc-20210411xxxxxx-xxxxxxxxxxxxx
  ```

  系统会返回如下类似的告警信息。

  ```
  [WARN] The job you are trying to stop is not in creating/running status, can not be sto
  pped.
  ```

- 停止一个正在运行的任务（假设ID为dlc-20210411xxxxxx-xxxxxxxxxxxxx的任务处于运行状态）

  ```
  dlc stop dlc-20210411xxxxxx-xxxxxxxxxxxxx --force
  ```

  系统返回如下类似结果。

  ```
  [INFO] Job [dlc-20210411xxxxxx-xxxxxxxxxxxxx] has been stopped.
  ```

5.3.1.7. 查询命令

您可以通过客户端工具查看任务日志、镜像、代码源及数据源。本文介绍查询相关的命令详情，包括调用格式、参数解释及使用示例。

背景信息

客户端工具提供以下查询相关命令：

- 查看任务日志（logs）
- 查看仓库中的镜像（get images）
- 查看代码源（get code_source）
- 查看数据源（get data_source）

查看任务日志（logs）

- 功能
  查看一个训练任务的日志详情。
- 格式

  ```
  dlc logs <yourJobId> <yourPodId> [--max_events_num <yourMaxNum>] [--start_time <yourStart
  Time>] [--end_time <yourStartTime>]
  ```

- 参数

参数	是否必选	描述	类型
<yourJobId>	是	待查看训练任务的ID。	STRING
<yourPodId>	是	待查看日志的Pod ID。在分布式任务场景下，存在多个Pod。	STRING
max_events_num <yourMaxNum>	否	返回的日志最大行数，默认值为2000。	INT
start_time <yourStartTime	否	日志查询的起始时间，默认值为7天前。例如，start_time 2020-11-08T16:00:00Z	STRING

参数	是否必选	描述	类型
end_time <yourStartTime>	否	日志查询的截止时间，默认值为当前时间。例如，end_time 2020-11-08T17:00:00Z。	STRING

- 示例
  针对分布式训练任务的0号Worker节点，获取十行日志。

```
dlc logs dlc-20210411xxxxxx-xxxxxxxxxxxxx dlc-20210411xxxxxx-xxxxxxxxxxxxx-worker-0 --max
_events_num 10
```

系统返回如下类似结果。

```
WARN: ./requirements.txt not found, skip installing requirements.
================================================
|  PAI Tensorflow powered by Aliyun PAI Team.  |
================================================
Network is under initialization...
Network successfully initialized.
[2021-04-16 12:27:56.368026] [INFO] [7#7] [tensorflow/core/platform/cpu_feature_guard.cc:
141] Your CPU supports instructions that this TensorFlow binary was not compiled to use:
AVX2 AVX512F FMA
[2021-04-16 12:27:56.375586] [INFO] [7#7] [tensorflow/core/distributed_runtime/master.cc:
80] ====================CPU Architecture=====================
[2021-04-16 12:27:56.375600] [INFO] [7#7] [tensorflow/core/distributed_runtime/master.cc:
84] Disable AVX512.
[2021-04-16 12:27:56.375605] [INFO] [7#7] [tensorflow/core/distributed_runtime/master.cc:
87] CPU Vendor ID: GenuineIntel
```

查看仓库中的镜像（get images）

- 功能
  从PAI-DLC的镜像仓库中获取可用的镜像信息，您可以在提交训练任务时使用这些镜像。
- 格式

```
dlc get images [--order=<yourOrder>] [--framework=<yourFramework>] [--device=<yourDeviceT
ype>] [--provider=<yourProvider>]
```

- 参数

参数	是否必选	描述	类型
order= <yourOrder>	否	排序顺序，取值包括： ○ desc：默认值，表示降序。 ○ asc：表示升序。	STRING
framework= <yourFramework>	否	镜像包含的框架类型，默认值为空，表示任意类型。该参数取值包括： ○ PyTorchJob ○ TFJob	STRING

参数	是否必选	描述	类型
device=<yourDeviceType>	否	支持的设备类型，默认值为空，表示所有类型。该参数取值包括： ○ cpu ○ gpu	STRING
provider=<yourProvider>	否	镜像提供源，默认值为空，表示所有来源。该参数取值包括： ○ Community：社区镜像。 ○ PAI：PAI提供的官方镜像。	STRING

- 示例
  从仓库中获取社区提供的Pytorch GPU镜像，命令如下。

```
dlc get images --framework=pytorch --device=gpu --provider=Community
```

  系统返回如下类似结果。

```
+-------------------------------------------------------------------------------------------------------------------------------------------+-------------------------------------------------------------------------------------------------------------------------------------------+-------------+---------------+-------------+-----------+
|                                                                                                                                           ImageUrl                                                                                                                                           |
|                                                                                                                                           ImageUrlVpc                                                                                                                                        |
| ImageProvider | Accelerator | Framework |
+-------------------------------------------------------------------------------------------------------------------------------------------+-------------------------------------------------------------------------------------------------------------------------------------------+-------------+---------------+-------------+-----------+
| registry.cn-beijing.aliyuncs.com/pai-dlc/pytorch-training:1.6.0-gpu-py37-cu101-ubuntu18.04 | registry-vpc.cn-beijing.aliyuncs.com/pai-dlc/pytorch-training:1.6.0-gpu-py37-cu101-ubuntu18.04 | Community     | gpu         | pytorch   |
| registry.cn-beijing.aliyuncs.com/pai-dlc/pytorch-training:1.7.1-gpu-py37-cu110-ubuntu18.04 | registry-vpc.cn-beijing.aliyuncs.com/pai-dlc/pytorch-training:1.7.1-gpu-py37-cu110-ubuntu18.04 | Community     | gpu         | pytorch   |
+-------------------------------------------------------------------------------------------------------------------------------------------+-------------------------------------------------------------------------------------------------------------------------------------------+-------------+---------------+-------------+-----------+
```

  提交任务时，您可以从上述返回结果的ImageUrl和ImageUrlVpc字段，获取节点的镜像信息。

查看代码源（get code_source）

- 功能
  获取代码源的信息。如果没有指定代码源的ID，则系统返回所有的代码源。如果指定了代码源，则返回这个特定代码源的信息。
- 格式

```
dlc get code_source [yourCodeSourceId] [--sort_by=<yourSortField>] [--order=<yourOrder>] [--display_name=<yourCodeSourceName>] [--page_num=<yourPageNum>] [--page_size=<yourPageSize>]
```

- 参数

参数	是否必选	描述	类型
yourCodeSourceId	否	代码源的ID，默认值为空。	STRING
sort_by=<yourSortField>	否	用于排序的字段，默认值为GmtCreateTime。	STRING
order=<yourOrder>	否	排序顺序。取值包括： ○ desc：默认值，表示降序。 ○ asc：表示升序。	STRING
display_name=<yourCodeSourceName>	否	代码源的显示名称，支持模糊匹配，默认值为空。	STRING
page_num=<yourPageNum>	否	取第几页的数据，默认值为1。	INT
page_size=<yourPageSize>	否	分页大小，默认值为10。	INT

- 示例
  - 获取所有的代码源信息

    ```
    dlc get code_source
    ```

    系统返回如下类似结果。

```
+----------------------------------+-------------+-------------+----------------------
----------------------+------------+------------+------------------+------------------
--+--------------------+----------------------+----------------------+
|           CodeSourceId           | DisplayName | Description |                  CodeR
epo                  | CodeBranch | CodeCommit | CodeRepoUserName | CodeRepoAccessToke
n |       UserId       |    GmtCreateTime     |    GmtModifyTime     |
+----------------------------------+-------------+-------------+----------------------
----------------------+------------+------------+------------------+------------------
--+--------------------+----------------------+----------------------+
| code-20210411205952-3lm541ppXXXX | hhh         |             | http://github.com/
| master     |            |                  |                     | 232113015457841879
| 2021-04-11T12:59:53Z | 2021-04-11T12:59:53Z |
| code-20210410224342-z9fnegctXXXX | new_cs      | new test    | https://github.com
| master     |            |                  |                     | 232113015457841879
| 2021-04-10T14:43:43Z | 2021-04-10T14:43:43Z |
| code-20210410224339-t0pp0drcXXXX | new_cs      | new test    | http://github.com
| master     |            |                  |                     | 232113015457841879
| 2021-04-10T14:43:39Z | 2021-04-10T14:43:39Z |
| code-20210410224329-fie1f5o8XXXX | new_cs      | new test    | https://github.com/NVI
DIA/FasterTransformer | master     |            |                     |
| 232113015457841879 | 2021-04-10T14:43:29Z | 2021-04-10T14:43:29Z |
+----------------------------------+-------------+-------------+----------------------
----------------------+------------+------------+------------------+------------------
--+--------------------+----------------------+----------------------+
|
Total         |          4          |
+----------------------------------+-------------+-------------+----------------------
----------------------+------------+------------+------------------+------------------
--+--------------------+----------------------+----------------------+
```

- 获取特定的代码源信息

```
dlc get code_source code-20210410224329-fie1f5o8XXXX
```

系统返回如下类似结果。

```
+---------------------+-------------------------------------------------+
|        FIELD        |                      VALUE                      |
+---------------------+-------------------------------------------------+
| CodeSourceId        | code-20210410224329-fie1f5o8XXXX                |
| DisplayName         | new_cs                                          |
| Description         | new test                                        |
| CodeRepo            | https://github.com/NVIDIA/FasterTransformer     |
| CodeBranch          | master                                          |
| CodeCommit          |                                                 |
| CodeRepoUserName    |                                                 |
| CodeRepoAccessToken |                                                 |
| MountPath           | /root/code/                                     |
| UserId              | 232113015457841879                              |
| GmtCreateTime       | 2021-04-10T14:43:29Z                            |
| GmtModifyTime       | 2021-04-10T14:43:29Z                            |
+---------------------+-------------------------------------------------+
```

查看数据源（get data_source）

- 功能
  获取数据源的信息。如果没有指定数据源的ID，则系统返回所有的数据源。如果指定了数据源，则返回这个特定数据源的信息。
- 格式

```
dlc get data_source [yourDataSourceId] [--sort_by=<yourSortField>] [--order=<yourOrder>] [--display_name=<yourCodeSourceName>] [--page_num=<yourPageNum>] [--page_size=<yourPageSize>] [--data_source_type=<yourCodeSourceType>]
```

- 参数

参数	是否必选	描述	类型
yourDataSourceId	否	数据源的ID，默认值为空。	STRING
sort_by=<yourSortField>	否	用于排序的字段，默认值为GmtCreateTime。	STRING
order=<yourOrder>	否	排序顺序。取值包括： o desc：默认值，表示降序。 o asc：表示升序。	STRING
display_name=<yourCodeSourceName>	否	代码源的显示名称，支持模糊匹配，默认值为空。	STRING
page_num=<yourPageNum>	否	取第几页的数据，默认值为1。	INT
page_size=<yourPageSize>	否	分页大小，默认值为10。	INT
data_source_type=<yourCodeSourceType>	否	数据源的类型，默认值为nas。当前仅支持NAS。	STRING

- 示例

- 获取所有的数据源信息

```
dlc get data_source
```

系统返回如下类似结果。

```
+----------------+----------------------------------+-----------------+--------------
----------------+--------------+--------------------+----------------------+----------
------------+
| DataSourceType |           DataSourceId           |   DisplayName    |          Descr
iption           | FileSystemId |       UserId       |    GmtCreateTime     |    GmtMod
ifyTime     |
+----------------+----------------------------------+-----------------+--------------
----------------+--------------+--------------------+----------------------+----------
------------+
| nas            | data-20210410224621-gml01wz0XXXX | new_test_1       | test once
| 16a0b4b17a   | 232113015457841879 | 2021-04-10T14:46:21Z | 2021-04-10T14:46:21Z |
| nas            | data-20210323171833-w2hslsl1XXXX | PyTorch-Face-Nas | Nas for PyTorc
h Face Detection | 16a0b4b17a   | 232113015457841879 | 2021-03-23T09:18:34Z | 2021-03-2
3T09:18:34Z |
| nas            | data-20210323171710-ap5jirtcXXXX | PyTorch-Face-Nas | Nas for PyTorc
h Face Detection | 16a0b4b17a   | 232113015457841879 | 2021-03-23T09:17:11Z | 2021-03-2
3T09:17:11Z |
+----------------+----------------------------------+-----------------+--------------
----------------+--------------+--------------------+----------------------+----------
------------+
|
Total         |          3           |
+----------------+----------------------------------+-----------------+--------------
----------------+--------------+--------------------+----------------------+----------
------------+
```

- 获取特定的数据源信息

```
dlc get data_source data-20210410224621-gml01wz0XXXX
```

系统返回如下类似结果。

```
+----------------+----------------------------------+
|     FIELD      |              VALUE               |
+----------------+----------------------------------+
| DataSourceType | nas                              |
| DataSourceId   | data-20210410224621-gml01wz0XXXX |
| DisplayName    | new_test_1                       |
| Description    | test once                        |
| FileSystemId   | 16a0b4b17a                       |
| MountPath      | /root/data/                      |
| UserId         | 232113015457841879               |
| GmtCreateTime  | 2021-04-10T14:46:21Z             |
| GmtModifyTime  | 2021-04-10T14:46:21Z             |
+----------------+----------------------------------+
```

5.3.2. eascmd客户端工具

5.3.2.1. 下载并认证客户端

您可以通过EASCMD客户端管理PAI-EAS服务。本文介绍如何下载客户端并对其进行用户认证。

操作步骤

1. 下载EASCMD客户端。

   各版本的客户端下载地址如下：

   - Linux 32版本
   - Linux 64版本
   - Mac 64版本
   - Windows 64版本

2. （可选）在命令行中，将下载的客户端文件修改为可执行文件，示例如下。如果您下载的Windows 64版本，则可以跳过该步骤。

   ```
   chmod +x <eascmd64>
   ```

   其中<eascmd64>表示下载的客户端文件名，需要根据实际情况修改。

3. 使用阿里云账号的AccessKey进行身份认证。

   不同版本的认证命令存在差异：

   - 如果您下载的Windows 64版本，则在Windows的命令行中执行以下命令。

     ```
     eascmdwin64.exe config -i <yourAccessKeyID> -k <yourAccessKeySecret> [-e Endpoint]
     ```

   - 如果您下载的其他版本，则在对应系统的命令行中执行以下命令。

     ```
     ./<eascmd64> config -i <yourAccessKeyID> -k <yourAccessKeySecret> [-e Endpoint]
     ```

   需要根据实际情况替换以下参数。

参数	描述
<eascmd64>	根据系统环境下载的客户端文件名。
<yourAccessKeyID>	阿里云账号的AccessKey ID。
<yourAccessKeySecret>	阿里云账号的AccessKey Secret。
Endpoint	默认的PAI-EAS服务地域为华东2（上海），如果需要将模型部署至其它地域，可以使用-e参数指定地域对应的Endpoint（取值请参见地域与Endpoint的对应关系）。例如客户端文件名称为*eascmd64*，可以使用如下命令，将地域指定为华北2（北京）。 `./eascmd64 config -i <yourAccessKeyID> -k <yourAccessKeySecret> -e pai-eas.cn-beijing.aliyuncs.com`

地域与Endpoint的对应关系

地域	Endpoint
华东2（上海）	pai-eas.cn-shanghai.aliyuncs.com
华北2（北京）	pai-eas.cn-beijing.aliyuncs.com
华东2（上海）（从PAI-DSW内部署）	pai-eas-share.cn-shanghai.aliyuncs.com
华北2（北京）（从PAI-DSW内部署）	pai-eas-share.cn-beijing.aliyuncs.com
华北2（北京）（政务云，从PAI-DSW内部署）	pai-eas.cn-north-2-gov-1.aliyuncs.com
华东1（杭州）	pai-eas.cn-hangzhou.aliyuncs.com
华南1（深圳）	pai-eas.cn-shenzhen.aliyuncs.com
中国（香港）	pai-eas.cn-hongkong.aliyuncs.com
新加坡（新加坡）	pai-eas.ap-southeast-1.aliyuncs.com
印度（孟买）	pai-eas.ap-south-1.aliyuncs.com
印度尼西亚（雅加达）	pai-eas.ap-southeast-5.aliyuncs.com
德国（法兰克福）	pai-eas.eu-central-1.aliyuncs.com
美国（弗吉尼亚）	pai-eas.us-east-1.aliyuncs.com

4. 认证成功后，系统输出如下类似结果。

```
Configuration saved to: /Users/test/.eas/config
```

5.3.2.2. 命令使用说明

您可以使用eascmd管理PAI-EAS服务。本文为您介绍如何使用eascmd客户端上传文件、创建服务、修改服务配置信息、切换服务版本、删除服务、查看服务列表、查看服务详细信息、查看服务进程及配置资源组网络的相关命令。

操作命令合集

使用eascmd命令行工具管理服务，相关的操作命令如下。

类型	功能	操作入口

类型	功能	操作入口
服务相关	• 上传文件 • 创建服务 • 修改配置 • 修改服务配置 • 增加服务版本 • 删除服务 • 蓝绿发布 • 切换版本 • 查看服务列表 • 查看服务信息 • 查看服务进程 • 删除服务实例（重启实例）	支持以下两种方式使用eascmd命令行工具： • 自行下载eascmd客户端，详细请参见下载并认证客户端。 • 在PAI-DSW的Terminal中，PAI-DSW已经内置了eascmd命令行工具。
资源组相关	• 查看资源组列表 • 查看资源组详情 • 查看资源组实例列表 • 配置资源组网络	

上传文件

- 功能
  PAI-EAS为每位用户提供了OSS仓库，通过eascmd的 `upload` 命令，您可以直接上传模型或Processor，并获取上传后的OSS地址。
- 格式

```
eascmd upload <filename> [--inner]
```

- 参数
  - <filename>：待上传的文件名称。
  - [--inner]：如果在PAI-DSW内执行上传命令，则需要增加该参数。
- 示例
  在PAI-DSW内，将打包好的SavedModel模型文件*savedmodel_example/savedmodel_example.tar.gz*上传至OSS。

```
eascmd upload savedmodel_example/savedmodel_example.tar.gz --inner
```

  系统输出如下类似结果。

```
[OK] oss endpoint:    [http://oss-cn-shanghai.aliyuncs.com]
[OK] oss target path: [oss://eas-model-shanghai/182848887922****/savedmodel_example/saved
model_example.tar.gz]
Succeed: Total num: 1, size: 33,013. OK num: 1(upload 1 files).
```

  其中 `oss://eas-model-shanghai/182848887922****/savedmodel_example/savedmodel_example.tar.gz` 为存储模型的OSS地址，可以用于服务部署。

创建服务

- 功能
  通过 `create` 命令创建服务。创建服务时，需要提供资源（模型或Processor）的HTTP或OSS地址，您可以将资源上传至OSS，并获取上传后的OSS地址。
- 命令

```
eascmd create <service_desc_json>
```

- 参数
  service_desc_json表示描述服务相关信息（模型存储位置及资源规格等）的JSON文件，该文件的示例如下。

```
{
  "name": "mnist_saved_model_example",
  "generate_token": "true",
  "model_path": "http://eas-data.oss-cn-shanghai.aliyuncs.com/models%2Fmnist_saved_model.
tar.gz",
  "processor": "tensorflow_cpu_1.12",
  "metadata": {
    "instance": 1,
    "cpu": 1
  }
}
```

服务相关信息JSON文件内的参数解释如下表所示。

参数	是否必选	描述
name	是	服务名称，必须在同一地域内唯一。
token	否	表示访问鉴权的Token字符串。如果未指定，则系统自动生成。

model_path	是	model_path与processor_path分别为模型和Processor的输入数据源地址，支持以下格式的地址：
		○ HTTP地址：所需文件必须为TAR.GZ、TAR、BZ2或ZIP等压缩包。
		○ OSS地址：地址链接可以是具体文件路径或文件夹路径。同时，还需要提供参数oss_endpoint，示例如下。
		`"model_path":"oss://wowei-beijing-tiyan/alink/",`
		`"oss_endpoint":"oss-cn-beijing.aliyuncs.com",`
		○ 本地路径：如果使用 `test` 命令进行本地调试，则可以使用本地路径。

oss_endpoint	否	OSS的Endpoint，例如oss-cn-beijing.aliyuncs.com其他取值请参见访问域名和数据中心。如果model_path使用OSS地址，则必须指定该参数。
model_entry	否	表示模型的入口文件，可以包含任意文件。如果未指定，则使用model_path中的文件名。主文件路径会传递给Processor中的Load()函数。
model_config	否	表示模型的配置，支持任意文本。该参数值会传递给Processor中LoadWithConfig()函数的第二个参数。

参数	是否必选	描述
processor	否	如果使用官方提供的预置Processor，则直接在此指定Processor Code即可。Processor在eascmd中使用的Code请参见预置Processor使用说明。 如果使用自定义Processor，则无需配置该参数，只需要配置processor_path、processor_entry、processor_mainclass及processor_type参数。
processor_path	否	processor相关的文件包路径，可以参见model_path参数的描述信息。
processor_entry	否	processor的主文件。例如*libprocessor.so*或*app.py*，其中包含了预测所需的 `initialize()` 函数和 `process()` 函数的实现。 当processor_type为cpp或python时，必须指定该参数。
processor_mainclass	否	processor的主文件，JAR包中的mainclass。例如com.aliyun.TestProcessor。 当processor_type为java时，必须指定该参数。
processor_type	否	processor实现的语言，取值范围如下： ○ cpp ○ java ○ python
metadata	是	服务的Meta信息，详细参数请参见metadata参数解释。

metadata参数解释

参数	是否必选	描述	
一般参数	workers	否	每个Instance中用于并发处理请求的线程数，默认值为5。
	instance	是	服务启动的Instance数量。
	cpu	否	每个Instance需要的CPU数量。
	memory	否	每个实例需要的内存数量，取值为整型，单位为MB。例如，`"memory": 4096` 表示每个实例需要4 GB内存。 仅在专属资源组（即resource字段不为空时）中支持设置内存字段。在公共资源组中，目前内存和CPU为1:4的固定比例，例如CPU为1，则memory自动设置为4000。
	gpu	否	每个Instance需要的GPU数量。
	gpu_memory	否	每个实例所需的GPU显存数量，取值为整型，单位为GB。 系统支持实例按显存进行调度，实现单卡共享多实例功能。如果使用显存调度，则需要将gpu字段设置为0。当gpu字段设置为1时，表示实例独占整张GPU卡，此时gpu_memory 字段会被忽略。 **注意** 当前未开启显存的严格隔离，业务方需自行控制各实例的显存使用量，不要超出申请量，避免出现显存OOM。

参数	是否必选	描述	
	qos	否	实例的服务质量，可选参数值为空或BestEffort。当qos指定为BestEffort时，表示进入CPU共享模式，该参数需配合gpu_memory参数使用，使实例完全按照显存调度，不再受节点的CPU数量限制，节点上的所有实例共享CPU，cpu字段表示按CPU共享方式单个实例能使用的最大配额。 **说明** 在GPU共享模式（即 `gpu_memory != 0` ）时才会考虑开启BestEffort。如果在非GPU共享模式（即 `gpu_memory = 0` ）时开启BestEffort，会因为实例过度调度资源导致竞争过度，从而影响服务性能和稳定性。
	resource	否	资源组ID，配置策略如下： ○ 如果服务部署在公共资源组，则可以忽略该参数，此时服务进行按量付费。 ○ 如果服务部署在专属资源组，则配置该参数为资源组ID。例如eas-r-6dbzve8ip0xnzte5rp。
高级参数（慎重调整）	rpc.batching	否	是否开启Server端Batching，用于GPU模型加速。取值范围如下： ○ false：默认值，关闭Server端Batching。 ○ true：开启Server端Batching。
	rpc.keepalive	否	单个请求的最长处理时间。如果请求处理时长超过该值，则服务端返回408超时并关闭连接。默认值为5000，单位为ms。
	rpc.io_threads	否	每个Instance用于处理网络IO的线程数量，默认值为4。
	rpc.max_batch_size	否	每个Batch的最大Size，默认值为16。仅rpc.batching取值为true时，该参数生效。
	rpc.max_batch_timeout	否	每个Batch的最大Timeout，默认值为50 ms。仅rpc.batching取值为true时，该参数生效。
	rpc.max_queue_size	否	队列大小，默认值为64。队列满时，服务端返回450并关闭连接。为保证服务端不会压力过载，队列可以提前通知客户端向其他Instance进行重试。对于RT较长的服务队列，可以适当减小队列长度，以避免请求在队列中堆积导致大量请求超时。
	rpc.worker_threads	否	每个Instance中用于并发处理请求的线程数，与参数workers含义相同，默认值为5。

- 示例（假设描述服务相关信息的JSON文件为*pmml.json*）

```
eascmd create pmml.json
```

系统输出如下类似结果。

```
[RequestId]: 1651567F-8F8D-4A2B-933D-F8D3E2DD****
+------------------+--------------------------------------------------------------------------------
--------+
| Intranet Endpoint | http://pai-eas-vpc.cn-shanghai.aliyuncs.com/api/predict/savedmodel_
exanple |
|             Token | YjQxZDYzZTBiZTZjMzQ5ZmE0MzczZjIxMGZiNzZmMDBkY2VjMDg4****
|
+------------------+--------------------------------------------------------------------------------
--------+
[OK] Creating api gateway
[OK] Building image [registry-vpc.cn-shanghai.aliyuncs.com/eas/savedmodel_exanple_cn-shan
ghai:v0.0.1-20190224001315]
[OK] Pushing image [registry-vpc.cn-shanghai.aliyuncs.com/eas/savedmodel_exanple_cn-shang
hai:v0.0.1-20190224001315]
[OK] Waiting [Total: 1, Pending: 1, Running: 0]
[OK] Waiting [Total: 1, Pending: 1, Running: 0]
[OK] Service is running
```

修改配置

- 功能
  对于Instnace和CPU等metadata信息，可以直接使用 `modify` 命令的 `-D` 参数进行修改。
- 命令

```
eascmd modify <service_name> -Dmetadata.<attr_name>=<attr_value>
```

  支持同时配置多个参数，详情请参见示例。
- 参数
  - <service_name>：服务名称。
  - <attr_name>：参数名称。
  - <attr_value>：参数取值。
- 示例
  将Instance数量配置为10，且每个Instance中的Quota数量为5（5核+20 GB）。

```
eascmd modify service_test -Dmetadata.instance=10 -Dmetadata.cpu=5
```

  扩缩容时，可以只修改服务的metadata.instance参数。如果指定的Instance数量大于服务当前的Instance数量，则系统启动新的Instance，以达到要求的Instance数量，原有实例的运行不受影响。如果指定的Instance数量小于当前Instance数量，则系统停止部分Instance，以达到要求的Instance数量，其他实例的运行不受影响。

> **说明** 只修改Instance的更新操作与全量更新不同，前者不会触发服务的滚动更新。

修改服务配置

- 功能
  通过 `modify` 命令可以对已部署的服务进行配置修改。
- 命令

```
eascmd modify <service_name> -s <service_desc_json>
```

- 参数
  - <service_name>：服务名称。
  - <service_desc_json>：服务描述文件。

> **说明** 在服务描述文件中，仅写需要修改的参数即可，其他不必要参数不写。如果写了模型文件信息及processor信息，则会被认定为增加一个新的服务版本。

增加服务版本

- 功能
  通过 `modify` 命令可以对已部署的服务增加服务版本。
- 命令

```
eascmd modify <service_name> -s <service_desc_json>
```

- 参数
  - <service_name>：服务名称。
  - <service_desc_json>：服务描述文件。

> **说明** 需要在服务描述文件中指定模型文件信息及processor信息。

停止服务

- 功能
  通过 `stop` 命令可以停止一个运行中的服务。
- 命令

```
eascmd stop <service_name>
```

- 参数
  <service_name>表示待停止的服务名称。

启动服务

- 功能
  通过 `start` 命令可以重新启动一个已停止的服务。
- 命令

```
eascmd start <service_name>
```

- 参数
  <service_name>表示待启动的服务名称。

删除服务

- 功能
  通过 `delete` 命令可以删除服务，但是只能删除当前地域的服务。
- 命令

```
eascmd delete <service_name>
```

- 参数

<service_name>表示待删除的服务名称。

- 示例
  假设服务名称为savedmodel_exanple，删除该服务的步骤如下：
  i. 执行删除服务的命令。

  ```
  eascmd delete savedmodel_exanple
  ```

  系统输出如下类似结果。

  ```
  Are you sure to delete the service [savedmodel_exanple] in [cn-shanghai]? [Y/n]
  ```

  ii. 输入*Y*。系统输出如下类似结果。

  ```
  [RequestId]: 1651567F-8F8D-4A2B-933D-F8D3E2DD****
  [OK] Service [savedmodel_exanple] in region [cn-shanghai] is terminating
  [OK] Service is terminating
  [OK] Service is terminating
  [OK] Service was deleted successfully
  ```

蓝绿发布

- 功能
  通过 `create -r` 命令可以对一个已存在的服务创建一个关联服务，再使用 `release` 命令根据需求随时切换流量比例，从而进行蓝绿发布。新服务的信息描述JSON文件中的name必须与旧服务同名，其它字段根据需求自由配置。系统会自动在旧服务名基础上增加随机后缀，从而得到新服务名。
  如果删除新服务，则流量会全部切换至旧服务。如果删除旧服务，则全部流量切换至新服务。蓝绿发布之前，最原始服务的Endpoint会成为后续发布迭代的流量入口，无论后续在这个基础上进行多少次蓝绿发布迭代，该入口的Endpoint始终保持不变（例如下面示例中的{domain}/api/predict/savedmodel_example），您无须修改客户端调用代码。

  > **说明** 蓝绿发布不适用于网络直连访问的方式。

- 命令
  i. 创建关联服务

  ```
  eascmd create <service_desc_json> -r
  ```

  ii. 对蓝绿服务进行切流。

  ```
  eascmd release <service_name> -w <weight>
  ```

- 参数
  - <service_desc_json>：服务信息描述的JSON文件。
  - <service_name>：创建的新服务名称。
  - <weight>：新服务承载的流量百分比。
- 示例（假设服务信息描述文件为*pmml.json*）
  i. 创建关联服务

  ```
  eascmd create pmml.json -r
  ```

  系统输出以下类似信息。

```
[RequestId]: 1651567F-8F8D-4A2B-933D-F8D3E2DD****
+------------------+------------------------------------------------------------------
------------+
| Intranet Endpoint | http://xxx.cn-shanghai.pai-eas.aliyuncs.com/api/predict/savedmo
del_example_9c16a222 |
|             Token | YjQxZDYzZTBiZTZjMzQ5ZmE0MzczZjIxMGZiNzZmMDBkY2VjMDg4****
|
+------------------+------------------------------------------------------------------
------------+
[OK] Building image [registry-vpc.cn-shanghai.aliyuncs.com/eas/savedmodel_exanple_9c1
6a222_cn-shanghai:v0.0.1-20190224001315]
[OK] Pushing image [registry-vpc.cn-shanghai.aliyuncs.com/eas/savedmodel_exanple_9c16
a222_cn-shanghai:v0.0.1-20190224001315]
[OK] Waiting [Total: 1, Pending: 1, Running: 0]
[OK] Waiting [Total: 1, Pending: 1, Running: 0]
[OK] Service is running
```

上述输出表示创建了一个名为savedmodel_example_9c16a222的服务，且两个服务分别有独立的流量入口，可以被单独调用。您对新服务可以进行独立测试，不会影响已有服务的线上运行，测试完成后，可以对服务进行切流操作。

ii. 对蓝绿服务进行流量切换。

```
eascmd release savedmodel_example_9c16a222 -w 20
```

上述命令表示切换20%的流量到新服务savedmodel_example_9c16a222，其余80%的流量在旧服务savedmodel_example上。此时，新服务的独立访问Endpoint（{domain}/api/predict/savedmodel_example_9c16a222）关闭，旧服务Endpoint（{domain}/api/predict/savedmodel_example）流量的20%会进入新服务，80%进入旧服务。
系统输出如下类似结果。

```
Confirmed to release this service at weight [20%]? [Y/n]
```

iii. 输入*Y*，并单击Enter键，系统输出如下类似结果。

```
[RequestId]: 9258EEDE-6F99-4C3B-841B-B6E9774F****
[OK] Service [savedmodel_example_9c16a222] is weighted to 20% now
```

切换版本

- 功能
  您可以先通过 `desc` 命令查看服务的最新版本和当前版本，再通过 `version` 命令切换服务至最新版本之前的任意版本。
- 命令

```
eascmd version <service_name> <version_id>
```

- 参数
  - <service_name>：服务名称。
  - <version_id>：待切换服务的版本ID。

查看服务列表

- 功能

使用 `list`（或缩写 `ls`）命令可以查看当前用户已部署的服务列表。

- 命令

```
eascmd ls
```

- 参数
  无
- 示例

```
eascmd ls
```

系统输出如下类似结果。

```
[RequestId]: 83945D4E-ED3E-4D35-A989-831E36BB****
+--------------------------+-------------+----------+--------------------+-------------
--------+---------+--------+----------------------------------------+
|        SERVICENAME       |   REGION    | INSTANCE |     CREATETIME     |     UPDATETI
ME      | STATUS  | WEIGHT |              SERVICEPATH               |
+--------------------------+-------------+----------+--------------------+-------------
--------+---------+--------+----------------------------------------+
| mnist_saved_model_example | cn-shanghai |        1 | 2019-02-21 16:35:41 | 2019-02-21 1
6:35:41 | Running |      0 | /api/predict/mnist_saved_model_example |
+--------------------------+-------------+----------+--------------------+-------------
--------+---------+--------+----------------------------------------+
```

查看服务信息

- 功能
  通过 `desc` 命令可以查看已部署服务的详情信息。
- 命令

```
eascmd desc <service_name>
```

- 参数
  <service_name>表示服务名称。
- 示例

```
eascmd desc mnist_saved_model_example
```

系统输出如下类似结果。

```
$ eascmd desc mnist_saved_model_example
+--------------------+-----------------------------------------------------------------------
-----------------------------------------+
|              Status | Running
|
|         ServiceName | mnist_saved_model_example
|
|              Region | cn-shanghai
|
|          CreateTime | 2019-02-21 16:35:41
|
|          UpdateTime | 2019-02-21 16:35:41
|
|         AccessToken |
```

```
|
|        PrivateToken | ZWNjMTNkNDExMmExNjZkYTM4YWQ5YTY0YmFjNjk3YWYzZTRjM2Y2****
|
|       TotalInstance | 1
|
|     RunningInstance | 1
|
|     PendingInstance | 0
|
|                 CPU | 1
|
|                 GPU | 0
|
|              Memory | 1000M
|
|               Image | registry-vpc.cn-shanghai.aliyuncs.com/eas/mnist_saved_model_examp
le_cn-shanghai:v0.0.1-20190221163541        |
|              Weight | 0
|
|       LatestVersion | 1
|
|      CurrentVersion | 1
|
|             Message | Service start successfully
|
|       APIGatewayUrl | 1c3b37ea83c047efa0dc6df0cacb****-cn-shanghai.alicloudapi.com/EAPI
_182848887922****_mnist_saved_model_example |
|    APIGatewayAppKey | 2564****
|
| APIGatewayAppSecret | 12562a7b8858bbba2c2e9c4517ff****
|
|    IntranetEndpoint | http://pai-eas-vpc.cn-shanghai.aliyuncs.com/api/predict/mnist_sav
ed_model_example                            |
|       ServiceConfig | {
|
|                     |   "generate_token": "false",
|
|                     |   "metadata": {
|
|                     |     "cpu": 1,
|
|                     |     "instance": 1,
|                                                                                 |
|                     |     "region": "cn-shanghai"
|
|                     |   },
|
|                     |   "model_path":
|
|                     | "http://eas-data.oss-cn-shanghai.aliyuncs.com/models%2Fmnist_save
d_model.tar.gz",                            |
|                     |   "name":
|
|                     | "mnist_saved_model_example",
```

```
|
|                     |   "processor":
|
|                     | "tensorflow_cpu"
|
|                     | }
|
+--------------------+-------------------------------------------------------------------------
-----------------------------------------+
```

查看服务进程

- 功能
  通过 `showworkers(w)` （或缩写 `w` ）命令可以查看服务正在运行的进程状态。
- 命令

```
eascmd w <service_name>
```

- 参数
  <service_name>表示服务名称。
- 示例

```
eascmd w mnist_saved_model_example
```

系统输出如下类似结果。

```
[RequestId]: B23BA8AC-CDEC-5704-935F-3CEC6606****
+------------------------------+-------------+-------------+---------------------+-----
-----+---------+-------+--------+-------------------------------------------------------------
-------------------------------------------------+
|          INSTANCENAME          |   INNERIP   |    HOSTIP    |       STARTAT       | REST
ARTS | STATUS  | READY | REASON |                                                    LASTSTAT
E                                                |
+------------------------------+-------------+-------------+---------------------+-----
-----+---------+-------+--------+-------------------------------------------------------------
-------------------------------------------------+
| network-test-69cf5dd6c7-5**** | 10.240.XX.XX | 10.224.XX.XX | 2021-09-27 15:04:22 |
1 | Running | [1/1] |        | {"exitCode":247,"finishedAt":"2021-09-27T07:04:21Z","reaso
n":"Error","startedAt":"2021-09-27T05:36:56Z"} |
+------------------------------+-------------+-------------+---------------------+-----
-----+---------+-------+--------+-------------------------------------------------------------
-------------------------------------------------+
```

返回结果中的参数解释如下表所示。

参数	描述
INSTANCENAME	服务实例的名称。
INNERIP	实例的内网IP。
HOSTIP	实例所在的节点IP。
STARTAT	实例的启动时间。

参数	描述
RESTARTS	实例的重启次数。实例每次发生OOM或代码Crash均会自动重启，RESTARTS取值会加1。
STATUS	实例的当前状态。其中Pending时表示在等待资源调度，长时间处于Pending状态表示资源不足，实例无法调度。
READY	实例中的容器状态，格式为`[当前就绪的容器数/当前实例中所有的容器数]`。 当READY为[0/1]时，表示容器正在启动或容器异常健康检查不通过，此时流量不会进入该实例中。 当一个服务中的所有实例Ready均为[0/1]时，表示整个服务不可用，请求服务时会返回`5xx`。
REASON	实例状态的简单描述。
LASTSTATE	实例上次重启的状态。LASTSTATE中reason为OOMKIlled时，表示实例发生了OOM。

删除服务实例（重启实例）

- 功能
  通过`deleteworkers`（或缩写`dw`）命令可以删除服务的一个或多个实例。由于删除实例后，系统会自动启动新实例进行替换，因此该命令也可以作为实例重启操作使用。
- 命令

```
eascmd dw <service_name> <instance_names>
```

- 参数
  - <service_name>：表示服务名称。
  - <instance_names>：表示待删除的服务实例名称，多个实例中间使用半角逗号（,）分隔。
- 示例

```
eascmd dw mnist_saved_model_example mnist-saved-model-example-69cf5dd6c7-5****
```

系统输出如下类似结果。

```
Are you sure to delete the instances [mnist-saved-model-example-69cf5dd6c7-5****] of serv
ice [mnist_saved_model_example] in [cn-shenzhen]? [Y/n]
```

输入*Y*后，系统输出如下类似结果。

```
[RequestId]: 564C8F56-D97A-555E-9E0B-22BE140A****
[OK] Instance(s) [mnist-saved-model-example-69cf5dd6c7-5****] for service [mnist_saved_mo
del_example] in region [cn-shenzhen] was deleted successfully
```

查看资源组列表

- 功能
  通过`resource list`（或缩写`resource ls`）命令可以查看当前账户下的资源组列表。
- 命令

```
eascmd resource ls
```

- 参数
  无
- 示例

```
eascmd resource ls
```

系统输出如下类似结果。

```
+--------------------------+-------------+---------------+----------+----------+---------
---------+--------------------+---------------+
|       RESOURCENAME       |  CLUSTERID  | INSTANCECOUNT | GPUCOUNT | CPUCOUNT |     OWNE
RUID     |     CREATETIME     |    STATUS     |
+--------------------------+-------------+---------------+----------+----------+---------
---------+--------------------+---------------+
| eas-r-lzo32vrdbtukr7te3i | cn-shanghai |             1 |        0 |       16 | 18284888
7922**** | 2020-03-18 13:09:24 | ResourceReady |
+--------------------------+-------------+---------------+----------+----------+---------
---------+--------------------+---------------+
```

查看资源组详情

- 功能
  通过 `resource desc` 命令可以查看某个资源组的详细信息。
- 命令

```
eascmd resource desc <resource_id>
```

- 参数
  <resource_id>表示待查看的资源组ID，即 `resource list(ls)` 命令返回结果中的RESOURCENAME字段。
- 示例

```
eascmd -c ~/.eas/shanghai2.conf resource desc eas-r-lzo32vrdbtukr7te3i
```

系统输出如下类似结果。

```
+---------+----------------+------------------------------------------------------------+
|   Basic | ResourceName   | eas-r-lzo32vrdbtukr7te3i                                   |
|         | Region         | cn-shanghai                                                |
|         | CpuCount       | 16                                                         |
|         | GpuCount       | 0                                                          |
|         | instanceCount  | 1                                                          |
|         | CreateTime     | 2020-03-18 13:09:24                                        |
|         | LastStatus     | ResourceReady                                              |
|         | Message        | Resource is ready                                          |
|         | RoleArn        | acs:ram::xxx:role/AliyunPAIAccessingENIRole                |
| Network | VpcId          | vpc-uf6s9pv47nu03srne****                                  |
|         | VSwitchId      | vsw-uf6voq53e893k56ws****                                  |
|         | SecurityGroupId | sg-uf6c5twkfar8l06c****                                   |
|         | DestinationCIDR |                                                            |
|         | AuxVSwitchList | []                                                         |
+---------+----------------+------------------------------------------------------------+
```

查看资源组实例列表

- 功能
  通过 `resource list_instance` （或缩写为 `resource li` ）命令可以查看某个资源组的实例列表及每个实例的资源使用情况。
- 命令

```
eascmd resource list_instance <resource_id>
```

- 参数
  <resource_id>表示待查看的资源组ID，即 `resource list(ls)` 命令返回结果中的RESOURCENAME字段。
- 示例

```
eascmd resource li eas-r-lzo32vrdbtukr7te3i
```

  系统输出如下类似结果。

```
+-----------------------------------+--------------+--------+----------------+----------
------+------------------+--------------------+---------------+------------+
|            INSTANCENAME           |  INSTANCEIP  | STATUS | TOTAL/USED CPU | TOTAL/USE
D GPU | TOTAL/USED MEMORY |     CREATETIME      |  INSTANCETYPE  | CHARGETYPE |
+-----------------------------------+--------------+--------+----------------+----------
------+------------------+--------------------+---------------+------------+
| cn-shanghai.i-uf6dj71ir6mh3gjmaz3a | 10.224.XX.XX | Ready  | 16/6           | 0/0
| 62240M/4200M      | 2020-03-18 13:09:34 | ecs.g6.4xlarge | PostPaid   |
+-----------------------------------+--------------+--------+----------------+----------
------+------------------+--------------------+---------------+------------+
```

配置资源组网络

- 功能
  通过 `resource network` 命令可以设置某个资源组的直连情况，用于连通PAI-EAS VPC和用户VPC之间的网络。一方面可以在用户VPC内以直连软负载的方式调用PAI-EAS服务，另一方面可以在PAI-EAS Processor中反向访问用户VPC中的内网资源（例如RDS、Redis等）。
- 命令

```
eascmd resource network <resource_id> -s <network_cfg.json>
```

- 参数
  - <resource_id>：表示查待看的资源组ID，即 `resource list(ls)` 命令返回结果中的RESOURCENAME字段。

- <network_cfg.json>：网络配置文件，该文件格式如下所示。

```
{
  "Action":"create",
  "VSwitchId": "vsw-8vbsunr5bkcbyxh94****",
  "SecurityGroupId": "sg-8vbhwowdxzx5fjcx****",
  "VSwitchIdList": ["vsw-8xbsunr5abcbyqh93****", "vsw-8xbs1y7gu6cxbvqzw****"],
  "DestinationCIDR": "192.XX.XX.XX/16"
}
```

各参数的含义如下表所示。

参数	描述	是否必选	默认值
Action	网络设置的操作，取值范围如下： ■ create：开通直连。 ■ delete：关闭直连，此时无需配置其他参数。	是	无
VSwitchId	待连通的目标主vSwitch ID，PAI-EAS会自动在该vSwitch中创建ENI弹性网卡，请不要主动删除该ENI，否则会导致网络连通性问题。	是	无
SecurityGroupId	客户端ECS所在的安全组ID。 **说明** 客户端ECS必需归属于该安全组中，否则会导致网络连通性问题。	是	无
VSwitchIdList	待打通的附属vSwitch列表，必须与主vSwitch在同一个VPC中，这些VSwitch的IP网段会自动被加入到PAI-EAS的路由表规则中。	否	空数组（[]）
DestinationCIDR	待打通的客户端目标网段，必须与主vSwitch在同一个VPC中，该网段会被自动加入到PAI-EAS的路由表规则中。	否	空字符串（""）

说明 VSwitchIdList与DestinationCIDR原理相同，均是为了连通PAI-EAS集群与用户某个网段的网络。如果需要连通多个指定vSwitch，则使用VSwitchIdList。如果需要连通一个大网段（例如整个VPC），则使用DestinationCIDR字段。建议不要使用10.0.0.0/8、10.224.0.0/16或10.240.0.0/16网段，否则会导致网络冲突问题。如果有其他需求，可以提交工单。

预测

进行预测调用时，可以根据创建服务时生成的HTTP URL访问服务。预测服务的输入输出格式由Processor自定义，详细请参见模型服务调用章节中的通用Processor服务请求数据构造部分。

6.相关内容

6.1. 相关协议

6.1.1. 阿里云产品服务协议

阿里云产品服务协议的详情，请参见阿里云产品及服务协议。

6.1.2. 服务等级协议（SLA）

机器学习PAI的服务等级协议（SLA）的详情，请参见阿里云机器学习PAI服务等级协议。

6.2. 用户交流

阿里云机器学习平台以社区化的方式进行用户运营，会经常举行用户活动，同时为企业级用户提供高效支持。如果您在产品使用过程中遇到问题，请通过工单系统和机器学习平台内部的问答机器人解决。用户交流群主要用来进行机器学习算法心得交流及相关活动的推广。
加入阿里云机器学习钉钉群，入群链接：邀请链接。

此外，也可以关注云栖社区的官方公众号，会有精彩文章推送。

相关地址

- 产品首页
- 计费说明
- 算法组件
- 深度学习框架
- 模型在线服务
- 离线调度
- 案例说明
- 产品BUG反馈、工单系统