ModelArts

Workflow

文档版本 01

发布日期 2022-05-26

华为技术有限公司

商标声明


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

注意

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

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

目 录

1 MLOps 简介 1
1.1 什么是 MLOps 1
1.2 DevOps 1
1.3 MLOps 功能介绍 2
2 什么是 Workflow 4
2.1 Workflow 能做什么 4
2.2 Workflow 的构成 5
2.3 运行第一条 Workflow 5
2.4 Workflow 提供的样例 10
3 如何使用 Workflow 11
3.1 从 AI Gallery 订阅的 Workflow 如何使用 11
3.2 Workflow 的配置 12
3.2.1 配置的入口 12
3.2.2 运行配置 13
3.2.3 资源配置 13
3.2.4 消息通知 14
3.2.5 输入与输出配置 14
3.2.6 节点参数配置 15
3.2.7 保存配置 15
3.3 启动/停止/复制/删除 Workflow 16
3.4 查看 Workflow 运行记录 17
3.5 重试/停止/继续运行节点 18
4 如何开发 Workflow 20
4.1 基本概念与功能概览 20
4.2 开发环境准备 20
4.2.1 Notebook-JupyterLab 20
4.2.2 本地 IDE(PyCharm) 22
4.3 开发第一条 Workflow 23
4.4 节点的数据类型 25
4.4.1 输入数据类型 26
4.4.2 输出数据类型 29
4.4.3 参数类型 33

4.4.4 统一存储 34
4.4.4.1 功能介绍 35
4.4.4.2 常用方式 35
4.4.4.3 进阶用法 35
4.4.4.4 使用案例 36
4.4.4.5 根路径配置 37
4.5 Workflow 支持的节点类型 38
4.5.1 数据集创建节点 38
4.5.2 数据集标注节点 40
4.5.3 数据集导入节点 42
4.5.4 数据集版本发布节点 45
4.5.5 作业类型节点 47
4.5.6 模型注册节点 51
4.5.7 服务部署节点 53
4.5.8 条件节点 54
4.6 如何在 Workflow 中使用大数据的能力（DLI/MRS） 56
4.7 如何将串联节点构建工作流 58
4.8 如何调试 Workflow 59
4.9 如何把 Workflow 发布给用户使用 60
4.10 如何分享自己的 Workflow 到 Gallery 60
4.10.1 准备 Workflow 60
4.10.2 发布 Workflow 资产 62
4.10.3 发布 Workflow 的新版本 62
4.11 最佳实践 63
4.11.1 全链路（服务部署） 63
4.11.2 全链路（服务更新） 64
4.11.3 全链路（condition 判断是否部署） 66
4.11.4 全链路（数据可迭代） 67

1 MLOps 简介

1.1 什么是 MLOps

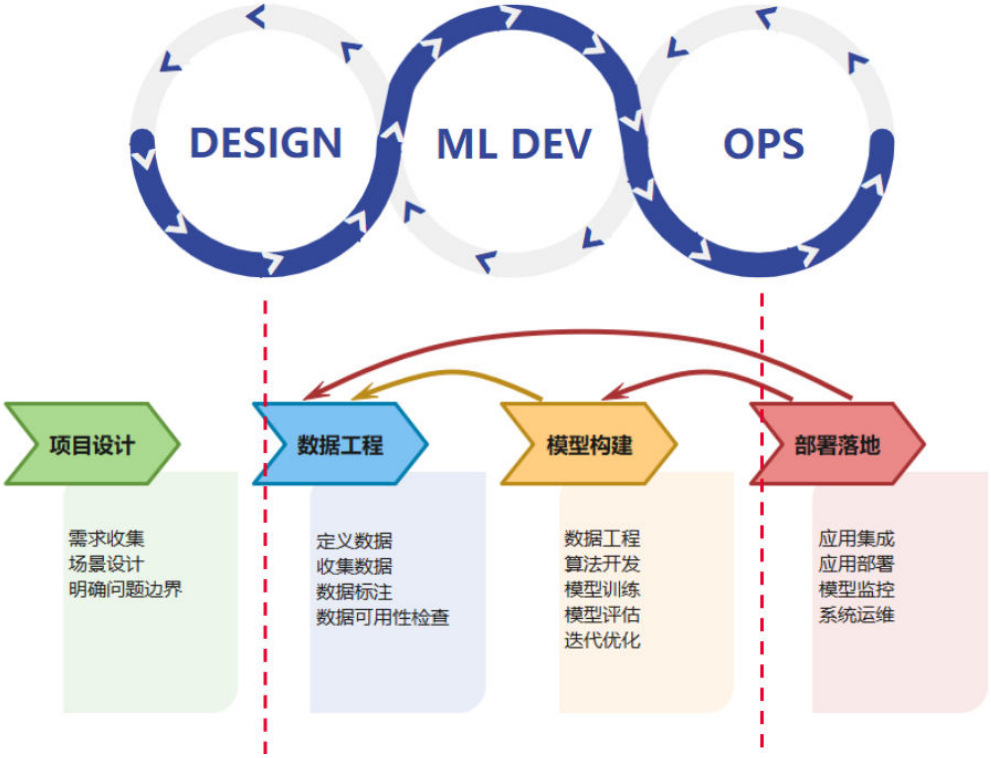
MLOps(Machine Learning Operation)是“机器学习”（Machine Learning）和“DevOps”（Development and Operations）的组合实践。随着机器学习的发展，人们对它的期待不仅仅是学术研究方面的领先突破，更希望这些技术能够系统化地落地到各个场景中。但技术的真实落地和学术研究还是有比较大的差别的。在学术研究中，一个AI算法的开发是面向固定的数据集（公共数据集或者某个特定场景固定数据集），基于这单个数据集上，不断做算法的迭代与优化，面向场景的AI系统化开发的过程中，除了模型的开发，还有整套系统的开发，于是软件系统开发中成功经验“DevOps”被自然地引入进来。但是，在人工智能时代，传统的DevOps已经不能完全覆盖一个人工智能系统开发的全流程了。

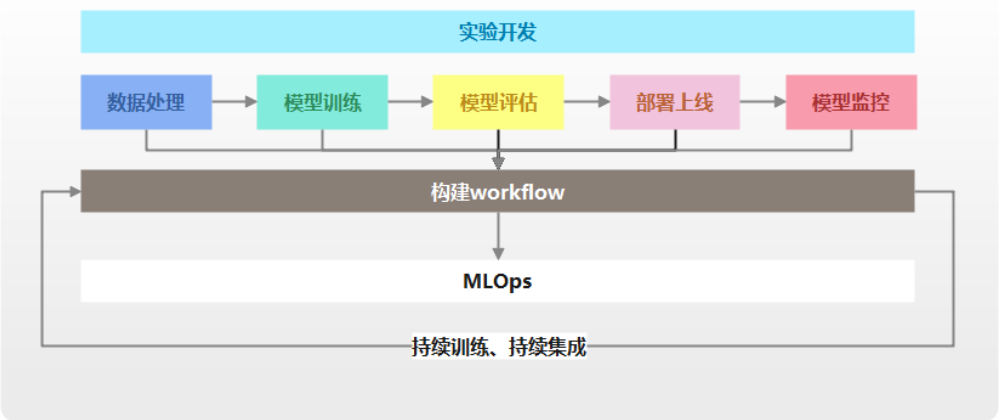
1.2 DevOps

DevOps，即Development and Operations，是一组过程、方法与系统的统称，用于促进软件开发、运维和质量保障部门之间的沟通、协作与整合。在大型的软件系统开发中，DevOps被验证是一个非常成功的方法。DevOps不仅可以加快业务与开发之间的互动与迭代，还可以解决开发与运维之间的冲突。开发侧很快，运维侧太稳，这个就是我们常说的开发与运维之间固有的、根因的冲突。在AI应用落地的过程中，也有类似的冲突。AI应用的开发门槛较高，需要有一定的算法基础，而且算法需要快速高效地迭代。专业的运维人员追求的更多是稳定、安全和可靠；专业知识也和AI算法大相径庭。运维人员需要去理解算法人员的设计与思路才能保障服务，这对于他们来说，门槛更高了。在这种情况下，更多时候可能需要一个算法人员去端到端负责，这样一来，人力成本就会过高。这种模式在少量模型应用的场景是可行的，但是当规模化落地AI应用时，人力问题将会成为瓶颈。

1.3 MLOps 功能介绍

机器学习开发流程主要可以定义为四个步骤：项目设计、数据工程、模型构建、部署落地。AI开发并不是一个单向的流水线作业，在开发的过程中，会根据数据和模型结果进行多轮的实验迭代（如上图箭头指向）。算法工程师会根据数据特征以及数据的标签做多样化的数据处理以及多种模型优化，以获得在已有的数据集上更好的模型效果。传统的AI应用交付会直接在实验迭代结束后以输出的模型为终点。当应用上线后，随着时间的推移，会出现模型漂移的问题。新的数据和新的特征在已有的模型上表现会越来越差。在MLOps中，实验迭代的产物将会是一条固化下来的流水线，这条流水线将会包含数据工程，模型算法，训练配置等。用户将会使用这条流水线在持续产生的数据中持续迭代训练，确保这条流水线生产出来的模型的AI应用始终维持在一个较好的状态。





MLOps的整条链路需要有一个工具去承载，MLOps打通了算法开发到交付运维的全流程。和以往的开发交付不同，以往的开发与交付过程是分离的，算法工程师开发完的模型，一般都需要交付给下游系统工程师，此时往往这个过程中，算法工程师参与度还是非常高的。企业内部一般都是有一个交付配合的机制。从项目管理角度上需要增加一个AI项目的工作流程机制管理，流程管理不是一个简单的流水线构建管理，它是一个任务管理体系。

这个工具需要具备以下的能力：

- 流程分析： 沉淀行业样例流水线，帮助用户能快速进行AI项目的参考设计，启动快速的AI项目流程设计。
- 流程定义与重定义：以流水线作为承载项，用户能快速定义AI项目，实现训练+推理上线的工作流设计。
- 资源分配：支持帐号管理机制给流水线中的参与人员（包含开发者和运维人员）分配相应的资源配额与权限，并查看相应的资源使用情况等。
- 时间安排：围绕子流水线配置相应的子任务安排，并加以通知机制，实现流程执行过程之间配合的运转高效管理。
- 流程质量与效率测评：提供流水线的任务执行过程视图，增加不同的检查点，如数据评估，模型评估，性能评估等，让AI项目管理者能很方便的查看流水线执行过程的质量与效率。
- 流程优化：围绕流水线每一次迭代下，用户可以自定义输出相关的核心指标，并获取相应的问题数据与原因等，从而基于这些指标下，快速决定下一轮迭代的执行优化。

2 什么是 Workflow

2.1 Workflow 能做什么

Workflow（也称工作流，下文中均可使用工作流进行描述）本质是开发者基于实际业务场景开发用于部署模型或应用的流水线工具。在机器学习的场景中，流水线可能会覆盖数据标注、数据处理、模型开发/训练、模型评估、应用开发、应用评估等步骤。

区别于传统的机器学习模型构建，开发者可以使用Workflow开发生产流水线。基于MLOps的概念，Workflow会提供运行记录、监控、持续运行等功能。根据角色的分工与概念，产品上将工作流的开发和持续迭代分开。

一条流水线由多个节点组成，Workflow SDK提供了流水线需要覆盖的功能以及功能需要的参数描述。用户在开发流水线的时候，使用SDK对节点以及节点之间串联的关系进行描述。对流水线的开发操作在Workflow中统称为Workflow的开发态。当确定好整条流水线后，开发者可以将流水线固化下来，提供给其他人使用。使用者无需关注流水线中包含什么算法，也不需要关注流水线是如何实现的。使用者只需要关注流水线生产出来的模型或者应用是否符合上线要求，如果不符合，是否需要调整数据和参数重新迭代。这种使用固化下来的流水线的状态，在Workflow中统称为运行态。

总的来说，Workflow有两种形态：

- 开发态：使用Workflow的Python SDK开发和调测流水线。
- 运行态：可视化配置运行生产好的流水线。

Workflow基于对当前ModelArts已有能力的编排。基于DevOps原则和实践，应用于AI开发过程中，提升AI应用开发与落地效率，以达到更快的模型实验和开发和更快的将模型部署到生产。

工作流的开发态和运行态分别实现了不同的功能。

开发态-开发工作流

开发者结合实际业务的需求，通过Workflow提供的Python SDK，将ModelArts模块的能力封装成流水线中的一个个步骤。对于AI开发者来说是非常熟悉的开发模式，而且灵活度极高。Python SDK主要提供以下能力。

- 调测：部分运行、全部运行、debug。
- 发布：发布到运行态。
- 实验记录：实验的持久化及管理。

运行态-运行工作流

Workflow提供了可视化的工作流运行方式。使用者只需要关注一些简单的参数配置，模型是否需要重新训练和模型当前的部署情况。运行态工作流的来源为：通过开发态发布通过AI Gallery订阅。

运行态主要提供以下能力。

- 统一配置管理：管理工作流需要配置的参数及使用的资源等。
- 操作工作流：启动、停止、复制、删除工作流。
- 运行记录：工作流历史运行的参数以及状态记录。

2.2 Workflow 的构成

工作流是对一个有向无环图的描述。开发者可以通过 Workflow 进行有向无环图（Directed Acyclic Graph，DAG）的开发。一个DAG是由节点和节点之间的关系描述组成的。开发者通过定义节点的执行内容和节点的执行顺序定义DAG。如下图，绿色的矩形表示为一个节点，节点与节点之间的连线则是节点的关系描述。整个DAG的执行其实就是有序的任务执行模板。

图 2-1 工作流

2.3 运行第一条 Workflow

了解Workflow的功能与构成后，可通过订阅workflow的方式尝试运行首条工作流，进一步了解Workflow的运行过程。

1. **数据集准备**。
2. **订阅工作流**。
3. **运行工作流**。

数据集准备

1. 前往AI Gallery，在“资产集市>数据>数据集”下载**常见生活垃圾图片**。

2. 选择下载方式选择“ModelArts数据集”，目标区域“华北-北京四”，目标位置为数据集的输出路径，数据集名称可自行修改。

3. 单击“确定”，自动跳转至AI Gallery的个人中心“我的下载”页签。等待下载完成即可。

4. 单击“目标数据集”即可跳转至管理控制台的数据集概览页。

订阅工作流

1. 登录MdoelArts管理控制台，左侧菜单栏选择“总览>Workflow（Beta）”,进入Workflow详情页。
2. 在详情页的Workflow列表区域，单击“**前往AI Gallery订阅**”。
3. 选择订阅“图像分类-ResNet_v1_50工作流”。

运行工作流

1. 订阅完成后，单击“运行”进入配置页面，资产版本默认，选择云服务区域“华北-北京四”，单击“导入”即可。

说明

工作流运行的云服务区域需要与创建的数据集所在区域保持一致，否则工作流配置时无法选到准备好的数据集。

2. 导入完成后会自动跳转至workflow的详情页面，在详情页单击右上方的“配置”按钮完成配置。配置参数填写参考**表2-1**。

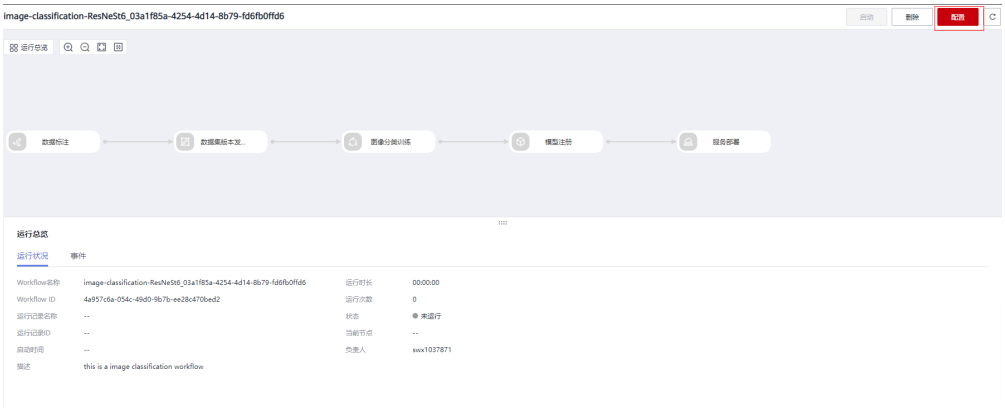


表 2-1 配置参数说明

参数	配置说明
运行配置	该参数为输出根目录配置，整个工作流的输出均会被保存在该目录下。
资源配置	训练资源规格配置，建议选用GPU规格。

参数	配置说明
数据标注节点配置	选择预先创建的数据集即可，版本可以不用选择。task_name填写需要创建的标注任务名称即可。 **说明** 首次运行需要配置，会自动创建新的标注任务，后续不建议进行修改，使用同一个标注任务进行数据标注。
训练相关参数配置	算法超参相关的配置，每个参数的具体含义已在输入框下方进行说明（可直接使用默认值）。
模型名称参数配置	• 配置生成的模型名称，工作流多次运行使用同一个模型名称会自动新增版本。 • 工作流运行完成后用户可以在Modelarts界面的AI应用进行查看。

3. 配置完成后单击右上方“保存配置”按钮，保存完成后单击“启动”开始运行工作流。工作流在运行过程中，需要用户在数据标注节点以及服务部署节点完成相关操作或者配置。

   a. 数据标注节点：标注节点启动后会等待用户确认数据标注是否完成，用户需单击“详情”前往数据集页面查看该数据集是否已完成标注。

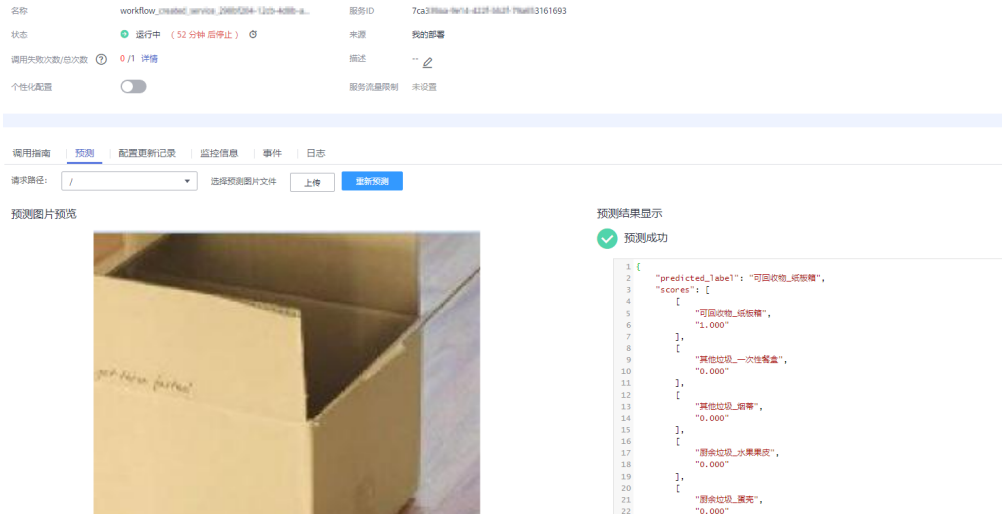
   

   b. 确认数据标注完成后返回执行页面，单击“继续运行”。

   c. 服务部署节点：“选择模型模板”默认选择最新模型版本，“计算节点规格”选择GPU类型，可单击开启“是否自动停止”，默认不开启。配置完成后单击“继续运行”即可，等待服务部署完成。

4. 测试推理服务：工作流运行完成后，在服务部署节点右侧单击“详情”进行跳转或者在ModelArts管理控制台，选择“部署上线>在线服务”，找到部署的推理服务，单击服务名称，进入服务详情，单击“预测”。右边可查看预测结果。



2.4 Workflow 提供的样例

ModelArts提供了丰富的基于场景的工作流样例，用户可以前往AI Gallery进行订阅，**AIGallery Workflow案例库**。

产品会在AI Gallery中持续提供越来越多的资产 。用户也可以自己开发Workflow样例，参见**如何开发Workflow**。

3 如何使用 Workflow

3.1 从 AI Gallery 订阅的 Workflow 如何使用

1. 登录**AI Gallery的Workflow案例库**。
2. 从AI Gallery选择并订阅一个Workflow。
3. 订阅完成后，单击“运行”后跳转到ModelArts控制台界面，选择**资产版本**和**云服务区域**，单击“导入”，进入该Workflow的详情页面。

   图 3-1 导入 Workflow

4. 单击右上角的“配置”后进入配置页面，根据您所订阅的工作流，配置Workflow需要的部分输入项和参数，单击右上角的“保存配置”。
5. 保存成功后，单击右上角的“启动”，启动Workflow。
6. Workflow进入运行页面，等待Workflow运行。
7. Workflow完成运行。

图 3-2 完成运行

3.2 Workflow 的配置

在运行工作流**之前**或**运行过程中**，需要去设置工作流需要的参数及使用的资源等内容，这些内容是通过工作流的开发者根据业务设计呈现给使用者的。用户在获取该条工作流后，可根据自己的需求进行配置的修改，使生产的模型或者应用更贴合自己的场景。

3.2.1 配置的入口

Workflow的配置包括运行前的配置和运行中的配置。

运行前配置

在ModeArts控制台“总览>workflow”进入工作流列表页，有如下两个入口用于运行前的工作流配置。

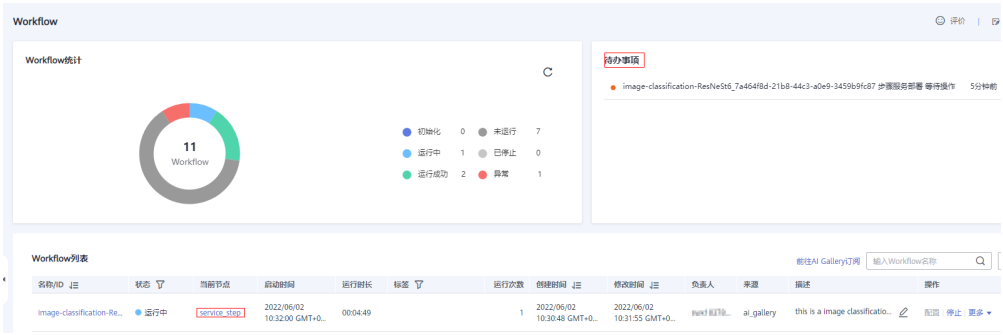
- 单击workflow列表页操作栏的“配置”，跳转到工作流配置页。

图 3-3 配置工作流

- 在工作流列表页，单击某条工作流的名称，进入工作流详情页，单击右上角的“配置”，跳转到工作流配置页。

运行中配置

工作流可能存在运行中的配置项，运行到该节点会暂停，等待用户输入。此类配置可从ModeArts控制台的“总览>workflow”进入Workflow页面，在Workflow页面的“待办事项”和运行中的工作流“当前节点”进入。



3.2.2 运行配置

Workflow可以针对工作目录做统一管理，根目录的配置在运行配置页签中完成。

3.2.3 资源配置

一条Workflow中有不止一个节点可以进行资源的配置，当前支持对不同节点进行不同的规格配置，所消耗的资源与训练作业/在线推理的费用一致。只有在节点运行时实际消耗费用，当节点未运行和等待操作的过程中均不消耗。训练资源规格配置，默认使用公共资源池。

当您需要使用专属资源池时，将开关打开即可。

推理资源规格配置如下图所示：

3.2.4 消息通知

Workflow使用了消息通知服务，支持用户在事件列表中选择需要监控的状态，并在事件发生时发送消息通知。如需订阅通知消息，则打开“订阅消息”开关。

1. 打开开关后，需要先指定SMN主题名，如未创建主题名，需前往消息通知服务创建主题。
2. 支持对Workflow中单个节点、多个节点以及流的相关事件进行订阅。订阅列表中，一行代表一个节点或者整条流的订阅。如需对多个节点的状态变化获取消息，则需增加多行订阅消息。

3. 对每一个订阅对象，可以选择多个订阅事件，包含：等待输入、运行成功、异常。

3.2.5 输入与输出配置

输入配置

输入配置需要填写的参数如下表所示：

表 3-1 输入参数填写说明

输入配置参数	填写说明
数据集	选择您的数据集，或者重新创建数据集。
OBS位置	选择您的OBS及对应的桶名称。
标注任务	选择您的数据集下的标注任务。

输出配置

这里的输出选择您所创建的OBS桶，用来存放输出数据。单击“选择”，选择您的桶。

3.2.6 节点参数配置

对每个节点，开发者都可以选择暴露不同的接口，包含类型：枚举、整数、浮点数、布尔值。

3.2.7 保存配置

通过“配置”入口进入配置页面，所有的配置参数编辑完成之后，单击界面右上角的“保存配置”按钮。

配置保存成功后，单击界面右上角的“启动”按钮，出现弹窗，单击确定，工作流就会启动并进入运行页面。

完成配置后，Workflow即可运行。

3.3 启动/停止/复制/删除 Workflow

启动

当工作流当前状态为非运行态时，可以通过单击“启动”按钮运行工作流，有3种操作方式。

- 工作流列表页：单击操作栏的“启动”按钮，出现启动Workflow询问弹窗，点击确定“确定”。
- 工作流运行页面：单击右上角的“启动”按钮，出现启动Workflow询问弹窗，单击“确定”。
- 工作流参数配置页面：单击右上角的“启动”按钮，出现启动Workflow询问弹窗，单击“确定”。

停止

可以通过“停止”按钮，主动停止正在运行的工作流，有2种操作方式：

- 工作流列表页：

  当工作流处于“运行中”时，操作栏会出现“停止”按钮。单击“停止”，出现停止Workflow询问弹窗，单击“确定”。
- 进入某条运行中的工作流，单击右上角的“停止”按钮，出现停止Workflow询问弹窗，单击确定。

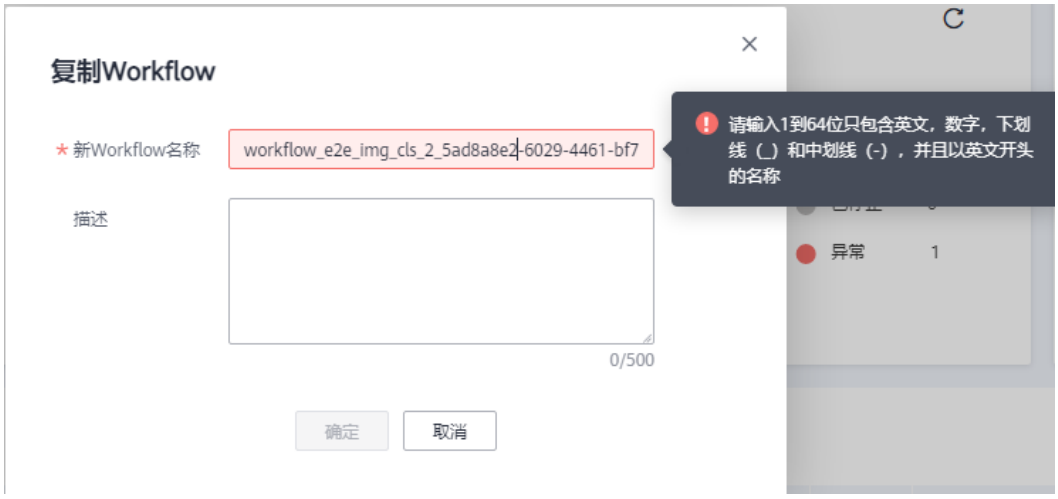
  说明

  只有处于“运行中”状态的工作流，才会出现“停止”按钮。

复制

某条工作流，目前只能存在一个正在运行的实例，如果用户想要使同一个工作流同时运行多次，可以使用复制工作流的功能。单击列表页的操作栏“更多”，选择“复制”，出现复制Workflow弹窗，新名称会自动生成（生成规则：原工作流名称 + '_copy'）。

用户也可以自行修改新工作流名称，但会有校验规则验证新名称是否符合规范。



删除

删除工作流有2种方式

- 工作流列表页
  a. 单击操作栏的“更多”，选择“删除”，出现删除Workflow询问弹窗。
  b. 输入“delete”，单击“确定”，删除Workflow。

- 工作流运行页面
  单击界面右上角的“删除”，出现删除Workflow弹窗，输入“delete”，单击“确定”，删除Workflow。

3.4 查看 Workflow 运行记录

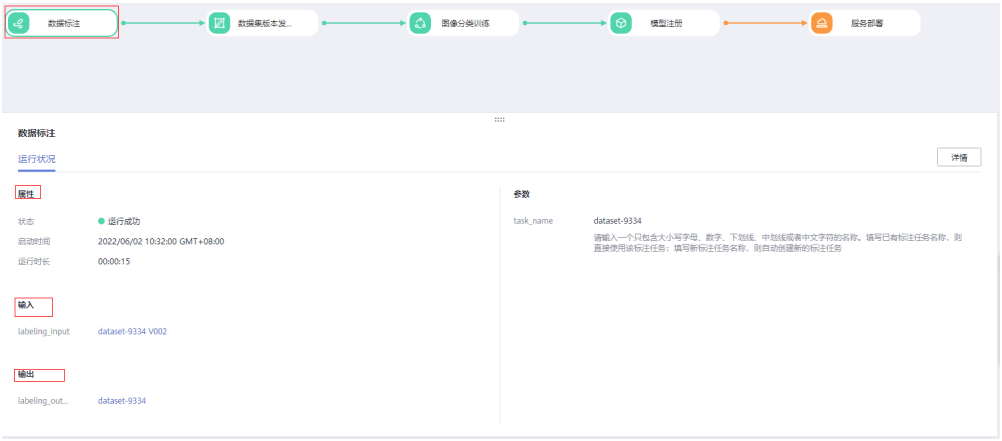
运行记录是展示某条工作流所有运行状态数据的地方。

图 3-4 查看运行记录

当单击“启动”运行工作流时，运行记录列表会自动刷新，并更新至最新一条的执行记录数据，并与DAG图和总览数据面板双向联动更新数据。每次启动后都会新增一条运行记录。

用户可以单击workflow详情页中任一节点进行节点运行状况查询。包括节点的属性（节点的运行状态、启动时间以及运行时长）、输入位置与输出位置以及参数（数据集的标注任务名称）。

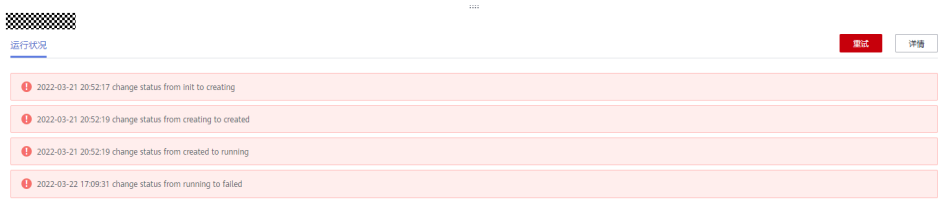
图 3-5 节点运行情况查看



3.5 重试/停止/继续运行节点

- 重试

当单个节点运行失败时，用户可以通过重试按钮重新执行当前节点，无需重新启动工作流。在重试之前您也可以前往全局配置页面进行配置修改，节点重试启动后新修改的配置信息可以在当前执行中立即生效。



- 停止

单击指定节点查看详情，可以对运行中的节点进行停止操作。

- 继续运行

对于单个节点中设置了需要运行中配置的参数时，节点运行会处于“等待输入”状态，用户完成相关数据的配置后，可单击“继续运行”按钮并确认继续执行当前节点。

4 如何开发 Workflow

4.1 基本概念与功能概览

开发者可以通过 Workflow 进行有向无环图（Directed Acyclic Graph，DAG）的开发。一个DAG是由节点和节点之间的关系描述组成的。节点之间的关系决定了节点的运行顺序，开发者通过定义节点的执行内容和节点的执行顺序定义DAG。如下图，绿色的矩形表示为一个节点，节点与节点之间的连线则是节点的关系描述。整个DAG的执行其实就是有序的任务执行模板。

当前，开发者需要对节点和节点之间的关系使用进行描述。节点在描述的时候带有其业务属性。可为：

数据标注、数据处理、数据注册、模型训练、模型评估、应用生成和应用部署等。Workflow使用声明节点运行需依赖哪些已完成的节点进行判断控制流，从而完成节点之间的关系描述和DAG图的构建。

开发者可以使用Workflow SDK开发工作流。对于熟练使用代码进行算法开发的开发者，会较快上手使用Workflow SDK进行工作流的开发，自由度也很高。

4.2 开发环境准备

4.2.1 Notebook-JupyterLab

创建 Notebook 实例

1. 登录ModelArts控制台。
2. 选择控制台区域为“华北-北京四”。

3. 在开发环境Notebook（New）中创建基于pytorch1.4-cuda10.1-cudnn7-ubuntu18.04镜像的Notebook，具体操作请参见**创建Notebook实例**章节。

图 4-1 创建 Notebook 实例

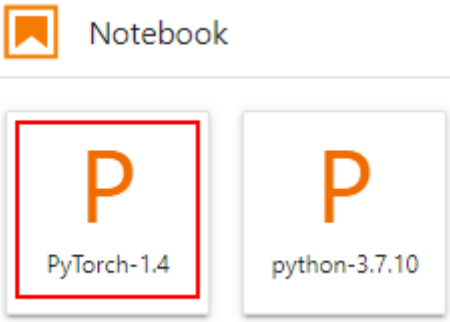
使用 JupyterLab 打开 Notebook 实例准备环境

1. 在Notebook列表中单击操作列的“打开”，进入JupyterLab页面。JupyterLab操作请参见**JupyterLab简介及常用操作**。

图 4-2 使用 JupyterLab 打开

2. 创建一个ipynb文件。

图 4-3 创建一个 ipynb 文件



3. 在Notebook的第一个cell运行如下命令进行环境准备。

```
!rm modelarts*.whl
!wget -N https://modelarts-cnnorth4-market.obs.cn-north-4.myhuaweicloud.com/workflow-apps/modelarts-1.3.0-py2.py3-none-any.whl
!wget -N https://modelarts-cnnorth4-market.obs.cn-north-4.myhuaweicloud.com/workflow-apps/v0.1.1/85719177/modelarts_workflow-0.1.1-py2.py3-none-any.whl
```



```
!pip uninstall -y modelarts modelarts-workflow
!pip install modelarts-1.3.0-py2.py3-none-any.whl
!pip install modelarts_workflow-0.1.1-py2.py3-none-any.whl
```

环境安装成功验证：

创建一个新的cell，运行如下命令，若能成功导入，则表示环境已安装成功：

```
from modelarts import workflow as wf
```

若导入失败，建议重新执行安装命令，或者重启kernel后再次执行安装命令。

4.2.2 本地 IDE(PyCharm)

创建 Notebook 实例

1. 登录ModelArts控制台。
2. 选择控制台区域为“华北-北京四”。
3. 在开发环境Notebook（New）中创建基于pytorch1.4-cuda10.1-cudnn7-ubuntu18.04镜像的Notebook，并开启SSH远程开发，具体操作请参见**创建Notebook实例**章节。

图 4-4 创建 Notebook 实例

使用本地 IDE 如 PyCharm 远程连接 Notebook 准备环境

使用本地IDE如PyCharm开发工作流，您只需专注于本地代码开发即可。PyCharm连接Notbook操作请参见**配置本地IDE（PyCharm ToolKit连接）**或**配置本地IDE（PyCharm手动连接）**。

在本地IDE的终端运行如下命令进行环境准备。Python版本要求:3.6.x/3.7.x.

```
rm modelarts*.whl
wget -N https://modelarts-cnnorth4-market.obs.cn-north-4.myhuaweicloud.com/workflow-apps/
```

```
modelarts-1.3.0-py2.py3-none-any.whl
wget -N https://modelarts-cnnorth4-market.obs.cn-north-4.myhuaweicloud.com/workflow-apps/v0.1.1/85719177/modelarts_workflow-0.1.1-py2.py3-none-any.whl

pip uninstall -y modelarts modelarts-workflow
!pip install modelarts-1.3.0-py2.py3-none-any.whl
!pip install modelarts_workflow-0.1.1-py2.py3-none-any.whl
```

使用本地IDE进行开发时，配置好Pycharm环境后，在代码中还需要使用AK-SK认证模式，示例代码如下。

```
from modelarts.session import Session
session = Session(access_key='***', secret_key='***', project_id='***', region_name='***')
```

4.3 开发第一条 Workflow

本章节以图像分类为例，阐述工作流的完整开发过程。您可以前往AI Gallery订阅**图像分类-ResNet_v1_50工作流**，根据工作流下方的使用指南进行订阅体验。

准备工作

- 准备一个图像分类算法，将该算法发布至AI Gallery并订阅（或者可以直接从AI Gallery订阅一个**图像分类算法**）。
- 准备一个图片类型的数据集，可从AI Gallery直接下载（例如：**生活垃圾分类**）

操作步骤

1. 图像分类工作流构建（只需将算法的订阅ID替换成您真实的订阅ID即可）。

```
from modelarts import workflow as wf

# 定义统一存储对象管理输出目录
output_storage = wf.data.OutputStorage(name="output_storage", description="输出目录统一配置")

# 创建标注任务
data = wf.data.DatasetPlaceholder(name="input_data")

label_step = wf.steps.LabelingStep(
    name="labeling",
    title="数据标注",
    properties=wf.steps.LabelTaskProperties(
        task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,
        task_name=wf.Placeholder(name="task_name", placeholder_type=wf.PlaceholderType.STR,
description="请输入一个只包含大小写字母、数字、下划线、中划线或者中文字符的名称。填写已有标注任务名称，则直接使用该标注任务；填写新标注任务名称，则自动创建新的标注任务")
    ),
    inputs=wf.steps.LabelingInput(name="labeling_input", data=data),
    outputs=wf.steps.LabelingOutput(name="labeling_output"),
)

# 对标注任务进行发布
release_step = wf.steps.ReleaseDatasetStep(
    name="release",
```

```
    title="数据集版本发布",
    inputs=wf.steps.ReleaseDatasetInput(name="input_data",
data=label_step.outputs["labeling_output"].as_input()),
    outputs=wf.steps.ReleaseDatasetOutput(name="labeling_output",
dataset_version_config=wf.data.DatasetVersionConfig(train_evaluate_sample_ratio="0.8")),
    depend_steps=[label_step]
)


# 创建训练作业
job_step = wf.steps.JobStep(
    name="training_job",
    title="图像分类训练",
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="fake_subscription_id", # 订阅算法的ID
        item_version_id="10.0.0", # 订阅算法的版本ID
        parameters=[
            wf.AlgorithmParameters(name="task_type", value="image_classification_v2"),
            wf.AlgorithmParameters(name="model_name", value="resnet_v1_50"),
            wf.AlgorithmParameters(name="do_train", value="True"),
            wf.AlgorithmParameters(name="do_eval_along_train", value="True"),
            wf.AlgorithmParameters(name="variable_update", value="horovod"),
            wf.AlgorithmParameters(name="learning_rate_strategy",
value=wf.Placeholder(name="learning_rate_strategy", placeholder_type=wf.PlaceholderType.STR,
default="0.002", description="训练的学习率策略(10:0.001,20:0.0001代表0-10个epoch学习率0.001，
10-20epoch学习率0.0001),如果不指定epoch, 会根据验证精度情况自动调整学习率，并当精度没有明显提
升时，训练停止")),
            wf.AlgorithmParameters(name="batch_size", value=wf.Placeholder(name="batch_size",
placeholder_type=wf.PlaceholderType.INT, default=64, description="每步训练的图片数量（单卡）")),
            wf.AlgorithmParameters(name="eval_batch_size",
value=wf.Placeholder(name="eval_batch_size", placeholder_type=wf.PlaceholderType.INT, default=64,
description="每步验证的图片数量（单卡）")),
            wf.AlgorithmParameters(name="evaluate_every_n_epochs",
value=wf.Placeholder(name="evaluate_every_n_epochs", placeholder_type=wf.PlaceholderType.FLOAT,
default=1.0, description="每训练n个epoch做一次验证")),
            wf.AlgorithmParameters(name="save_model_secs",
value=wf.Placeholder(name="save_model_secs", placeholder_type=wf.PlaceholderType.INT, default=60,
description="保存模型的频率（单位：s)")),
            wf.AlgorithmParameters(name="save_summary_steps",
value=wf.Placeholder(name="save_summary_steps", placeholder_type=wf.PlaceholderType.INT,
default=10, description="保存summary的频率（单位：步)")),
            wf.AlgorithmParameters(name="log_every_n_steps",
value=wf.Placeholder(name="log_every_n_steps", placeholder_type=wf.PlaceholderType.INT,
default=10, description="打印日志的频率（单位：步)")),
            wf.AlgorithmParameters(name="do_data_cleaning",
value=wf.Placeholder(name="do_data_cleaning", placeholder_type=wf.PlaceholderType.STR,
default="True", description="是否做数据清洗, 数据格式异常会导致训练失败，建议开启，保证训练稳定
性。数据量过大时，数据清洗可能耗时较久，可自行线下清洗（支持BMP.JPEG,PNG格式, RGB三通道）。
建议用JPEG格式数据")),
            wf.AlgorithmParameters(name="use_fp16", value=wf.Placeholder(name="use_fp16",
placeholder_type=wf.PlaceholderType.STR, default="True", description="是否使用混合精度, 混合精度可
以加速训练，但是可能会造成一点精度损失，若对精度无极严格的要求，建议开启")),
            wf.AlgorithmParameters(name="xla_compile", value=wf.Placeholder(name="xla_compile",
placeholder_type=wf.PlaceholderType.STR, default="True", description="是否开启xla编译，加速训练，
默认启用")),
            wf.AlgorithmParameters(name="data_format", value=wf.Placeholder(name="data_format",
placeholder_type=wf.PlaceholderType.ENUM, default="NCHW", enum_list=["NCHW", "NHWC"],
description="输入数据类型，NHWC表示channel在最后，NCHW表channel在最前，默认值NCHW（速度
有提升）")),
            wf.AlgorithmParameters(name="best_model", value=wf.Placeholder(name="best_model",
placeholder_type=wf.PlaceholderType.STR, default="True", description="是否在训练过程中保存并使用精
度最高的模型，而不是最新的模型。默认值True,保存最优模型。在一定误差范围内，最优模型会保存最新
的高精度模型")),
            wf.AlgorithmParameters(name="jpeg_preprocess",
value=wf.Placeholder(name="jpeg_preprocess", placeholder_type=wf.PlaceholderType.STR,
default="True", description="是否使用jpeg预处理加速算子(仅支持jpeg格式数据)，可加速数据读取，提升
性能，默认启用。若数据格式不是jpeg格式，开启数据清洗功能即可使用"))
        ]
    ),
```



```
    inputs=[wf.steps.JobInput(name="data_url",
data=release_step.outputs["labeling_output"].as_input())],
    outputs=[wf.steps.JobOutput(name="train_url",
                                obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/
train_output/")))],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="training_flavor",
                                  placeholder_type=wf.PlaceholderType.JSON,
                                  description="训练资源规格"
                                  )
        )
    ),
    depend_steps=[release_step]
)

model_name = wf.Placeholder(name="model_name", placeholder_type=wf.PlaceholderType.STR,
description="请输入一个1至64位且只包含大小写字母、中文、数字、中划线或者下划线的名称。工作流第一次运行建议填写新的模型名称，后续运行会自动在该模型上新增版本")

# 模型注册
model_step = wf.steps.ModelStep(
    name="model_step",
    title="模型注册",
    inputs=[wf.steps.ModelInput(name="model_input",
                                data=job_step.outputs["train_url"].as_input())],
    outputs=[wf.steps.ModelOutput(name="model_output",
                                  model_config=wf.steps.ModelConfig(model_name=model_name,
model_type="TensorFlow"))],
    depend_steps=[job_step]
)


# 服务部署
service_step = wf.steps.ServiceStep(
    name="service_step",
    title="服务部署",
    inputs=[wf.steps.ServiceInput(name="service_input",
                                  data=wf.data.ServiceInputPlaceholder(name="service_model",
model_name=model_name))],
    outputs=[wf.steps.ServiceOutput(name="service_output")],
    depend_steps=[model_step]
)

# 构建工作流对象
workflow = wf.Workflow(name="image-classification-ResNeSt",
                       desc="this is a image classification workflow",
                       steps=[label_step, release_step, job_step, model_step, service_step],
                       storages=[output_storage]
                       )
```

2. 在开发态进行工作流的调试。

   使用run模式运行全部节点

   ```
   workflow.run(steps=[label_step, release_step, job_step, model_step, service_step], experiment_id="实验记录ID")
   ```

3. 调试完成后发布至运行态进行配置运行。

   ```
   workflow.release()
   # 上述命令执行完成后，若日志打印显示发布成功，则可前往MA的workflow页面中查看新发布的工作流。
   ```

4. 分享**工作流至AI Gallery**给客户使用。

4.4 节点的数据类型

该章节主要介绍workflow中使用的相关数据对象，涵盖了输入、输出、参数以及统一存储四种数据类型，用户在构建节点时可以根据需要自行选择。

4.4.1 输入数据类型

主要分为两种：

- 真实的数据对象，在工作流构建时直接指定：

  Dataset：用于定义已有的数据集，常用于数据标注，模型训练等场景

  LabelTask: 用于定义已有的标注任务，常用于数据标注，数据集版本发布等场景

  OBSPath：用于定义指定的OBS路径，常用于模型训练，数据集导入，模型导入等场景

  ServiceData：用于定义一个已有的服务，只用于服务更新的场景

  SWRImage：用于定义已有的SWR路径，常用于模型注册场景

- 占位符式的数据对象，在工作流运行时指定：

  DatasetPlaceholder：用于定义在运行时需要确定的数据集，常用于数据标注，模型训练等场景

  LabelTaskPlaceholder: 用于定义在运行时需要确定的标注任务，常用于数据标注，数据集版本发布等场景

  OBSPlaceholder：用于定义在运行时需要确定的OBS路径，常用于模型训练，数据集导入，模型导入等场景

  ServiceUpdataPlaceholder: 用于定义在运行时需要确定的已有服务，只用于服务更新的场景

  SWRImagePlaceholder ： 用于定义在运行时需要确定的SWR路径，常用于模型注册场景

  ServiceInputPlaceholder：用于定义在运行时需要确定服务部署所需的模型相关信息，只用于服务部署及服务更新场景

Dataset

属性	描述	是否必填	数据类型
dataset_name	数据集名称	是	str
version_name	数据集版本名称	否	str

示例：

```
example = Dataset(dataset_name = "dataset_name", version_name = "dataset_version_name")
# 通过ModelArts的数据集，获取对应的数据集名称及相应的版本名称。
```

说明

当Dataset对象作为节点的输入时，需根据业务需要自行决定是否填写version_name字段（比如LabelingStep, ReleaseDatasetStep不需要填写，JobStep必须填写）

LabelTask

属性	描述	是否必填	数据类型
dataset_name	数据集名称	是	str
task_name	标注任务名称	是	str

示例：

```
example = LabelTask(dataset_name = "dataset_name", task_name = "label_task_name")
# 通过ModelArts的新版数据集，获取对应的数据集名称及相应的标注任务名称
```

OBSPath

属性	描述	是否必填	数据类型
obs_path	OBS路径	是	str

示例：

```
example = OBSPath(obs_path = "obs_path")
# 通过对象存储服务，获取OBS路径值
```

ServiceData

属性	描述	是否必填	数据类型
service_id	服务的ID	是	str

示例：

```
example = ServiceData(service_id = "service_id")
# 通过ModelArts的在线服务，获取对应服务的服务ID，描述指定的在线服务。用于服务更新的场景
```

SWRImage

属性	描述	是否必填	数据类型
swr_path	容器镜像的SWR路径	是	str

示例：

```
example = SWRImage(swr_path = "swr_path")
# 容器镜像地址，用于模型注册节点的输入
```

DatasetPlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str
data_type	数据类型	否	DataTypeEnum

示例：

```
example = DatasetPlaceholder(name = "dataset_placeholder_name", data_type =
DataTypeEnum.IMAGE_CLASSIFICATION )
# 数据集对象的占位符形式，可以通过指定data_type限制数据集的数据类型
```

OBSPlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str
object_type	OBS路径，可以是文件或者文件夹	是	str

示例：

```
example = OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" )
# OBS对象的占位符形式，object_type只支持两种类型， "file" 以及 "directory"
```

LabelTaskPlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str

示例：

```
example = LabelTaskPlaceholder(name = "label_task_placeholder_name")
# LabelTask对象的占位符形式
```

ServiceUpdatePlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str

示例：

```
example = ServiceUpdatePlaceholder(name = "service_update_placeholder_name")
# ServiceData对象的占位符形式，用于服务更新节点的输入
```

SWRImagePlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str

示例：


```
example = SWRImagePlaceholder(name = "swrimage_placeholder_name" )
# SWRImage对象的占位符形式，用于模型注册节点的输入
```

ServiceInputPlaceholder

属性	描述	是否必填	数据类型
name	名称	是	str
model_name	模型名称	是	str
model_version	模型版本	否	str
envs	环境变量	否	dict
delay	服务部署相关信息是否在节点运行时配置，默认为True	否	bool

示例：

```
example = ServiceInputPlaceholder(name = "service_input_placeholder_name" , model_name = "model_name")
# 用于服务部署或者服务更新节点的输入
```

4.4.2 输出数据类型

主要分为四种：

- OBSOutputConfig：输出到OBS时的相关配置信息，常用于模型训练等场景
- DatasetVersionConfig：数据集版本发布时版本相关的配置信息，只用于数据集版本发布场景
- ModelConfig：模型注册相关的配置信息，只用于模型注册场景
- ServiceConfig：服务部署相关的配置信息，只用于服务部署场景

OBSOutputConfig

属性	描述	是否必填	数据类型
obs_path	OBS路径	是	str
metric_file	存储metric信息的文件名称	否	str

示例：

```
example = OBSOutputConfig(obs_path = "obs_path" , metric_file = "metric_file_name")
# 主要在作业类型节点的输出中使用
```

说明

当您后续需要使用输出的metric信息时，应当在obs_path路径下将metric信息以json体的格式保存在文件中，文件名称由metric_file字段指定。

DatasetVersionConfig

属性	描述	是否必填	数据类型
version_name	数据集版本名称，推荐使用类似V001的格式，不填则默认从V001往上递增	否	str或者Placeholder
version_format	版本格式，默认为"Default"，也可支持"CarbonData"	否	str
train_evaluate_sample_ratio	训练-验证集比例，默认值为"1.00"；取值范围为0-1.00， 例如"0.8"表示训练集比例为80%，验证集比例为20%	否	str
clear_hard_property	是否清除难例，默认为True	否	bool
remove_sample_usage	是否清除数据集已有的usage信息，默认为True	否	bool
label_task_type	标注任务的类型。当输入是数据集时，该字段必填，用来指定数据集版本的标注场景；输入是标注任务时该字段不用填写	否	LabelTaskTypeEnum
description	版本描述信息	否	str

示例：

```
example = DatasetVersionConfig(version_name = "V001", version_format = "Default",
train_evaluate_sample_ratio = "1.00", clear_hard_property = True, description = "this is test")
# 在数据集版本发布节点的输出中使用
```

说明

如果您没有特殊需求，则可直接使用内置的默认值，例如example = DatasetVersionConfig()

ModelConfig

属性	描述	是否必填	数据类型
model_type	模型的类型，支持的格式有（"TensorFlow", "MXNet", "Caffe", "Spark_MLlib", "Scikit_Learn", "XGBoost", "Image", "PyTorch", "Template"）	是	str
model_name	模型的名称，支持1-64位可见字符（含中文），名称可以包含字母、中文、数字、中划线、下划线	否	str
model_version	模型的版本，格式需为“数值.数值.数值”，其中数值为1-2位正整数, 注意：版本不可以出现例如01.01.01等以0开头的版本号形式	否	str, Placeholder
runtime	模型运行时环境，runtime可选值与model_type相	否	str, Placeholder
description	模型备注信息，1-100位长度，不能包含&!'"<>=	否	str
execution_code	执行代码存放的OBS地址，默认值为空，名称固定为“customize_service.py”。推理代码文件需存放在模型“model”目录。该字段不需要填，系统也能自动识别出model目录下的推理代码	否	str
dependencies	推理代码及模型需安装的包，默认为空。从配置文件读取	否	str

属性	描述	是否必填	数据类型
model_metrics	模型精度信息，从配置文件读取	否	str
apis	模型所有的apis入参出参信息（选填），从配置文件中解析出来	否	str
initial_config	模型配置相关数据	否	dict

示例：

```
example = ModelConfig(model_type = "TensorFlow", model_name = "model_test")
# 主要在模型注册节点的输出中使用，多次运行模型版本会自动增加
```

ServiceConfig

属性	描述	是否必填	数据类型
infer_type	推理方式：取值为real-time/batch/edge ● real-time代表在线服务，将模型部署为一个Web Service。 ● batch为批量服务，批量服务可对批量数据进行推理，完成数据处理后自动停止。 ● edge表示边缘服务，通过华为云智能边缘平台，在边缘节点将模型部署为一个Web Service，需提前在IEF（智能边缘服务）创建好节点。	是	str
description	服务备注，默认为空，不超过100个字符	否	str
workspace_id	服务所属的工作空间ID，默认为0，代表默认工作空间	否	str, Placeholder
vpc_id	在线服务实例部署的虚拟私有云ID，默认为空，此时ModelArts会为每个用户分配一个专属的VPC，用户之间隔离；如需要在服务实例中访问名下VPC内的其他服务组件，则可配置此参数为对应VPC的ID。VPC一旦配置，不支持修改。当vpc_id与cluster_id一同配置时，只有专属资源池参数生效。	否	str

属性	描述	是否必填	数据类型
subnet_network_id	子网的网络ID，默认为空，当配置了vpc_id则此参数必填。需填写虚拟私有云控制台子网详情中显示的“网络ID”。通过子网可提供与其他网络隔离的、可以独享的网络资源。	否	str
security_group_id	安全组，默认为空，当配置了vpc_id则此参数必填。安全组起着虚拟防火墙的作用，为服务实例提供安全的网络访问控制策略。安全组须包含至少一条入方向规则，对协议为TCP、源地址为0.0.0.0/0、端口为8080的请求放行。	否	str
cluster_id	专属资源池ID，默认为空，不使用专属资源池；使用专属资源池部署服务时需确保集群状态正常；配置此参数后，则使用集群的网络配置，vpc_id参数不生效;与下方real-time config中的cluster_id同时配置时，优先使用real-time config中的cluster_id参数。	否	str
additional_properties	附加的相关配置信息。	否	dict
apps	服务部署支持APP认证。	否	str, Placeholder, list

示例：

```
example = ServiceConfig()
# 主要在服务部署节点的输出中使用
```

如果您没有特殊需求，可直接使用内置的默认值。

4.4.3 参数类型

参数类型使用Placeholder对象来表示，以占位符的形式实现用户数据运行时配置的能力；该对象的具体字段及含义如下表所示：

Placeholder

属性	描述	是否必填	数据类型
name	参数名称，需要全局唯一	是	str

属性	描述	是否必填	数据类型
placeholder_type	参数类型，与真实数据类型的映射关系如下： PlaceholderType.INT -> int PlaceholderType.STR -> str PlaceholderType.BOOL -> bool PlaceholderType.FLOAT -> float PlaceholderType.ENUM -> Enum PlaceholderType.JSON -> dict	是	PlaceholderType
default	参数默认值，数据类型需要与placeholder_type一致	否	Any
placeholder_format	支持的format格式数据，当前支持obs, train_flavor	否	str
delay	参数是否运行时输入，默认为False，在工作流启动运行前进行配置；设置为True，则在使用的相应节点运行时卡点配置	否	bool
description	参数描述信息	否	str
enum_list	参数枚举值列表，只有当参数类型为PlaceholderType.ENUM时才需要填写	否	list

使用示例：

```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR,
default="default_value", description="这是一个str类型的参数")
```

4.4.4 统一存储

4.4.4.1 功能介绍

统一存储主要用于目录管理，帮助用户统一管理一个工作流中的所有存储路径，主要分为以下两个功能：

- 输入目录管理：开发者在编辑开发工作流时可以对所有数据的存储路径做统一管理，规定用户按照自己的目录规划来存放数据，而存储的根目录可以根据用户自己的需求自行配置。
- 输出目录管理：开发者在编辑开发工作流时可以对所有的输出路径做统一管理，用户无需手动创建输出目录，只需要在工作流运行前配置存储根路径，并且可以根据开发者的目录编排规则在指定目录下查看输出的数据信息；此外同一个工作流的多次运行将输出到不同的目录下，对不同的执行做了很好的数据隔离。

4.4.4.2 常用方式

1. **InputStorage**（路径拼接）

   该对象主要用于帮助用户统一管理输入的目录，使用示例如下：

   ```
   import modelarts.workflow as wf
   storage = wf.data.InputStorage(name="storage_name", title="title_info",
   description="description_info") # name字段必填，title, description可选填
   input_data = wf.data.OBSPath(obs_path = storage.join("directory_path")) # 注意，如果是目录则最后需要加"/"，例如：storage.join("/input/data/")

   工作流运行时，若storage对象配置的根路径为"/root/"，则最后得到的路径为"/root/directory_path"
   ```

2. **OutputStorage**（目录创建）

   该对象主要用于帮助用户统一管理输出的目录，保证工作流每次执行输出到新目录，使用示例如下：

   ```
   import modelarts.workflow as wf
   storage = wf.data.OutputStorage(name="storage_name", title="title_info",
   description="description_info") # name字段必填，title, description可选填
   output_path = wf.data.OBSOutputConfig(obs_path = storage.join("directory_path")) # 注意，只能创建目录，不能创建文件

   工作流运行时，若storage对象配置的根路径为"/root/"，则系统自动创建相对目录，最后得到的路径为"/root/执行ID/directory_path"
   ```

4.4.4.3 进阶用法

Storage

该对象是InputStorage和OutputStorage的基类，包含了两者的所有能力，可以供用户灵活使用。

属性	描述	是否必填	数据类型
name	名称	是	str
title	不填默认使用name的值	否	str
description	描述信息	否	str
create_dir	是否创建目录，默认为False	否	bool

属性	描述	是否必填	数据类型
with_execution_id	创建目录时是否拼接execution_id，默认为False	否	bool

使用示例如下：

- 实现**InputStorage**相同的能力

```
import modelarts.workflow as wf
# 构建一个Storage对象, with_execution_id=False, create_dir=False
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info",
with_execution_id=False, create_dir=False)
input_data = wf.data.OBSPath(obs_path = storage.join("directory_path")) # 注意，如果是目录则最后需要加"/"，例如：storage.join("/input/data/")

工作流运行时，若storage对象配置的根路径为"/root/"，则最后得到的路径为"/root/directory_path"
```

- 实现**OutputStorage**相同的能力

```
import modelarts.workflow as wf
# 构建一个Storage对象, with_execution_id=True, create_dir=True
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info",
with_execution_id=True, create_dir=True)
output_path = wf.data.OBSOutputConfig(obs_path = storage.join("directory_path")) # 注意，只能创建目录，不能创建文件

工作流运行时，若storage对象配置的根路径为"/root/"，则系统自动创建相对目录，最后得到的路径为"/root/执行ID/directory_path"
```

- 通过join方法的参数实现同一个Storage的不同用法

```
import modelarts.workflow as wf
# 构建一个Storage对象, 并且假设Storage配置的根目录为"/root/"
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info",
with_execution_id=False, create_dir=False)
input_data1 = wf.data.OBSPath(obs_path = storage) # 得到的路径为：/root/
input_data2 = wf.data.OBSPath(obs_path = storage.join("directory_path")) # 得到的路径为：/root/directory_path
output_path1 = wf.data.OBSOutputConfig(obs_path = storage.join(directory="directory_path",
with_execution_id=False, create_dir=True)) # 系统自动创建目录，得到的路径为：/root/directory_path
output_path2 = wf.data.OBSOutputConfig(obs_path = storage.join(directory="directory_path",
with_execution_id=True, create_dir=True)) # 系统自动创建目录，得到的路径为：/root/执行ID/directory_path
```

Storage可实现链式调用

使用示例如下：

```
import modelarts.workflow as wf
# 构建一个基类Storage对象, 并且假设Storage配置的根目录为"/root/"
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info",
with_execution_id=False, create_dir=Fals)
input_storage = storage.join("directory_path_1") # 得到的路径为：/root/directory_path_1
input_storage_next = input_storage.join("directory_path_2") # 得到的路径为： /root/directory_path_1/directory_path_2
```

4.4.4.4 使用案例

```
from modelarts import workflow as wf

# 构建一个InputStorage对象, 并且假设配置的根目录为"/root/input-data/"
input_storage = wf.data.InputStorage(name="input_storage_name", title="title_info",
description="description_info") # name字段必填，title, description可选填
```



```
# 构建一个OutputStorage对象, 并且假设配置的根目录为"/root/output/"
output_storage = wf.data.OutputStorage(name="output_storage_name", title="title_info",
description="description_info") # name字段必填，title, description可选填

# 通过JobStep来定义一个训练节点，输入数据来源为OBS，并将训练结果输出到OBS中
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(subscription_id="subscription_ID",
item_version_id="item_version_ID"), # 训练使用的算法对象，示例中使用AIGallery订阅的算法
    inputs=[
        wf.steps.JobInput(name="data_url_1", data=wf.data.OBSPath(obs_path = input_storage.join("/
dataset1/new.manifest"))),  # 获得的路径为：/root/input-data/dataset1/new.manifest
        wf.steps.JobInput(name="data_url_2", data=wf.data.OBSPath(obs_path = input_storage.join("/
dataset2/new.manifest")))   # 获得的路径为：/root/input-data/dataset2/new.manifest
    ],
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/model/"))), # 训练输出的路径为：/
root/output/执行ID/model/
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
                flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")
        ),
        log_export_path=wf.steps.job_step.LogExportPath(obs_url=output_storage.join("/logs/"))  # 日志输出的
路径为：/root/output/执行ID/logs/
    )# 训练资源规格信息
)

# 定义一个只包含job_step的工作流
workflow = wf.Workflow(
    name="test-workflow",
    desc="this is a test workflow",
    steps=[job_step],
    storages=[input_storage, output_storage] # 注意在整个工作流中使用到的Storage对象需要在此处添加
)
```

4.4.4.5 根路径配置

开发态配置

调用工作流对象的run方法，在开始运行时展示输入框，等待用户输入，如下所示：

运行态配置：

调用工作流对象的release方法将工作流发布到运行态，在Modelarts管理控制台，“总览>workflow”找到相应的工作流进行根路径配置，如下所示：

4.5 Workflow 支持的节点类型

4.5.1 数据集创建节点

功能介绍

通过对ModelArts数据集能力进行封装，实现新版数据集的创建功能。

应用场景

主要用于通过创建数据集对已有数据（已标注/未标注）进行统一管理的场景，后续可接数据集导入节点或者数据集标注节点。

最佳使用案例

您可以使用**CreateDatasetStep**来构建数据集创建节点，定义**CreateDatasetStep**示例如下：

枚举类型	枚举值
DataTypeEnum	IMAGE TEXT AUDIO TABULAR VIDEO FREE_FORMAT

枚举类型	枚举值
AnnotationFormatEnum	MA_IMAGE_CLASSIFICATION_V1 MA_IMAGENET_V1 MA_PASCAL_VOC_V1 YOLO MA_IMAGE_SEGMENTATION_V1 MA_TEXT_CLASSIFICATION_COMBINE_V1 MA_TEXT_CLASSIFICATION_V1 MA_AUDIO_CLASSIFICATION_DIR_V1

- **基于未标注的数据创建数据集**

  数据准备：存储在OBS文件夹中的未标注的数据。

```
from modelarts import workflow as wf
# 通过CreateDatasetStep将存储在OBS中的数据创建成一个新版数据集

# 定义数据集输出路径参数
dataset_output_path = wf.Placeholder(name="dataset_output_path",
placeholder_type=wf.PlaceholderType.STR, placeholder_format="obs")

# 定义数据集名称参数
dataset_name = wf.Placeholder(name="dataset_name", placeholder_type=wf.PlaceholderType.STR)

create_dataset = wf.steps.CreateDatasetStep(
    name="create_dataset",# 数据集创建节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="数据集创建", # 标题信息，不填默认使用name值
    inputs=wf.steps.CreateDatasetInput(name="input_name",
data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")),# CreateDatasetStep的输入，数据在运行时进行配置；data字段也可使用wf.data.OBSPath(obs_path="fake_obs_path")对象表示
    outputs=wf.steps.CreateDatasetOutput(name="output_name",
config=wf.data.OBSOutputConfig(obs_path=dataset_output_path)),# CreateDatasetStep的输出
    properties=wf.steps.DatasetProperties(
        dataset_name=dataset_name, # 该名称对应的数据集若不存在，则创建新的数据集;若已存在，则直接使用该名称对应的数据集
        data_type=wf.data.DataTypeEnum.IMAGE, # 数据集对应的数据类型, 示例为图像
    )
)
# 注意dataset_name这个参数配置的数据集名称需要用户自行确认在该账号下未被他人使用，否则会导致期望的数据集未被创建，而后续节点错误使用了他人创建的数据集
```

- **基于已标注的数据创建数据集，并导入标注信息。**

  数据准备：存储在OBS文件夹中的已标注数据。

  OBS目录导入已标注数据的规范：可参见**OBS目录导入数据规范说明**。

```
from modelarts import workflow as wf
# 通过CreateDatasetStep将存储在OBS中的数据创建成一个新版数据集

# 定义数据集输出路径参数
dataset_output_path = wf.Placeholder(name="dataset_placeholder_name",
placeholder_type=wf.PlaceholderType.STR, placeholder_format="obs")

# 定义数据集名称参数
dataset_name = wf.Placeholder(name="dataset_placeholder_name",
```



```
placeholder_type=wf.PlaceholderType.STR)

create_dataset = wf.steps.CreateDatasetStep(
    name="create_dataset",# 数据集创建节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="数据集创建", # 标题信息，不填默认使用name值
    inputs=wf.steps.CreateDatasetInput(name="input_name", data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")),# CreateDatasetStep的输入，数据在运行时进行配置；data字段也可使用wf.data.OBSPath(obs_path="fake_obs_path")对象表示
    outputs=wf.steps.CreateDatasetOutput(name="output_name", config=wf.data.OBSOutputConfig(obs_path=dataset_output_path)),# CreateDatasetStep的输出
    properties=wf.steps.DatasetProperties(
        dataset_name=dataset_name, # 该名称对应的数据集若不存在，则创建新的数据集;若已存在，则直接使用该名称对应的数据集
        data_type=wf.data.DataTypeEnum.IMAGE, # 数据集对应的数据类型, 示例为图像
        import_config=wf.steps.ImportConfig(
            annotation_format_config=[
                wf.steps.AnnotationFormatConfig(
                    format_name=wf.steps.AnnotationFormatEnum.MA_IMAGE_CLASSIFICATION_V1, # 已标注数据的标注格式
                    scene=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION) # 标注的场景类型
            ]
        )
    )
)
# 注意dataset_name这个参数配置的数据集名称需要用户自行确认在该账号下未被他人使用，否则会导致期望的数据集未被创建，而后续节点错误使用了他人创建的数据集
```

4.5.2 数据集标注节点

功能介绍

通过对ModelArts数据集能力进行封装，实现数据集的标注功能。数据集标注节点主要用于创建标注任务或对已有的标注任务进行卡点标注。

应用场景

主要用于需要对数据进行人工标注的场景。

最佳使用案例

您可以使用**LabelingStep**来创建数据集标注节点，该节点有两种使用方式。

1. 基于用户指定的数据集创建标注任务，并等待用户标注完成
2. 用户基于指定的标注任务进行标注。

定义**LabelingStep**示例如下：

枚举类型	枚举值
LabelTaskTypeEnum	IMAGE_CLASSIFICATION OBJECT_DETECTION IMAGE_SEGMENTATION TEXT_CLASSIFICATION NAMED_ENTITY_RECOGNITION TEXT_TRIPLE AUDIO_CLASSIFICATION SPEECH_CONTENT SPEECH_SEGMENTATION DATASET_TABULAR VIDEO_ANNOTATION FREE_FORMAT

- **用户基于指定的数据集创建标注任务，并等待用户标注完成**。

  使用场景：

  - 用户只创建了一个未标注完成的数据集，需要在工作流运行时对数据进行人工标注。
  - 可以放在数据集导入节点之后，对导入的新数据进行人工标注。

  数据准备：提前在ModelArts管理控制台创建一个数据集。

```
from modelarts import workflow as wf
# 通过LabelingStep给输入的数据集对象创建新的标注任务，并等待用户标注完成

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 定义标注任务的名称参数
task_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

labeling = wf.steps.LabelingStep(
    name="labeling", # 数据集标注节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="数据集标注", # 标题信息，不填默认使用name值
    properties=wf.steps.LabelTaskProperties(
        task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,  # 标注任务的类型，以图像分类为例
        task_name=task_name  # 该名称对应的标注任务若不存在则创建，若存在则直接使用该任务
    ),
    inputs=wf.steps.LabelingInput(name="input_name", data=dataset), # LabelingStep的输入，数据集对象在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="fake_dataset_name")表示
    outputs=wf.steps.LabelingOutput(name="output_name"), # LabelingStep的输出
)
```

- **用户基于指定的标注任务进行标注**

  使用场景：

  - 用户基于数据集自主创建了一个标注任务，需要在工作流运行时对数据进行人工标注。

- 可以放在数据集导入节点之后，对导入的新数据进行人工标注。

数据准备：提前在ModelArts管理控制台，基于使用的数据集创建一个标注任务。

```
from modelarts import workflow as wf
# 用户输入标注任务，等待用户标注完成

# 定义数据集的标注任务对象
label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

labeling = wf.steps.LabelingStep(
    name="labeling", # 数据集标注节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="数据集标注", # 标题信息，不填默认使用name值
    inputs=wf.steps.LabelingInput(name="input_name", data=label_task), # LabelingStep的输入，标注任务对象在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")来表示
    outputs=wf.steps.LabelingOutput(name="output_name"), # LabelingStep的输出
)
```

- **基于数据集创建节点，构建数据标注节点**

  使用场景：数据集创建节点的输出作为数据集数据标注节点的输入。

```
from modelarts import workflow as wf

# 定义标注任务的名称参数
task_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

labeling = wf.steps.LabelingStep(
    name="labeling", # 数据集标注节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="数据集标注", # 标题信息，不填默认使用name值
    properties=wf.steps.LabelTaskProperties(
        task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,  # 标注任务的类型，以图像分类为例
        task_name=task_name  # 该名称对应的标注任务若不存在则创建，若存在则直接使用该任务
    ),
    inputs=wf.steps.LabelingInput(name="input_name", data=create_dataset.outputs["create_dataset_output"].as_input()), # LabelingStep的输入，data数据来源为数据集创建节点的输出
    outputs=wf.steps.LabelingOutput(name="output_name"), # LabelingStep的输出
    depend_steps=create_dataset # 依赖的数据集创建节点对象
)
# create_dataset是 wf.steps.CreateDatasetStep的一个实例，create_dataset_output是wf.steps.CreateDatasetOutput的name字段值
```

4.5.3 数据集导入节点

功能介绍

通过对ModelArts数据集能力进行封装，实现数据集的数据导入功能。数据集导入节点主要用于将指定路径下的数据导入到数据集或者标注任务中。

应用场景

- 适用于数据不断迭代的场景，可以将一些新增的原始数据或者已标注数据导入到标注任务中，并通过后续的数据集标注节点进行标注。
- 对于一些已标注好的原始数据，可以直接导入到数据集或者标注任务中，并通过后续的数据集版本发布节点获取带有版本信息的数据集对象。

最佳使用案例

您可以使用**DatasetImportStep**来创建数据集导入节点，该节点有两种使用方式：

1. 将指定存储路径下的数据导入到目标数据集中。
2. 将指定存储路径下的数据导入到指定标注任务中。

定义**DatasetImportStep**示例如下：

- **将指定存储路径下的数据导入到目标数据集中**

  使用场景：适用于需要对数据集进行数据更新的操作。

  - 用户将指定路径下已标注的数据导入到数据集中(同时导入标签信息)，后续可增加数据集版本发布节点进行版本发布。

    数据准备：提前在ModelArts管理控制台，创建数据集，并将已标注的数据上传至OBS中。

    ```
    from modelarts import workflow as wf
    # 通过DatasetImportStep将指定路径下的数据导入到数据集中，输出数据集对象

    # 定义数据集对象
    dataset = wf.data.DatasetPlaceholder(name="input_dataset")

    # 定义OBS数据对象
    obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

    data_import = wf.steps.DatasetImportStep(
        name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
        title="数据集导入", # 标题信息，不填默认使用name值
        inputs=[
            wf.steps.DatasetImportInput(name="input_name_1", data=dataset), # 目标数据集在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="dataset_name")表示
            wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
        ],# DatasetImportStep的输入
        outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep的输出
        properties=wf.steps.ImportDataInfo(
            annotation_format_config=[
                wf.steps.AnnotationFormatConfig(
                    format_name=wf.steps.AnnotationFormatEnum.MA_IMAGE_CLASSIFICATION_V1, # 已标注数据的标注格式，示例为图像分类
                    scene=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION # 标注的场景类型
                )
            ]
        )
    )
    ```

  - 用户将指定路径下未标注的数据导入到数据集中，后续可增加数据集标注节点对新增数据进行标注。

    数据准备：提前在ModelArts界面创建数据集，并将未标注的数据上传至OBS中。

    ```
    from modelarts import workflow as wf
    # 通过DatasetImportStep将指定路径下的数据导入到数据集中，输出数据集对象

    # 定义数据集对象
    dataset = wf.data.DatasetPlaceholder(name="input_dataset")

    # 定义OBS数据对象
    obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

    data_import = wf.steps.DatasetImportStep(
        name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
        title="数据集导入", # 标题信息，不填默认使用name值
        inputs=[
    ```



```
        wf.steps.DatasetImportInput(name="input_name_1", data=dataset), # 目标数据集在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="dataset_name")表示
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
    ],# DatasetImportStep的输入
    outputs=wf.steps.DatasetImportOutput(name="output_name") # DatasetImportStep的输出
)
```

- **将指定存储路径下的数据导入到指定标注任务中**。

  使用场景：适用于需要对标注任务进行数据更新的操作。

  - 用户将指定路径下已标注的数据导入到标注任务中(同时导入标签信息)，后续可增加数据集版本发布节点进行版本发布。

    数据准备：基于使用的数据集，提前创建标注任务，并将已标注的数据上传至OBS中。

    ```
    from modelarts import workflow as wf
    # 通过DatasetImportStep将指定路径下的数据导入到标注任务中，输出标注任务对象

    # 定义标注任务对象
    label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

    # 定义OBS数据对象
    obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

    data_import = wf.steps.DatasetImportStep(
        name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
        title="数据集导入", # 标题信息，不填默认使用name值
        inputs=[
            wf.steps.DatasetImportInput(name="input_name_1", data=label_task), # 目标标注任务对象，在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")表示
            wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
        ],# DatasetImportStep的输入
        outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep的输出
        properties=wf.steps.ImportDataInfo(
            annotation_format_config=[
                wf.steps.AnnotationFormatConfig(
                    format_name=wf.steps.AnnotationFormatEnum.MA_IMAGE_CLASSIFICATION_V1, # 已标注数据的标注格式，示例为图像分类
                    scene=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION # 标注的场景类型
                )
            ]
        )
    )
    ```

  - 用户将指定路径下未标注的数据导入到标注任务中，后续可增加数据集标注节点对新增数据进行标注。

    数据准备：基于使用的数据集，提前创建标注任务，并将未标注的数据上传至OBS中。

    ```
    from modelarts import workflow as wf
    # 通过DatasetImportStep将指定路径下的数据导入到标注任务中，输出标注任务对象

    # 定义标注任务对象
    label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

    # 定义OBS数据对象
    obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

    data_import = wf.steps.DatasetImportStep(
        name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step
    ```



```
名称不能重复
    title="数据集导入", # 标题信息，不填默认使用name值
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=label_task), # 目标标注任务对象，在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")表示
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
    ],# DatasetImportStep的输入
    outputs=wf.steps.DatasetImportOutput(name="output_name") # DatasetImportStep的输出
)
```

- **基于数据集创建节点，构建数据集导入节点**

  使用场景：数据集创建节点的输出作为数据集导入节点的输入。

```
from modelarts import workflow as wf
# 通过DatasetImportStep将指定路径下的数据导入到数据集中，输出数据集对象

# 定义OBS数据对象
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

data_import = wf.steps.DatasetImportStep(
    name="data_import", # 数据集导入节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="数据集导入", # 标题信息，不填默认使用name值
    inputs=[
        wf.steps.DatasetImportInput(name="input_name_1", data=create_dataset.outputs["create_dataset_output"].as_input()), # 数据集创建节点的输出作为导入节点的输入
        wf.steps.DatasetImportInput(name="input_name_2", data=obs) # 导入的数据存储路径，运行时配置；data字段也可使用wf.data.OBSPath(obs_path="obs_path")表示
    ],# DatasetImportStep的输入
    outputs=wf.steps.DatasetImportOutput(name="output_name"), # DatasetImportStep的输出
    depend_steps=create_dataset # 依赖的数据集创建节点对象
)
# create_dataset是 wf.steps.CreateDatasetStep的一个实例，create_dataset_output是wf.steps.CreateDatasetOutput的name字段值
```

4.5.4 数据集版本发布节点

功能介绍

通过对ModelArts数据集能力进行封装，实现数据集的版本自动发布的功能。数据集版本发布节点主要用于将已存在的数据集或者标注任务进行版本发布，每个版本相当于数据的一个快照，可用于后续的数据溯源。

应用场景

- 对于数据标注这种操作，可以在标注完成后自动帮助用户发布新的数据集版本，结合as_input的能力提供给后续节点使用。
- 当模型训练需要更新数据时，可以使用数据集导入节点先导入新的数据，然后再通过该节点发布新的版本供后续节点使用。

最佳使用案例

您可以使用**ReleaseDatasetStep**来创建数据集版本发布节点，该节点有两种使用方式：

1. 基于数据集发布版本
2. 基于标注任务发布版本。

定义**ReleaseDatasetStep**示例如下：

- **基于数据集发布版本**

  使用场景：当数据集更新了数据时，可以通过该节点发布新的数据集版本供后续的节点使用。

```
from modelarts import workflow as wf
# 通过ReleaseDatasetStep将输入的数据集对象发布新的版本，输出带有版本信息的数据集对象

# 定义数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 定义训练验证切分比参数
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR, default="0.8")

release_version = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # 数据集发布节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="数据集版本发布", # 标题信息，不填默认使用name值
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=dataset), # ReleaseDatasetStep的输入，数据集对象在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="dataset_name")表示
    outputs=wf.steps.ReleaseDatasetOutput(
        name="output_name",
        dataset_version_config=wf.data.DatasetVersionConfig(
            label_task_type=wf.data.LabelTaskTypeEnum.IMAGE_CLASSIFICATION,  # 数据集发布版本时需要指定标注任务的类型
            train_evaluate_sample_ratio=train_ration # 数据集的训练验证切分比
            )
    ) # ReleaseDatasetStep的输出
)
```

- **基于标注任务发布版本**

  当标注任务更新了数据或者标注信息时，可以通过该节点发布新的数据集版本供后续的节点使用。

```
from modelarts import workflow as wf
# 通过ReleaseDatasetStep将输入的标注任务对象发布新的版本，输出带有版本信息的数据集对象

# 定义标注任务对象
label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

# 定义训练验证切分比参数
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR, default="0.8")

release_version = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # 数据集发布节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="数据集版本发布", # 标题信息，不填默认使用name值
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=label_task), # ReleaseDatasetStep的输入，
标注任务对象在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")表示
    outputs=wf.steps.ReleaseDatasetOutput(name="output_name", dataset_version_config=wf.data.DatasetVersionConfig(train_evaluate_sample_ratio=train_ration)), # 数据集的训练验证切分比
)
```

- **基于数据集标注节点，构建数据集版本发布节点**

  使用场景：数据集标注节点的输出作为数据集版本发布节点的输入。

```
from modelarts import workflow as wf
# 通过ReleaseDatasetStep将输入的标注任务对象发布新的版本，输出带有版本信息的数据集对象
```



```
# 定义标注任务对象
label_task = wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")

# 定义训练验证切分比参数
train_ration = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR, default="0.8")

release_version = wf.steps.ReleaseDatasetStep(
    name="release_dataset", # 数据集发布节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="数据集版本发布", # 标题信息，不填默认使用name值
    inputs=wf.steps.ReleaseDatasetInput(name="input_name", data=label_task), # ReleaseDatasetStep的输入，
标注任务对象在运行时配置；data字段也可使用wf.data.LabelTask(dataset_name="dataset_name", task_name="label_task_name")表示
    outputs=wf.steps.ReleaseDatasetOutput(name="output_name", dataset_version_config=wf.data.DatasetVersionConfig(train_evaluate_sample_ratio=train_ration)), # 数据集的训练验证切分比
)
```

4.5.5 作业类型节点

功能介绍

该节点通过对算法、输入、输出的定义，实现ModelArts作业管理的能力。主要用于数据处理、模型训练、模型评估等场景。

应用场景

- 当需要对图像进行增强，对语音进行除噪等操作时，可以使用该节点进行数据的预处理。
- 对于一些物体检测，图像分类等AI应用场景，可以根据已有的数据使用该节点进行模型的训练。

最佳使用案例

您在创建作业类型节点之前可以通过以下操作来获取该帐号所支持的训练资源规格列表以及引擎规格列表：

```
from modelarts.session import Session
from modelarts.estimatorV2 import TrainingJob
from modelarts.workflow.client.job_client import JobClient

# 若您在本地IDEA环境中开发工作流，则Session初始化使用如下方式
session = Session(
    access_key="***", # 账号的AK信息
    secret_key="***", # 账号的SK信息
    region_name="***", # 账号所属的region
    project_id="***" # 账号的项目ID
)

# 若您在Notebook环境中开发工作流，则Session初始化使用如下方式
session = Session()

# 公共资源池规格列表查询
spec_list = TrainingJob(session).get_train_instance_types(session) # 返回的类型为list,可按需打印查看

# 运行中的专属资源池列表查询
pool_list = JobClient(session).get_pool_list() # 返回专属资源池的详情列表
pool_id_list = JobClient(session).get_pool_id_list() # 返回专属资源池ID列表
GPU/NPU专属资源池规格ID列表如下，根据所选资源池的实际规格自行选择：
    1. modelarts.pool.visual.xlarge 对应1卡
```



```
  2. modelarts.pool.visual.2xlarge 对应2卡
  3. modelarts.pool.visual.4xlarge 对应4卡
  4. modelarts.pool.visual.8xlarge 对应8卡

# 引擎规格查询
engine_dict = TrainingJob(session).get_engine_list(session) # 返回的类型为dict,可按需打印查看
```

您可以使用**JobStep**来创建作业类型节点。定义**JobStep**示例如下。

- **使用订阅自AIGallery的算法**。

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
description="description_info") # name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")


# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
   name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
   title="图像分类训练", # 标题信息，不填默认使用name
   algorithm=wf.AIGalleryAlgorithm(
      subscription_id="subscription_id", # 算法订阅ID
      item_version_id="item_version_id", # 算法订阅版本ID
      parameters=[
         wf.AlgorithmParameters(
            name="parameter_name",
            value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
         ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
      ]

   ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值若无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值


   inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表示
   outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
   spec=wf.steps.JobSpec(
      resource=wf.steps.JobResource(
         flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

      )
   )# 训练资源规格信息
)
```

- **使用算法管理中的算法**

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
description="description_info") # name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
   name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
   title="图像分类训练", # 标题信息，不填默认使用name
```



```
    algorithm=wf.Algorithm(
        algorithm_id="algorithm_id", # 算法ID

        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中的算法来源于算法管理；部分算法超参的值若无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值

    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段
也可使用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表
示
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    )# 训练资源规格信息
)
```

- **使用自定义算法（代码目录+启动文件+官方镜像）**

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
description="description_info") # name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线
（-），并且只能以英文字母开头，长度限制为64字符），一个workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.BaseAlgorithm(
        code_dir="fake_code_dir", # 代码目录存储的路径
        boot_file="fake_boot_file", # 启动文件存储路径，需要在代码目录下
        engine=wf.steps.JobEngine(engine_name="fake_engine_name",
engine_version="fake_engine_version"), # 官方镜像的名称以及版本信息

        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), 自定义算法使用代码目录+启动文件+官方惊险的方式实现


    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段
也可使用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表
示
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
```



```
    )# 训练资源规格信息
)
```

- **使用自定义算法（代码目录+脚本命令+自定义镜像）**

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
description="description_info") # name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.BaseAlgorithm(
        code_dir="fake_code_dir", # 代码目录存储的路径
        command="fake_command", # 执行的脚本命令
        engine=wf.steps.JobEngine(image_url="fake_image_url"), # 自定义镜像的url
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), 自定义算法使用代码目录+脚本命令+自定义镜像的方式实现

    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表示
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")

        )
    )# 训练资源规格信息
)
```

上述四种方式使用据集对象作为输入，若您需要使用OBS路径作为输入时，只需将JobInput中的data数据替换为**data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")**或者**data=wf.data.OBSPath(obs_path="fake_obs_path")**即可。

此外，在构建工作流时就指定好数据集对象或者OBS路径的方式可以减少配置操作，方便您在开发态进行调试。但是对于发布到运行态或者gallery的工作流，更推荐的方式是采用数据占位符的方式进行编写，您可以在工作流启动之前对参数进行配置，自由度更高。

- **基于数据集版本发布节点构建作业类型节点**

  使用场景：数据集版本发布节点的输出作为作业类型节点的输入。

```
from modelarts import workflow as wf

# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
description="description_info") # name字段必填，title, description可选填

# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
job_step = wf.steps.JobStep(
```



```
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR, default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值若无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值


    inputs=wf.steps.JobInput(name="data_url", data=release_version_step.outputs["output_name"].as_input()), # 使用数据集版本发布节点的输出作为JobStep的输入
    outputs=wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格")

        )
    ), # 训练资源规格信息
    depend_steps=release_version_step # 依赖的数据集版本发布节点对象
)
# release_version_step是wf.steps.ReleaseDatasetStep的实例对象，output_name是wf.steps.ReleaseDatasetOutput的name字段值
```

五. 基于数据集版本发布节点构建作业类型节点

4.5.6 模型注册节点

功能介绍

通过对ModelArts模型管理的能力进行封装，实现将训练后的结果注册到模型管理中，便于后续服务部署、更新等步骤的执行。

应用场景

- 注册ModelArts训练作业中训练完成的模型。
- 注册自定义镜像中的模型。

最佳使用案例

您可以使用**ModelStep**来创建模型注册节点。定义**ModelStep**的几种示例如下。

- **从训练作业中注册模型（模型输入来源JobStep的输出）。**

```
import modelarts.workflow as wf
# 通过ModelStep来定义一个模型注册节点，输入来源于JobStep的输出

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration",  # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="模型注册",  # 标题信息
```



```
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()),  # ModelStep的输入来源于依赖的JobStep的输出

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")), # ModelStep的输出
    depend_steps=job_step # 依赖的作业类型节点对象
)
# job_step是wf.steps.JobStep的 实例对象，train_url是wf.steps.JobOutput的name字段值
```

- **从训练作业中注册模型（模型输入来源OBS路径, 训练完成的模型已存储到OBS路径）。**

```
import modelarts.workflow as wf
# 通过ModelStep来定义一个模型注册节点，输入来源于OBS中

# 定义OBS数据对象
obs = wf.data.OBSPlaceholder(name = "obs_placeholder_name", object_type = "directory" ) # object_type必须是file或者directory

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration",  # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="模型注册",  # 标题信息
    inputs=wf.steps.ModelInput(name='model_input', data=obs),  # ModelStep的输入在运行时配置；data字段的值也可使用wf.data.OBSPath(obs_path="fake_obs_path")表示

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow"))# ModelStep的输出
)
```

- **从自定义镜像中注册模型**

```
import modelarts.workflow as wf
# 通过ModelStep来定义一个模型注册节点，输入来源于自定义镜像地址

# 定义镜像数据
swr = wf.data.SWRImagePlaceholder(name="placeholder_name")

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration",  # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="模型注册",  # 标题信息
    inputs=wf.steps.ModelInput(name="input",data=swr), # ModelStep的输入在运行时配置；data字段的值也可使用wf.data.SWRImage(swr_path="fake_path")表示

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow"))# ModelStep的输出
)
```

对于model_type字段的说明：

model_type支持的类型有："**TensorFlow**"、"**MXNet**"、"**Caffe**"、"**Spark_MLlib**"、 "**Scikit_Learn**"、"**XGBoost**"、 "**Image**"、"**PyTorch**"。

在wf.steps.ModelConfig对象中，若model_type字段未填写，则表示默认使用"**TensorFlow**"。

- 若您构建的工作流对注册的模型类型没有修改的需求，则按照上述示例使用即可。
- 若您工作流的多次运行可以修改模型类型，则可使用占位符参数的方式进行编写：

model_type = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.ENUM, default="TensorFlow", enum_list=["TensorFlow", "MXNet", "Caffe", "Spark_MLlib", "Scikit_Learn", "XGBoost", "Image", "PyTorch"], description="模型类型")。

4.5.7 服务部署节点

功能介绍

通过对ModelArts服务管理能力的封装，实现Workflow新增服务和服务更新的能力。

应用场景

- 将模型部署为一个Web Service。
- 更新已有服务，支持灰度更新等能力。

最佳使用案例

您可以使用**ServiceStep**来创建服务部署节点。定义**ServiceStep**示例如下。

- **新增在线服务**

```
import modelarts.workflow as wf
# 通过ServiceStep来定义一个服务部署节点，输入指定的模型进行服务部署

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

service_step = wf.steps.ServiceStep(
    name="service_step", # 服务部署节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="新增服务", # 标题信息
    inputs=wf.steps.ServiceInput(name="si_service_ph",
data=wf.data.ServiceInputPlaceholder(name="si_placeholder1",
                                                                # 模型名称的限制/约束,在运行态只能选择该模型名称；一般与模型注册节点中的model_name使用同一个参数对象
                                                                model_name=model_name)),# ServiceStep的输入列表
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep的输出
)
```

- **服务更新：**

  使用场景：使用新版本的模型对已有的服务进行更新，需要保证新版本的模型与已部署服务的模型名称一致。

```
import modelarts.workflow as wf
# 通过ServiceStep来定义一个服务部署节点，输入指定的模型对已部署的服务进行更新

# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

# 定义服务对象
service = wf.data.ServiceUpdatePlaceholder(name="placeholder_name")

service_step  = wf.steps.ServiceStep(
    name="service_step", # 服务部署节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="服务更新", # 标题信息
    inputs=[wf.steps.ServiceInput(name="si2",
data=wf.data.ServiceInputPlaceholder(name="si_placeholder2",
                                                                # 模型名称的限制/约束,在运行态只能选择该模型名称
                                                                model_name=model_name)),
        wf.steps.ServiceInput(name="si_service_data", data=service) # 已部署的服务在运行时配置；
```



```
    data也可使用wf.data.ServiceData(service_id="fake_service")表示
    ], # ServiceStep的输入列表
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep的输出
)
```

服务部署相关信息配置：

在开发态中（一般指Notebook），节点启动运行后，用户根据日志打印的输入格式进行配置，如下所示：

```
Please enter the ServiceInputPlaceholder "service_model" in the following format:
"[{'model_name': '******', 'model_version': '*.*.*', 'specification': [redacted], 'weight': 50}, {'model_name': '******', 'model_version': '*.*.*', 'specification':'******', 'weight': 30,
'envs':{'k1': 'v1', 'k2': 'v2'}},{'model_name': '******', 'model_version': '*.*.*', 'specification':'******', 'weight': 20, 'envs':{'******': '******', '******': '******'}},]"
Note that:(1) The "[" at the beginning and "]" at the end are required.
          (2) The sum of the weights must be equal to 100.
          (3) All model must have the same model name. Two model versions cannot be the same.
```

在ModelArts管理控制台workflow页面，服务部署节点启动之后会等待用户设置相关配置信息，配置完成后直接单击继续运行即可，如下所示：

4.5.8 条件节点

功能介绍

主要用于执行流程的条件分支选择，可以简单的进行数值比较来控制执行流程，也可以根据节点输出的metric相关信息决定后续的执行流程。

应用场景

- 可以用于需要根据不同的输入值来决定后续执行流程的场景。
- 例如需要根据训练节点输出的精度信息来决定是重新训练还是进行模型的注册操作时可以使用该节点来实现流程的控制。

使用案例

您可以使用**ConditionStep**来创建条件节点。定义**ConditionStep**示例如下。

```
import modelarts.workflow as wf
condition = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=9, right=10) # 条件对象，包含类型以及左右值
condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # 条件节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    conditions=condition, # 条件对象,允许多个条件，条件之间的关系为&&
    if_then_steps="step_1", # 当condition结果为true时，名称为step_1的节点允许执行，名称为step_2的节点跳过不执行
    else_then_steps="step_2" # 当condition结果为false时，名称为step_2的节点允许执行，名称为step_1的节点跳过不执行
)
```

示例说明：

1. **Condition对象（由三部分组成：条件类型，左值以及右值）**
   - 条件类型使用ConditionTypeEnum来获取，支持"==", ">", ">=", "in", "<", "<=", "!=", "or"操作符，具体映射关系如下表所示

枚举类型	操作符
ConditionTypeEnum.EQ	==
ConditionTypeEnum.GT	>
ConditionTypeEnum.GTE	>=
ConditionTypeEnum.IN	in
ConditionTypeEnum.LT	<
ConditionTypeEnum.LTE	<=
ConditionTypeEnum.NOT	!=
ConditionTypeEnum.OR	or

   - 左右值支持的类型有：int, float, str, bool, Placeholder, Sequence, Condition, MetricInfo。
   - 一个ConditionStep支持多个Condition对象，使用list表示，多个Condition之间进行&&操作。
2. **if_then_steps 和 else_then_steps**。
   - if_then_steps表示的是当Condition比较的结果为true时允许执行的节点列表，存储的是节点名称；此时else_then_steps中的step跳过不执行。
   - else_then_step表示的是当Condition比较的结果为false时允许执行的节点列表，存储的是节点名称；此时if_then_steps中的step跳过不执行。

进阶示例，多个Condition条件：

```
import modelarts.workflow as wf
parameter = wf.Placeholder(name="param_1", placeholder_type=wf.PlaceholderType.INT, default=10)
condition1 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=parameter, right=10)
condition2 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.GT, left=10, right=9)
condition3 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.LT, left=10, right=9)
condition4 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.OR, left=condition2, right=condition3)
```



```
condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # 条件节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    conditions=[condition1, condition4], # 条件对象,允许多个条件，条件之间的关系为&&
    if_then_steps=["true_step_1", "true_step_2"], # 当condition结果为true时，名称为true_step_1和true_step_2的节点允许执行，名称为false_step_1和false_step_2的节点跳过不执行
    else_then_steps=["false_step_1", "false_step_2"] # 当condition结果为false时，名称为false_step_1和false_step_2的节点允许执行，名称为true_step_1和true_step_2的节点跳过不执行
)
```

示例说明：

condition1的结果为**true**, condition2的结果为**true**, condition3的结果为**false**, condition4的结果为**true**。所以condition_step的比较结果最后为true。

结果：

名称为"true_step_1", "true_step_2"的step节点允许执行。

名称为"false_step_1", "false_step_2"的step节点跳过不执行。

名称为"false_step_1", "false_step_2"的step节点跳过不执行

4.6 如何在 Workflow 中使用大数据的能力（DLI/MRS）

MRS作业节点。

功能介绍

该节点通过调用MRS服务，提供大数集群计算能力。主要用于数据批量理、模型训练等场景。

应用场景

需要使用MRS Spark组件进行大量数据的计算时，可以根据已有数据使用该节点进行训练计算。

使用案例

在华为云MRS服务下查看自己帐号下可用的MRS集群，如果没有，则需要创建，当前需要集群有Spark组件，安装时，注意勾选上。

您可以使用MrsStep来创建作业类型节点。定义MrsStep示例如下。

- **指定启动脚本与集群**

```
from modelarts import workflow as wf
# 通过MrsStep来定义一个MrsJobStep节点，

algorithm = wf.steps.MrsJobAlgorithm(
    boot_file="obs://spark-sql/wordcount.py",    //执行脚本OBS路径
    parameters=[wf.AlgorithmParameters(name="run_args", value="--master,yarn-cluster")]
)
```



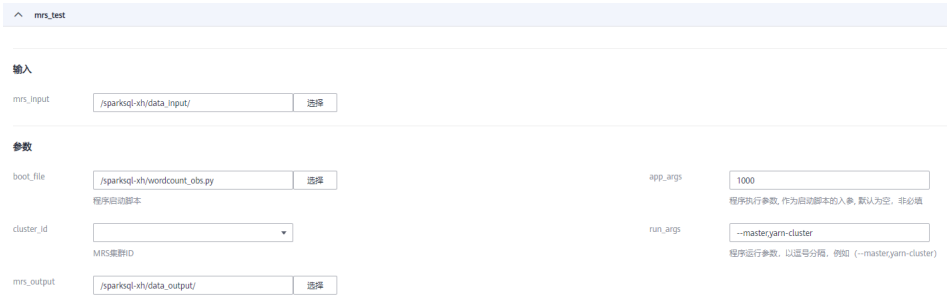
```
    inputs = wf.steps.MrsJobInput(name="mrs_input", data=OBSPath(obs_path="/spark-sql/
mrs_input/")) //输入数据的OBS路径
    outputs = wf.steps.MrsJobOutput(name="mrs_output", obs_config=OBSOutputConfig(obs_path="/
spark-sql/mrs_output")) //输出的OBS路径
    step = wf.steps.MrsJobStep(
            name="mrs_test",          //step名称，可自定义
            mrs_algorithm=algorithm,
            inputs=inputs,
            outputs=outputs,
            cluster_id="cluster_id_xxx"   //MRS集群ID
)
```

- **使用选取集群和启动脚本的形式**

```
from modelarts import workflow as wf
# 通过MrsJobStep来定义一个节点
run_arg_description = "程序执行参数, 作为程序运行环境参数, 默认为（ --master,yarn-cluster)"
app_arg_description = "程序执行参数, 作为启动脚本的入参, 例如（ --param_a=3,--param_b=4 ）默认为
空，非必填"
mrs_outputs_description = "数据输出路径, 可以通过从参数列表中获取--train_url参数获取"
cluster_id_description = "cluster id of MapReduce Service"

algorithm = wf.steps.MrsJobAlgorithm(
    boot_file=wf.Placeholder(name="boot_file",
                             description="程序启动脚本",
                             placeholder_type=wf.PlaceholderType.STR,
                             placeholder_format="obs"),
    parameters=[wf.AlgorithmParameters(name="run_args",
                                       value=wf.Placeholder(name="run_args",
                                                            description=run_arg_description,
                                                            default="--master,yarn-cluster",
                                                            placeholder_type=wf.PlaceholderType.STR),
                                       ),
                wf.AlgorithmParameters(name="app_args",
                                       value=wf.Placeholder(name="app_args",
                                                            description=app_arg_description,
                                                            default="",
                                                            placeholder_type=wf.PlaceholderType.STR)
                                       )
                ]
    )

    inputs = wf.steps.MrsJobInput(name="data_url",
                                  data=wf.data.OBSPlaceholder(name="data_url",object_type="directory"))

    outputs = wf.steps.MrsJobOutput(name="train_url",
obs_config=OBSOutputConfig(obs_path=wf.Placeholder(name="train_url",
placeholder_type=wf.PlaceholderType.STR,
placeholder_format="obs",description=mrs_outputs_description)))

    mrs_job_step = wf.steps.MrsJobStep(
        name="mrs_job_test",
        mrs_algorithm=algorithm,
        inputs=inputs,
        outputs=outputs,
        cluster_id=wf.Placeholder(name="cluster_id", placeholder_type=wf.PlaceholderType.STR,
description=cluster_id_description,                                placeholder_format="cluster"))
```

- **前端如何使用MRS节点**

  workflow 发布后，在workflow配置页，配置节点的数据输入，输出，启动脚本，集群ID等参数。



4.7 如何将串联节点构建工作流

1. **构建节点**

```
# 准备环境
from modelarts import workflow as wf
# 通过JobStep来定义一个训练节点，输入使用数据集，并将训练结果输出到OBS
# 构建一个OutputStorage对象，对训练输出目录做统一管理
storage = wf.data.OutputStorage(name="storage_name", title="title_info",
description="description_info") # name字段必填，title, description可选填

# 定义输入的数据集对象
dataset = wf.data.DatasetPlaceholder(name="input_dataset")

job_step = wf.steps.JobStep(
    name="training_job", # 训练节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="图像分类训练", # 标题信息，不填默认使用name
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值若无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name="data_url", data=dataset), # JobStep的输入在运行时配置；data字段也可使用wf.data.Dataset(dataset_name="fake_dataset_name", version_name="fake_version_name")表示
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))), # JobStep的输出
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="训练资源规格")
        )
    )# 训练资源规格信息
)

# 通过ModelStep将训练好的模型注册到模型管理中
# 定义模型名称参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # 模型注册节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="模型注册", # 标题信息, 不填默认使用name
    inputs=wf.steps.ModelInput(name="model_input",
                               data=job_step.outputs["train_url"].as_input()),
    outputs=wf.steps.ModelOutput(name="model_output",
```



```
                        model_config=wf.steps.ModelConfig(model_name=model_name,
model_type="TensorFlow")),
    depend_steps=[job_step] # 定义了步骤的依赖关系，表示model_step依赖job_step
)

# 通过ServiceStep将依赖的model_step注册的模型进行在线部署
service_step = wf.steps.ServiceStep(
    name="service_step", # 服务部署节点的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)，一个workflow里的两个step名称不能重复
    title="服务部署", # 标题信息, 不填默认使用name
    inputs=wf.steps.ServiceInput(name="service_input",
                          data=wf.data.ServiceInputPlaceholder(name="service_model",
model_name=model_name)),
    outputs=wf.steps.ServiceOutput(name="service_output"),
    depend_steps=[model_step] # 定义了步骤的依赖关系，表示service_step依赖model_step
```

2. **构建工作流**

   构建工作流有两种方式：‘

   - 当您在Notebook环境中使用时，系统自动获取环境变量，无需手动构造Session对象

     ```
     from modelarts import workflow as wf
     # 构建工作流
     workflow =wf.Workflow(name="image_cls", # 工作流的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)
                     desc="this is a demo workflow", # 工作流的描述信息
                     steps=[job_step, model_step, service_step], # 工作流包含的节点
                     storages=[storage] # 统一存储对象
     )
     ```

   - 当您在本地IDEA环境中使用时，需要手动初始化Session对象

     ```
     from modelarts import workflow as wf
     from modelarts.session import Session
     # 初始化Session对象
     session = Session(
         access_key="***", # 账号的AK信息
         secret_key="***", # 账号的SK信息
         region_name="***",# 账号所属的区域，例如"cn-north-4"表示北京4
         project_id="***" # 账号的项目ID
     )
     # 构建工作流
     workflow =wf.Workflow(name="image_cls", # 工作流的名称，命名规范(只能包含英文字母、数字、下划线（_）、中划线（-），并且只能以英文字母开头，长度限制为64字符)
                     desc="this is a demo workflow", # 工作流的描述信息
                     steps=[job_step, model_step, service_step], # 工作流包含的节点
                     storages=[storage], # 统一存储对象
                     session=session # 工作流运行时与ModelArts服务交互所需的会话对象
     )
     ```

4.8 如何调试 Workflow

先构建一条服务部署全链路的工作流**最佳实践**。

假设当前的工作流对象Workflow有label_step、release_step、 job_step、model_step、service_step五个节点。

- **使用run模式**
  - 运行全部节点

    ```
    workflow.run(steps=[label_step, release_step, job_step, model_step, service_step],
    experiment_id="实验记录ID")
    ```

  - 指定运行job_step, model_step, service_step

    ```
    workflow.run(steps=[job_step, model_step, service_step], experiment_id="实验记录ID")
    ```

- **使用debug模式**

debug模式只能在Notebook环境中使用，并且需要配合Workflow的next方法来一起使用，示例如下：

a. 启动debug模式。
```
workflow.debug(steps=[label_step, release_step, job_step, model_step, service_step], experiment_id="实验记录ID")
```

b. 执行第一个节点。
```
workflow.next()
```
此时出现两种情况：

i. 该节点运行需要的数据已经准备好，则直接启动运行

ii. 该节点数据未准备好，则打印日志信息提醒用户按照指示给该节点设置数据, 设置数据有两种方式：

- 对于单个的参数类型数据：
```
workflow.set_placeholder("参数名称", 参数值)
```
- 对于数据对象：
```
workflow.set_data(数据对象的名称, 数据对象)
# 示例：设置数据集对象
workflow.set_data("对象名称", Dataset(dataset_name="数据集名称", version_name="数据集版本名称"))
```

c. 当上一个节点执行完成后，继续调用workflow.next()启动后续节点，重复操作直至节点全部运行完成。

说明

在开发态调试工作流时，系统只会监控运行状态并打印相关日志信息，涉及到每个节点的详细运行状况需用户自行前往ModelArts管理控制台的相应服务处进行查看。

4.9 如何把 Workflow 发布给用户使用

工作流调试完成后，可以调用Workflow对象的release()方法发布到运行态进行配置执行（console界面）。

```
# 示例
workflow.release()
# 上述命令执行完成后，若日志打印显示发布成功，则可前往MA的workflow页面中查看新发布的工作流
```

4.10 如何分享自己的 Workflow 到 Gallery

4.10.1 准备 Workflow

工作流示例：

```
# 环境准备
import modelarts.workflow as wf
from modelarts.session import Session
session = Session()

# 定义输出的统一存储对象
output_storage = wf.data.OutputStorage(name="output_storage", title="输出目录", description="产物输出目录统一配置")

# 定义数据集标注节点
step_1_label_step = wf.steps.LabelingStep(
    name="labeling",
```



```
    title="数据集标注",
    inputs=wf.steps.LabelingInput(name="label_input",
data=wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")),
    outputs=wf.steps.LabelingOutput(name="label_output")

)

# 定义数据集版本发布节点
step_2_release_data_step = wf.steps.ReleaseDatasetStep(
    name="release_dataset",
    inputs=wf.steps.ReleaseDatasetInput(name="dataset",
data=step_1_label_step.outputs["label_output"].as_input()),
    outputs=wf.steps.ReleaseDatasetOutput(name="dataset_version"),
    depend_steps=[step_1_label_step],
)

# 定义作业节点
step_3_training_step = wf.steps.JobStep(
    name="image_classification_training",
    title="图片分类训练",
    algorithm=wf.AIGalleryAlgorithm(subscription_id="fake_subscription_id",
item_version_id="fake_version_id",
                                    parameters=[wf.AlgorithmParameters(name='parameter_name',
value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")),]), # 算法的超参，具有默认值的超参可以不用填写，只需
要填写自定义超参以及需要修改值的超参
    inputs=wf.steps.JobInput(name='data_url',
data=step_2_release_data_step.outputs["dataset_version"].as_input()),
    outputs=wf.steps.JobOutput(name="train_url",
            obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/train_output/"))),
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(flavor=wf.Placeholder(name="train_flavor",
placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格"))),
    depend_steps=[step_2_release_data_step]
)

# 定义模型名称为可配置参数
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

# 定义模型注册节点
step_4_model_registration = wf.steps.ModelStep(name="model_registration",
                                    title="模型注册",
                                    inputs=wf.steps.ModelInput(name='model_input',
data=step_3_training_step.outputs["train_url"].as_input()),
                                    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),
                                    depend_steps=[step_3_training_step]
                                    )

# 定义服务部署节点
step_5_service_step = wf.steps.ServiceStep(
    name="service_step",
    title="部署服务",
    inputs=wf.steps.ServiceInput(name="service_input",
data=wf.data.ServiceInputPlaceholder(name="service_input_ph", model_name=model_name)),
    outputs=wf.steps.ServiceOutput(name="service_output"),
    depend_steps=[step_4_model_registration]
)

# 定义Workflow
workflow =wf.Workflow(name="image_cls",
                      desc="this is a demo workflow",
                      steps=[step_1_label_step,step_2_release_data_step, step_3_training_step,
step_4_model_registration,step_5_service_step],
                      session=session,
                      storages=[output_storage]
)
```

注意事项：

- 所有节点的输入数据只能使用AbstractDataPlaceholders对象，例如（DatasetPlaceholder, LabelTaskPlaceholder, OBSPlaceholder, ServiceUpdataPlaceholder, SWRImagePlaceholder, ServiceInputPlaceholder等）。
- JobStep节点只能使用来自Gallery中的订阅算法。

4.10.2 发布 Workflow 资产

在发布工作流新版本之前需要先使用create_content方法在Gallery中发布一个工作流类型的资产（对于同一个工作流create_content方法一般只需执行一次）

create_content方法的参数列表

参数名称	是否必填	参数类型
title	是	str
visibility	是	str
group_users	否（domainID的列表）	list[str]

visibility的可选值为:

- public：公共可见
- private：仅自己可见
- group：指定用户组可见

当visibility的值设置为group时，group_users需要至少指定一个domainID

示例：

```
content_id = workflow.create_content(title="workflow手写体测试", visibility="public")
print(content_id)
```

create_content方法的返回值为资产ID。

4.10.3 发布 Workflow 的新版本

可以通过release_to_gallery方法发布新的工作流版本到指定资产下

release_to_gallery方法的参数列表

参数名称	是否必填	参数类型
content_id	是（来源为create_content方法的返回值）	str
version	是（推荐格式为：x.x.x）	str
desc	否	str

示例：


```
workflow.release_to_gallery(content_id="fake_content_id", version="x.x.x", desc="description_info")
```

执行成功后可通过ModelArts界面中的AIGallery链接到市场查询发布的新工作流，其它授权用户可进行订阅使用

4.11 最佳实践

4.11.1 全链路（服务部署）

Workflow全链路（服务部署）的示例如下所示，您也可以点击此**Notebook链接** 0代码体验。

```
# 环境准备
import modelarts.workflow as wf
from modelarts.session import Session
session = Session()

# 定义输出的统一存储对象
output_storage = wf.data.OutputStorage(name="output_storage", title="输出目录", description="产物输出目录统一配置")

# 定义数据集标注节点
step_1_label_step = wf.steps.LabelingStep(
    name="labeling",
    title="数据集标注",
    inputs=wf.steps.LabelingInput(name="label_input",
data=wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")),
    outputs=wf.steps.LabelingOutput(name="label_output")

)
# 定义数据集版本发布节点
step_2_release_data_step = wf.steps.ReleaseDatasetStep(
    name="release_dataset",
    inputs=wf.steps.ReleaseDatasetInput(name="dataset",
data=step_1_label_step.outputs["label_output"].as_input()),
    outputs=wf.steps.ReleaseDatasetOutput(name="dataset_version"),
    depend_steps=[step_1_label_step],
)

# 定义作业节点
step_3_training_step = wf.steps.JobStep(
    name="image_classification_training",
    title="图片分类训练",

    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]

    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值若无需修改，则在parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name='data_url',
data=step_2_release_data_step.outputs["dataset_version"].as_input()),
    outputs=wf.steps.JobOutput(name="train_url",
              obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/train_output/"))),
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(flavor=wf.Placeholder(name="train_flavor",
placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格")
```



```
)),
    depend_steps=[step_2_release_data_step]
)

# 定义模型名称参数
model_name = wf.Placeholder(name="model_name", placeholder_type=wf.PlaceholderType.STR)

# 定义模型注册节点
step_4_model_registration = wf.steps.ModelStep(name="model_registration",
                                        title="模型注册",
                                        inputs=wf.steps.ModelInput(name='model_input',
data=step_3_training_step.outputs["train_url"].as_input()),
                                        outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),
                                        depend_steps=[step_3_training_step]
                                        )

# 定义服务部署节点
step_5_service_step = wf.steps.ServiceStep(
    name="service_step",
    title="部署服务",
    inputs=wf.steps.ServiceInput(name="service_input",
data=wf.data.ServiceInputPlaceholder(name="service_input_ph", model_name=model_name)),
    outputs=wf.steps.ServiceOutput(name="service_output"),
    depend_steps=[step_4_model_registration]
)

# 定义Workflow
workflow =wf.Workflow(name="image_cls",
                      desc="this is a demo workflow",
                      steps=[step_1_label_step, step_2_release_data_step, step_3_training_step,
step_4_model_registration, step_5_service_step],
                      session=session,
                      storages=[output_storage]
)

# 运行Workflow
workflow.run(
    # 运行指定steps, 指定的steps须在定义Workflow时定义
    steps=[step_1_label_step, step_2_release_data_step, step_3_training_step, step_4_model_registration,
step_5_service_step],
    # Workflow实验记录的文件夹名称
    experiment_id="workflow_e2e_1")
```

4.11.2 全链路（服务更新）

Workflow全链路（服务更新）的示例如下所示，您也可以点击此**Notebook链接** 0代码体验。

```
# 环境准备
import modelarts.workflow as wf
from modelarts.session import Session
session = Session()

# 定义输出的统一存储对象
output_storage = wf.data.OutputStorage(name="output_storage", title="输出目录", description="产物输出目
录统一配置")

# 定义数据集标注节点
step_1_label_step = wf.steps.LabelingStep(
    name="labeling",
    title="数据集标注",
    inputs=wf.steps.LabelingInput(name="label_input",
data=wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")),
    outputs=wf.steps.LabelingOutput(name="label_output")

)
# 定义数据集版本发布节点
step_2_release_data_step = wf.steps.ReleaseDatasetStep(
```

```
    name="release_dataset",
    inputs=wf.steps.ReleaseDatasetInput(name="dataset",
data=step_1_label_step.outputs["label_output"].as_input()),
    outputs=wf.steps.ReleaseDatasetOutput(name="dataset_version"),
    depend_steps=[step_1_label_step],
)

# 定义作业节点
step_3_training_step = wf.steps.JobStep(
    name="image_classification_training",
    title="图片分类训练",

    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]

    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值若无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name='data_url',
data=step_2_release_data_step.outputs["dataset_version"].as_input()),
    outputs=wf.steps.JobOutput(name="train_url",
              obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/train_output/"))),
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(flavor=wf.Placeholder(name="train_flavor",
placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格")

)),
    depend_steps=[step_2_release_data_step],
)

# 定义模型名称参数
model_name = wf.Placeholder(name="model_name", placeholder_type=wf.PlaceholderType.STR)

# 定义模型注册节点
step_4_model_registration = wf.steps.ModelStep(name="model_registration",
                                        title="模型注册",
                                        inputs=wf.steps.ModelInput(name='model_input',
data=step_3_training_step.outputs["train_url"].as_input()),
                                        outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),
                                        depend_steps=[step_3_training_step]
                                        )

# 定义服务对象
service = wf.data.ServiceUpdatePlaceholder(name="service_placeholder_name")

# 定义服务部署节点
step_5_service_step = wf.steps.ServiceStep(
    name="service_step",
    title="服务更新",
    inputs=[wf.steps.ServiceInput(name="service_input",
data=wf.data.ServiceInputPlaceholder(name="service_input_ph", model_name=model_name)),
            wf.steps.ServiceInput(name="si_service_id", data=service)],
    outputs=wf.steps.ServiceOutput(name="service_output"),
    depend_steps=[step_4_model_registration]
)

# 定义Workflow
workflow =wf.Workflow(name="image_cls",
                      desc="this is a demo workflow",
                      steps=[step_1_label_step, step_2_release_data_step, step_3_training_step,
step_4_model_registration, step_5_service_step],
```



```
                session=session,
                storages=[output_storage]
)

# 运行Workflow
workflow.run(
    # 运行指定steps, 指定的steps须在定义Workflow时定义
    steps=[step_1_label_step, step_2_release_data_step, step_3_training_step, step_4_model_registration,
step_5_service_step],
    # Workflow实验记录的文件夹名称
    experiment_id="workflow_e2e_2")
```

4.11.3 全链路（condition 判断是否部署）

Workflow全链路，当满足condition时进行部署的示例如下所示，您也可以点击此**Notebook链接** 0代码体验。

```
# 环境准备
import modelarts.workflow as wf
from modelarts.session import Session
session = Session()

# 定义输出的统一存储对象
output_storage = wf.data.OutputStorage(name="output_storage", title="输出目录", description="产物输出目
录统一配置")

# 定义数据集标注节点
step_1_label_step = wf.steps.LabelingStep(
    name="labeling",
    title="数据集标注",
    inputs=wf.steps.LabelingInput(name="label_input",
data=wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")),
    outputs=wf.steps.LabelingOutput(name="label_output")

)
# 定义数据集版本发布节点
step_2_release_data_step = wf.steps.ReleaseDatasetStep(
    name="release_dataset",
    inputs=wf.steps.ReleaseDatasetInput(name="dataset",
data=step_1_label_step.outputs["label_output"].as_input()),
    outputs=wf.steps.ReleaseDatasetOutput(name="dataset_version"),
    depend_steps=[step_1_label_step],
)

# 定义作业节点
step_3_training_step = wf.steps.JobStep(
    name="image_classification_training",
    title="图片分类训练",

    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值若无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name='data_url',
data=step_2_release_data_step.outputs["dataset_version"].as_input()),
    outputs=wf.steps.JobOutput(name="train_url",
              obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/train_output/"))),
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(flavor=wf.Placeholder(name="train_flavor",
placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格")
```



```
)),
    depend_steps=[step_2_release_data_step],
)

# 定义条件的left为step_3_training_step输出的metric信息
metric = wf.steps.MetricInfo(input_data=step_3_training_step.outputs["train_url"].as_input(),
json_key="f1_score")
condition_greater = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.GT, left=metric,
right=0.8)

# 定义当step_3_training_step输出的metric信息f1_score大于0.8,才服务部署
condition_step = wf.steps.ConditionStep(
    name="condition_step_test",
    conditions=[condition_greater],
    if_then_steps=["model_registration", "service_step"],
    depend_steps=[step_3_training_step]
)

# 定义模型名称参数
model_name = wf.Placeholder(name="model_name", placeholder_type=wf.PlaceholderType.STR)

# 定义模型注册节点
step_4_model_registration = wf.steps.ModelStep(name="model_registration",
                                        title="模型注册",
                                        inputs=wf.steps.ModelInput(name='model_input',
data=step_3_training_step.outputs["train_url"].as_input()),
                                        outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),
                                        depend_steps=[step_3_training_step]
                                        )

# 定义服务部署节点
step_5_service_step = wf.steps.ServiceStep(
    name="service_step",
    title="部署服务",
    inputs=[wf.steps.ServiceInput(name="service_input",
data=wf.data.ServiceInputPlaceholder(name="service_input_ph", model_name=model_name))],

    outputs=wf.steps.ServiceOutput(name="service_output"),
    depend_steps=[step_4_model_registration]
)

# 定义Workflow
workflow =wf.Workflow(name="image_cls",
                      desc="this is a demo workflow",
                      steps=[step_1_label_step,step_2_release_data_step, step_3_training_step,condition_step,
step_4_model_registration,step_5_service_step],
                      session=session,
                      storages=[output_storage]
)

# 运行Workflow
workflow.run(
    # 运行指定steps, 指定的steps须在定义Workflow时定义
    steps=[step_1_label_step,step_2_release_data_step, step_3_training_step, condition_step,
step_4_model_registration,step_5_service_step],
    # Workflow实验记录的文件夹名称
    experiment_id="workflow_e2e_3"
)
```

4.11.4 全链路（数据可迭代）

Workflow全链路，导入数据和发布数据集的示例如下所示，您也可以点击此**Notebook链接** 0代码体验。

适用场景：推理之后，数据回流到数据导入形成数据回路。


```
# 环境准备
import modelarts.workflow as wf
from modelarts.session import Session
session = Session()

# 定义输出的统一存储对象
output_storage = wf.data.OutputStorage(name="output_storage", title="输出目录", description="产物输出目
录统一配置")

# 定义数据集导入节点
step_0_data_import = wf.steps.DatasetImportStep(
    name="data_import",
    title="数据集导入",
    inputs=[
        wf.steps.DatasetImportInput(name="input_label_task",
data=wf.data.LabelTaskPlaceholder(name="label_task_placeholder_name")),
        wf.steps.DatasetImportInput(name="input_obs",
data=wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type = "directory" ))
    ],
    outputs=wf.steps.DatasetImportOutput(name="import_output")
)

# 定义数据集标注节点
step_1_label_step = wf.steps.LabelingStep(
    name="labeling",
    title="数据集标注",
    inputs=wf.steps.LabelingInput(name="label_input",
data=step_0_data_import.outputs["import_output"].as_input()),
    outputs=wf.steps.LabelingOutput(name="label_output"),
    depend_steps=[step_0_data_import]
)
# 定义数据集版本发布节点
step_2_release_data_step = wf.steps.ReleaseDatasetStep(
    name="release_dataset",
    inputs=wf.steps.ReleaseDatasetInput(name="dataset",
data=step_1_label_step.outputs["label_output"].as_input()),
    outputs=wf.steps.ReleaseDatasetOutput(name="dataset_version"),
    depend_steps=[step_1_label_step],
)

# 定义作业节点
step_3_training_step = wf.steps.JobStep(
    name="image_classification_training",
    title="图片分类训练",



    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id", # 算法订阅ID
        item_version_id="item_version_id", # 算法订阅版本ID
        parameters=[
            wf.AlgorithmParameters(
                name="parameter_name",
                value=wf.Placeholder(name="parameter_name", placeholder_type=wf.PlaceholderType.STR,
default="fake_value",description="description_info")
            ) # 算法超参的值使用Placeholder对象来表示，支持int, bool, float, str四种类型
        ]
    ), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值若无需修改，则在
parameters字段中可以不填写，系统自动填充相关超参值
    inputs=wf.steps.JobInput(name='data_url',
data=step_2_release_data_step.outputs["dataset_version"].as_input()),
    outputs=wf.steps.JobOutput(name="train_url",
              obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/train_output/"))),
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(flavor=wf.Placeholder(name="train_flavor",
placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格")

)),
    depend_steps=[step_2_release_data_step],
```



```
)

# 定义模型名称参数
model_name = wf.Placeholder(name="model_name", placeholder_type=wf.PlaceholderType.STR)

# 定义模型注册节点
step_4_model_registration = wf.steps.ModelStep(name="model_registration",
                                        title="模型注册",
                                        inputs=wf.steps.ModelInput(name='model_input',
data=step_3_training_step.outputs["train_url"].as_input()),
                                        outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),
                                        depend_steps=[step_3_training_step]
                                        )

# 定义服务部署节点
step_5_service_step = wf.steps.ServiceStep(
    name="service_step",
    title="部署服务",
    inputs=wf.steps.ServiceInput(name="service_input",
data=wf.data.ServiceInputPlaceholder(name="service_input_ph", model_name=model_name)),
    outputs=wf.steps.ServiceOutput(name="service_output"),
    depend_steps=[step_4_model_registration]
)

# 定义Workflow
workflow =wf.Workflow(name="image_cls",
                      desc="this is a demo workflow",
                      steps=[step_0_data_import, step_1_label_step, step_2_release_data_step,
step_3_training_step, step_4_model_registration, step_5_service_step],
                      session=session,
                      storages=[output_storage]
)

# 运行Workflow
workflow.run(
    # 运行指定steps, 指定的steps须在定义Workflow时定义
    steps=[step_0_data_import, step_1_label_step, step_2_release_data_step, step_3_training_step,
step_4_model_registration, step_5_service_step],
    # Workflow实验记录的文件夹名称
    experiment_id="workflow_e2e_4"
)
```

