

ModelArts

Workflow

文档版本 01
发布日期 2024-04-03



版权所有 © 华为云计算技术有限公司 2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

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

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

目录

| | |
|----------------------------------|-----------|
| 1 MLOps 简介..... | 1 |
| 2 什么是 Workflow..... | 3 |
| 3 如何使用 Workflow..... | 6 |
| 3.1 配置 Workflow..... | 6 |
| 3.1.1 配置入口..... | 6 |
| 3.1.2 运行配置..... | 7 |
| 3.1.3 资源配置..... | 7 |
| 3.1.4 标签配置..... | 9 |
| 3.1.5 消息通知..... | 10 |
| 3.1.6 输入与输出配置..... | 10 |
| 3.1.7 节点参数配置..... | 11 |
| 3.1.8 保存配置..... | 11 |
| 3.2 启动/停止/查找/复制/删除 Workflow..... | 11 |
| 3.3 查看 Workflow 运行记录..... | 13 |
| 3.4 重试/停止/继续运行节点..... | 15 |
| 3.5 部分运行..... | 15 |
| 4 如何开发 Workflow..... | 16 |
| 4.1 核心概念..... | 16 |
| 4.1.1 Workflow..... | 16 |
| 4.1.2 Step..... | 17 |
| 4.1.3 Data..... | 18 |
| 4.1.4 开发态..... | 23 |
| 4.1.5 运行态..... | 23 |
| 4.2 参数配置..... | 24 |
| 4.2.1 功能介绍..... | 24 |
| 4.2.2 属性总览..... | 24 |
| 4.2.3 使用案例..... | 25 |
| 4.3 统一存储..... | 25 |
| 4.3.1 功能介绍..... | 26 |
| 4.3.2 常用方式..... | 26 |
| 4.3.3 进阶用法..... | 26 |
| 4.3.4 使用案例..... | 27 |

| | |
|----------------------|----|
| 4.3.5 相关配置操作..... | 28 |
| 4.4 节点类型..... | 29 |
| 4.4.1 数据集创建节点..... | 29 |
| 4.4.1.1 功能介绍..... | 29 |
| 4.4.1.2 属性总览..... | 29 |
| 4.4.1.3 使用案例..... | 32 |
| 4.4.2 数据集标注节点..... | 34 |
| 4.4.2.1 功能介绍..... | 34 |
| 4.4.2.2 属性总览..... | 34 |
| 4.4.2.3 使用案例..... | 37 |
| 4.4.3 数据集导入节点..... | 39 |
| 4.4.3.1 功能介绍..... | 39 |
| 4.4.3.2 属性总览..... | 39 |
| 4.4.3.3 使用案例..... | 43 |
| 4.4.4 数据集版本发布节点..... | 47 |
| 4.4.4.1 功能介绍..... | 47 |
| 4.4.4.2 属性总览..... | 47 |
| 4.4.4.3 使用案例..... | 50 |
| 4.4.5 作业类型节点..... | 51 |
| 4.4.5.1 功能介绍..... | 51 |
| 4.4.5.2 属性总览..... | 52 |
| 4.4.5.3 资源规格查询..... | 56 |
| 4.4.5.4 使用案例..... | 57 |
| 4.4.6 模型注册节点..... | 66 |
| 4.4.6.1 功能介绍..... | 66 |
| 4.4.6.2 属性总览..... | 66 |
| 4.4.6.3 使用案例..... | 69 |
| 4.4.7 服务部署节点..... | 74 |
| 4.4.7.1 功能介绍..... | 74 |
| 4.4.7.2 属性总览..... | 74 |
| 4.4.7.3 使用案例..... | 77 |
| 4.4.7.4 相关配置操作..... | 79 |
| 4.4.8 条件节点..... | 79 |
| 4.4.8.1 功能介绍..... | 79 |
| 4.4.8.2 属性总览..... | 79 |
| 4.4.8.3 使用案例..... | 81 |
| 4.5 分支控制..... | 85 |
| 4.6 多输入支持数据选择..... | 90 |
| 4.7 编写 Workflow..... | 92 |
| 4.8 调试 Workflow..... | 92 |
| 4.9 发布 Workflow..... | 93 |
| 4.9.1 发布运行态..... | 93 |

| | |
|--|----|
| 4.9.2 发布到 AI Gallery..... | 93 |
| 4.10 高阶能力..... | 94 |
| 4.10.1 部分运行..... | 95 |
| 4.10.2 在 Workflow 中使用大数据能力（DLI/MRS）..... | 95 |
| 4.11 常见问题..... | 96 |
| 4.11.1 开发态调试时，如何查询训练规格？..... | 97 |
| 4.11.2 如何实现多分支？..... | 97 |
| 4.11.3 如何导入对象？..... | 97 |
| 4.11.4 如何定位运行报错？..... | 98 |

1 MLOps 简介

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），基于单个数据集，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。

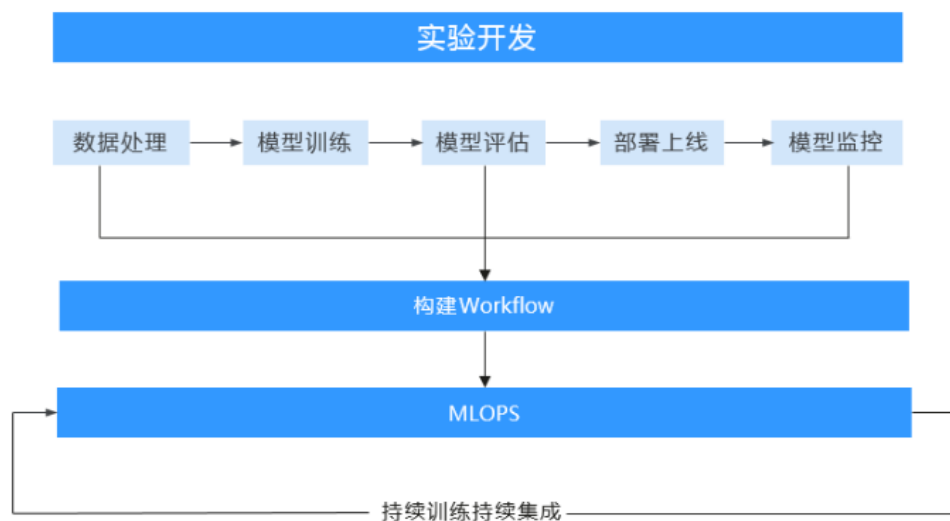
DevOps

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，这对于运维人员来说，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。

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。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。

图 1-1 MLOps



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。MLOps和以往的开发交付不同，在这个过程中，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。

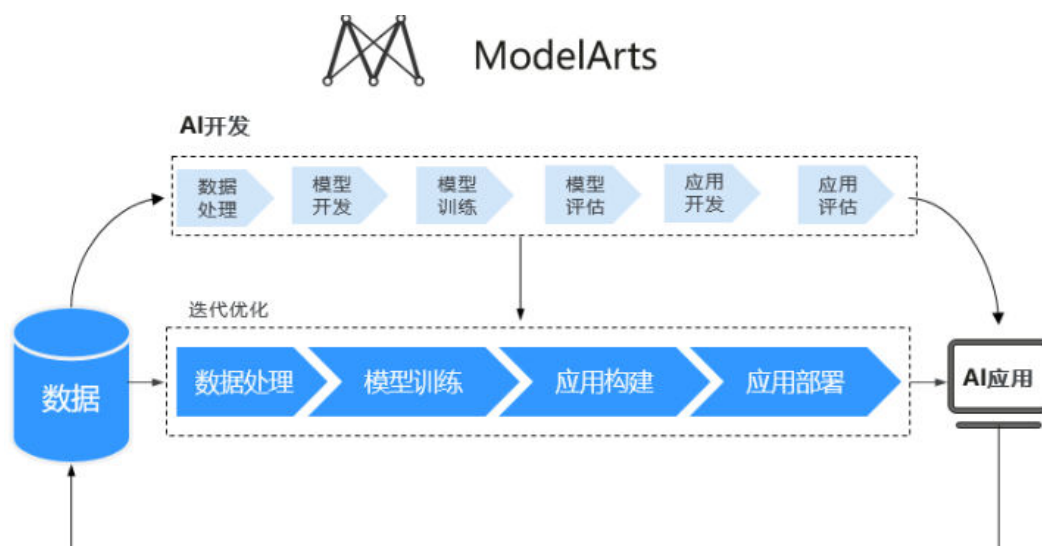
这个工具需要具备以下的能力：

- 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：支持账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，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，快速决定下一轮迭代的执行优化。

2 什么是 Workflow

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。

图 2-1 Workflow



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，提升了AI应用开发与落地效率，更快的进行模型实验和开发，并更快的将模型部署到生产环境。

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 workflow

开发者结合实际业务的需求，通过Workflow提供的Python SDK，将ModelArts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 workflow

Workflow提供了可视化的 workflow 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 workflow 的来源为：通过开发态发布，或者通过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 workflow：启动、停止、复制、删除 workflow。
- 查看运行记录：查看 workflow 历史运行的参数以及状态记录。

Workflow 的构成

workflow 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。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。

Workflow 提供的样例

ModelArts提供了丰富的基于场景的 workflow 样例，用户可以[前往AI Gallery进行订阅](#)。

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的Workflow资产页面，选择并订阅一个Workflow，勾选“我已阅读《数据安全与隐私风险承担条款》和《华为云AI Gallery服务协议》”后，单击“继续订阅”。
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**、**Workflow名称**、**云服务区域**以及**工作空间**，单击“导入”，进入该Workflow的详情页面。

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，参数配置完成后，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. 每一个节点运行状况页面的“状态”为此节点的运行状态，运行成功会自动执行下一个节点的运行，直至所有节点运行成功，代表Workflow完成运行。

3 如何使用 Workflow

[配置Workflow](#)

[启动/停止/查找/复制/删除Workflow](#)

[查看Workflow运行记录](#)

[重试/停止/继续运行节点](#)

[部分运行](#)

3.1 配置 Workflow

3.1.1 配置入口

在运行工作流之前或运行过程中，需要设置工作流相应的参数和使用的资源等内容，这些内容是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，可根据自己的需求修改配置，使生成的模型或应用更贴合业务场景。

Workflow的配置包括运行前的配置和运行中的配置。

运行前配置

登录ModelArts控制台，在左侧导航栏选择**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。

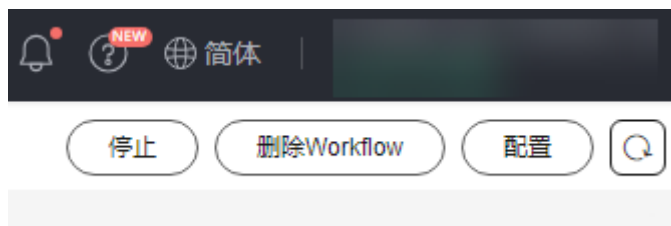
- 单击Workflow列表页操作栏的“配置”，跳转到工作流配置页。

图 3-1 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。

图 3-2 配置 workflow



运行中配置

Workflow可能存在运行中的配置项，运行到该节点会暂停，等待用户输入操作。

在Workflow总览页面，右方查看待办事项，单击您的工作流名称，可直接跳转至该工作流对应的需要输入的节点，在该节点完成相关参数填写后，单击“继续运行”即可。

图 3-3 Workflow 待办项

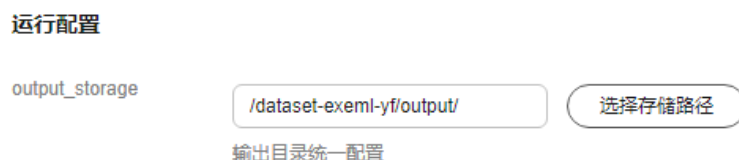


3.1.2 运行配置

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。

1. 在Workflow列表页，单击某条工作流名称，进入工作流详情页面。
2. 单击页面右上角的“配置”。进入工作流配置详情页。
3. 在“Workflow配置”页签，完成“运行配置”。

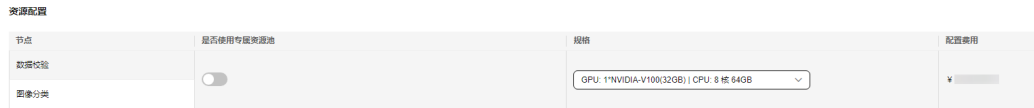
图 3-4 运行配置



3.1.3 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配置不同的规格。所消耗的资源与训练作业/在线推理的费用一致。只有在节点运行时才会产生消耗费用，当节点未运行或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。

图 3-5 资源配置



当您需要使用专属资源池时，将“是否使用专属资源池”的开关打开即可。

推理资源规格配置：

workflow 运行到服务部署节点时，需要手动输入推理的资源规格：

1. 待 workflow 运行至服务部署节点，状态为“等待输入”
2. 在“输入”区域选择推理需要使用到的资源规格。
3. 完成后选择“继续运行”。

图 3-6 输入配置

输入

说明

- **计算节点规格**：华北-北京四可支持限时免费的规格，但每个用户只限创建一个基于此免费规格的实例。其余规格均按需计费，使用完之后请及时停止 Workflow，避免产生不必要的费用。
- 如果您购买了套餐包，计算节点规格可选择您的套餐包，同时在“配置费用”页签还可查看您的套餐包余量以及超出部分的计费方式，请您务必关注，避免造成不必要的资源浪费。

3.1.4 标签配置

Workflow支持通过标签快速识别对应的工作流，用户在使用的过程中可通过标签筛选出匹配的工作流，实现了工作流分类分块的功能，极大程度上节省了用户的时间。

配置标签


1. 在ModelArts管理控制台，左侧菜单栏单击“Workflow”。进入Workflow列表页。
2. 在列表页根据Workflow工作流名称，找到需要打标签的工作流，单击工作流名称，进入工作流详情页。
3. 在工作流详情页，单击左上角编辑按钮。
4. 在弹出的编辑Workflow弹窗中，在标签框中输入相应的标签后，单击“新增标签”，新生成的标签会展示在标签行的下方，您可以同时增加多个标签。标签增加完成后，单击“确定”，标签即可生成。

图 3-7 编辑

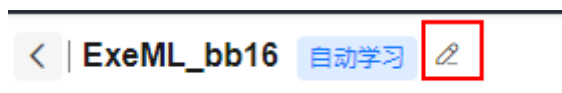


图 3-8 新增标签



通过标签搜索 workflow

生成了标签的 Workflow，支持在搜索框中按照标签筛选对应的 Workflow。

1. 在 Workflow 列表页，上方搜索栏中，属性类型选择“标签”。

图 3-9 标签搜索



2. 系统会关联出当前页面中所有的 Workflow 包含的标签，再选择您需要的标签筛选条件后，Workflow 列表页所展示的 Workflow 为您所需要的工作流。

3.1.5 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。

- 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。
- 支持对 Workflow 中单个节点、多个节点以及工作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工作流的订阅。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。
- 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”三种事件。

3.1.6 输入与输出配置

输入参数与输出参数的配置，可在配置页面进行配置，也可在运行的过程中进行配置。

当运行 Workflow 时，节点处于“等待输入”状态，也可进行相关参数的配置。

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路径。 |
| 标注任务 | 选择您的数据集下的标注任务。 |
| 运行服务 | 选择您已部署的在线服务。 |
| 容器镜像 | 选择您注册模型需要的镜像存储路径。 |

输出配置

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

3.1.7 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。

3.1.8 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。

图 3-10 保存配置



完成配置并保存成功后，单击界面右上角的“启动”按钮，出现启动Workflow的弹窗，单击“确定”，工作流就会启动并进入运行页面。

3.2 启动/停止/查找/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，单击“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”。

- workflow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”。

📖 说明

启动Workflow后，运行过程中将会按需收费，请关注实例状态，完成后的工作流请及时停止，避免产生不必要的费用。

查找

在Workflow列表页，您可以通过搜索框，根据工作流的属性类型快速搜索过滤到相应的工作流，可节省您的时间。


1. 登录ModelArts管理控制台，在左侧导航栏选择Workflow，进入Workflow总览页面。
2. 在工作流列表上方的搜索框中，根据您需要的属性类型，例如：名称、状态、当前节点、启动时间、运行时长或标签等，过滤出相应的工作流。
3. 单击搜索框右侧的  按钮，可设置Workflow列表页需要展示的内容和展示效果。
 - 表格内容折行：默认为关闭状态。启用此功能可以让Workflow列表页中的内容在显示时自动换行。禁用此功能可截断文本，Workflow列表页中仅显示部分内容。
 - 操作列：默认为开启状态，启用此能力可让操作列固定在最后一列永久可见。
 - 自定义显示列：默认所有显示项全部勾选，您可以根据实际需要定义您的显示列。

图 3-11 设置

设置

基础设置

表格内容折行 自动折行
启用此能力可让表格内容自动折行，禁用此功能可截断文本。

操作列 列固定
启用此能力可让操作列固定在最后一列永久可见。


自定义显示列

🔍 搜索

- 名称 (默认)
- 状态
- 当前节点
- 启动时间
- 运行时长
- 标签
- 运行次数
- 创建时间
- 修改时间
- 创建人
- 来源
- 描述

取消

确定

4. 设置完成后，单击“确定”即可。
5. 同时可支持对Workflow显示列进行排序，单击表头中的箭头，就可对该列进行排序。

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。
停止Workflow后，关联的训练作业和在线服务也会停止。

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要求。

说明

新的Workflow名称，必须为1~64位只包含英文、数字、下划线（_）和中划线（-）且以英文开头的名称。

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。

说明

- 删除后的Workflow无法恢复，请谨慎操作。
- 删除Workflow后，对应的训练作业和在线服务不会随之被删除，需要分别在“训练管理>训练作业”和“部署上线>在线服务”页面中手动删除任务。

3.3 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。

1. 在Workflow列表页，单击某条工作流的名称，进入该工作流的详情页面。
2. 在工作流的详情页，左侧区域即为该条工作流的所有运行记录。

图 3-12 查看运行记录



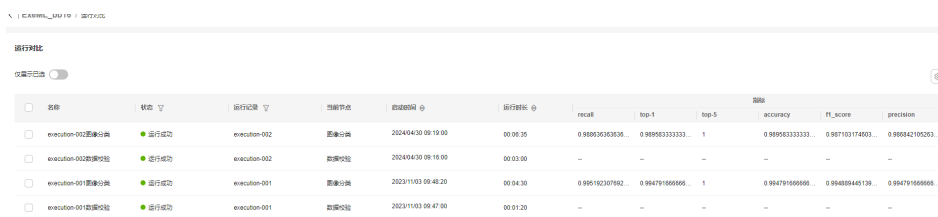
3. 您可以对当前工作流的所有运行记录，进行删除、编辑以及重新运行的操作。
 - 删除：若该条运行记录不再需要，您可以单击“删除”，在弹出的确认框中单击“确定”即可完成运行记录的删除。
 - 编辑：若您想对您当前的工作流下的所有运行记录进行区分，您可以单击“编辑”，对每一条运行记录添加相应的标签予以区分。
 - 重新运行：可以单击“重新运行”直接在某条记录上运行该工作流。
4. 您可以对该条工作流的所有运行记录进行筛选和对比。
 - 筛选：该功能支持您对所有运行记录按照“运行状态”和“运行标签”进行筛选。

图 3-13 筛选



- 对比：针对某条工作流的所有运行记录，按照状态、运行记录、启动时间、运行时长、参数等进行对比。

图 3-14 对比



| 名称 | 状态 | 运行记录 | 当前节点 | 创建时间 | 运行时长 | recall | top-1 | top-5 | accuracy | F1_score | precision |
|-------------------|------|---------------|------|---------------------|----------|----------------|----------------|-------|----------------|----------------|----------------|
| execution-002数据校验 | 运行成功 | execution-002 | 数据校验 | 2024/04/30 09:19:00 | 00:00:35 | 0.9880363636 | 0.9899333333 | 1 | 0.9899333333 | 0.98710314503 | 0.9864210263 |
| execution-002数据校验 | 运行成功 | execution-002 | 数据校验 | 2024/04/30 09:19:00 | 00:03:00 | -- | -- | -- | -- | -- | -- |
| execution-001数据校验 | 运行成功 | execution-001 | 数据校验 | 2023/11/03 09:48:20 | 00:04:30 | 0.995182307692 | 0.994791666666 | 1 | 0.994791666666 | 0.994888445139 | 0.994791666666 |
| execution-001数据校验 | 运行成功 | execution-001 | 数据校验 | 2023/11/03 09:47:00 | 00:01:20 | -- | -- | -- | -- | -- | -- |

当单击“启动”运行 workflow 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 详情页中任一节点查询节点运行状况。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）。

3.4 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 workflow。在当前节点的运行状况页面，单击“重试”。在重试之前您也可以前往全局配置页面修改配置，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“等待操作”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。

3.5 部分运行

针对大型、复杂的 Workflow，为节省重复运行消耗的时间，在运行业务场景时，用户可以选择其中的部分节点作为业务场景运行，workflow 在执行时将会按顺序执行部分运行节点。

- **创建**
通过 SDK 创建工作流时，预先定义好部分运行场景，具体可参考[部分运行](#)。
- **配置**
在配置 workflow 时，打开“部分运行”开关，选择需要执行的部分运行场景，并填写完善相关节点的参数。
- **启动**
保存上一步的配置后，单击“启动”按钮即可启动部分运行场景。

4 如何开发 Workflow

- [核心概念](#)
- [参数配置](#)
- [统一存储](#)
- [节点类型](#)
- [分支控制](#)
- [多输入支持数据选择](#)
- [编写Workflow](#)
- [调试Workflow](#)
- [发布Workflow](#)
- [高阶能力](#)
- [常见问题](#)

4.1 核心概念

4.1.1 Workflow

Workflow是一个有向无环图（Directed Acyclic Graph, DAG），由节点和节点之间的关系描述组成。

节点与节点之间的依赖关系由单箭头的线段来表示，依赖关系决定了节点的执行顺序，示例中的工作流在启动后将从左往右顺序执行。DAG也支持多分支结构，用户可根据实际场景进行灵活设计，在多分支场景下，并行分支的节点支持并行运行。

表 4-1 Workflow

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------|--|------|------------------------|
| name | 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位字符 | 是 | str |
| desc | 工作流的描述信息 | 是 | str |
| steps | 工作流包含的节点列表 | 是 | list[Step] |
| storage_s | 统一存储对象列表 | 否 | Storage或者list[Storage] |
| policy | 工作流的配置策略，主要用于部分运行场景 | 否 | Policy |

4.1.2 Step

Step是组成Workflow的最小单元，体现在DAG中就是一个一个的节点，不同的Step类型承载了不同的服务能力，主要构成如下。

表 4-2 Step

| 属性 | 描述 | 是否必填 | 数据类型 |
|------------|--|------|--------------------------------------|
| name | 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 | 是 | str |
| title | 节点的标题信息，主要用于在DAG中的展示，如果该字段未填写，则默认使用name进行展示 | 否 | str |
| step_type | 节点的类型，决定了节点的功能 | 是 | enum |
| inputs | 节点的输入列表 | 否 | AbstractInput或者list[AbstractInput] |
| outputs | 节点的输出列表 | 否 | AbstractOutput或者list[AbstractOutput] |
| properties | 节点的属性信息 | 否 | dict |

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------|---|------|------------------|
| policy | 节点的执行策略，主要包含节点调度运行的时间间隔、节点执行的超时时间、以及节点执行是否跳过的相关配置 | 否 | StepPolicy |
| depend_steps | 依赖节点的列表，该字段决定了DAG的结构，也决定了节点执行的顺序 | 否 | Step或者list[Step] |

表 4-3 StepPolicy

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------------------|-------------------------|------|------------------------|
| poll_interval_seconds | 节点调度时间周期，默认为1秒 | 是 | str |
| max_execution_minutes | 节点运行超时时间，默认为10080分钟，即7天 | 是 | str |
| skip_conditions | 节点是否跳过的条件列表 | 否 | Condition或者Condition列表 |

Step是节点的超类，主要用于概念上的承载，用户不直接使用。根据功能的不同，构建了不同类型的节点，主要包括**CreateDatasetStep**、**LabelingStep**、**DatasetImportStep**、**ReleaseDatasetStep**、**JobStep**、**ModelStep**、**ServiceStep**、**ConditionStep**等，详情请见[节点类型](#)。

4.1.3 Data

数据对象用于节点的输入，主要可分为以下三种类型：

- 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
 - GalleryModel: 用于定义从gallery订阅的模型，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，对应Dataset对象，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，对应LabelTask对象，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，对应OBSPath对象，常用于模型训练，数据集导入，模型导入等场景

- ServiceUpdatePlaceholder: 用于定义在运行时需要确定的已有服务，对应 ServiceData对象，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，对应 SWRImage对象，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
 - DataSelector: 支持多种数据类型的选择，当前仅支持在JobStep节点中使用（仅支持选择OBS或者数据集）
- 数据选择对象：
 - DataConsumptionSelector: 用于在多个依赖节点的输出中选择一个有效输出作为数据输入，常用于存在条件分支的场景中（在构建工作流时未能确定数据输入来源为哪个依赖节点的输出，需根据依赖节点的实际执行情况进行自动选择）

表 4-4 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 = "", 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）。

表 4-5 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 = "", 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
```

表 4-6 OBSPath

| 属性 | 描述 | 是否必填 | 数据类型 |
|----------|-------|------|--------------|
| obs_path | OBS路径 | 是 | str, Storage |

示例:

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

表 4-7 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 = "**")  
# 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。
```

表 4-8 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 = "**")  
# 容器镜像地址, 用于模型注册节点的输入
```

表 4-9 GalleryModel

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------------|-----------|------|------|
| subscription_id | 订阅模型的订阅ID | 是 | str |
| version_num | 订阅模型的版本号 | 是 | str |

示例:

```
example = GalleryModel(subscription_id="**", version_num="**")  
# 订阅的模型对象, 用于模型注册节点的输入
```

表 4-10 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 |

| 属性 | 描述 | 是否必填 | 数据类型 |
|---------|----------------------------|------|---------|
| delay | 标志数据对象是否在节点运行时配置，默认为 False | 否 | bool |
| default | 数据对象的默认值 | 否 | Dataset |

示例：

```
example = DatasetPlaceholder(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
```

表 4-11 OBSPlaceholder

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------------|----------------------------------|------|---------|
| name | 名称 | 是 | str |
| object_type | 表示OBS对象类型，仅支持"file"或者"directory" | 是 | str |
| delay | 标志数据对象是否在节点运行时配置，默认为 False | 否 | bool |
| default | 数据对象的默认值 | 否 | OBSPath |

示例：

```
example = OBSPlaceholder(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"
```

表 4-12 LabelTaskPlaceholder

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------|----------------------------|------|-------------------|
| name | 名称 | 是 | str |
| task_type | 表示标注任务的类型 | 否 | LabelTaskTypeEnum |
| delay | 标志数据对象是否在节点运行时配置，默认为 False | 否 | bool |

示例：

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

表 4-13 ServiceUpdatePlaceholder

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------|----------------------------|------|------|
| name | 名称 | 是 | str |
| delay | 标志数据对象是否在节点运行时配置，默认为 False | 否 | bool |

示例:

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

表 4-14 SWRImagePlaceholder

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------|----------------------------|------|------|
| name | 名称 | 是 | str |
| delay | 标志数据对象是否在节点运行时配置，默认为 False | 否 | bool |

示例:

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

表 4-15 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或者 Placeholder |
| 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 = "**", model_name = "model_name")  
# 用于服务部署或者服务更新节点的输入
```

表 4-16 DataSelector

| 属性 | 描述 | 是否必填 | 数据类型 |
|----------------|----------------------------|------|------|
| name | 名称 | 是 | str |
| data_type_list | 支持的数据类型列表，当前仅支持obs、dataset | 是 | list |
| delay | 标志数据对象是否在节点运行时配置，默认为False | 否 | bool |

示例：

```
example = DataSelector(name = "***", data_type_list=["obs", "dataset"])  
# 用于作业类型节点的输入
```

表 4-17 DataConsumptionSelector

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------|---------------|------|------|
| data_list | 依赖节点的输出数据对象列表 | 是 | list |

示例：

```
example = DataConsumptionSelector(data_list=[step1.outputs["step1_output_name"].as_input(),  
step2.outputs["step2_output_name"].as_input()])  
# 从step1以及step2中选择有效输出作为输入，当step1跳过无输出，step2执行有输出时，将step2的有效输出作为输入（需保证data_list中同时只有一个有效输出）
```

4.1.4 开发态

开发态主要是指使用Workflow的Python SDK开发和调试工作流，使用上具有一定的门槛，适用于MLOps相关的开发者。该使用方式对于AI开发者来说是非常熟悉的一种开发模式，而且灵活度极高，主要提供以下能力。

- 开发构建：使用python代码灵活编排构建工作流。
- 调试：支持debug以及run两种模式，其中run模式支持节点部分运行、全部运行。
- 发布：支持将调试后的工作流进行固化，发布至运行态，支持配置运行。
- 共享：支持将工作流作为资产发布至AI Gallery，分享给其他用户使用。

4.1.5 运行态

Workflow提供了整套可视化的工作流运行方式，简称为运行态。使用者不需要了解工作流的内部细节，只需要关注一些简单的参数配置即可启动运行工作流。运行态的工作流来源主要为：通过开发态发布或者从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、重试、复制、删除工作流等。
- 运行记录：工作流历史运行的参数以及状态记录。

4.2 参数配置

4.2.1 功能介绍

参数相关的配置使用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，当前支持的数据类型包括：int、str、bool、float、Enum、dict、list。开发者可根据场景需要，将节点中的相关字段（如算法超参）通过Placeholder的形式透出，支持设置默认值，供用户修改配置使用。

4.2.2 属性总览

Placeholder

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------------|---|------|-----------------|
| name | 参数名称，需要保证全局唯一。 | 是 | str |
| 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 PlaceholderType.LIST -> list <ul style="list-style-type: none">• 当类型为PlaceholderType.ENUM时，enum_list字段不能为空。• 当类型为PlaceholderType.LIST时，placeholder_format字段不能为空，且只能填写str/int/float/bool，用来表示list中的数据类型。 | 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、flavor、train_flavor、swr。 | 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 |

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------------|--|------|------|
| 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 |
| constraint | 参数相关的约束配置，当前该字段仅支持训练规格的约束，且用户不感知。 | 否 | dict |
| required | 参数是否必填标记。 <ul style="list-style-type: none">默认required=True。Delay参数不能设required=False。 运行时前端可以不填此参数。 | 否 | bool |

4.2.3 使用案例

- **int类型参数**

```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_int", placeholder_type=wf.PlaceholderType.INT, default=1,
description="这是一个int类型的参数")
```
- **str类型参数**

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

```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_bool", placeholder_type=wf.PlaceholderType.BOOL, default=True,
description="这是一个bool类型的参数")
```
- **float类型参数**

```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_float", placeholder_type=wf.PlaceholderType.FLOAT, default=0.1,
description="这是一个float类型的参数")
```
- **Enum类型参数**

```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_enum", placeholder_type=wf.PlaceholderType.ENUM, default="a",
enum_list=["a", "b"], description="这是一个enum类型的参数")
```
- **dict类型参数**

```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_dict", placeholder_type=wf.PlaceholderType.JSON, default={"key":
"value"}, description="这是一个dict类型的参数")
```
- **list类型参数**

```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_list", placeholder_type=wf.PlaceholderType.LIST, default=[1, 2],
placeholder_format="int", description="这是一个list类型的参数，并且value类型为int")
```

4.3 统一存储

4.3.1 功能介绍

统一存储主要用于工作流的目录管理，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。该方式只做目录的编排，不会自动创建新的目录。
- 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支持输出到不同的目录下，对不同的执行做了很好的数据隔离。

4.3.2 常用方式

- **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，如果storage对象配置的根路径为"/root/"，则最后得到的路径为"/root/directory_path"

- **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，如果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"

4.3.3 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 | 表示是否自动创建目录，默认为“False”。 | 否 | bool |

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------------------|---|------|------|
| with_execution_id | 表示创建目录时是否拼接 execution_id，默认为“False”。该字段只有在create_dir为True时才支持设置为True。 | 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/")
```

workflow运行时, 如果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
```

workflow运行时, 如果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, 需要用户自行保证该路径存在
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=False)
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
```

4.3.4 使用案例

统一存储主要用于JobStep中，下面代码示例全部以单训练节点为例。

```
from modelarts import workflow as wf
# 构建一个InputStorage对象, 并且假设置置的根目录为"/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对象需要在这里添加
)
```

4.3.5 相关配置操作

开发态配置

调用 workflow 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:

图 4-1 等待用户输入



```
INFO:root:start running workflow...
Please input the path of Storage "input_storage": [input path]
Please input the path of Storage "output_storage": [output path]
```

要求用户输入已存在的路径, 否则会报错, 路径格式要求为: /桶名称/文件夹路径/。

运行态配置

调用 workflow 对象的 release 方法将 workflow 发布到运行态, 在 ModelArts 管理控制台, 单击 Workflow 找到相应的工作流进行根路径配置, 如下所示:

图 4-2 根目录配置

运行配置



4.4 节点类型

4.4.1 数据集创建节点

4.4.1.1 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。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，后续常接数据集导入节点或者数据集标注节点。

4.4.1.2 属性总览

您可以使用CreateDatasetStep来构建数据集创建节点，CreateDatasetStep及相关对象结构如下。

表 4-18 CreateDatasetStep

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------------|--|------|---|
| name | 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。 | 是 | str |
| inputs | 数据集创建节点的输入列表。 | 是 | CreateDatasetInput或者CreateDatasetInput的列表 |
| outputs | 数据集创建节点的输出列表。 | 是 | CreateDatasetOutput或者CreateDatasetOutput的列表 |
| properties | 数据集创建相关的配置信息。 | 是 | DatasetProperties |
| title | title信息，主要用于前端的名称展示。 | 否 | str |
| description | 数据集创建节点的描述信息。 | 否 | str |

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------|--------------|------|---------------|
| policy | 节点执行的policy。 | 否 | StepPolicy |
| depend_steps | 依赖的节点列表。 | 否 | Step或者Step的列表 |

表 4-19 CreateDatasetInput

| 属性 | 描述 | 是否必填 | 数据类型 |
|------|--|------|---|
| name | 数据集创建节点的输入名称，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)。同一个Step的输入名称不能重复。 | 是 | str |
| data | 数据集创建节点的输入数据对象。 | 是 | OBS相关对象，当前仅支持OBSPath、OBSConsumption、OBSPlaceholder、DataConsumption Selector |

表 4-20 CreateDatasetOutput

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------|--|------|----------------------|
| name | 数据集创建节点的输出名称，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)。同一个Step的输出名称不能重复。 | 是 | str |
| config | 数据集创建节点的输出相关配置。 | 是 | 当前仅支持OBSOutputConfig |

表 4-21 DatasetProperties

| 属性 | 描述 | 是否必填 | 数据类型 |
|----------------|---|------|--------------------|
| dataset_name | 数据集的名称，只能是中文、字母、数字、下划线或中划线组成的合法字符串，长度为1-100位。 | 是 | str、Placeholder |
| dataset_format | 数据集格式，默认为0，表示文件类型。 | 否 | 0: 文件类型 1: 表格类型 |
| data_type | 数据类型，默认为FREE_FORMAT。 | 否 | DataTypeEnum |
| description | 描述信息。 | 否 | str |
| import_data | 是否要导入数据，当前只支持表格数据，默认为False。 | 否 | bool |
| work_path_type | 数据集输出路径类型，当前仅支持OBS，默认为0。 | 否 | int |
| import_config | 标签导入的相关配置，默认为None，当基于已标注的数据创建数据集时，可指定该字段导入相关标注信息。 | 否 | ImportConfig |

表 4-22 Importconfig

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------------------|--|------|------------------------------|
| import_annotations | 是否自动导入输入目录下的标注信息，支持检测/图像分类/文本分类。可选值如下： <ul style="list-style-type: none"> • true: 导入输入目录下的标注信息（默认值） • false: 不导入输入目录下的标注信息 | 否 | str、Placeholder |
| import_type | 导入方式。可选值如下： <ul style="list-style-type: none"> • dir: 目录导入 • manifest: 按manifest文件导入 | 否 | 0: 文件类型 ImportTypeEnum |
| annotation_format_config | 导入的标注格式的配置参数。 | 否 | DAnnotationFormatTypeEnum的列表 |

表 4-23 AnnotationFormatConfig

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------------|------------|------|----------------------|
| format_name | 标注格式的名称。 | 否 | AnnotationFormatEnum |
| scene | 标注场景，可选参数。 | 否 | LabelTaskTypeEnum |

| 枚举类型 | 枚举值 |
|----------------------|--|
| ImportTypeEnum | DIR MANIFEST |
| DataTypeEnum | IMAGE TEXT AUDIO TABULAR VIDEO FREE_FORMAT |
| 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 |

4.4.1.3 使用案例

主要包含两种场景的用例。

- 基于未标注数据创建数据集
- 基于已标注的数据创建数据集，并自动导入标注信息

基于未标注数据创建数据集

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如果不存在, 则创建新的数据集;如果已存在, 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

workflow = wf.Workflow(
    name="create-dataset-demo",
    desc="this is a demo workflow",
    steps=[create_dataset]
)
```

基于已标注数据创建数据集，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如果不存在, 则创建新的数据集;如果已存在, 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
                    the label format
                    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  
  
workflow = wf.Workflow(  
    name="create-dataset-demo",  
    desc="this is a demo workflow",  
    steps=[create_dataset]  
)
```

4.4.2 数据集标注节点

4.4.2.1 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，主要用于需要对数据进行人工标注的场景。

4.4.2.2 属性总览

您可以使用LabelingStep来构建数据集标注节点，LabelingStep结构如下：

表 4-24 LabelingStep

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------------|---|------|-----------------------------------|
| name | 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 | 是 | str |
| inputs | 数据集标注节点的输入列表 | 是 | LabelingInput或者LabelingInput的列表 |
| outputs | 数据集标注节点的输出列表 | 是 | LabelingOutput或者LabelingOutput的列表 |
| properties | 数据集标注相关的配置信息 | 是 | LabelTaskProperties |
| title | title信息，主要用于前端的名称展示 | 否 | str |
| description | 数据集标注节点的描述信息 | 否 | str |

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------|-------------|------|---------------|
| policy | 节点执行的policy | 否 | StepPolicy |
| depend_steps | 依赖的节点列表 | 否 | Step或者Step的列表 |

表 4-25 LabelingInput

| 属性 | 描述 | 是否必填 | 数据类型 |
|------|---|------|---|
| name | 数据集标注节点的输入名称，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)。同一个Step的输入名称不能重复 | 是 | str |
| data | 数据集标注节点的输入数据对象 | 是 | 数据集或标注任务相关对象，当前仅支持Dataset, DatasetConsumption, DatasetPlaceholder, LabelTask, LabelTaskPlaceholder, LabelTaskConsumption, DataConsumptionSelector |

表 4-26 LabelingOutput

| 属性 | 描述 | 是否必填 | 数据类型 |
|------|---|------|------|
| name | 数据集标注节点的输出名称，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)。同一个Step的输出名称不能重复 | 是 | str |

表 4-27 LabelTaskProperties

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------------------|---|------|-------------------|
| task_type | 标注任务类型，返回指定标注任务类型的任务列表。 | 是 | LabelTaskTypeEnum |
| task_name | 标注任务名称，名称只能包含中文、字母、数字、中划线和下划线，长度为1-100位。 当输入是数据集对象时，该参数必填 | 否 | str、Placeholder |
| labels | 待创建的标签列表 | 否 | Label |
| properties | 标注任务的属性，可扩展字段，可以记录自定义信息。 | 否 | dict |
| auto_sync_dataset | 标注任务的标注结果是否自动同步至数据集。可选值如下： <ul style="list-style-type: none">• true：标注任务的标注结果自动同步至数据集（默认值）• false：标注任务的标注结果不自动同步至数据集 | 否 | bool |
| content_labeling | 语音分割标注任务是否开启内容标注，默认开启。 | 否 | bool |
| description | 标注任务描述信息，长度为0-256位，不能包含^!<>=&""特殊字符。 | 否 | str |

表 4-28 Label

| 属性 | 描述 | 是否必填 | 数据类型 |
|----------|--------------------|------|---------------------|
| name | 标签名称 | 否 | str |
| property | 标签基本属性键值对，如颜色、快捷键等 | 否 | str、dic、Placeholder |

| 属性 | 描述 | 是否必填 | 数据类型 |
|------|------|------|---------------|
| type | 标签类型 | 否 | LabelTypeEnum |

| 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 |

4.4.2.3 使用案例

主要包含三种场景的用例：

- 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。
- 基于指定的标注任务进行标注。
- 基于数据集创建节点的输出创建标注任务。

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如果不存在则创建, 如果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
)

workflow = wf.Workflow(
    name="labeling-step-demo",
    desc="this is a demo workflow",
    steps=[labeling]
)
```

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
)

workflow = wf.Workflow(
    name="labeling-step-demo",
    desc="this is a demo workflow",
    steps=[labeling]
)
```

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="create_dataset_output",
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如果不存在，则创建新的数据集;如果已存在，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
)
)

# 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如果不存在则创建，如果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
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

workflow = wf.Workflow(
    name="labeling-step-demo",
    desc="this is a demo workflow",
    steps=[create_dataset, labeling]
)
```

4.4.3 数据集导入节点

4.4.3.1 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，主要应用场景如下：

- 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。

4.4.3.2 属性总览

您可以使用**DatasetImportStep**来构建数据集导入节点，**DatasetImportStep**结构如下。

表 4-29 DatasetImportStep

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------|---|------|---|
| name | 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。 | 是 | str |
| inputs | 数据集导入节点的输入列表。 | 是 | DatasetImportInput或者DatasetImportInput的列表 |
| outputs | 数据集导入节点的输出列表。 | 是 | DatasetImportOutput或者DatasetImportOutput的列表 |
| properties | 数据集导入相关的配置信息。 | 是 | ImportDataInfo |
| title | title信息，主要用于前端的名称展示。 | 否 | str |
| description | 数据集导入节点的描述信息。 | 否 | str |
| policy | 节点执行的policy。 | 否 | StepPolicy |
| depend_steps | 依赖的节点列表。 | 否 | Step或者Step的列表 |

表 4-30 DatasetImportInput

| 属性 | 描述 | 是否必填 | 数据类型 |
|------|---|------|------|
| name | 数据集导入节点的输入名称，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。同一个Step的输入名称不能重复。 | 是 | str |

| 属性 | 描述 | 是否必填 | 数据类型 |
|------|-----------------|------|---|
| data | 数据集导入节点的输入数据对象。 | 是 | 数据集、OBS或标注任务相关对象，当前仅支持 Dataset, DatasetConsumption, DatasetPlaceholder, OBSPath, OBSConsumption, OBSPlaceholder, LabelTask, LabelTaskPlaceholder, LabelTaskConsumption, DataConsumptionSelector |

表 4-31 DatasetImportOutput

| 属性 | 描述 | 是否必填 | 数据类型 |
|------|--|------|------|
| name | 数据集导入节点的输出名称，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)。同一个Step的输出名称不能重复。 | 是 | str |

表 4-32 ImportDataInfo

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------------------|---------------|------|------------------------|
| annotation_format_config | 导入的标注格式的配置参数。 | 否 | AnnotationFormatConfig |
| excluded_labels | 不导入包含指定标签的样本。 | 否 | Label的列表 |

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------------|---|------|----------------|
| import_annotated | 用于导入智能标注结果的任务，是否导入原数据集中已标注的样本到待确认，默认值为"false"即不导入原数据集中已标注的样本到待确认。可选值如下： <ul style="list-style-type: none">• true: 导入原数据集中已标注的样本到待确认• false: 不导入原数据集中已标注的样本到待确认 | 否 | bool |
| import_annotations | 是否导入标签。可选值如下： <ul style="list-style-type: none">• true: 导入标签（默认值）• false: 不导入标签 | 否 | bool |
| import_samples | 是否导入样本。可选值如下： <ul style="list-style-type: none">• true: 导入样本（默认值）• false: 不导入样本 | 否 | bool |
| import_type | 导入方式。可选值如下： <ul style="list-style-type: none">• dir: 目录导入• manifest: 按manifest文件导入 | 否 | ImportTypeEnum |
| included_labels | 导入包含指定标签的样本。 | 否 | Label的列表 |
| label_format | 标签格式，此参数仅文本类数据集使用。 | 否 | LabelFormat |

表 4-33 AnnotationFormatConfig

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------------|------------|------|-----------------------------|
| format_name | 标注格式的名称。 | 否 | AnnotationFormat Enum |
| parameters | 标注格式的高级参数。 | 否 | AnnotationFormat Parameters |
| scene | 标注场景，可选参数。 | 否 | LabelTaskTypeEnum |

表 4-34 AnnotationFormatParameters

| 属性 | 描述 | 是否必填 | 数据类型 |
|------------------------|---|------|----------|
| difficult_only | 是否只导入难例。可选值如下： <ul style="list-style-type: none">• true: 只导入难例样本• false: 导入全部样本（默认值） | 否 | bool |
| included_labels | 导入包含指定标签的样本。 | 否 | Label的列表 |
| label_separator | 标签与标签之间的分隔符，默认为逗号分隔，分隔符需转义。分隔符仅支持一个字符，必须为大小写字母，数字和“!@#\$%^&* _= ?/!/:;,” 其中的某一字符。 | 否 | str |
| sample_label_separator | 文本与标签之间的分隔符，默认为Tab键分隔，分隔符需转义。分隔符仅支持一个字符，必须为大小写字母，数字和“!@#\$%^&* _= ?/!/:;,” 其中的某一字符。 | 否 | str |

4.4.3.3 使用案例

主要包含三种场景的用例：

- 将指定存储路径下的数据导入到目标数据集中
 - 基于已标注的数据导入到数据集中。
 - 基于未标注的数据导入到数据集中。
- 将指定存储路径下的数据导入到指定标注任务中。
 - 基于已标注的数据导入到标注任务中。

- 基于未标注的数据导入到标注任务中。
- 基于数据集创建节点构建数据集导入节点。

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

dataset_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
            )
        ]
    )
)

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

- 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

dataset_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
)

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

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

dataset_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
            )
        ]
    )
)

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

- 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

dataset_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
)

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

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
必须是file或者directory

dataset_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

workflow = wf.Workflow(
    name="dataset-import-demo",
    desc="this is a demo workflow",
    steps=[dataset_import]
)
```

4.4.4 数据集版本发布节点

4.4.4.1 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。主要应用场景如下：

- 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。

4.4.4.2 属性总览

您可以使用**ReleaseDatasetStep**来构建数据集版本发布节点，**ReleaseDatasetStep**结构如下：

表 4-35 ReleaseDatasetStep

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------|---|------|---|
| name | 数据集版本发布节点的名称，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 | 是 | str |
| inputs | 数据集版本发布节点的输入列表 | 是 | ReleaseDatasetInput或者ReleaseDatasetInput的列表 |
| outputs | 数据集版本发布节点的输出列表 | 是 | ReleaseDatasetOutput或者ReleaseDatasetOutput的列表 |
| title | title信息，主要用于前端的名称展示 | 否 | str |
| description | 数据集版本发布节点的描述信息 | 否 | str |
| policy | 节点执行的policy | 否 | StepPolicy |
| depend_steps | 依赖的节点列表 | 否 | Step或者Step的列表 |

表 4-36 ReleaseDatasetInput

| 属性 | 描述 | 是否必填 | 数据类型 |
|------|---|------|--|
| name | 数据集版本发布节点的输入名称，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)。同一个Step的输入名称不能重复 | 是 | str |
| data | 数据集版本发布节点的输入数据对象 | 是 | 数据集或标注任务相关对象，当前仅支持 Dataset, DatasetConsumption, DatasetPlaceholder, LabelTask, LabelTaskPlaceholder, LabelTaskConsumption, DataConsumptionSelector |

表 4-37 ReleaseDatasetOutput

| 属性 | 描述 | 是否必填 | 数据类型 |
|------------------------|---|------|----------------------|
| name | 数据集版本发布节点的输出名称，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)。同一个Step的输出名称不能重复 | 是 | str |
| dataset_version_config | 数据集版本发布相关配置信息 | 是 | DatasetVersionConfig |

表4 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 |

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------------------------|---|------|--|
| 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或者Placeholder |
| clear_hard_property | 是否清除难例，默认为"True"。 | 否 | bool或者Placeholder |
| remove_sample_usage | 是否清除数据集已有的usage信息，默认为"True"。 | 否 | bool或者Placeholder |
| 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 支持以下几种类型： <ul style="list-style-type: none">• IMAGE_CLASSIFICATION（图像分类）• OBJECT_DETECTION = 1（物体检测）• IMAGE_SEGMENTATION（图像分割）• TEXT_CLASSIFICATION（文本分类）• NAMED_ENTITY_RECOGNITION（命名实体）• TEXT_TRIPLE（文本三元组）• AUDIO_CLASSIFICATION（声音分类）• SPEECH_CONTENT SPEECH_SEGMENTATION（语音内容） SPEECH_SEGMENTATION（语音分割）• TABLE（表格数据）• VIDEO_ANNOTATION（视频标注） |
| description | 版本描述信息。 | 否 | str |

📖 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()`

4.4.4.3 使用案例

主要包含三种场景的用例。

- 基于数据集发布版本
- 基于标注任务发布版本
- 基于数据集标注节点的输出发布版本

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_ratio = 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_ratio # 数据集的训练验证切分比
        )
    ) # ReleaseDatasetStep的输出
)

workflow = wf.Workflow(
    name="dataset-release-demo",
    desc="this is a demo workflow",
    steps=[release_version]
)
```

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_ratio = 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_ratio)), # 数据集的
    训练验证切分比
)

workflow = wf.Workflow(
    name="dataset-release-demo",
    desc="this is a demo workflow",
    steps=[release_version]
)
```

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

# 定义训练验证切分比参数
train_ratio = 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=labeling_step.outputs["output_name"].as_input()), # 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_ratio)), # 数据集的
    训练验证切分比
    depend_steps = [labeling_step] # 依赖的数据集标注节点对象
)
# labeling_step是wf.steps.LabelingStep的实例对象, output_name是wf.steps.LabelingOutput的name字段值

workflow = wf.Workflow(
    name="dataset-release-demo",
    desc="this is a demo workflow",
    steps=[release_version]
)
```

4.4.5 作业类型节点

4.4.5.1 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。主要应用场景如下:

- 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。

4.4.5.2 属性总览

您可以使用JobStep来构建作业类型节点，JobStep结构如下

表 4-38 JobStep

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------|---|------|--|
| name | 作业节点的名称，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 | 是 | str |
| algorithm | 算法对象 | 是 | BaseAlgorithm, Algorithm, AIGalleryAlgorithm |
| spec | 作业使用的资源规格相关配置 | 是 | JobSpec |
| inputs | 作业节点的输入列表 | 是 | JobInput或者JobInput的列表 |
| outputs | 作业节点的输出列表 | 是 | JobOutput或者JobOutput的列表 |
| title | title信息，主要用于前端的名称展示 | 否 | str |
| description | 作业节点的描述信息 | 否 | str |
| policy | 节点执行的policy | 否 | StepPolicy |
| depend_steps | 依赖的节点列表 | 否 | Step或者Step的列表 |

表 4-39 JobInput

| 属性 | 描述 | 是否必填 | 数据类型 |
|------|---|------|---|
| name | 作业类型节点的输入名称，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。同一个Step的输入名称不能重复 | 是 | str |
| data | 作业类型节点的输入数据对象 | 是 | 数据集或OBS相关对象，当前仅支持Dataset, DatasetPlaceholder, DatasetConsumption, OBSPath, OBSConsumption, OBSPlaceholder, DataConsumption Selector |

表 4-40 JobOutput

| 属性 | 描述 | 是否必填 | 数据类型 |
|----------------|---|------|-----------------|
| name | 作业类型节点的输出名称，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。同一个Step的输出名称不能重复 | 是 | str |
| obs_config | 输出的OBS相关配置 | 否 | OBSOutputConfig |
| model_config | 输出的模型相关配置 | 否 | ModelConfig |
| metrics_config | metrics相关配置 | 否 | MetricsConfig |

表 4-41 OBSOutputConfig

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------------|-----------------|------|-------------------------|
| obs_path | 已存在的OBS目录 | 是 | str、Placeholder、Storage |
| metric_file | 存储metric信息的文件名称 | 否 | str、Placeholder |

表 4-42 BaseAlgorithm

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------------|-----------|------|-------------------------|
| id | 算法管理的算法ID | 否 | str |
| subscription_id | 订阅算法的订阅ID | 否 | str |
| item_version_id | 订阅算法的版本号 | 否 | str |
| code_dir | 代码目录 | 否 | str、Placeholder、Storage |
| boot_file | 启动文件 | 否 | str、Placeholder、Storage |
| command | 启动命令 | 否 | str、Placeholder |
| parameters | 算法超参 | 否 | AlgorithmParameters的列表 |
| engine | 作业使用的镜像信息 | 否 | JobEngine |
| environments | 环境变量 | 否 | dict |

表 4-43 Algorithm

| 属性 | 描述 | 是否必填 | 数据类型 |
|--------------|-----------|------|------------------------|
| algorithm_id | 算法管理的算法ID | 是 | str |
| parameters | 算法超参 | 否 | AlgorithmParameters的列表 |

表 4-44 AIGalleryAlgorithm

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------------|-----------|------|------------------------|
| subscription_id | 订阅算法的订阅ID | 是 | str |
| item_version_id | 订阅算法的版本号 | 是 | str |
| parameters | 算法超参 | 否 | AlgorithmParameters的列表 |

表 4-45 AlgorithmParameters

| 属性 | 描述 | 是否必填 | 数据类型 |
|-------|---------|------|---|
| name | 算法超参的名称 | 是 | str |
| value | 算法超参的值 | 是 | int, bool, float, str, Placeholder, Storage |

表 4-46 JobEngine

| 属性 | 描述 | 是否必填 | 数据类型 |
|----------------|-------|------|------------------|
| engine_id | 镜像ID | 否 | str, Placeholder |
| engine_name | 镜像名称 | 否 | str, Placeholder |
| engine_version | 镜像版本 | 否 | str, Placeholder |
| image_url | 镜像url | 否 | str, Placeholder |

表 4-47 JobSpec

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------------|-------------|------|----------------|
| resource | 资源信息 | 是 | JobResource |
| log_export_path | 日志输出路径 | 否 | LogExportPath |
| schedule_policy | 作业调度配置策略 | 否 | SchedulePolicy |
| volumes | 作业挂载的文件系统信息 | 否 | list[Volume] |

表 4-48 JobResource

| 属性 | 描述 | 是否必填 | 数据类型 |
|------------|----------------------|------|------------------|
| flavor | 资源规格 | 是 | Placeholder |
| node_count | 节点个数，默认为1，多节点表示支持分布式 | 否 | int, Placeholder |

表 4-49 SchedulePolicy

| 属性 | 描述 | 是否必填 | 数据类型 |
|----------|-------------------------------------|------|------------------|
| priority | 作业调度的优先级，仅支持配置为1、2、3，分别对应低、中、高三种优先级 | 是 | int, Placeholder |

表 4-50 Volume

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----|-----------|------|------|
| nfs | NFS文件系统对象 | 否 | NFS |

表 4-51 NFS

| 属性 | 描述 | 是否必填 | 数据类型 |
|-----------------|--------------|------|-------------------|
| nfs_server_path | NFS文件系统的服务地址 | 是 | str, Placeholder |
| local_path | 挂载到容器里面的路径 | 是 | str, Placeholder |
| read_only | 是否只读的方式挂载 | 否 | bool, Placeholder |

4.4.5.3 资源规格查询

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

- 导包

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

- **session初始化**
 - # 如果您在本地IDEA环境中开发工作流，则Session初始化使用如下方式
 - # 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
 - # 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

```
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_SDK_SK"]
```
 - # 如果进行了加密还需要进行解密操作
 - ```
session = Session(
 access_key=__AK, # 账号的AK信息
 secret_key=__SK, # 账号的SK信息
 region_name="****", # 账号所属的region
 project_id="****", # 账号的项目ID
)
```
  - # 如果您在Notebook环境中开发工作流，则Session初始化使用如下方式
  - ```
session = Session()
```
- **公共池查询**
 - # 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
print(spec_list)
```
- **专属池查询**
  - # 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
```
 - 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**
 - # 引擎规格查询
 - ```
engine_dict = TrainingJob(session).get_engine_list(session) # 返回的类型为dict,按需打印查看
print(engine_dict)
```

#### 4.4.5.4 使用案例

主要包含七种场景的用例：

- 使用订阅自AI Gallery的算法
- 使用算法管理中的算法
- 使用自定义算法（代码目录+启动文件+官方镜像）
- 使用自定义算法（代码目录+脚本命令+自定义镜像）
- 基于数据集版本发布节点构建作业类型节点
- 作业类型节点结合可视化能力
- 输入使用DataSelector对象，支持选择OBS或者数据集

#### 使用订阅自 AI Gallery 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，也可直接填写版本号
 item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
 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；部分算法超参的值如果无需修改，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
)

workflow = wf.Workflow(
 name="job-step-demo",
 desc="this is a demo workflow",
 steps=[job_step],
 storages=[storage]
)
```

## 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；部分算法超参的值如果无需修改，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
)

workflow = wf.Workflow(
 name="job-step-demo",
 desc="this is a demo workflow",
 steps=[job_step],
 storages=[storage]
)
```

## 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
)

workflow = wf.Workflow(
 name="job-step-demo",
 desc="this is a demo workflow",
 steps=[job_step],
 storages=[storage]
)
```



## 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，格式为：组织名/镜像名
 # 版本:版本号，不需要携带相应的域名地址；如果image_url需要设置为运行态可配置，则使用如下方式：
 image_url=wf.Placeholder(name="image_url", placeholder_type=wf.PlaceholderType.STR,
 placeholder_format="swr", description="自定义镜像")
 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
)

workflow = wf.Workflow(
 name="job-step-demo",
 desc="this is a demo workflow",
 steps=[job_step],
 storages=[storage]
)
```

### 📖 说明

上述四种方式使用数据集对象作为输入，如果您需要使用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
```

```
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_ratio = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR,
default="0.8")

release_version_step = 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_ratio # 数据集的训练验证切分比
)
) # ReleaseDatasetStep的输出
)

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AI Gallery订阅的算法; 部分算法超参的值如果无需修改, 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

workflow = wf.Workflow(
 name="job-step-demo",
 desc="this is a demo workflow",
 steps=[release_version_step, job_step],
 storages=[storage]
)
```

## 作业类型节点结合可视化能力

节点可视化特性将用户在使用Workflow时产生的一些衡量指标进行一个可视化的展示，支持数据的实时可视化，并且允许独立呈现可视化外挂节点。形态上基于作业类型节点原有的使用方式，新增一个针对metrics信息展示的输出，通过MetricsConfig对象进行配置。

表 4-52 MetricsConfig

| 属性                     | 描述                   | 是否必填 | 数据类型                                  |
|------------------------|----------------------|------|---------------------------------------|
| metric_files           | metrics输出文件列表        | 是    | list，列表内元素支持（str、Placeholder、Storage） |
| realtime_visualization | 输出的metrics信息是否需要实时展示 | 否    | bool，默认为False                         |
| visualization          | 是否呈现独立的可视化节点         | 否    | bool，默认为True                          |

对于输出的metrics文件，数据内容必须为标准的json数据，大小限制为1M，并且与当前支持的几种数据格式保持一致：

- 键值对类型的数据

```
[
 {
 "key": "loss",
 "title": "loss",
 "type": "float",
 "data": {
 "value": 1.2
 }
 },
 {
 "key": "accuracy",
 "title": "accuracy",
 "type": "float",
 "data": {
 "value": 1.6
 }
 }
]
```

- 折线图数据(type是line chart)

```
[
 {
 "key": "metric",
 "title": "metric",
 "type": "line chart",
 "data": {
 "x_axis": [
 {
 "title": "step/epoch",
 "value": [
 1,
 2,
 3
]
 }
]
 }
 }
]
```

```
 },
],
 "y_axis": [
 {
 "title": "value",
 "value": [
 0.5,
 0.4,
 0.3
]
 }
]
 }
}
```

- 柱状图数据(type是histogram)

```
[
 {
 "key": "metric",
 "title": "metric",
 "type": "histogram",
 "data": {
 "x_axis": [
 {
 "title": "step/epoch",
 "value": [
 1,
 2,
 3
]
 }
],
 "y_axis": [
 {
 "title": "value",
 "value": [
 0.5,
 0.4,
 0.3
]
 }
]
 }
 }
]
```

- 混淆矩阵

```
[
 {
 "key": "confusion_matrix",
 "title": "confusion_matrix",
 "type": "table",
 "data": {
 "cell_value": [
 [
 1,
 2
],
 [
 2,
 3
]
]
 },
 "col_labels": {
 "title": "labels",
 "value": [
 "daisy",
 "dandelion"
]
 }
 }
]
```

```
]
 },
 "row_labels": {
 "title": "predictions",
 "value": [
 "daisy",
 "dandelion"
]
 }
}
]
```

- 一维表格

```
[
 {
 "key": "Application Evaluation Results",
 "title": "Application Evaluation Results",
 "type": "one-dimensional-table",
 "data": {
 "cell_value": [
 [
 10,
 2,
 0.5
]
],
 "labels": [
 "samples",
 "maxResTine",
 "p99"
]
 }
 }
]
```

#### 使用案例:

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

```
构建一个Storage对象, 对训练输出目录做统一管理
```

```
storage = wf.data.Storage(name="storage_name", title="title_info", description="description_info",
with_execution_id=True, create_dir=True) # 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", # 算法订阅版本号, 也可直接填写版本号
```

```
 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AI Gallery订阅的算法; 部分算法超参的值如果无需修改, 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
 wf.steps.JobOutput(name="metrics_output",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # 相关metrics信息由作业脚本代码自行输出到配置的路径下
],
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
)

workflow = wf.Workflow(
 name="job-step-demo",
 desc="this is a demo workflow",
 steps=[job_step],
 storages=[storage]
)
```

### 📖 说明

Workflow不会自动获取训练输出的指标信息，要求用户自行在算法代码中获取指标信息并且按照指定的数据格式构造出metrics.json文件，自行上传到MetricsConfig中配置的OBS路径下，Workflow只进行数据的读取以及渲染展示。

## 输入使用 DataSelector 对象，支持选择 OBS 或者数据集

该方式主要用于输入支持可选择的场景，使用DataSelector对象作为输入时，用户在页面配置时可自由选择数据集对象或者OBS对象作为训练的输入，代码示例如下：

```
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DataSelector对象
data_selector = wf.data.DataSelector(name="input_data", data_type_list=["dataset", "obs"])

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，也可直接填写版本号
 item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
 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；部分算法超参的值如果无需修改，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=data_selector), # JobStep的输入在运行时配置,可自由选择
OBS或者数据集作为输入
 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
)

workflow = wf.Workflow(
 name="job-step-demo",
 desc="this is a demo workflow",
 steps=[job_step],
 storages=[storage]
)
```

#### 📖 说明

使用DataSelector作为输入时，需要用户自行保证算法的输入同时支持数据集或者OBS。

## 4.4.6 模型注册节点

### 4.4.6.1 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。主要应用场景如下：

- 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。

### 4.4.6.2 属性总览

您可以使用**ModelStep**来构建模型注册节点，**ModelStep**结构如下：

表 4-53 ModelStep

| 属性          | 描述                                                                                 | 是否必填 | 数据类型                        |
|-------------|------------------------------------------------------------------------------------|------|-----------------------------|
| name        | 模型注册节点的名称。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 | 是    | str                         |
| inputs      | 模型注册节点的输入列表                                                                        | 否    | ModelInput或者ModelInput的列表   |
| outputs     | 模型注册节点的输出列表                                                                        | 是    | ModelOutput或者ModelOutput的列表 |
| title       | title信息，主要用于前端的名称展示                                                                | 否    | str                         |
| description | 模型注册节点的描述信息                                                                        | 否    | str                         |
| policy      | 节点执行的policy                                                                        | 否    | StepPolicy                  |

| 属性           | 描述      | 是否必填 | 数据类型          |
|--------------|---------|------|---------------|
| depend_steps | 依赖的节点列表 | 否    | Step或者Step的列表 |

表 4-54 ModelInput

| 属性   | 描述                                                                           | 是否必填 | 数据类型                                                                                                                                |
|------|------------------------------------------------------------------------------|------|-------------------------------------------------------------------------------------------------------------------------------------|
| name | 模型注册节点的输入名称，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。同一个Step的输入名称不能重复 | 是    | str                                                                                                                                 |
| data | 模型注册节点的输入数据对象                                                                | 是    | OBS、SWR或订阅模型相关对象，当前仅支持OBSPath, SWRImage, OBSConsumption, OBSPlaceholder, SWRImagePlaceholder, DataConsumptionSelector, GalleryModel |

表 4-55 ModelOutput

| 属性           | 描述                                                                           | 是否必填 | 数据类型        |
|--------------|------------------------------------------------------------------------------|------|-------------|
| name         | 模型注册节点的输出名称，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。同一个Step的输出名称不能重复 | 是    | str         |
| model_config | 模型注册相关配置信息                                                                   | 是    | ModelConfig |



表 4-56 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", "Custom") 默认为 TensorFlow。 | 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、Placeholder  |
| 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。该字段不填时，版本号自动增加。<br><b>注意</b><br>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相同。                                                                                                                 | 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”。<br>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              |
| 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             |
| template          | 模板的相关配置项，使用模板导入模型(即model_type为Template)时必选                                                                                                        | 否    | Template         |
| dynamic_load_mode | 动态加载模式，当前仅支持"Single"                                                                                                                              | 否    | str、Placeholder  |
| prebuild          | 模型是否提前构建，默认为False                                                                                                                                 | 否    | bool、Placeholder |

| 属性           | 描述                                                 | 是否必填 | 数据类型      |
|--------------|----------------------------------------------------|------|-----------|
| install_type | 模型的安装类型，支持"real_time", "edge", "batch"，该字段不填时默认均支持 | 否    | list[str] |

表 4-57 Template

| 属性              | 描述                         | 是否必填 | 数据类型                          |
|-----------------|----------------------------|------|-------------------------------|
| template_id     | 所使用的模板ID，模板中会内置一个输入输出模式    | 是    | str、Placeholder               |
| infer_format    | 输入输出模式ID，提供时覆盖模板中的内置输入输出模式 | 否    | str、Placeholder               |
| template_inputs | 模板输入项配置，即配置模型的源路径          | 是    | list of TemplateInputs object |

表 4-58 TemplateInputs

| 属性       | 描述                                                                                                     | 是否必填 | 数据类型                    |
|----------|--------------------------------------------------------------------------------------------------------|------|-------------------------|
| input_id | 输入项ID，从模板详情中获取                                                                                         | 是    | str、Placeholder         |
| input    | 模板输入路径，可以是OBS文件路径或OBS目录路径。使用多输入项的模板创建模型时，如果模板定义的目标路径input_properties是一样的，则此处输入的obs目录或者obs文件不能重名，否则会覆盖。 | 是    | str、Placeholder、Storage |

#### 4.4.6.3 使用案例

主要包含六种场景的用例：

- 基于JobStep的输出注册模型
- 基于OBS数据注册模型
- 使用模板方式注册模型
- 使用自定义镜像注册模型
- 使用自定义镜像+OBS的方式注册模型
- 使用订阅模型+OBS的方式注册模型

## 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

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，也可直接填写版本号
 item_version_id="item_version_id", # 算法订阅版本ID，也可直接填写版本号
 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；部分算法超参的值如果无需修改，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=mo
 del_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

workflow = wf.Workflow(
 name="model-step-demo",
 desc="this is a demo workflow",
 steps=[job_step, model_registration],
 storages=[storage]
)
```

## 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
)

workflow = wf.Workflow(
 name="model-step-demo",
 desc="this is a demo workflow",
 steps=[model_registration]
)
```

## 使用模板的方式注册模型

```
import modelarts.workflow as wf
通过ModelStep来定义一个模型注册节点，并通过预置模板进行注册

定义预置模板对象，Template对象中的字段可使用Placeholder表示
template = wf.steps.Template(
 template_id="fake_template_id",
 infer_format="fake_infer_format",
 template_inputs=[
 wf.steps.TemplateInputs(
 input_id="fake_input_id",
 input="fake_input_file"
)
]
)

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

workflow = wf.Workflow(
 name="model-step-demo",
 desc="this is a demo workflow",
 steps=[model_registration]
)
```

## 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
)

workflow = wf.Workflow(
 name="model-step-demo",
 desc="this is a demo workflow",
 steps=[model_registration]
)
```

## 使用自定义镜像+OBS 的方式注册模型

```
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OBS模型数据
model_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="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
 wf.steps.ModelInput(name="input",data=model_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="Custom",
 dynamic_load_mode="Single"
)
)# ModelStep的输出
)

workflow = wf.Workflow(
 name="model-step-demo",
 desc="this is a demo workflow",
 steps=[model_registration]
)
```

## 使用订阅模型+OBS 的方式注册模型

该方式本质上与自定义镜像+OBS的方式没有区别，只是自定义镜像变成从订阅模型中获取。

具体使用案例：

```
import modelarts.workflow as wf

定义订阅的模型对象
base_model = wf.data.GalleryModel(subscription_id="fake_subscription_id", version_num="fake_version") #
从gallery订阅的模型，一般由开发者自行创建发布

定义OBS模型数据
model_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="input",data=base_model) # ModelStep的输入使用订阅的模型
 wf.steps.ModelInput(name="input",data=model_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="Custom",
 dynamic_load_mode="Single"
)
)# ModelStep的输出
)

workflow = wf.Workflow(
 name="model-step-demo",
 desc="this is a demo workflow",
 steps=[model_registration]
)
```

上述案例中，系统会自动获取订阅模型中的自定义镜像，然后结合输入的OBS模型路径，注册生成一个新的模型，其中model\_obs可以替换成JobStep的动态输出。

### 📖 说明

model\_type支持的类型有："TensorFlow"、"MXNet"、"Caffe"、"Spark\_MLlib"、"Scikit\_Learn"、"XGBoost"、"Image"、"PyTorch"、"Template"、"Custom"。

在wf.steps.ModelConfig对象中，如果model\_type字段未填写，则表示默认使用"TensorFlow"。

- 如果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。
- 如果您构建的工作流需要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", "Template", "Custom"], description="模型类型")
```

## 4.4.7 服务部署节点

### 4.4.7.1 功能介绍

通过对ModelArts服务管理能力的封装，实现Workflow新增服务和更新服务的能力。主要应用场景如下：

- 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。

### 4.4.7.2 属性总览

您可以使用**ServiceStep**来构建服务部署节点，**ServiceStep**结构如下

表 4-59 ServiceStep

| 属性           | 描述                                                                                       | 是否必填 | 数据类型                            |
|--------------|------------------------------------------------------------------------------------------|------|---------------------------------|
| name         | 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 | 是    | str                             |
| inputs       | 服务部署节点的输入列表                                                                              | 否    | ServiceInput或者ServiceInput的列表   |
| outputs      | 服务部署节点的输出列表                                                                              | 是    | ServiceOutput或者ServiceOutput的列表 |
| title        | title信息，主要用于前端的名称展示                                                                      | 否    | str                             |
| description  | 服务部署节点的描述信息                                                                              | 否    | str                             |
| policy       | 节点执行的policy                                                                              | 否    | StepPolicy                      |
| depend_steps | 依赖的节点列表                                                                                  | 否    | Step或者Step的列表                   |

表 4-60 ServiceInput

| 属性   | 描述                                                                                 | 是否必填 | 数据类型 |
|------|------------------------------------------------------------------------------------|------|------|
| name | 服务部署节点的输入名称，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)。同一个Step的输入名称不能重复 | 是    | str  |

| 属性   | 描述            | 是否必填 | 数据类型                                                                             |
|------|---------------|------|----------------------------------------------------------------------------------|
| data | 服务部署节点的输入数据对象 | 是    | 模型列表或服务相关对象，当前仅支持 ServiceInputPlaceholder, ServiceData, ServiceUpdatePlaceholder |

表 4-61 ServiceOutput

| 属性             | 描述                                                                                 | 是否必填 | 数据类型          |
|----------------|------------------------------------------------------------------------------------|------|---------------|
| name           | 服务部署节点的输出名称，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)。同一个Step的输出名称不能重复 | 是    | str           |
| service_config | 服务部署相关配置信息                                                                         | 是    | ServiceConfig |

表4 ServiceConfig

| 属性           | 描述                                                                                                                                                                                                                                                            | 是否必填 | 数据类型            |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|-----------------|
| infer_type   | 推理方式：取值可为real-time/batch/edge。默认为real-time。 <ul style="list-style-type: none"><li>real-time代表在线服务，将模型部署为一个Web Service。</li><li>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。</li><li>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。</li></ul> | 是    | str             |
| service_name | 服务名称，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。<br><b>说明</b><br>该字段不填时默认为自动生成的服务名称。                                                                                                                                                                             | 否    | str、Placeholder |



| 属性                    | 描述                                                                                                                                                    | 是否必填 | 数据类型                 |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|------|----------------------|
| description           | 服务备注，默认为空，不超过100个字符。                                                                                                                                  | 否    | str                  |
| 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                  |
| 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。支持填入多个app name。                                                                                                                           | 否    | str、Placeholder、list |
| envs                  | 环境变量                                                                                                                                                  | 否    | dict                 |

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。

### 4.4.7.3 使用案例

主要包含三种场景的用例：

- 新增在线服务
- 更新在线服务
- 服务部署输出推理地址

#### 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
)

workflow = wf.Workflow(
 name="service-step-demo",
 desc="this is a demo workflow",
 steps=[service_step]
)
```

#### 更新在线服务

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
)

workflow = wf.Workflow(
 name="service-step-demo",
```

```
desc="this is a demo workflow",
steps=[service_step]
)
```

## 服务部署输出推理地址

服务部署节点支持输出推理地址，通过`get_output_variable("access_address")`方法获取输出值，并在后续节点中使用。

- 针对部署在公共资源池的服务，可以通过`access_address`属性从输出中获取注册在公网的推理地址。
- 针对部署在专属资源池的服务，除了可以获取注册在公网的推理地址，还能通过`cluster_inner_access_address`属性从输出中获取内部使用的推理地址，并且该地址只能在其他推理服务中进行访问。

```
import modelarts.workflow as wf

定义模型名称参数
sub_model_name = wf.Placeholder(name="si_placeholder1",
placeholder_type=wf.PlaceholderType.STR)

sub_service_step = wf.steps.ServiceStep(
 name="sub_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="子服务", # 标题信息
 inputs=wf.steps.ServiceInput(
 name="si_service_ph",
 data=wf.data.ServiceInputPlaceholder(name="si_placeholder1", model_name=sub_model_name)
),# ServiceStep的输入列表
 outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep的输出
)

main_model_name = wf.Placeholder(name="si_placeholder2",
placeholder_type=wf.PlaceholderType.STR)

获取子服务输出的推理地址，并通过envs传递给主服务
main_service_config = wf.steps.ServiceConfig(
 infer_type="real-time",
 envs={"infer_address":
sub_service_step.outputs["service_output"].get_output_variable("access_address")} # 获取子服务输出的
推理地址，并通过envs传递到主服务中
)

main_service_step = wf.steps.ServiceStep(
 name="main_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="主服务", # 标题信息
 inputs=wf.steps.ServiceInput(
 name="si_service_ph",
 data=wf.data.ServiceInputPlaceholder(name="si_placeholder2",
model_name=main_model_name)
),# ServiceStep的输入列表
 outputs=wf.steps.ServiceOutput(name="service_output", service_config=main_service_config), #
ServiceStep的输出
 depend_steps=sub_service_step
)

workflow = wf.Workflow(
 name="service-step-demo",
 desc="this is a demo workflow",
 steps=[sub_service_step, main_service_step]
)
```

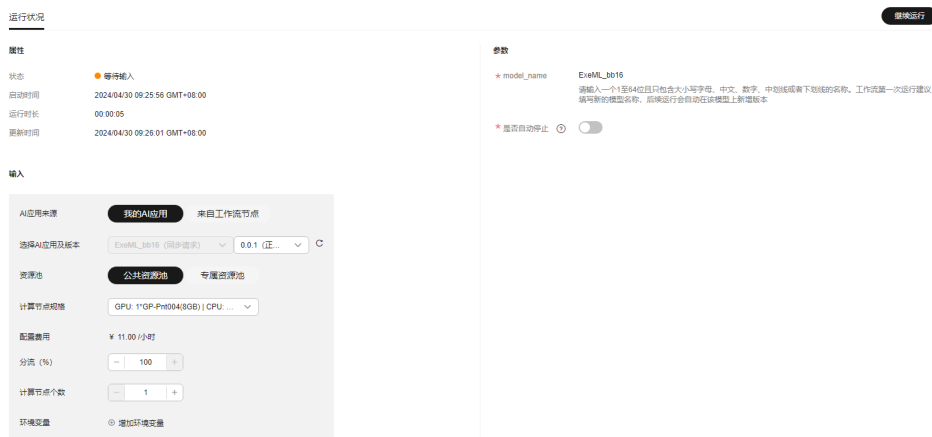
#### 4.4.7.4 相关配置操作

##### 同步推理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": "v1", "specification": "*****", "weight": 50}, {"model_name": "*****", "model_version": "v2", "specification": "*****", "weight": 30},
"envs": {"k1": "v1", "k2": "v2"}}, {"model_name": "*****", "model_version": "v1", "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.
```

1. 在ModelArts管理控制台，左侧菜单栏选择“Workflow”进入Workflow页面。
2. 在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。



##### 异步推理服务部署相关信息配置

1. 在ModelArts管理控制台，左侧菜单栏选择“Workflow”进入Workflow页面。
2. 在服务部署节点启动之后会等待用户设置相关配置信息，选择AI应用及版本为异步推理模型，设置**服务启动参数**，配置完成后直接单击**继续运行**即可。

##### 📖 说明

其中**服务启动参数**与您选择的异步推理AI应用相关，选择了需要的AI应用及版本后，系统会自动匹配响应的**服务启动参数**。

#### 4.4.8 条件节点

##### 4.4.8.1 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。主要应用场景如下：

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。

##### 4.4.8.2 属性总览

您可以使用**ConditionStep**来构建条件节点，**ConditionStep**结构如下：

表 4-62 ConditionStep

| 属性              | 描述                                                                                     | 是否必填 | 数据类型                    |
|-----------------|----------------------------------------------------------------------------------------|------|-------------------------|
| name            | 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 | 是    | str                     |
| conditions      | 条件列表，列表中的多个Condition执行'逻辑与'操作                                                          | 是    | Condition或者Condition的列表 |
| if_then_steps   | 条件表达式计算结果为True时，执行的step列表                                                              | 否    | str或者str列表              |
| else_then_steps | 条件表达式计算结果为False时，执行的step列表                                                             | 否    | str或者str列表              |
| title           | title信息，主要用于前端节点的名称展示                                                                  | 否    | str                     |
| description     | 条件节点的描述信息                                                                              | 否    | str                     |
| depend_steps    | 依赖的节点列表                                                                                | 否    | Step或者Step的列表           |

表 4-63 Condition

| 属性             | 描述                                                    | 是否必填 | 数据类型                                                                |
|----------------|-------------------------------------------------------|------|---------------------------------------------------------------------|
| condition_type | 条件类型，支持"==", ">", ">=", "in", "<", "<=", "!="，"or"操作符 | 是    | ConditionTypeEnum                                                   |
| left           | 条件表达式的左值                                              | 是    | int, float, str, bool, Placeholder, Sequence, Condition, MetricInfo |
| right          | 条件表达式的右值                                              | 是    | int, float, str, bool, Placeholder, Sequence, Condition, MetricInfo |

表 4-64 MetricInfo

| 属性         | 描述                              | 是否必填 | 数据类型       |
|------------|---------------------------------|------|------------|
| input_data | metric文件的存储对象，当前仅支持JobStep节点的输出 | 是    | JobStep的输出 |
| json_key   | 需要获取的metric信息对应的key值            | 是    | str        |

结构内容详解：

- **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。

- **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。

### 4.4.8.3 使用案例

#### 简单示例

- 通过参数配置实现

```
import modelarts.workflow as wf
```

```
left_value = wf.Placeholder(name="left_value", placeholder_type=wf.PlaceholderType.BOOL,
 default=True)

条件对象
condition = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=left_value,
 right=True) # 条件对象, 包含类型以及左右值

条件节点
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="job_step_1", # 当condition结果为true时, 名称为job_step_1的节点允许执行, 名称为
 job_step_2的节点跳过不执行
 else_then_steps="job_step_2" # 当condition结果为false时, 名称为job_step_2的节点允许执行, 名称为
 job_step_1的节点跳过不执行
)

该节点仅作为示例使用, 其他字段需自行补充
job_step_1 = wf.steps.JobStep(
 name="job_step_1",
 depend_steps=condition_step
)

该节点仅作为示例使用, 其他字段需自行补充
model_step_1 = wf.steps.ModelStep(
 name="model_step_1",
 depend_steps=job_step_1
)

该节点仅作为示例使用, 其他字段需自行补充
job_step_2 = wf.steps.JobStep(
 name="job_step_2",
 depend_steps=condition_step
)

该节点仅作为示例使用, 其他字段需自行补充
model_step_2 = wf.steps.ModelStep(
 name="model_step_2",
 depend_steps=job_step_2
)

workflow = wf.Workflow(
 name="condition-demo",
 desc="this is a demo workflow",
 steps=[condition_step, job_step_1, job_step_2, model_step_1, model_step_2]
)
```

### 📖 说明

场景说明: job\_step\_1和job\_step\_2表示两个训练节点, 并且均直接依赖于condition\_step。condition\_step通过参数配置决定后继节点的执行行为。

执行情况分析:

- 参数left\_value默认值为True, 则condition逻辑表达式计算结果为True: job\_step\_1执行, job\_step\_2跳过, 并且以job\_step\_2为唯一根节点的分支所包含的所有节点也将跳过, 即model\_step\_2会跳过, 因此最终执行的节点有condition\_step、job\_step\_1、model\_step\_1。
- 如果设置left\_value的值为False, 则condition逻辑表达式计算结果为False: job\_step\_2执行, job\_step\_1跳过, 并且以job\_step\_1为唯一根节点的分支所包含的所有节点也将跳过, 即model\_step\_1会跳过, 因此最终执行的节点有condition\_step、job\_step\_2、model\_step\_2。

- 通过获取JobStep输出的相关metric指标信息实现

```
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.Storage(name="storage_name", title="title_info", with_execution_id=True,
create_dir=True, description="description_info") # name字段必填, title, description可选填

定义输入的OBS对象
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

通过JobStep来定义一个训练节点, 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, 也可直接填写版本号
 parameters=[]
), # 训练使用的算法对象, 示例中使用AIGallery订阅的算法; 部分算法超参的值如果无需修改, 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=obs_data),
 outputs=[

wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
 wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # 指定metric的输出路径, 相关指标信息由作业脚本代码根据指定的数据格式自行输出(示例中需要将metric信息输出到训练输出目录下的metrics.json文件中)
],
 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
)

定义条件对象
condition_lt = wf.steps.Condition(
 condition_type=wf.steps.ConditionTypeEnum.LT,
 left=wf.steps.MetricInfo(job_step.outputs["metrics"].as_input(), "accuracy"),
 right=0.5
)

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_lt, # 条件对象, 允许多个条件, 条件之间的关系为&&
 if_then_steps="training_job_retrain", # 当condition结果为true时, 名称为training_job_retrain的节点允许执行, 名称为model_registration的节点跳过不执行
 else_then_steps="model_registration", # 当condition结果为false时, 名称为model_registration的节点允许执行, 名称为training_job_retrain的节点跳过不执行
 depend_steps=job_step
)

通过JobStep来定义一个训练节点, 并将训练结果输出到OBS
job_step_retrain = wf.steps.JobStep(
 name="training_job_retrain", # 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="图像分类重新训练训练", # 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, 也可直接填写版本号
 parameters=[]
), # 训练使用的算法对象, 示例中使用AIGallery订阅的算法; 部分算法超参的值如果无需修改, 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=obs_data),
 outputs=[
```



```
wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_retrain")),
wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path_retrain/metrics.json",
create_dir=False))) # 指定metric的输出路径, 相关指标信息由作业脚本代码根据指定的数据格式自行输出
(示例中需要将metric信息输出到训练输出目录下的metrics.json文件中)
],
spec=wf.steps.JobSpec(
resource=wf.steps.JobResource(
flavor=wf.Placeholder(name="train_flavor_retrain",
placeholder_type=wf.PlaceholderType.JSON, description="训练资源规格")
), # 训练资源规格信息
depend_steps=condition_step
)

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
inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()), #
job_step的输出作为输入
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=condition_step,
)

workflow = wf.Workflow(
name="condition-demo",
desc="this is a demo workflow",
steps=[job_step, condition_step, job_step_retrain, model_step],
storages=storage
)
```

案例中ConditionStep节点通过获取job\_step输出的accuracy指标信息与预置的值进行比较, 决定重新训练还是模型注册。当job\_step输出的accuracy指标数据小于阈值0.5时, condition\_lt的计算结果为True, 此时job\_step\_retrain运行, model\_step跳过; 反之job\_step\_retrain跳过, model\_step执行。

### 📖 说明

job\_step输出的metric文件格式要求可参考[作业类型节点](#)部分, 并且在Condition中只支持使用type为float类型的指标数据作为输入。

此案例中metrics.json的内容示例如下:

```
[
 {
 "key": "loss",
 "title": "loss",
 "type": "float",
 "data": {
 "value": 1.2
 }
 },
 {
 "key": "accuracy",
 "title": "accuracy",
 "type": "float",
 "data": {
 "value": 0.8
 }
 }
]
```

## 进阶示例

```
import modelarts.workflow as wf

left_value = wf.Placeholder(name="left_value", placeholder_type=wf.PlaceholderType.BOOL, default=True)
condition1 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=left_value, right=True)

internal_condition_1 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.GT, left=10, right=9)
internal_condition_2 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.LT, left=10, right=9)

condition2的结果为internal_condition_1 || internal_condition_2
condition2 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.OR, left=internal_condition_1,
right=internal_condition_2)

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, condition2], # 条件对象,允许多个条件, 条件之间的关系为&&
 if_then_steps=["job_step_1"], # 当condition结果为true时, 名称为job_step_1的节点允许执行, 名称为
job_step_2的节点跳过不执行
 else_then_steps=["job_step_2"] # 当condition结果为false时, 名称为job_step_2的节点允许执行, 名称为
job_step_1的节点跳过不执行
)

该节点仅作为示例使用, 其他字段需自行补充
job_step_1 = wf.steps.JobStep(
 name="job_step_1",
 depend_steps=condition_step
)

该节点仅作为示例使用, 其他字段需自行补充
job_step_2 = wf.steps.JobStep(
 name="job_step_2",
 depend_steps=condition_step
)

workflow = wf.Workflow(
 name="condition-demo",
 desc="this is a demo workflow",
 steps=[condition_step, job_step_1, job_step_2],
)
```

ConditionStep支持多条件节点的嵌套使用, 用户可以基于不同的场景灵活设计。

### 📖 说明

条件节点只支持双分支的选择执行, 局限性较大, 推荐您使用新的分支功能, 可以在不添加新节点的情况下完全覆盖ConditionStep的能力, 详情请参见[分支控制](#)章节。

## 4.5 分支控制

### 功能介绍

支持单节点通过参数配置或者获取训练输出的metric指标信息来决定执行是否跳过, 同时可以基于此能力完成对执行流程的控制。

### 应用场景

主要用于存在多分支选择执行的复杂场景, 在每次启动执行后需要根据相关配置信息决定哪些分支需要执行, 哪些分支需要跳过, 达到分支部分执行的目的, 与ConditionStep的使用场景类似, 但功能更加强大。当前该能力适用于数据集创建节

点、数据集标注节点、数据集导入节点、数据集版本发布节点、作业类型节点、模型注册节点以及服务部署节点。

## 使用案例

- 控制单节点的执行

- 通过参数配置实现

```
from modelarts import workflow as wf

condition_equal = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)

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OBS对象
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

通过JobStep来定义一个训练节点，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，也可直接填写版本号
 parameters=[]
), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，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=obs_data),
 # JobStep的输入在运行时配置；data字段也可使用
 data=wf.data.OBSPath(obs_path="fake_obs_path")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
 policy=wf.steps.StepPolicy(
 skip_conditions=[condition_equal] # 通过skip_conditions中的计算结果决定job_step是否跳过
)
)

workflow = wf.Workflow(
 name="new-condition-demo",
 desc="this is a demo workflow",
 steps=[job_step],
 storages=storage
)
```

案例中job\_step配置了相关的跳过策略，并且通过一个bool类型的参数进行控制。当name为is\_skip的Placeholder参数配置为True时，condition\_equal的计算结果为True，此时job\_step会被置为跳过，反之job\_step正常执行，其中Condition对象详情可参考[条件节点](#)。

- 通过获取JobStep输出的相关metric指标信息实现

```
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.Storage(name="storage_name", title="title_info", with_execution_id=True,
create_dir=True, description="description_info") # name字段必填，title, description可选填

定义输入的OBS对象
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

通过JobStep来定义一个训练节点，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，也可直接填写版本号
 parameters=[]

), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，
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=obs_data),
 outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.joi
n("directory_path"))),
 wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # 指定metric的输出路径，相关指标信息由作业脚本代码根据指定的数据格式自行
输出（示例中需要将metric信息输出到训练输出目录下的metrics.json文件中）
],
 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
)

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

定义条件对象
condition_lt = wf.steps.Condition(
 condition_type=wf.steps.ConditionTypeEnum.LT,
 left=wf.steps.MetricInfo(job_step.outputs["metrics"].as_input(), "accuracy"),
 right=0.5
)

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
 inputs=wf.steps.ModelInput(name='model_input',
data=job_step.outputs["train_url"].as_input()), # job_step的输出作为输入
 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
 policy=wf.steps.StepPolicy(skip_conditions=condition_lt) # 通过skip_conditions中的计算结果决
定model_step是否跳过
)

workflow = wf.Workflow(
 name="new-condition-demo",
 desc="this is a demo workflow",
 steps=[job_step, model_step],
```

```
storages=storage
)
```

案例中model\_step配置了相关的跳过策略，并且通过获取job\_step输出的accuracy指标信息与预置的值进行比较，决定是否需要模型注册。当job\_step输出的accuracy指标数据小于阈值0.5时，condition\_lt的计算结果为True，此时model\_step会被置为跳过，反之model\_step正常执行。

### 说明

job\_step输出的metric文件格式要求可参考[分支控制](#)部分，并且在Condition中只支持使用type为float类型的指标数据作为输入。

此案例中metrics.json的内容示例如下：

```
[
 {
 "key": "loss",
 "title": "loss",
 "type": "float",
 "data": {
 "value": 1.2
 }
 },
 {
 "key": "accuracy",
 "title": "accuracy",
 "type": "float",
 "data": {
 "value": 0.8
 }
 }
]
```

- **控制多分支的部分执行**

```
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.Storage(name="storage_name", title="title_info", with_execution_id=True,
create_dir=True, description="description_info") # name字段必填，title, description可选填
```

```
定义输入的OBS对象
```

```
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")
```

```
condition_equal_a = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="job_step_a_is_skip", placeholder_type=wf.PlaceholderType.BOOL),
right=True)
```

```
通过JobStep来定义一个训练节点，并将训练结果输出到OBS
```

```
job_step_a = wf.steps.JobStep(
 name="training_job_a", # 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，也可直接填写版本号
 parameters=[]
```

```
), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，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=obs_data),
```

```
outputs=[wf.steps.JobOutput(name="train_url",
```

```
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_a"))],
```

```
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
```

```

 policy=wf.steps.StepPolicy(skip_conditions=condition_equal_a)
)

condition_equal_b = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="job_step_b_is_skip", placeholder_type=wf.PlaceholderType.BOOL),
right=True)

通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step_b = wf.steps.JobStep(
 name="training_job_b", # 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，也可直接填写版本号
 parameters=[]

), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，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=obs_data),
 outputs=[wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_b")))],
 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
 policy=wf.steps.StepPolicy(skip_conditions=condition_equal_b)
)

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
 inputs=wf.steps.ModelInput(name='model_input',
data=wf.data.DataConsumptionSelector(data_list=[job_step_a.outputs["train_url"].as_input(),
job_step_b.outputs["train_url"].as_input()])), # 选择job_step_a或者job_step_b的输出作为输入
 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_a, job_step_b], # 依赖的作业类型节点对象
)

workflow = wf.Workflow(
 name="new-condition-demo",
 desc="this is a demo workflow",
 steps=[job_step_a, job_step_b, model_step],
 storages=storage
)

```

案例中job\_step\_a和job\_step\_b均配置了跳过策略，并且都使用参数进行控制。当参数值配置不同时，model\_step的执行可以分为以下几种情况（model\_step没有配置跳过策略，因此会遵循默认规则）：

| job_step_a_is_skip参数值 | job_step_b_is_skip参数值 | model_step是否执行 |
|-----------------------|-----------------------|----------------|
| True                  | True                  | 跳过             |
|                       | False                 | 执行             |
| False                 | True                  | 执行             |

| job_step_a_is_skip参数值 | job_step_b_is_skip参数值 | model_step是否执行 |
|-----------------------|-----------------------|----------------|
|                       | False                 | 执行             |

**⚠ 注意**

默认规则：当某个节点依赖的所有节点状态均为跳过时，该节点自动跳过，否则正常执行，此判断逻辑可扩展至任意节点。

在上述案例的基础上，如果需要打破默认规则，在job\_step\_a以及job\_step\_b跳过时，model\_step也允许执行，则只需要在model\_step中也配置跳过策略即可（跳过策略的优先级高于默认规则）。

## 4.6 多输入支持数据选择

### 功能介绍

仅用于存在多分支执行的场景，在编写构建工作流节点时，节点的数据输入来源暂不确定，可能是多个依赖节点中任意一个节点的输出。只有当依赖节点全部执行完成后，才会根据实际执行情况自动获取有效输出作为输入。

### 使用案例

```

from modelarts import workflow as wf

condition_equal = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="is_true", placeholder_type=wf.PlaceholderType.BOOL), right=True)
condition_step = wf.steps.ConditionStep(
 name="condition_step",
 conditions=[condition_equal],
 if_then_steps=["training_job_1"],
 else_then_steps=["training_job_2"],
)

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OBS对象
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step_1 = wf.steps.JobStep(
 name="training_job_1", # 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，也可直接填写版本号
 parameters=[]
), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，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=obs_data),
 # JobStep的输入在运行时配置；data字段也可使用data=wf.data.OBSPath(obs_path="fake_obs_path")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
depend_steps=[condition_step]
)

通过JobStep来定义一个训练节点，并将训练结果输出到OBS
job_step_2 = wf.steps.JobStep(
 name="training_job_2", # 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，也可直接填写版本号
 parameters=[]

), # 训练使用的算法对象，示例中使用AIGallery订阅的算法；部分算法超参的值如果无需修改，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=obs_data),
 # JobStep的输入在运行时配置；data字段也可使用data=wf.data.OBSPath(obs_path="fake_obs_path")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
 depend_steps=[condition_step]
)

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
 inputs=wf.steps.ModelInput(name='model_input',
data=wf.data.DataConsumptionSelector(data_list=[job_step_1.outputs["train_url"].as_input(),
job_step_2.outputs["train_url"].as_input()])), # 选择job_step_1或者job_step_2的输出作为输入
 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_1, job_step_2] # 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

workflow = wf.Workflow(name="data-select-demo",
 desc="this is a test workflow",
 steps=[condition_step, job_step_1, job_step_2, model_step],
 storages=storage
)
```



## 📖 说明

案例中的Workflow存在两个并行分支，并且同时只有一条分支会执行，由condition\_step的相关配置决定。model\_step的输入来源为job\_step\_1或者job\_step\_2的输出，当job\_step\_1节点所在分支执行，job\_step\_2节点所在分支跳过时，model\_step节点执行时自动获取job\_step\_1的输出作为输入，反之自动获取job\_step\_2的输出作为输入。

## 4.7 编写 Workflow

Workflow的编写主要在于每个节点的定义，您可以参考[节点类型](#)章节，按照自己的场景需求选择相应的代码示例模板进行修改。编写过程主要分为以下几个步骤。

1. 梳理场景，了解预置Step的功能，确定最终的DAG结构。
2. 单节点功能，如训练、推理等在ModelArts相应服务中调试通过。
3. 根据节点功能选择相应的代码模板，进行内容的补充。
4. 根据DAG结构编排节点，完成Workflow的编写。

## 4.8 调试 Workflow

Workflow编写完成后，可以在开发态进行调试，当前支持run以及debug两种调试模式。假设某工作流对象Workflow有label\_step、release\_step、job\_step、model\_step、service\_step五个节点，调试过程如下：

- **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（部分运行时需要自行保证数据依赖的正确性）

```
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：

- 该节点运行需要的数据已经准备好，则直接启动运行。
- 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 发布 Workflow

### 4.9.1 发布运行态

工作流调试完成后，可以进行固化保存，调用Workflow对象的release()方法发布到运行态进行配置执行（在管理控制台Workflow页面配置）。

执行如下命令：

```
workflow.release()
```

上述命令执行完成后，如果日志打印显示发布成功，则可前往ModelArts的Workflow页面中查看新发布的工作流，工作流相关的配置执行操作可参考[如何使用Workflow](#)。

基于release()方法，提供了release\_and\_run()方法，支持用户在开发态发布并运行工作流，节省了前往console配置执行的操作。

#### ⚠️ 注意

使用该方法时需要注意以下几个事项：

- Workflow中所有出现占位符相关的配置对象时，均需要设置默认值，或者直接使用固定的数据对象
- 方法的执行依赖于Workflow对象的名称：当该名称的工作流不存在时，则创建新工作流并创建新执行；当该名称的工作流已存在时，则更新存在的工作流并基于新的工作流结构创建新的执行

```
workflow.release_and_run()
```

### 4.9.2 发布到 AI Gallery

Workflow支持发布到gallery，分享给其他用户使用，执行如下代码即可完成发布。

```
workflow.release_to_gallery()
```

发布完成后可前往gallery查看相应的资产信息，资产权限默认为private，可在资产的console页面自行修改。

其中release\_to\_gallery()方法包含以下入参：

| 参数名称       | 描述                      | 是否必填 | 参数类型 |
|------------|-------------------------|------|------|
| content_id | Workflow资产ID            | 否    | str  |
| version    | Workflow资产的版本号，格式为x.x.x | 否    | str  |

| 参数名称        | 描述                                                                            | 是否必填 | 参数类型      |
|-------------|-------------------------------------------------------------------------------|------|-----------|
| desc        | Workflow资产版本的描述信息                                                             | 否    | str       |
| title       | Workflow资产名称, 该参数未填写时默认使用Workflow的名称作为资产名称                                    | 否    | str       |
| visibility  | Workflow资产可见性, 支持"public"-公开、"group"-白名单、"private"-私有, 仅自己可见三种, 默认为"private"。 | 否    | str       |
| group_users | 白名单列表, 仅支持填写domain_id, 当visibility为"group"时才需要填写该字段                           | 否    | list[str] |

根据方法的入参不同, 主要可分为以下两种使用场景:

- Workflow.release\_to\_gallery(title="资产名称")发布Workflow新资产, 版本号为"1.0.0"; 如果Workflow包含非gallery的算法, 则自动将依赖算法发布至gallery, 版本号为"1.0.0"。
- Workflow.release\_to\_gallery(content\_id="\*\*\*", title="资产名称")基于指定的Workflow资产, 发布新的版本, 版本号自动增加; 如果Workflow包含gallery的算法, 则自动将依赖的算法资产发布新版本, 版本号也自动增加。

#### Workflow资产白名单设置:

在资产第一次发布时, 可以通过release\_to\_gallery方法的visibility+group\_users字段进行设置, 后续需要对指定资产进行用户白名单添加或删除操作时, 可执行如下命令:

```
from modelarts import workflow as wf

添加指定的白名单用户列表
wf.add_whitelist_users(content_id="***", version_num="*.*.*", user_groups=["***", "***"])

删除指定的白名单用户列表
wf.delete_whitelist_users(content_id="***", version_num="*.*.*", user_groups=["***", "***"])
```

#### 说明

在给Workflow资产添加或删除指定白名单用户列表时, 会自动查询该版本依赖的算法资产信息, 同步对算法资产进行相应的白名单设置。

## 4.10 高阶能力

## 4.10.1 部分运行

Workflow通过支持预置场景的方式来实现部分运行的能力，在开发工作流时按照场景的不同对DAG进行划分，之后在运行态可选择任意场景单独运行。具体代码示例如下所示：

```
workflow =wf.Workflow(
 name="image_cls",
 desc="this is a demo workflow",
 steps=[label_step, release_data_step, training_step, model_step, service_step],
 policy=wf.policy.Policy(
 scenes=[
 wf.policy.Scene(
 scene_name="模型训练",
 scene_steps=[label_step, release_data_step, training_step]
),
 wf.policy.Scene(
 scene_name="服务部署",
 scene_steps=[model_step, service_step]
),
]
)
)
```

该示例中Workflow包含了五个节点（节点相关定义已省略），在policy中定义了两个预置场景：模型训练和服务部署，工作流发布至运行态后，部分运行的开关默认关闭，节点全部运行。用户可在全局配置页面打开开关，选择指定的场景进行运行。

### 说明

部分运行能力支持同一个节点被定义在不同的运行场景中，但是需要用户自行保证节点之间数据依赖的正确性。另外，部分运行能力仅支持在运行态进行配置运行，不支持在开发态进行调试。

## 4.10.2 在 Workflow 中使用大数据能力（DLI/MRS）

### 功能介绍

该节点通过调用MRS服务，提供大数据集群计算能力。主要用于数据批量处理、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服务下查看自己账号下可用的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=wf.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=wf.data.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得--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=wf.data.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")
)
```

- **在控制台上如何使用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。

## 4.11 常见问题

### 4.11.1 开发态调试时，如何查询训练规格？

Workflow在开发态进行调试运行时，需要手动配置训练规格，详情请见[资源规格查询](#)章节

### 4.11.2 如何实现多分支？

当前支持两种方式实现多分支的能力，详情请见[条件节点](#)和[分支控制](#)章节，推荐使用分支控制的方式，使用上更灵活。

### 4.11.3 如何导入对象？

在编写Workflow过程中，相关对象都通过Workflow包进行导入，梳理如下：

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

Data包相关内容导入：

```
wf.data.DatasetTypeEnum
wf.data.Dataset
wf.data.DatasetVersionConfig
wf.data.DatasetPlaceholder
wf.data.ServiceInputPlaceholder
wf.data.ServiceData
wf.data.ServiceUpdatePlaceholder
wf.data.DataTypeEnum
wf.data.ModelData
wf.data.GalleryModel
wf.data.OBSPath
wf.data.OBSOutputConfig
wf.data.OBSPlaceholder
wf.data.SWRImage
wf.data.SWRImagePlaceholder
wf.data.Storage
wf.data.InputStorage
wf.data.OutputStorage
wf.data.LabelTask
wf.data.LabelTaskPlaceholder
wf.data.LabelTaskConfig
wf.data.LabelTaskTypeEnum
wf.data.MetricsConfig
wf.data.TripartiteServiceConfig
wf.data.DataConsumptionSelector
```

policy包相关内容导入：

```
wf.policy.Policy
wf.policy.Scene
```

steps包相关内容导入：

```
wf.steps.MetricInfo
wf.steps.Condition
wf.steps.ConditionTypeEnum
wf.steps.ConditionStep
wf.steps.LabelingStep
wf.steps.LabelingInput
wf.steps.LabelingOutput
wf.steps.LabelTaskProperties
wf.steps.ImportDataInfo
wf.steps.DataOriginTypeEnum
wf.steps.DatasetImportStep
wf.steps.DatasetImportInput
wf.steps.DatasetImportOutput
wf.steps.AnnotationFormatConfig
wf.steps.AnnotationFormatParameters
```

```
wf.steps.AnnotationFormatEnum
wf.steps.Label
wf.steps.ImportTypeEnum
wf.steps.LabelFormat
wf.steps.LabelTypeEnum
wf.steps.ReleaseDatasetStep
wf.steps.ReleaseDatasetInput
wf.steps.ReleaseDatasetOutput
wf.steps.CreateDatasetStep
wf.steps.CreateDatasetInput
wf.steps.CreateDatasetOutput
wf.steps.DatasetProperties
wf.steps.SchemaField
wf.steps.ImportConfig
wf.steps.JobStep
wf.steps.JobMetadata
wf.steps.JobSpec
wf.steps.JobResource
wf.steps.JobTypeEnum
wf.steps.JobEngine
wf.steps.JobInput
wf.steps.JobOutput
wf.steps.LogExportPath
wf.steps.MrsJobStep
wf.steps.MrsJobInput
wf.steps.MrsJobOutput
wf.steps.MrsJobAlgorithm
wf.steps.ModelStep
wf.steps.ModelInput
wf.steps.ModelOutput
wf.steps.ModelConfig
wf.steps.Template
wf.steps.TemplateInputs
wf.steps.ServiceStep
wf.steps.ServiceInput
wf.steps.ServiceOutput
wf.steps.ServiceConfig
wf.steps.StepPolicy
```

Workflow包相关内容导入：

```
wf.workflow
wf.Subgraph
wf.Placeholder
wf.PlaceholderType
wf.AlgorithmParameters
wf.BaseAlgorithm
wf.Algorithm
wf.AIGalleryAlgorithm
wf.resource
wf.SystemEnv
wf.add_whitelist_users
wf.delete_whitelist_users
```

#### 4.11.4 如何定位运行报错？

使用run模式运行工作流报错时，分析解决思路如下：

1. 确认安装的SDK包是否是最新版本，避免出现包版本不一致问题。
2. 检查编写的SDK代码是否符合规范，具体可参考相应的代码示例。
3. 检查运行过程中输入的内容是否正确，格式是否与提示信息中要求的一致。
4. 根据具体报错信息定位到报错的代码行，分析上下文逻辑。

## 历史 SDK 包常见的报错如下

- 服务部署节点运行报错

```
'weight': 100, 'specification':'custom',
'cluster_id':'*****', 'custom_spec':{'cpu':2.5, 'memory':
1024}, '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.  
[{'model\_name':'', 'model\_version':'', 'specification':'', 'weight':100}]  
Traceback (most recent call last):

- 输入服务相关的参数后，执行报错如下：

```
return self.name == obj.name and \

AttributeError: 'str' object has no attribute 'name'
```

## 解决方案

以上两种常见报错均可通过升级最新的SDK包解决。