大数据智能体工作台 DataBuddy

数据科学

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

数据科学
数据科学概述

最近更新时间：2026-09-11 20:42:30

数据科学是 DataBuddy 面向数据科学家、机器学习工程师提供的端到端机器学习开发的模块，覆盖从实验追踪、特征工程、模型注册到模型服务部署的完整流程。该模块基于开源 MLflow 平台构建，将机器学习工件与 DataBuddy Catalog 统一治理体系打通，实现机器学习资产与数据资产的同源管理。

> **提示**
> 更多 MLflow 功能与用法请参见 MLflow 官方文档 。

模块定位

数据科学模块基于 **MLOps** 理念构建，为机器学习模型的全生命周期提供标准化、自动化、可持续改进的过程管理，帮助企业稳定、规模化、高质效地持续生产模型为业务赋能。其核心理念为：

- 以业务目标为牵引推进 AI 项目研发；
- 以数据为中心实现 AI 研发；
- 以模块化平台驱动全生命周期：数据探索、特征工程、模型训练、在线服务；
- 以自动化流程实现持续训练、持续集成与持续交付。

功能概述

DataBuddy 数据科学模块建设了实验管理、特征管理、模型管理和模型服务四个核心功能模块，承接开发工作台、工作流、数据质量、计算资源等周边产品，实现实验追踪、特征复用、模型版本管理 、在线推理服务的全生命周期的端到端能力，搭建 MLOps 闭环。
开发流程通常从在**开发工作台** 中加载并清洗数据开始，将带主键和时间戳的数据登记为特征表以便复用；随后创建实验，通过自定义训练或 AutoML 记录参数、指标与模型工件，并对比多次运行选出最佳模型；将所需模型注册到 Catalog 后统一进行管理，最终部署为在线推理服务对外提供能力。

核心模块

模块	核心功能	详细文档
模型实验	模型实验面板跟踪训练的参数、指标、工件；支持自定义模型实验、智能体实验及 AutoML 实验（分类 / 回归 / 时序预测）等无代码开发三种类型的实验	模型实验
特征管理	在 **开发工作台** 中对于每一个特征表可使用特征工程 SDK 完成创建、写入、读取、查找、同步、消费等操作，并在特征管理页面中进行查看、管理特征，实现特征统一管理、统一消费	特征管理

模型管理	可以在实验中通过调用 MLflow 的相关函数来注册模型，或者在实验管理中执行可视化的模型注册。在管理界面中支持查看已注册模型的关键信息，以及与实验/运行、服务等关联关系。	模型管理
模型服务	将注册模型部署为在线推理 API，支持版本切换、流量分发与资源监控等功能	模型服务

周边模块

模块	与数据科学的协作关系
开发工作台	提供开发环境，用户可在 **开发工作台** 中编辑、调试、运行代码，并且调用 MLflow 和特征工程 API，实现特征表增删改查、模型训练、模型注册等操作。
Catalog	模型、特征表的统一存储与治理底座，提供权限、标签、血缘
工作流	提供自动化流程的工作区，在 **开发工作台** 中调试好代码后，可提交至工作流设置周期性调度，实现模型的自动化、周期性生产。
数据质量	模型服务推理表、特征表、训练数据表都可以通过发起数据质量任务，查看相应的字段分析、漂移分析、模型指标等质量信息。

必要说明及前置条件

类型	说明
开发工作台	必须开通 **开发工作台** 才可使用数据科学相关功能。**开发工作台** 是 AI 开发的主要工作区，用户在其中编辑、调试、运行代码，并调用 MLflow 和特征工程 API，完成特征表增删改查、模型训练、模型注册等操作。
计算资源支持	DataBuddy **开发工作台** 的计算资源可用于模型训练、实验上报、特征管理、模型注册；AutoML 实验需在创建计算资源时勾选**机器学习** 镜像（非默认基础镜像）。
MLflow	实验管理兼容 MLflow >= 3.0.0（< 4.0.0），相关 SDK 在 Kernel 启动时由预执行脚本自动安装（非预装在镜像中）。连接 **开发工作台** 运行环境后可执行以下命令检查： %pip list | grep mlflow 官方文档：MLflow 文档

版本	
离线特征存储	离线特征表仅支持 Catalog 中的表；具有主键和时间戳的表会被自动识别为特征表。注册特征表时必须指定主键及时间戳键，后续以该主键和时间戳键进行特征索引。
在线特征存储	DataBuddy 当前仅支持 PostgreSQL 作为在线特征存储，准备步骤如下： • 前往云数据库创建 PostgreSQL 实例，地域、网络需与调度资源组、所使用的引擎保持一致。 • 在 DataBuddy 的 **平台管理 > 数据源管理 ** 添加 PostgreSQL 数据源，测试可连通调度资源组。 • 在 **开发工作台** 中连接远程内核后，验证 PostgreSQL 是否连通： import psycopg2PG_HOST = "myhost" # 示例："10.0.0.51"PG_PORT = "myport" # 示例："5432"PG_DB = "mydb"PG_USER = "myuser"PG_PWD = "mypwd" # 如果没有密码可注释掉此行及下方 password 参数try: conn = psycopg2.connect(host=PG_HOST, port=PG_PORT, dbname=PG_DB, user=PG_USER, password=PG_PWD, connect_timeout=3,) conn.close() print("PostgreSQL 可连接")except Exception as e: print("PostgreSQL 连接失败：", e)
表操作权限	需在 **计算资源 > 对应资源 > 更多 > 权限 ** 授权，并在 **DataBuddy 目录 > 对应 Catalog / Schema / 表 > 权限** 授权。
模型操作权限	需在 DataBuddy **目录 > 对应模型 Catalog > 权限** 对用户授权。
特征工程包	连接引擎后执行以下命令安装特征工程包（要求 0.2.3 及以上）： %pip install "wedata3-feature-engineering[mlflow3]>=0.2.3" 可执行以下命令检查版本： %pip list | grep wedata3-feature-engineering
环境变量	DataBuddy 中 MLflow / Feast 服务地址、临时凭证等由 Kernel 启动时的预执行脚本自动注入，用户无需手动设置。

设置	

相关文档

- 模型实验
- 特征管理
- 模型管理
- 模型服务
- 开发工作台概述
- 数据目录概述
- 计算资源概述

模型实验
什么是模型实验

最近更新时间：2026-09-11 20:42:31

模型实验是 DataBuddy 数据科学模块中组织与追踪机器学习训练过程的能力。您在 Studio 中训练模型时，可以通过模型实验模块系统化地记录每次运行的参数、指标与工件；训练完成后，在实验面板中对比多次运行的表现，选出最优模型并注册到 Catalog 进行统一管理，无需手动维护训练日志或逐个比对结果。

关键概念

运行（Run）

Run 是一次具体的训练或调参过程。每次在 Notebook 中调用 `mlflow.start_run()` 就会启动一次新的 Run。Run 内可以记录：

- **运行 ID 与运行名称**：唯一标识，运行名称在所属实验内唯一。
- **运行状态**：运行中、已完成、失败。
- **运行时长**：自动统计，单位会按 ms / s / min / h 自动切换。
- **创建人、创建时间、来源 Notebook**：自动从执行环境采集。
- **数据集**：本次训练使用的输入数据，可在详情中查看 Schema。

父子 Run

在父 Run 上下文中再次调用 `mlflow.start_run(nested=True)`，新的 Run 会作为父 Run 的子 Run，常用于超参数调优场景。示例：

```
import mlflow

with mlflow.start_run(run_name="Parent Run"):
    mlflow.log_param("model_type", "CNN")
    for i in range(3):
        with mlflow.start_run(run_name=f"Child Run {i}", nested=True):
            mlflow.log_param("learning_rate", 0.01 * (i + 1))
            mlflow.log_metric("accuracy", 0.8 + 0.05 * i)
```

实验详情的运行任务列表会以嵌套方式展示父 Run 与其下的子 Run。

参数（Parameter）

训练开始时记录的输入配置。值类型可以是字符串、整数或浮点数。常见示例：

- `learning_rate: 0.01`

- `batch_size: 64`
- `n_estimators: 100`
- `max_depth: 6`

通过 `mlflow.log_param(key, value)` 记录，记录后不可修改。

指标（Metric）

训练过程中或训练结束时记录的模型表现度量。指标值可以随训练步（step）变化，便于绘制曲线图。常见示例：

- `accuracy`、`AUC`、`F1`、`precision`、`recall`（分类）
- `RMSE`、`MAE`、`R2`（回归）
- `loss`、`val_loss`（深度学习）

通过 `mlflow.log_metric(key, value, step=None)` 记录，可多次调用以记录训练曲线。

工件（Artifact）

本次 Run 产生的所有附属文件，每个 Run 拥有独立的存储空间，目录树状展示。常见内容：

- **训练得到的模型文件（`.pkl` / `.pt` / `.onnx` 等）**
- **日志文件（`stdout`、`stderr`、`log.txt`）**
- **配置文件（`config.yaml`、`params.json`）**
- **评估结果（预测输出、评估指标、图片）**

通过 `mlflow.log_artifact(local_path, artifact_path=None)` 或 `mlflow.<flavor>.log_model(...)` 记录。

系统指标（System Metrics）

MLflow 自动采集的系统资源占用指标，用于评估训练对计算资源的消耗。默认记录以下指标：

指标	含义
`cpu_utilization_percentage`	CPU 使用率
`system_memory_usage_megabytes`	系统内存占用
`system_memory_usage_percentage`	系统内存占用率
`network_receive_megabytes`	网络接收流量
`network_transmit_megabytes`	网络发送流量
`disk_usage_megabytes`	磁盘已用容量
`disk_available_megabytes`	磁盘可用容量

数据集（Dataset）

本次 Run 使用的输入数据，在 Run 详情中可点击展开数据集详情侧边弹窗，查看数据来源与字段信息。一个 Run 可以关联多个数据集。

模型（Model）与已注册模型

在 Run 中通过 `mlflow.<flavor>.log_model(...)` 记录的模型对象会作为 Run 的工件保存。模型有以下两种状态：

- **未注册模型**：仅作为 Run 的工件存在，不进入 Catalog 治理。
- **已注册模型**：通过 **注册模型** 操作或在代码中传入 `registered_model_name=` 参数后，模型会注册到 Catalog 下的 MLflow Model Registry，拥有 `catalog.schema.model_name` 三段式全名与递增的版本号。

模型管理的详细规则请参考 模型管理 。

最佳运行（Best Run）

实验内根据评估指标自动识别的最优 Run。在实验详情列表中，最佳运行默认置顶展示。

实验、Run 与模型的关系

实验、Run 与模型在不同场景下存在不同的对应关系：

场景	关系
一个 Run 训练并保存一个模型（基础场景）	Run 与模型一一对应
一个 Run 在多个 epoch 保存多个模型快照（深度学习）	一个 Run 对应多个模型工件（如 `torch-iris-10` 、`torch-iris-50` 、`torch-iris-100` ）
同一模型架构用不同参数多次训练（调参）	同一注册模型对应多个 Run，分别为不同模型版本
父子 Run 各自保存模型（如调参试验）	一个父 Run 关联多个子 Run，每个子 Run 关联一个模型工件

权限模型

实验权限复用基础平台权限体系。每个实验默认创建者与 Admin 账号可管理，其他成员可在实验详情右上角 **权限** 弹窗中按以下三级授权：

权限级别	可执行操作

读取（Can Read）	查看实验基本信息、所有 Run 的参数 / 指标 / 模型 / 工件、运行历史；导出实验结果；查看权限设置（不可修改）
编辑（Can Edit）	拥有读取的所有能力；新建 Run、修改 Run 内的参数 / 指标 / 工件；编辑实验描述与标签
管理（Can Manage）	拥有编辑的所有能力；删除实验、为其他用户分配权限

> 无权限（No Permissions）状态下用户完全不可访问该实验。

权限管理的具体步骤请参考 **成员与权限管理** 。

相关文档

- **模型实验概述**
- **创建实验**
- **记录训练指标**
- **对比实验结果**
- **模型管理概述**
- **核心概念与术语表**
- **MLflow 官方文档**

创建实验

最近更新时间：2026-09-11 20:42:31

DataBuddy 数据科学模块支持通过 UI 或 SDK 创建机器学习实验。

前提条件

在创建实验前，请确保满足以下条件：

- 已开通 DataBuddy 服务并完成空间初始化。
- 当前账号在目标 Workspace 中拥有创建 Notebook 与实验的权限。
- 计划使用 AutoML 实验时：可用的训练数据已通过 Catalog 进行管理，且您拥有读取该表的权限。
- 计划使用深度学习的自定义实验时：已在 计算资源概述 中准备好带 GPU 的计算集群。

创建自定义模型训练实验

1.通过 UI 创建

1. 登录 DataBuddy 控制台，在左侧导航栏选择 **数据科学** > **模型实验** 。
2. 点击**自定义模型**，在弹出的创建窗口中填写：
   - **名称**：必填，自定义实验名称。
   - **描述**（可选）：实验用途说明。
1. 点击**仅创建实验** 。系统会创建实验，实时显示在实验列表中。用户可通过点击实验名称进入实验详情页，点击 **前往 Notebook 创建任务** 创建 Notebook 任务。点击**创建实验并进入 Notebook**，自动创建实验，跳转到 Notebook，创建 Notebook 后将示例代码复制进单元格按需修改后运行。示例如下：

```
import mlflow
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# 设置实验 ID
mlflow.set_experiment(experiment_id="3")

# 自动记录实验过程（如参数、模型、指标等）
mlflow.autolog()

# 加载数据集
db = load_diabetes()
X_train, X_test, y_train, y_test = train_test_split(db.data, db.target)
```

```
# 创建并训练模型
rf = RandomForestRegressor(n_estimators=100, max_depth=6,
max_features=3)
rf.fit(X_train, y_train)
predictions = rf.predict(X_test)
```

2.通过 Notebook 创建

在**开发工作台** 点击**创建**， 可直接创建 Notebook，在中直接调用 MLflow API：

```
import mlflow

# 创建新实验（如果同名实验已存在则报错）
experiment_id = mlflow.create_experiment(name="customer_churn_v1")

# 或：设置实验。如果同名实验不存在则自动创建
mlflow.set_experiment("customer_churn_v1")
```

实验创建后会自动出现在 **数据科学** > **模型实验** 列表中。

创建 AutoML 实验

1.创建 AutoML 分类实验

适合业务分析师在不写算法代码的前提下，通过界面配置自动训练分类模型。

> ⓘ **提示**
> AutoML 实验由系统自动调用 Serverless CPU 资源池，无需手动选择计算集群。

1. 登录 DataBuddy 控制台，在左侧导航栏选择 **数据科学** > **模型实验** 。
2. 单击 **分类模型** ，进入创建表单页面。
3. 按照以下区域完成表单填写：

- 基础配置
  - **实验名称**： 输入不超过 50 个字符的实验名称。
  - **实验描述**： 可选，输入不超过 100 个字符的用途说明。
  - **资源组**： 下拉选择用于训练的资源组。
- 数据配置

- **训练数据：** 下拉选择 Catalog 中的训练数据表（格式为 `Catalog 名 / Schema / 表名`）。选定后右侧自动展示数据配置区。
- **Join 特征表：** 可选，单击 **展开配置** 添加特征表并指定主键和时间戳键，用于拼接额外特征列。默认收起。
- **结果列：** 单选训练数据中作为分类标签的列。选定后该列在右侧数据配置区中自动标记为用于结果。
- **训练列：** 多选参与模型训练的特征列。未选中的列不参与训练。
- **数据配置区（右侧面板）：** 选定训练数据后，右侧自动展示数据表的字段列表，您可以在此查看和调整各列配置：

列	说明
用于结果	标记为分类标签的列（单选，与左侧结果列联动）
用于训练	勾选参与训练的特征列（多选，与左侧训练列联动）
列名	字段名称
来源表	字段所属的数据表，点击后可跳转至 Catalog 中对应表详情界面
数据类型	字段类型如 integer、decimal、string 等
缺失值	下拉选择该列的缺失值补全方式

- 训练配置
  - **评估指标：** 下拉选择模型评估指标，默认自动选择。
  - **搜索空间：** 多选需要自动测试的模型框架（默认全选，可选 sklearn / xgboost / lightgbm）。
  - **最大运行时间：** 设置单次实验的最长执行时长，单位为分钟，超时将停止实验。
  - **最大运行次数：** 设置实验最多生成的运行任务数量。
  - **最大运行并发：** 设置同时运行的运行任务数量上限。

> **提示**
> 最大运行并发为上限值，实际并发数还受限于资源组可用资源。

1. 单击**确定**，系统自动启动 AutoML 训练并跳转到实验详情页。

2.创建 AutoML 回归实验

操作流程与分类实验一致，单击 **回归模型** 进入创建表单，区别如下：

- 数据配置中 **结果列** 改为选择回归预测的目标列（数值类型）。
- 训练配置中 **评估指标** 可选：Deviance / RMSE / MAE /R2 / MSE。

其余配置项与分类实验完全一致。

3.创建 AutoML 时序预测实验

操作流程与分类实验一致，单击 **时序预测模型** 进入创建表单，区别如下：

- 数据配置

除训练数据外，还需配置以下时序特有字段：

- **选择时间列**：单选训练数据中的时间维度列，必须为 datetime 或 timestamp 类型。
- **预测范围**：点击下拉选择时间序列单位（可选秒、分钟、小时、天、周、月、年）。
- **预测结果保存**：选择 Catalog 中保存预测结果的路径，并输入表名。

数据配置区与分类实验一致，缺失值补全额外支持按时序补全。

- 训练配置
  - **评估指标**：可选 SMAPE / MSE / RMSE / MAE / MDAPE。
  - **搜索空间**：可选 Prophet / ARIMA / Deep-AR。

其余训练配置项与分类实验一致。

4.AutoML 实验自动生成的 Notebook

AutoML 实验启动后，系统会在 Workspace 用户根目录的 `/automl/<实验名>/` 下自动生成以下结构：

路径	内容
`<实验名>`	实验对象本体
`<实验名>+preprocessing`	数据清洗、特征工程、缺失值处理、类别编码
`<实验名>+training`	模型训练主流程、参数设置、训练过程、结果展示
`<实验名>+tuning`	超参数搜索与优化过程
`trials/`	每次试验的独立 Notebook，命名为时间戳+模型类型+随机编码
`failed-notebooks/`	运行失败的 Notebook，记录失败参数、代码片段、错误日志

您可以在实验详情页点击对应运行名称进入详情界面，再点击 **进入对应 Notebook** 进行复查或二次开发。

创建 Agent 实验

适合需要构建智能体（Agent）应用的场景，支持基于 OpenAI 框架或 LangGraph 框架快速创建 Agent，内置代码模板、调试工具与一键部署能力，并通过 MLflow 自动追踪 Agent 行为。

1. 登录 DataBuddy 控制台，在左侧导航栏选择 **数据科学** > **模型实验** 。
2. 在 **智能体** 页签下，选择框架模板：

- **代码开发（OpenAI 框架）**：基于 OpenAI Agents SDK 开发，支持 Function Calling。
- **代码开发（LangGraph 框架）**：有状态多步 Agent，支持工作流编排。

1. 单击对应模板，进入创建表单页面，按照以下区域填写：

- 基础信息
  - **智能体名称**：输入英文字母、数字、短横杠组合的名称，最大长度 30 个字符。
  - **智能体描述**：推荐填写，描述该智能体的功能用途，最多 500 个字符。
- 资源配置
  - **MLflow 实验**：下拉选择一个 MLflow 实验或直接创建新的实验，用于追踪 Agent 的行为记录。

1. 单击 **确定**，系统创建 Agent 并跳转到开发页面。

常见问题

AutoML 实验创建失败，提示资源不足怎么办？

AutoML 实验依赖 Serverless CPU 资源池。如果当前 Workspace 资源池占用过高，请稍后重试或联系工作空间管理员申请扩容。

实验删除后如何恢复？

删除实验会进入 Workspace 回收站，您可在 Workspace 回收站找回。删除的实验对应的 Notebook 会同步进入回收站。

> ⓘ **注意**
> 删除运行任务后，运行任务不直接进入回收站。如需找回，请使用 `mlflow.restore_runs` API 配合实验 ID 与删除时间戳恢复：

```
import mlflow

runs_restored = mlflow.restore_runs(
    experiment_id="<实验 ID>",
    min_timestamp_millis=<删除时间戳>,
    max_runs=10,
)
```

相关文档

- 什么是模型实验
- 记录训练指标
- 对比实验结果
- Notebook
- 计算资源概述

记录训练指标

最近更新时间：2026-09-11 20:42:31

在 DataBuddy 数据科学模块中，训练指标的记录基于 MLflow 协议。您在 Notebook 中调用 MLflow API 记录的参数、指标、工件，会自动汇聚到对应实验的运行中，供后续查看与对比。
本文介绍如何在 Notebook 中通过 MLflow API 记录这三类信息，并在实验详情页查看记录结果。

前提条件

在开始记录前，请确保：

- 已创建目标实验（参见 创建实验 ），并获取实验 ID 或实验名称。
- 当前账号在该实验上拥有编辑或管理权限。
- 已在 Notebook 中安装 MLflow 客户端（DataBuddy 默认环境已预置 MLflow，无需额外安装）。

MLflow 协议中可记录的三类信息为：参数（Parameter）、指标（Metric）、工件（Artifact）。各类型的含义与典型示例请参见 什么是模型实验 > 关键概念 。

记录方式选择

DataBuddy 支持两种记录方式，可按场景灵活选择：

方式	适用场景
自动记录	使用 sklearn / xgboost / lightgbm / pytorch / tensorflow 等主流框架，无需手写 `log_*` ，框架训练时自动捕获参数、指标与模型
手动记录	自定义训练流程；需要记录非框架默认捕获的指标，完全控制记录时机与内容

自动记录

在 `mlflow.start_run()` 之前调用 `mlflow.autolog()` ，框架训练时会自动捕获参数、指标、模型与训练数据集：

```
import mlflow
from sklearn.datasets import load_diabetes
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

mlflow.set_experiment(experiment_id="3")
mlflow.autolog()  # 自动记录
```

```
with mlflow.start_run():
    db = load_diabetes()
    X_train, X_test, y_train, y_test = train_test_split(db.data,
db.target)
    rf = RandomForestRegressor(n_estimators=100, max_depth=6)
    rf.fit(X_train, y_train)
    # 参数（n_estimators、max_depth）、训练指标、模型工件均自动记录
```

> **提示**
> Autologging 是注册模型的推荐方式之一：自动记录的模型默认携带签名（Signature），可直接用于 Catalog 注册。模型签名是注册模型的强制前置条件。

手动记录

1.手动记录参数

在 `with mlflow.start_run()` 上下文中调用 `mlflow.log_param`：

```
import mlflow

with mlflow.start_run(run_name="manual_run"):
    # 单个参数
    mlflow.log_param("learning_rate", 0.01)
    mlflow.log_param("batch_size", 64)
    mlflow.log_param("optimizer", "Adam")

    # 批量记录
    mlflow.log_params({
        "n_estimators": 100,
        "max_depth": 6,
        "max_features": 3,
    })
```

取值约束：

- `key` 为字符串，最大长度由 MLflow 协议规定。
- `value` 类型可以是字符串、整数或浮点数。
- 同一 Run 内同名参数 **只能记录一次**，重复记录会报错。

2.手动记录指标

在 `with mlflow.start_run()` 上下文中调用 `mlflow.log_metric`，可指定 `step` 用于绘制曲线：

```
import mlflow

with mlflow.start_run(run_name="manual_run"):
    # 单个指标
    mlflow.log_metric("accuracy", 0.92)
    mlflow.log_metric("auc", 0.88)

    # 训练过程中按 epoch 记录损失
    for epoch in range(100):
        loss = train_one_epoch(...)
        mlflow.log_metric("loss", loss, step=epoch)

    # 批量记录
    mlflow.log_metrics({
        "precision": 0.91,
        "recall": 0.89,
        "f1": 0.90,
    })
```

> **提示**
>
> - 指标值类型必须是数字（整数或浮点数）。
> - 同一 `key` 可多次记录，按 `step` 串成时间序列，在实验详情图表视图中展示为折线图。
> - 系统指标如 CPU 使用率、内存占用、网络流量、磁盘 IO 等由 DataBuddy 自动采集，无需手工记录。

3.手动记录工件（Artifact）

工件包括序列化的模型、日志、配置、图片等任意文件。记录方式分为两类：

1. 记录通用工件文件

```
import mlflow

with mlflow.start_run():
    # 记录单个文件
    mlflow.log_artifact("output/eval_report.html")

    # 记录到子目录
```

```
    mlflow.log_artifact("config/params.yaml", artifact_path="configs")

    # 记录整个目录
    mlflow.log_artifacts("output/figures", artifact_path="figures")
```

1. 记录模型工件

不同 ML 框架使用对应的 `log_model` 方法。常用示例：

```
# scikit-learn
mlflow.sklearn.log_model(
    sk_model=model,
    name="sklearn-model",
    input_example=X_train,  # 提供输入样例，自动生成签名
)

# PyTorch
mlflow.pytorch.log_model(model, "model")

# 自定义 Python 模型
mlflow.pyfunc.log_model(artifact_path="model", python_model=my_model)
```

> ⓘ **注意**
> **注册到 Catalog 的模型必须携带签名（Signature）。建议在 `log_model` 时传入 `input_example=` 让 MLflow 自动推断签名；或在调用前显式构造 `mlflow.models.signature.ModelSignature`。无签名的模型在后续注册时将被拒绝。**

1. 在记录时直接注册模型

通过 `registered_model_name=` 参数可在 `log_model` 时同步注册到 Catalog（同一模型名首次注册为 V1，再次注册版本号自动 +1）：

```
mlflow.sklearn.log_model(
    sk_model=model,
    name="sklearn-model",
    input_example=X_train,
    registered_model_name="my_catalog.my_schema.churn_model",
)
```

模型注册的详细规则参见 模型版本管理 。

完整示例

下面是一个端到端的示例，演示在一次训练中同时记录参数、指标、模型并完成注册：

```
import mlflow
import mlflow.sklearn
from sklearn.datasets import make_regression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

mlflow.set_experiment("regression_demo")

with mlflow.start_run() as run:
    X, y = make_regression(n_features=4, n_informative=2,
random_state=0, shuffle=False)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    params = {"max_depth": 2, "random_state": 42}
    model = RandomForestRegressor(**params)
    model.fit(X_train, y_train)

    # 记录参数
    mlflow.log_params(params)

    # 记录指标
    y_pred = model.predict(X_test)
    mlflow.log_metrics({"mse": mean_squared_error(y_test, y_pred)})

    # 记录并注册模型（同步注册到 Catalog）
    mlflow.sklearn.log_model(
        sk_model=model,
        name="sklearn-model",
        input_example=X_train,
        registered_model_name="catalog_name.schema_name.model_name",
    )
```

父子 Run 与超参搜索

在父 Run 上下文中嵌套子 Run，常用于超参数搜索场景：

```
import mlflow

with mlflow.start_run(run_name="Parent Run") as parent:
    mlflow.log_param("model_type", "RandomForest")
    for n in [50, 100, 200]:
        with mlflow.start_run(run_name=f"n_estimators={n}",
nested=True):
            mlflow.log_param("n_estimators", n)
            # ... 训练并记录 metric ...
```

实验详情中的运行任务列表会以嵌套方式展示父 Run 与其下的子 Run，便于分层查看。

实验详情

1. 如何查看实验详情

记录完成后，进入实验详情查看：

1.1 进入**数据科学** > **模型实验**，点击目标实验名称进入详情。

1.2 默认在 **运行任务** 中以列表展示所有 Run。

1.3 点击运行任务名称进入 Run 详情，包含 4 个 Tab 如下：

Tab	内容
概览信息	关于此运行任务的基本信息。支持点击跳转数据集、模型、代码文件界面。支持编辑标签，搜索参数配置、指标
模型指标	展示通过 `log_metric` 记录的指标的可视化图表（柱状图、折线图等）
系统指标	CPU 使用率、内存、网络、磁盘等系统级监控图表
模型文件	当前 Run 的所有工件文件，支持浏览、复制、下载

2. 添加自定义标签

在 **概览信息** 页面，您可以为该运行任务添加自定义键值对标签，用于分类与检索。

操作方式：在**添加标签** 区域输入键值对，每输入一组后点击保存按钮，系统会自动新增一行；点击行末删除按钮可移除该行。

3. 验证结果

记录完成后，您可以通过以下方式确认记录结果：

- **概览信息** 中可看到所有记录的参数、指标、自定义标签。
- **模型指标** 与 **系统指标** 中可看到对应的图表。
- **模型文件** 中可看到记录的工件文件树状结构。
- 如果同步注册了模型，可在 **模型管理** 列表中找到对应的注册模型。

常见问题

调用 `mlflow.log_param` 时报错 Parameter already logged 怎么办?

同一 Run 内同名参数只能记录一次。请检查代码是否在循环中重复记录同一参数；如果确实需要按时间步记录变化的值，请改用 `mlflow.log_metric(key, value, step=...)` 。

指标曲线为什么只显示一个点?

`mlflow.log_metric(key, value)` 不传 `step` 时默认按记录顺序累加。如果只调用了一次，曲线只有一个点。在训练循环中按 epoch 调用 `log_metric(key, value, step=epoch)` 即可绘制完整曲线。

模型注册时提示模型没有签名怎么办?

注册模型要求模型必须携带签名。请：

1. 在 `log_model` 调用时传入 `input_example=` 让 MLflow 自动推断签名；或
2. 显式构造 `mlflow.models.signature.ModelSignature` 并通过 `signature=` 参数传入；或
3. 启用 `mlflow.autolog()` 由框架自动记录带签名的模型。

若模型没有签名，则模型不会被记录，**模型文件** 区不会显示已记录的模型文件。

手动启动的 Run 一直停留在运行中状态怎么办?

如果代码异常退出且未正确关闭 Run，可能导致 Run 状态滞留。建议始终使用 `with mlflow.start_run() as run:` 上下文管理器，异常退出时也会自动调用 `mlflow.end_run()` 关闭 Run。

相关文档

- 什么是模型实验
- 创建实验
- 对比实验结果
- Notebook

对比实验结果

最近更新时间：2026-09-11 20:42:31

DataBuddy 支持两类对比场景：**对比不同实验** 和 **对比同一实验下的不同运行任务** 。本文介绍这两种对比方式的操作步骤及相关功能。

前提条件

- 已创建实验且实验下至少存在 1 个运行任务（参见 记录训练指标 ）。
- 当前账号在目标实验上拥有读取及以上权限。

> ⓘ **注意**
> 智能体实验暂不支持实验对比。

对比不同实验

当您在多个同类型实验之间进行横向比较时。可在实验列表页进行跨实验对比。

1. 登录 DataBuddy 控制台，在左侧导航栏选择**数据科学** > **模型实验** 。
2. 在实验列表中通过左侧复选框选中两个或更多同类型实验。
3. 点击列表上方出现的**对比实验** ，进入跨实验对比窗口。

> ⓘ **提示**
> 跨实验对比主要用于宏观比较不同方案的最终效果，如需细粒度对比各运行任务之间的差异，请进入单个实验内进行运行任务对比。

对比同一实验下的不同运行任务

同一实验内通常包含多个运行任务（不同的超参组合、不同的训练数据等）。进入实验详情后，您可以通过表格视图、图表视图查看各运行任务表现，并选中多个运行任务进行逐项对比。

1.进入实验详情

1. 登录 DataBuddy 控制台，在左侧导航栏选择 **数据科学** > **模型实验** 。
2. 在实验列表中点击目标实验名称，进入实验详情。

2. 查看运行任务与模型

进入实验详情后，通过列表上方的视图切换器选择查看方式：

- **运行任务视图（默认）**：以运行任务为单位展示所有训练记录。
  - **表格模式**：展示运行任务基础信息。
    - 可按照创建时间选择正序、倒序对运行任务进行排序。

- 列表上方搜索框支持按照运行任务名称、运行任务 id 进行搜索。
- **图表模式**：单击视图切换器切换为图表，以柱状图形式对比各运行任务的指标。支持搜索图表指标、隐藏、显示指定运行任务、下载指标图片等操作。

- **模型视图**：单击切换器切换到模型视图，查看该实验下已注册到 Catalog 的模型版本。
  - **表格模式**：展示模型基础信息。操作列提供将模型注册到 Catalog 和删除模型两项操作。
  - **图表模式**：同运行任务图表模式，您可以搜索指标、直观的对比各模型版本的评估指标，便于直观判断不同版本的表现差异。

> **提示**
> - 同一运行任务注册同一个模型的多个版本会拆成多行显示；同一运行任务注册多个模型则在同一行内合并显示。
> - 如果运行任务生成的模型在记录时没有绑定实验（来源为空或仅为 Notebook），该模型不会出现在模型视图中。

3. 多选对比

在运行任务表格视图中通过左侧复选框选中两个或更多运行任务，点击上方**对比任务** 按钮后进入对比页面。任务对比页面提供两种对比模式（通过顶部 Tab 切换）：

- **数值对比**：将所选运行任务横向并排展示，逐项对照，提供 **仅看差异项** 开关。开启后只显示在所选运行任务之间存在差异的字段，便于聚焦关键差别。
- **图表对比**：可选不同参数配置、图表类型（平行坐标系、散点图、箱线图等），对结果进行可视化对比。

4. 选定最佳运行

实验详情会基于评估指标自动识别最佳运行，置顶显示在实验面板上。

常见问题

Q：表格视图中找不到我刚训练完的运行任务？

请检查：

- 是否在正确的实验下查看（左上角面包屑确认实验名称）。
- 运行任务是否成功结束（在概览中查看运行任务状态）。
- 是否被筛选条件过滤掉（清空筛选条件再查看）。

Q：图表视图中显示的运行任务数量比实际少？

当运行任务数量超过 10 个时，图表视图默认只展示前 10 条，其他运行任务自动隐藏。在指标列表中手动展开隐藏项即可全部查看。

Q：对比窗口中无法滚动到底部怎么办？

当对比指标过多时，对比窗口可能存在滚动条不易触达的问题，建议先通过差异项勾选减少展示行数，再进行对比。

Q：模型视图为空，但运行任务视图中明明有产出模型?

模型视图仅展示已经绑定实验的模型。如果运行任务生成的模型来源为空或仅为 Notebook（未绑定到实验），它们不会出现在模型视图。请在 `log_model` 时确认 `experiment_id` 设置正确。

相关文档

- 什么是模型实验
- 创建实验
- 记录训练指标

特征管理
界面操作

最近更新时间：2026-09-11 18:24:34

本文介绍特征管理在 DataBuddy 控制台中的界面操作，包括特征表的检索与筛选、特征表详情查看，以及将离线特征表发布为在线特征表。

> **提示**
> 特征管理页面为**纯索引视图**，不提供创建特征表或批量导入按钮。特征表的创建通过 特征工程 SDK 的 `create_table` 完成，或通过下文自动识别方式将已有表纳入管理。

前提条件

- 已开通 DataBuddy 服务并完成空间初始化。
- 当前账号在目标 Catalog / Schema 上拥有相应的查看或元数据维护权限。

特征管理列表检索与筛选

在左侧导航栏选择 **数据科学** > **特征管理**，进入特征管理列表页。列表以纯索引视图展示当前 Workspace 中所有已识别的特征表。

顶部操作区

- **搜索特征表**：按特征表名称模糊检索。
- **范围筛选**：按特征表**责任人**、所属 **Catalog**（支持多选）、**标签** 进行筛选。
- **刷新**：重新加载列表，获取最新状态。
- **Buddy**：打开 Buddy 入口，与智能体实时互动。

特征表列表区

字段	说明
特征表名称	`catalog.schema.table` 格式，点击跳转至特征表详情页
责任人	该表在 Catalog 中登记的责任人
在线表	关联的在线特征表所在的 Catalog 名称（鼠标悬停可查看全名）；未发布在线表时为空
最近更新时间	表数据或元数据的最近一次更新时间
标签	Catalog 中维护的标签集合
描述信息	Catalog 中维护的表描述

查看特征表详情

特征表本质是 **“用特征工程 SDK 声明了主键和时间戳键的 Catalog 中的表”** ，因此其详情页即该表在数据目录中的资产详情页。可通过以下任一方式进入：

- **从特征管理列表进入：** 在特征管理列表中点击特征表名称。
- **从数据目录进入：** 在左侧导航栏选择 **目录**（Catalog），展开目标 Catalog > Schema > 表，点击对应的特征表。

顶部操作栏

页面顶部展示当前特征表的名称、路径（ **快速访问** `/ <catalog> / <database> / <table>` ）以及一组操作入口：

操作	说明
搜索字段名称	在当前表的字段列表中按字段名快速过滤
根据表创建	下拉菜单，基于当前特征表快速发起下游操作（详见下表）
修改名称	修改特征表名称
删除表	删除当前特征表（等价于 SDK 的 `drop_table` ，操作不可恢复）
Buddy	一键带着当前表上下文进入 Buddy 进行智能分析

根据表创建 下拉项：

入口	说明
在线表	将当前离线特征表发布为在线特征表（详见 特征工程 SDK ）
NoteBook	新建 Notebook 并自动带入当前表，便于直接读写特征数据
SQL 探索	打开 SQL 探索并预填当前表，便于即席查询
仪表盘	基于当前表创建可视化仪表盘
分析空间	进入 Buddy 对当前表做对话式分析
质量任务	为当前表配置数据质量校验任务
Power BI 连接插件	获取 Power BI 连接当前表的配置

标签页

详情页提供 **概览、数据预览、血缘、权限、详情、变更历史、质量** 等标签页：

- **概览**：维护特征表元信息的主入口。可点击 **添加描述** 补充表的业务说明、编辑 **责任人** 、**添加标签** 维护表级标签；在字段列表中可编辑字段的 **字段描述** 、为字段 **添加标签**（主键字段带 **PK** 标识、时间戳键带相应标识）、添加**脱敏策略** 。
- **数据预览**：查看特征表数据样本。
- **血缘**：查看该表的上下游血缘关系。
- **权限**：查看与管理该表的访问权限。
- **详情**：查看表的 Schema、字段定义、存储信息等。
- **变更历史**：查看表结构与数据的变更记录。
- **质量**：查看数据质量校验结果。

> ⓘ **提示**
> 表级描述、责任人、字段描述、字段标签等元数据均由 Catalog 统一维护，修改后会同步到特征管理列表。建议在特征表正式投入训练之前补充完整，帮助下游使用者理解每个特征的业务含义。

自动识别已有表为特征表

Catalog 中任意一张**同时** 具备主键和时间戳键的表都会被**自动识别** 为特征表，无需调用 SDK 或在页面上执行任何额外操作。表在创建时已由数据工程流程或特征工程 SDK 指定好主键与时间戳键，则该表会立即出现在特征管理列表中。
表详情页 **概览** Tab 的字段列表中，主键字段右侧以 PK 标识展示、时间戳键字段右侧以时间戳图标展示，但这两个属性是**在表创建时确定并由后端读取的元数据** ，**当前 UI 不提供标记主键或标记时间戳键的可编辑入口** 。要让一张已有表被识别为特征表，需要在表的源头（DML / SQL DDL / 特征工程 SDK `create_table` ）中保证其同时具备主键与时间戳键。

> ⓘ **注意**
> 仅当主键与时间戳键**同时** 存在时，该表才会被识别为特征表。仅有主键或仅有时间戳的表不会出现在特征管理列表中。

验证方式

如需确认一张表是否被识别为特征表：

1. 在左侧导航栏选择 **目录**（Catalog），进入对应 Catalog > Schema > Table 详情页。
2. 查看字段列表中是否有字段被标记为 **PK**（主键）以及是否有字段带有 **时间戳** 图标。
3. 同时满足两项时，进入 **数据科学** > **特征管理** ，可在列表中找到该表。可点击列表页的 **刷新** 按钮重新加载。

发布在线特征表

如果模型需要部署为在线推理服务，则其依赖的离线特征表必须先被发布为在线特征表，由同步任务持续把离线数据同步到在线存储（PostgreSQL）。

1. 进入 **数据目录** ，打开目标离线特征表详情页。

2. 在 **根据表创建** 中选择 **在线表** ，弹出在线表创建窗口。
3. 在弹窗中配置参数并提交。系统会自动创建在线特征表，并同时创建一条特征同步工作流。

> **提示**
> 仅 Catalog 中具有主键和时间戳的离线特征表会展示 **创建在线表** 入口；不满足条件的表无法发布为在线表。

弹窗参数：

参数	说明	是否必填
计算资源	运行同步任务的计算资源	是
存储地址	用于在线表的 PostgreSQL 数据源	是
数据目录 / Schema / 表名	在线特征表的目标 Catalog、Schema 与表名	是
同步方式	快照 / 定时触发 / 持续运行	是

> **提示**
> 开通数据源指引参考 开通数据源 。

选择并配置同步方式：

同步方式	适用场景	触发逻辑
快照	一次性把离线特征表的当前快照同步到在线表，常用于首次发布或临时刷新	提交后立即执行单次同步
定时触发	周期性从离线特征表更新到在线表，例如每天凌晨同步当日产出的特征	按调度周期或 Cron 表达式定时触发
持续运行	实时性要求较高时连续运行同步任务	任务完成一次后立即开始下一次执行

调度配置项的详细规则与 Workflow 调度保持一致，详见 调度与触发器。本场景下需注意以下差异：

1. 持续运行模式不支持排队模式与最大并发数配置。
2. 任务重试模式（仅持续运行模式）：
   - **失败**：只要其他任务仍在首次尝试运行，失败的任务就会重试；如果所有任务都进入重试状态，会创建一个新的作业运行。
   - **从不**：失败后不再重试。

> **注意**
> 特征同步工作流的 Notebook 文件由系统创建并放置在固定目录，不允许修改任务文件内容、删除工作流或调整任务结构；您仅能修改其调度配置并启动运行。

> **提示**
> 除页面操作外，也可在 Notebook 中通过 SDK 的 `publish_table` 一键发布在线表，详见 特征工程 SDK – 在线特征表操作 。

常见问题

Q：在特征管理列表中没有看到刚创建、标记的表?

请确认目标表是否**同时** 具备 **主键** 与 **时间戳键** 。在 Catalog 表详情页检查字段标记；只有同时满足两项条件的表才会被识别为特征表。可点击列表页的 **刷新** 重新加载。

Q：特征表的字段描述、字段标签为什么要在 Catalog 里维护，而不是特征管理模块?

在 DataBuddy 中，特征表本质是 Catalog 中的表，所有元数据维护逻辑统一在 Catalog 完成。特征管理模块通过跳转复用同一份元信息，避免出现两套描述不一致。

Q：特征表的负责人转交后，原负责人是否仍能维护标签?

标签维护权限随 Catalog 中的表权限走。负责人转交后，原负责人若未保留对应权限，将无法继续维护表元数据；此机制与 Catalog 资产管理一致。

相关文档

- 特征工程 SDK
- 最佳实践与异常排查
- 数据目录概览
- 调度与触发器

特征工程 SDK

最近更新时间：2026-09-11 18:24:34

本文介绍 DataBuddy 特征工程 SDK（`wedata3-feature-engineering`）的完整 API 用法，覆盖客户端初始化、特征数据库管理、特征表的创建、写入、读取、元数据维护、删除、离线特征消费（训练集构建、模型记录、批量推理）以及在线特征表操作。

前提条件

- 已开通 DataBuddy 服务并完成空间初始化。
- 当前账号在目标 Catalog / Schema 上拥有相应的数据读写权限。
- 已创建并启用一个可用的计算资源（创建方法详见 创建计算资源）。
- 运行特征工程代码的内核环境 **Python 版本 ≥ 3.10**。
- 已在 Notebook / Python 内核中安装特征工程 SDK（要求 **0.2.3 及以上**）：

```
%pip install "wedata3-feature-engineering[mlflow3]>=0.2.3"
```

> **注意**
> 本文示例基于 0.2.3+ 版本验证。SDK 依赖 `mlflow >= 3.10.0, < 3.11.0`，若 `import wedata` 时提示 mlflow 版本不满足，请确认安装的是 `[mlflow3]` extra（即 `wedata3-feature-engineering[mlflow3]`），且不要在同一 Notebook 内混装 mlflow 2.x 与 3.x。

> **提示**
> 在 DataBuddy Notebook 内核中运行时，鉴权所需环境变量会被自动注入，无需手动配置；若在本地 IDE 调试，则需自行 `export` 环境变量，否则初始化 `FeatureEngineeringClient(...)` 时会抛 `EnvironmentError` / `KeyError`。

初始化客户端

在代码单元中导入并初始化特征工程客户端。DataBuddy Notebook 内核已预置 `spark` 会话，直接传入即可；本地调试时需自行创建 SparkSession 并传入。

```
%pip install "wedata3-feature-engineering[mlflow3]>=0.2.3"
%pip list | grep wedata3-feature-engineering
```

```
from wedata.feature_engineering.client import FeatureEngineeringClient
from wedata.common.constants.engine_types import EngineTypes
```

```
# DataBuddy Notebook 内核已预置 spark 会话；本地调试时需自行创建并传入
fe = FeatureEngineeringClient(spark)
```

特征数据库管理

特征表归属于某个 Schema（特征数据库）。若目标 Schema 尚未创建，可先通过 SDK 幂等准备；若已规划好 Schema，可跳过本节直接建表。

```
# 幂等创建特征数据库（已存在则跳过）；支持单段名或二段式 catalog.schema
fe.create_database("<schema_name>", catalog_name="<catalog_name>",
comment="<数据库描述>")

# 列出指定 Catalog 下的全部数据库
fe.list_databases(catalog_name="<catalog_name>")

# 判断数据库是否存在（任何内部异常都会被吞掉并返回 False）
fe.database_exists("<schema_name>", catalog_name="<catalog_name>")

# 删除数据库，默认非级联以避免误删；级联删除请慎用
fe.drop_database("<schema_name>", catalog_name="<catalog_name>")
```

接口	作用	关键参数
`create_database`	幂等创建特征数据库（`CREATE DATABASE IF NOT EXISTS`）	`database_name`（必填，支持二段式 `catalog.schema`）、`catalog_name`、`comment`、`location`
`drop_database`	删除特征数据库	`database_name`、`catalog_name`、`cascade`（默认 `False`，是否级联删表）、`if_exists`（默认 `True`）
`list_databases`	列出指定 Catalog 下的全部数据库	`catalog_name`（不传则为当前 Catalog）
`database_exists`	幂等检测数据库是否存在	`database_name`、`catalog_name`

> ⓘ **注意**

> `drop_database` 在数据库非空且 `cascade=False` 时会由底层抛错；设置 `cascade=True` 会级联删除其下所有表，生产环境请谨慎使用。

创建特征表

调用 `create_table` 创建特征表。表结构可由传入的 `df` 自动推断，也可显式传入 `schema`；`engine_type` 为必填项，用于指定底层表引擎：

```
from wedata.common.constants.engine_types import EngineTypes

# 方式一：传入 df，自动推断 schema 并写入首批数据
fe.create_table(

    name="<catalog>.<schema>.<table_name>",        # 特征表全限定名
    primary_keys=["<主键列表>"],                     # 单键传字符串，复合主键传列表
    timestamp_key=["<时间戳列名>"],                   # 时间戳键列名（单数）
    df=<Spark DataFrame>,                          # 传入 df 可自动推断 schema 并写入首批数据
    engine_type=EngineTypes.ICEBERG_ENGINE,        # 必填：ICEBERG_ENGINE 或 HIVE_ENGINE
    description=["<表描述>"],                       # 可选
)

# 方式二：不传 df，显式提供 schema 建空表
fe.create_table(
    name="<catalog>.<schema>.<table_name>",
    primary_keys=["<主键列名>"],
    timestamp_key="<时间戳列名>",
    schema=<PySpark StructType>,                    # df为空时必填
    engine_type=EngineTypes.ICEBERG_ENGINE,
)
```

> **提示**
>
> 1. `primary_keys` 类型为 `Union[str, List[str]]`，单主键可直接传字符串，复合主键传列表。
> 2. `schema` 与 `df` 二选一：当 `df` 已提供时 `schema` 可省略（由 df 推断）；仅当 `df=None` 时 `schema` 必填。

3. 其他可选参数：`catalog_name` / `database_name`（在 `name` 未使用全限定名时指定）、`tags`（表级标签字典）、`partition_columns`（分区列）。

> **提示**
> 引擎选择（`engine_type`）：`EngineTypes` 提供 `ICEBERG_ENGINE`（Iceberg 表，推荐）与 `HIVE_ENGINE`（Hive 表）两种取值。优先选用 Iceberg：它支持 schema evolution 与 time travel，在按时间点回溯特征查询时性能更优。

> **注意**
> **写入前的数据类型治理：** 特征工程 SDK 不识别 `decimal(p, s)` 类型——它会导致记录模型时无法推断模型签名（`signature`），进而引发在线、离线推理的类型不匹配。建议在写入特征表之前，把所有 `decimal` 列统一 `cast("double")`，并将该处理沉淀为可复用的工具函数。时间戳列应保留为时间类型，不要转为字符串，否则无法用作时间戳键。

执行成功后，新建的特征表将自动出现在特征管理列表中。

> **提示**
> **主键与时间戳的选择建议：** 主键用于唯一定位一条特征记录；时间戳键用于实现按时间点回溯特征值。建议将业务实体的唯一标识作为主键、特征生成时间作为时间戳键。

写入特征表

如需追加写入更多特征数据，调用 `write_table`：

```
fe.write_table(
    name="<catalog>.<schema>.<table_name>",        # 目标特征表全限定名
    df=<Spark DataFrame>,                          # 待写入的特征数据
    mode="append",                                 # 写入模式：append（默认）/ overwrite
)
```

流式写入时需额外提供 checkpoint 路径：

```
fe.write_table(
    name="<catalog>.<schema>.<table_name>",
    df=<Streaming DataFrame>,
    mode="append",
    checkpoint_location="<checkpoint 路径>",       # 流式写入必填
    trigger={"processingTime": "10 seconds"},      # 可选，流式触发器配置
```

```
)
```

写入模式说明：

- **append**：追加写入，新数据按主键合并入表（默认模式）。
- **overwrite**：覆盖整张表的全部数据，慎用于生产场景。

> **提示**
> 当前 SDK（0.2.4）`write_table` **不支持** `merge` **模式。增量写入请使用默认的** `append` **。**

获取与读取特征表

调用 `get_table` 可以编程方式拿到特征表的完整元数据；调用 `read_table` 读取特征表数据（返回 Spark DataFrame）：

```
# 获取元数据
table_info = fe.get_table(name="<catalog_name>.<schema_name>.
<table_name>")
print(table_info.primary_keys)
print(table_info.timestamp_keys)
print(table_info.tags)

# 读取数据
df = fe.read_table(name="<catalog_name>.<schema_name>.<table_name>")
df.show(5)
```

> **提示**
> `read_table` 读取的是离线特征表中的全量数据。在线特征表（PostgreSQL）中的数据用于在线推理服务在收到请求时自动查找特征，无需调用方在 SDK 中显式读取；如需核对在线表数据，可在数据目录中查看对应的在线特征表。

维护特征表标签

特征表标签可用于版本号、所属业务线、特征类别等业务分类，可通过 SDK 管理：

```
# 设置或更新一个标签
fe.set_feature_table_tag(
    name="<catalog_name>.<schema_name>.<table_name>",
    key="<tag_key>",           # 如 "business_line"、"version" 等
    value="<tag_value>",       # 如 "growth"、"v1.2.0" 等
```

```
)

# 删除一个标签
fe.delete_feature_table_tag(
    name="<catalog_name>.<schema_name>.<table_name>",
    key="<tag_key>",
)
```

设置或删除完成后，标签会同步显示在 Catalog 详情页与特征管理列表的 **标签** 列。推荐为每张特征表至少维护以下标签，便于检索与治理：

标签键	取值示例	用途
`owner`	`ml-team`	责任人 / 责任团队
`purpose`	`test` / `prod`	用途（实验 / 生产）
`domain`	`user` / `order` / `product`	业务域
`sla`	`daily` / `hourly`	数据更新频率

> **注意**
> `primary_keys` 、 `timestamp_keys` 等是 SDK 的保留字段，**不能** 作为自定义标签的 key 使用。

> **提示**
> **版本管理建议：** DataBuddy 当前未提供独立的特征表版本号对象，建议使用 `version` 标签（如 `v1.0.0` 、 `v1.1.0` ）配合 Catalog 表本身的数据变更历史来标识业务版本演进。如需在不同版本之间快速切换，可采用按版本建独立特征表、在训练代码中显式选择的方式。

删除特征表

当一张特征表不再被任何模型与下游使用时，可通过 SDK 删除：

```
fe.drop_table(name="<catalog_name>.<schema_name>.<table_name>")
```

> **警告**
> 删除特征表会同时删除底层数据文件，该操作不可恢复。删除前请确认：
> - 没有训练任务、推理任务、Workflow 仍在依赖该表。
> - 没有已发布的在线特征表以该表为来源（需先 `drop_online_table` ）。

- 没有模型工件中的 `feature_spec.yaml` 仍在引用该表。

构建训练数据集

以一组主键（如实体 ID、事件时间戳）作为 DataFrame，调用 `create_training_set` 拼接多张特征表生成训练数据：

```
from wedata.feature_engineering.client import FeatureEngineeringClient
from wedata.common.entities.feature_lookup import FeatureLookup

fe = FeatureEngineeringClient(spark)

# 1. 主键 DataFrame：包含训练标签 (label) + 主键 + 事件时间戳
labels_df = spark.table("<catalog_name>.<schema_name>.
<label_table_name>")

# 2. 定义需要拼接的特征表与字段（强烈推荐使用三段式全限定名，避免依赖默认 catalog）
feature_lookups = [
    FeatureLookup(
        table_name="<catalog_name>.<schema_name>.
<feature_table_name_1>",
        feature_names=["<feature_column_1>", "<feature_column_2>"],
        lookup_key=["<primary_key>"],
        timestamp_lookup_key="<timestamp_key>",
    ),
    FeatureLookup(
        table_name="<catalog_name>.<schema_name>.
<feature_table_name_2>",
        feature_names=["<feature_column_3>"],
        lookup_key=["<primary_key>"],
        timestamp_lookup_key="<timestamp_key>",
    ),
]

# 3. 创建训练数据集
training_set = fe.create_training_set(
    df=labels_df,
    feature_lookups=feature_lookups,
    label="<label_column>",
    exclude_columns=[],
```

```
)

training_df = training_set.load_df()
```

> **提示**
> `timestamp_lookup_key` 用于按时间点回溯特征值（point-in-time correctness），即对每条样本拼接其事件时间点之前最近一次的特征值，避免特征穿越。如需限制回溯的时间范围，可在 `FeatureLookup` 中配合 `lookback_window`（`datetime.timedelta`）使用，仅查找该时间窗内的特征值。

> **注意**
> **数据类型治理：** 写入与拼接前务必将 `decimal(p,s)` 转为 `double`。SDK 不识别 `decimal(p,s)` 类型，否则可能导致类型解析失败或在线、离线值不一致。

回溯窗口与多表 / On-Demand 特征组合

1. **回溯窗口：** `lookback_window` 必须与 `timestamp_lookup_key` 同时使用，且取值为**非负** 的 `datetime.timedelta`（`0` 表示精确匹配，不传表示不限制），否则会抛 `ValueError`。
2. **多 `FeatureLookup` + `FeatureFunction` 组合：** 可同时声明多张特征表查找，并通过 `FeatureFunction` 引入按需计算（On-Demand）特征：

```
import datetime
from wedata.common.entities.feature_lookup import FeatureLookup
from wedata.common.entities.feature_function import FeatureFunction

feature_lookups = [
    FeatureLookup(
        table_name="<catalog_name>.<schema_name>.<feature_table_a>",
        lookup_key=["<primary_key_a>"],
        feature_names=["<feature_col_1>", "<feature_col_2>"],
        rename_outputs={"<feature_col_1>": "<renamed_col_1>"},  # 输出列重
命名，避免同名冲突
    ),
    FeatureLookup(
        table_name="<catalog_name>.<schema_name>.<feature_table_b>",
        lookup_key=["<primary_key_b>"],
        timestamp_lookup_key="<timestamp_key>",
        lookback_window=datetime.timedelta(days=30),  # 仅取近 30 天内的特
征值
    ),
```

```
    FeatureFunction(
        # 按需计算特征（On-Demand UDF）
        udf_name="<catalog_name>.<schema_name>.<udf_name>",
        input_bindings={"<udf_input_param>": "<renamed_col_1>"},
        output_name="<udf_output_col>",
    ),
]
```

训练并记录模型

使用拼接好的训练 DataFrame 训练模型，并通过 `log_model` 把模型连同特征来源信息一起记录到 MLflow：

```
import mlflow
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import GradientBoostingClassifier

with mlflow.start_run() as run:
    pdf = training_df.toPandas()
    X = pdf.drop(columns=["<label_column>"])
    y = pdf["<label_column>"]

    # 预处理（OHE / Scaler 等）必须封装进 Pipeline，禁止在 Pipeline 外部用
pd.get_dummies 等手工预处理，
    # 否则在线推理时无法复现训练期的特征变换，导致线上线下不一致。
    preprocessor = ColumnTransformer(
        transformers=[
            ("cat", OneHotEncoder(handle_unknown="ignore"), ["
<categorical_feature>"]),
            ("num", StandardScaler(), ["<numeric_feature_1>", "
<numeric_feature_2>"]),
        ]
    )
    model = Pipeline(steps=[
        ("preprocessor", preprocessor),
        ("clf", GradientBoostingClassifier()),
    ])
    model.fit(X, y)
```

```
    fe.log_model(
        model=model,
        artifact_path="<model_artifact_path>",
        flavor=mlflow.sklearn,
        training_set=training_set,
        registered_model_name="<catalog_name>.<schema_name>.
<model_name>",
        infer_input_example=True,
        # 自动根据训练数据生成 input_example 与模型签名
    )
```

> ⓘ **注意**
>
> 关于 `log_model` 需遵守：
>
> 1. 必须用 `fe.log_model`，不能用 `mlflow.sklearn.log_model`：只有前者会把 `FeatureSpec` 打包进 MLmodel，后者会导致 `score_batch` 找不到特征来源。
> 2. 必须传 `training_set`：模型签名与 `input_example` 由 SDK 根据 `training_set.feature_spec` 自动推断，对齐的是特征表中的原始列。
> 3. `infer_input_example=True` 自动从训练数据生成 `input_example` 与签名，是部署在线推理服务的前置条件，务必开启；不要再手动 `infer_signature(...)` 或手动传 `input_example=...`，否则会与 `feature_spec` 的原始列错位、污染契约。
> 4. 所有预处理（OneHotEncoder、StandardScaler、自定义函数等）必须封装进 `sklearn.Pipeline`，传入的 `model` 是整条 Pipeline。不能在 Pipeline 外部用 `pd.get_dummies` / 手动 `StandardScaler.fit_transform` 后再 `model.fit(...)` ——外部预处理步骤不会被序列化进 MLmodel，推理时 SDK 从特征表 lookup 出来的是原始 string 列，喂给只认 OHE 后列名的裸模型，轻则报 `Feature names unseen at fit time`，重则所有样本静默输出训练集均值。
> 5. `registered_model_name` 建议使用三段式 `catalog.schema.model_name`，便于 Model Registry 治理。

`fe.log_model` 会在模型工件中写入一份 `feature_spec.yaml`，记录该模型在推理时所依赖的特征表、字段、主键与时间戳关系。这份文件将在后续的 `score_batch` 与在线推理中被自动读取。

离线批量推理

完成训练后，调用 `score_batch` 进行离线批量推理。此时只需提供主键 DataFrame，特征列由 SDK 根据模型工件中的 `feature_spec.yaml` 自动从特征表中查找并补齐：

```
# 主键 DataFrame：仅包含主键 + 事件时间戳，不需要特征列
# 注意：时间戳列需与特征表 schema 类型一致（如需 cast 为 timestamp）
```

```
keys_df = spark.table("<catalog_name>.<schema_name>.
<scoring_table_name>")

result_df = fe.score_batch(
    model_uri="models:/<catalog_name>.<schema_name>.
<model_name>/<version>",  # 版本号对齐训练产出的最新版本
    df=keys_df,
    result_type="double",       # 可选，预测结果列的数据类型，默认 "double"
    timestamp_key="<timestamp_key>",  # 训练含 timestamp_lookup_key 时强烈
建议显式指定，保证 point-in-time 一致性
)

result_df.show()
```

> ⓘ **注意**
>
> `score_batch` 的输入 `df` 有以下硬性约束：
>
> 1. 不能包含 `prediction` 列，也不能是流式 DataFrame；
> 2. 必须包含 `feature_spec.yaml` 中声明的所有 `lookup_key`；
> 3. 训练时声明了 `timestamp_lookup_key` 的场景下，强烈建议在 `df` 中显式带上同名时间戳列以保证 point-in-time 一致性。若不带，SDK 会自动注入 `current_timestamp()` 取「当前最新可见」的特征，等同于「用今天的特征预测历史样本」，语义不严谨。

执行流程：

1. SDK 根据 `model_uri` 加载模型与 `feature_spec.yaml`。
2. 根据模型签名识别需要的特征列；对未在 `keys_df` 中提供的列，自动通过 Feature Lookup 从特征表中查找。
3. 拼装完整的输入 DataFrame 进入模型推理，输出预测结果。

在线特征表操作

发布在线特征表

除页面操作外，也可在 Notebook 中通过 SDK 的 `publish_table` 一键发布：

```
fe.publish_table(
    catalog_name="<offline_catalog_name>",          # 离线特征表所在 Catalog
    schema_name="<offline_schema_name>",            # 离线特征表所在 Schema
    table_name="<offline_table_name>",              # 离线特征表名
    online_db_name="<online_schema_name>",          # 在线表所在的库
```

```
    online_table_name="<online_table_name>",          # 在线表名
    # trigger 不传则默认每天 0 点（Asia/Shanghai）全量同步一次
)
```

> ⓘ **注意**
>
> 1. 调用 `publish_table` 要求工作空间下至少有：① 一个运行中的数据计算资源组；② 一个 PostgreSQL 类型连接。否则会抛 `RuntimeError`（错误信息中带 `request_id` 便于定位）。
> 2. 如需更精细的调度（如每小时整点），可通过 `trigger` 参数自定义同步周期。
>
> **补充说明（特定版本、环境）**：部分 SDK 版本或工作空间配置下，服务端会要求显式指定在线表所属的 Catalog（对应服务端 `TargetPath.CatalogName`）。若调用时报错 `TargetPath.CatalogName 不能为空`，可在上述参数基础上补传 `online_catalog_name="<online_catalog_name>"`（具体名称参考所在工作空间已配置的在线 Catalog）。

删除在线特征表

当在线特征表不再需要时，可通过 SDK 删除：

```
fe.drop_online_table(name="<catalog_name>.<schema_name>.<table_name>")
```

> ⓘ **注意**
>
> 删除在线特征表会同时删除 PostgreSQL 中的在线表数据及对应的同步工作流，该操作不可恢复。删除前请确认没有在线推理服务正在依赖该在线表，否则服务会在下次请求时报特征查找失败。离线特征表不受影响。

命名与组织规范

统一的命名与组织规范有助于团队协作、特征复用与权限治理。建议遵循以下约定：

层级	命名规则	示例
Catalog	`<业务域>`	`showcase` / `risk` / `recommend`
Schema（特征数据库）	`<团队>_<场景>_db`	`growth_database` / `risk_churn_db`
特征表	`<实体>_<用途>_<版本>`	`orders_purchase_delivery_feature_v2`

此外建议：

- 引用特征表统一使用三段式全限定名 `catalog.schema.table`，避免依赖默认 Catalog 带来的副作用，迁移到不同工作空间时也无需改代码。
- 标签列建议用动词短语命名（如 `order_purchase_delivery_time`），避免与特征列重名。

参数说明

`create_table` 关键参数

参数	说明	是否必填	默认值
`name`	特征表的全限定名，格式 `catalog.schema.table`	是	—
`primary_keys`	主键列名，类型 `Union[str, List[str]]`，单键传字符串、复合主键传列表	是	—
`timestamp_key`	时间戳键列名（单数）	否	`None`
`engine_type`	底层表引擎，`EngineTypes.ICEBERG_ENGINE` 或 `EngineTypes.HIVE_ENGINE`	是	—
`df`	初始数据 Spark DataFrame，传入后可自动推断 `schema` 并写入首批数据	否	`None`
`schema`	表结构（PySpark `StructType`）。当 `df=None` 时必填	否	`None`
`catalog_name`	Catalog 名称（`name` 未使用全限定名时指定）	否	`None`
`database_name`	Schema / 数据库名称（`name` 未使用全限定名时指定）	否	`None`
`partition_columns`	分区列名列表	否	`None`
`tags`	表级标签字典	否	`None`
`description`	表描述，会显示在特征管理列表与 Catalog 详情页	否	空

`write_table` 关键参数

参数	说明	是否必填	默认值
`name`	目标特征表的全限定名	是	—
`df`	写入数据，Spark DataFrame（支持批 / 流式 DataFrame）	是	—
`mode`	写入模式，`append` / `overwrite`	否	`append`
`checkpoint_location`	流式写入时的 checkpoint 路径	流式写入必填	`None`
`trigger`	流式写入触发器配置	否	`None`

`get_table` / `read_table` 参数

参数	说明	是否必填	默认值
`name`	特征表名（可为全限定名，或配合 `catalog_name` / `database_name` 使用）	是	—
`database_name`	Schema / 数据库名称	否	`None`
`catalog_name`	Catalog 名称	否	`None`

`get_table` 与 `create_table` 返回的均为 `FeatureTable` 实体（`wedata.common.entities.feature_table.FeatureTable`），由接口构造返回，**不应由调用方直接实例化**。字段包括：`name`、`table_id`、`description`、`primary_keys`、`partition_columns`、`features`、`timestamp_keys`、`creation_timestamp`、`online_stores`、`tags`。

> ⓘ **注意**
>
> `tags` 字段为惰性加载，未加载时直接访问会抛 `ValueError`。建议先确认表已正确返回后再读取标签。

`set_feature_table_tag` / `delete_feature_table_tag` 参数

参数	说明	是否必填	默认值

`name`	特征表名（可为全限定名，或配合 `catalog_name` / `database_name` 使用）	是	—
`database_name`	Schema / 数据库名称	否	`None`
`catalog_name`	Catalog 名称	否	`None`
`key`	标签键	是	`""`
`value`	标签值（仅 `set_feature_table_tag`）	是	`""`

`FeatureLookup` 关键参数

参数	说明	是否必填	默认值
`table_name`	特征表名，**强烈推荐使用三段式全限定名** `catalog.schema.table`	是	—
`lookup_key`	用于匹配的主键列名（可复合主键）	是	—
`feature_names`	需要查找的特征列名列表；不传则查找全部非主键列	否	全部非主键列
`timestamp_lookup_key`	用于按时间点回溯的时间戳列名	否	`None`
`lookback_window`	时间点回溯的时间窗，类型 `Optional[datetime.timedelta]`，配合 `timestamp_lookup_key` 使用	否	`None`
`rename_outputs`	输出字段重命名映射，避免多个特征表同名列冲突	否	`None`
`is_online`	是否为在线查找场景	否	`None`
`online_config`	在线查找的相关配置	否	`None`

提示

> `FeatureLookup` 共 9 个参数，其中包含 2 个已弃用参数（`feature_name`、`output_name`，请改用 `feature_names`、`rename_outputs`）。

`create_training_set` 关键参数

参数	说明	是否必填	默认值
`df`	主键 DataFrame，包含主键、时间戳、训练标签	是	—
`feature_lookups`	`FeatureLookup` 列表（与 `feature_spec` 互斥）	否	—
`feature_spec`	已有的特征配置（与 `feature_lookups` 互斥，二者择一）	否	—
`label`	训练标签列名	是	—
`exclude_columns`	不参与训练的列名列表	否	`[]`
`database_name`	Schema / 数据库名称	否	`None`
`catalog_name`	Catalog 名称	否	`None`

`log_model` 关键参数

参数	类型	说明	是否必填	默认值
`model`	`Any`	训练好的模型对象（必须使用 `training_set.load_df()` 返回的 DataFrame 训练）	是	—
`artifact_path`	`str`	MLflow artifact 路径（如 `"model"`）	是	—
`flavor`	`ModuleType`	MLflow flavor 模块，如 `mlflow.sklearn`、`mlflow.xgboost`	是	—
`training_se`	`Optional[Tr`	训练集对象；不传则不携带 `FeatureSpec`（推理时不会自动 lookup）	否	`Non`

t	ainingSet]			e
registered_model_name	Optional[str]	模型注册名；建议用三段式 catalog.schema.model_name	否	None
model_registry_uri	Optional[str]	Model Registry URI	否	None
await_registration_for	int	等待注册完成的秒数	否	300
infer_input_example	bool	是否自动从 training_set.load_df() 生成 input_example	否	False
**kwargs	—	透传给对应 flavor 的 log_model	否	—

> ⓘ **注意**
> 模型 signature 与 input_example 由 training_set.feature_spec 自动生成，**无法通过参数覆盖**，以保证 score_batch 时 lookup 列契约一致。请勿再手动 infer_signature(...) 或手动传 input_example=...。

score_batch 关键参数

参数	说明	是否必填	默认值
model_uri	MLflow 模型 URI，如 runs:/<run_id>/model、models:/<name>/<version> 或 models:/<name>/<stage>	是	—
df	主键 DataFrame，须包含 feature_spec.yaml 中所有 lookup_key；不能含 prediction 列，不能是流式 DataFrame	是	—
result_type	预测结果列的数据类型，取值同 mlflow.pyfunc.spark_udf 的 result_type	否	"double"
timestamp_key	推理 DataFrame 中的时间戳列名；不传时若模型训练含 timestamp_lookup_key，SDK 会注入 current_timestamp() 取最新特征	否	None

publish_table 参数

参数	说明	是否必	默认

		填	值
`catalog_name`	离线特征表所在 Catalog	是	—
`schema_name`	离线特征表所在 Schema / 数据库	是	—
`table_name`	离线特征表名	是	—
`online_db_name`	在线表所在的库	是	—
`online_table_name`	在线表名	是	—
`trigger`	同步调度配置；不传则默认每天 0 点（Asia/Shanghai）全量同步一次	否	`None`

> **提示**
> 部分 SDK 版本、工作空间配置下，服务端会额外要求 `online_catalog_name`（在线表所属的 Catalog，对应服务端 `TargetPath.CatalogName`）。仅在报错 `TargetPath.CatalogName` **不能为空** 时按需补传，取值参考所在工作空间已配置的在线 Catalog。

完整的调度参数详见 调度与触发器。

`drop_table` / `drop_online_table` 参数

参数	说明	是否必填	默认值
`name`	特征表名（可为全限定名，或配合 `catalog_name` / `database_name` 使用）	是	—
`database_name`	Schema / 数据库名称	否	`None`
`catalog_name`	Catalog 名称	否	`None`

相关文档

- 界面操作
- 最佳实践与异常排查
- 模型列表

- 部署在线推理服务
- 全局数据源配置
- 创建计算资源

最佳实践与异常排查

最近更新时间：2026-09-11 20:42:31

本文汇总在 DataBuddy 特征管理全链路（建表 → 训练 → 推理 → 在线发布）中的关键最佳实践，以及最常见的异常类型、错误速查表与排错清单，帮助您规避高频陷阱并快速定位问题。

最佳实践

1. 数据类型治理：写入前统一 cast double

特征工程 SDK 不识别 `decimal(p, s)` 类型。若特征表中存在 `decimal` 列，`fe.log_model` 记录模型时将无法推断模型签名（`signature`），进而引发在线、离线推理的类型不匹配，甚至 MLmodel 文件中没有 signature 段。
做法： 在写入特征表之前，把所有 `decimal` 列统一 `cast("double")`，并将该处理沉淀为可复用的工具函数，在所有建表、写入管道中统一调用。时间戳列应保留为时间类型，**不要** 转为字符串，否则无法用作时间戳键。

2. Pipeline 铁律：预处理必须封装进 sklearn.Pipeline

所有预处理（`OneHotEncoder`、`StandardScaler`、自定义变换等）**必须** 封装进 `sklearn.Pipeline`，把整条 Pipeline 作为 `model` 传给 `fe.log_model`。
禁止 在 Pipeline 外部用 `pd.get_dummies` / 手动 `StandardScaler.fit_transform` 后再 `model.fit(...)`：外部预处理步骤不会被序列化进 MLmodel，推理时 SDK 从特征表 lookup 出来的是原始 string 列，喂给只认 OHE 后列名的裸模型，轻则报 `Feature names unseen at fit time`，重则所有样本静默输出训练集均值。

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

preprocessor = ColumnTransformer(transformers=[
    ("cat", OneHotEncoder(handle_unknown="ignore"), ["
<categorical_feature>"]),
    ("num", StandardScaler(), ["<numeric_feature>"]),
])
model = Pipeline(steps=[("preprocessor", preprocessor), ("clf",
<estimator>)])
```

3. 模型记录：必须用 fe.log_model 且传 training_set

- 必须用 `fe.log_model` 而非 `mlflow.<flavor>.log_model`：只有前者会把 `FeatureSpec` 打包进 MLmodel，否则 `score_batch` 找不到特征来源。

- 必须传 `training_set`，并开启 `infer_input_example=True`：模型签名与 `input_example` 由 SDK 自动推断，是部署在线推理服务的前置条件。
- 不要 再手动 `infer_signature(...)` 或手动传 `input_example=...`，否则会与 `feature_spec` 的原始列错位、污染契约。

4. Point-in-time 一致性：显式带上时间戳列

训练时声明了 `timestamp_lookup_key` 的场景下，`score_batch` 的输入 `df` 强烈建议显式带上同名时间戳列 以保证 point-in-time 一致性。若不带，SDK 会自动注入 `current_timestamp()` 取当前最新可见的特征，等同于用今天的特征预测历史样本，语义不严谨。
如需限制回溯时间范围，在 `FeatureLookup` 中配合 `lookback_window`（非负 `datetime.timedelta`）使用，且必须与 `timestamp_lookup_key` 同时出现。

5. 命名规范：统一使用三段式全限定名

引用特征表、注册模型统一使用三段式全限定名 `catalog.schema.table` / `catalog.schema.model_name`，避免依赖默认 Catalog 带来的副作用，迁移到不同工作空间时也无需改代码。

异常分类

异常类型	触发场景	处理建议
`ImportError`	运行环境中的 MLflow 版本低于 SDK 要求	安装带 `mlflow3` extra 的 SDK，或升级 MLflow 至要求版本
`EnvironmentError`（`wedata.common.utils.env_utils`）	必需环境变量未设置（项目 ID、地域、引擎名等）	在 DataBuddy Notebook 内核中运行，或在本地配置好对应环境变量
`KeyError`	临时密钥环境变量缺失	在 DataBuddy Notebook 内核中运行，或本地配置好对应环境变量
`ValueError`	业务参数校验失败	按错误信息修正参数
`RuntimeError`	底层云 API、计算资源、连接调用失败	查看错误信息中的 `request_id`，结合 DataBuddy 控制台排查
`TencentCloudSDKException`	调用云 API 时返回错误（鉴权失败、资源不存在、限流等）	根据 `code` / `message` / `request_id` 联系平台支持
`IllegalArgumentException` / `Exception`（PySpark）	Spark SQL 执行失败（DDL、写入、读取）	排查 Spark 日志与表权限

ValueError 常见错误速查

`ValueError` 多由参数不合法引起，可对照错误信息快速修正：

错误信息（节选）	场景	修复方式
`name must be a non-empty string`	表名、库名传入空字符串	检查参数
`name '...' contains too many segments`	表名段数超过 3	使用 `catalog.schema.table` 三段式或更短形式
`database_name '...' contains too many segments`	数据库名段数超过 2	使用 `catalog.schema` 二段式或单段名
`catalog_name '...' conflicts with catalog '...'`	显式 `catalog_name` 与多段名解析结果不一致	二者保持一致，或仅传一种
`Primary keys have duplicates: [...]`	`create_table` 主键重复	去重；复合主键传列表而非字符串
`Engine type ... is not supported`	`engine_type` 不在受支持的枚举值内	使用 `EngineTypes.ICEBERG_ENGINE` 或 `EngineTypes.HIVE_ENGINE`
`Invalid write mode '...'`	`write_table` 的 `mode` 取值非法	使用 `append` 或 `overwrite`
`Streaming write requires checkpoint_location parameter`	流式写入未传 `checkpoint_location`	显式指定 checkpoint 路径
`table '...' not exists`	读 / 写 / 删 / 打标的表不存在	先 `create_table` 或检查 catalog / database / 表名
`properties must be a non-empty dictionary`	设置标签时未提供有效 key / value	传入非空 `key`、`value`
`All properties are reserved and cannot be set: [...]`	标签 key 命中系统保留字段（如主键、时间戳键等）	换一个非保留 key

`Either 'feature_spec' or 'feature_lookups' must be provided, but not both.`	`create_training_set` 同时传或都不传两者	二者择一
`Invalid feature_spec format`	`feature_spec` 不是三段式	改成 `catalog.schema.spec_name`
`FeatureLookup must specify a table_name`	`FeatureLookup.table_name` 为空	显式指定表名
`When table_name '...' is a single-level name, both 'catalog_name' and 'database_name' must be provided.`	`FeatureLookup` 用了单段表名但未补 catalog / database	改用三段式表名（推荐），或显式传 `catalog_name` + `database_name`
`When table_name '...' is a two-level name (schema.table), 'catalog_name' must be provided.`	`FeatureLookup` 用了两段表名但未传 catalog	补 `catalog_name` 或改用三段式
`Unexpected lookback_window value: ...`	`lookback_window` 未配套 `timestamp_lookup_key`	同时设置 `timestamp_lookup_key`
`only non-negative datetime.timedelta allowed.`	`lookback_window` 不是 `timedelta` 或为负	使用非负的 `datetime.timedelta(...)`
`Setting multiple timestamp lookup keys is not supported.`	`timestamp_lookup_key` 传入多元素列表	仅传单列名（`str`）或单元素列表
`cloud_secret_id is empty` / `cloud_secret_key is empty`	临时密钥未注入	检查鉴权环境变量

RuntimeError 常见错误速查

`RuntimeError` 多与计算资源、数据源连接或云 API 调用有关，错误信息中通常带 `request_id` 便于定位：

错误信息（节选）	场景	修复方式
`Failed to delete table '...'`	底层 `DROP TABLE` 失败	查看异常 cause、表锁 / 权限

`Failed to modify properties for table '...'`	修改表标签失败	查看异常 cause
`Failed to list compute resources: no compute resources found`	工作空间无运行中的计算资源组	在 DataBuddy 控制台创建 / 启用数据计算资源组
`Unable to get compute resource basic info`	资源组基础信息缺失	联系工作空间管理员
`Failed to list connections: no connections found`	发布在线表时工作空间无 PostgreSQL 类型连接	在控制台创建 PostgreSQL 类型连接
`Failed to create online table '...'`	`publish_table` 调用创建在线表失败	凭 `request_id` 联系平台支持
`Failed to drop online table`	`drop_online_table` 失败	同上
`TargetPath.CatalogName` 不能为空	特定版本 / 环境下 `publish_table` 缺少在线 Catalog 信息	补传 `online_catalog_name`（具体名称参考所在工作空间已配置的在线 Catalog）

全链路排错清单

下表按现象 → 根因 → 解决方法整理了从建表到在线发布全链路中最易踩的坑：

现象	根因	解决方法
MLmodel 文件中没有 signature 段	特征表存在 `decimal(p, s)` 列，SDK 类型映射不支持	写入前把所有 `decimal` 列 `cast("double")`
`log_model` 推断 signature 失败 / `score_batch` 列对不上	使用了原生 `mlflow.<flavor>.log_model`，或手动传了 `signature` / `input_example`	改用 `fe.log_model(..., training_set=..., infer_input_example=True)`
`ImportError: ... mlflow >= ...`	环境里的 MLflow 版本过低	重装带 `mlflow3` extra 的 SDK
`EnvironmentError` / `KeyError`	本地调试缺环境变量	在 DataBuddy Notebook 中运行，或本地手动配置对应环境变量

`When table_name '...' is a single-level name ...`	`FeatureLookup` 用了单段表名但没补 catalog / db	改三段式表名（推荐），或显式传 `catalog_name` + `database_name`
`Unexpected lookback_window value ...`	`lookback_window` 没有配套 `timestamp_lookup_key`	同时设置 `timestamp_lookup_key`
`only non-negative datetime.timedelta allowed.`	`lookback_window` 不是 `timedelta` 或为负数	使用非负的 `datetime.timedelta(days=N)`
`Failed to list compute resources: no compute resources found`	工作空间无运行中的计算资源组	在 DataBuddy 控制台启动一个数据计算资源组
`Failed to list connections: no connections found`	发布在线表时无 PostgreSQL 连接	在控制台先创建 PostgreSQL 类型连接
`TargetPath.CatalogName` 不能为空	特定版本 / 环境下 `publish_table` 缺少在线 Catalog 信息	补传 `online_catalog_name`（名称参考所在工作空间已配置的在线 Catalog）
`Primary keys have duplicates`	`create_table` 主键重复	去重；复合主键传列表而非字符串
`Streaming write requires checkpoint_location parameter`	流式 `write_table` 未传 checkpoint	显式传 `checkpoint_location`
`score_batch` 抛 schema 不匹配	推理 DataFrame 类型与训练时查出的特征类型不一致（典型：`decimal` vs `double`）	训练前统一 `cast`，所有特征表存 `double`
`score_batch` 抛 `Feature names unseen at fit time`，或所有样本预测值完全相同	训练时在 `Pipeline` 外裸调用了 `pd.get_dummies` / `StandardScaler` 等预处理，未被序列化进模型	用 `sklearn.Pipeline` + `ColumnTransformer` + `OneHotEncoder(handle_unknown="ignore")` 把预处理与模型一起作为 `model` 传给 `fe.log_model`

错误处理建议

在生产化调用中，建议对不同异常类型分别捕获，便于区分参数错误与资源、云 API 错误，并保留 `request_id` 以便排查：

```
from tencentcloud.common.exception.tencent_cloud_sdk_exception import
TencentCloudSDKException
from wedata.common.utils.env_utils import EnvironmentError as
WedataEnvError

try:
    fe.publish_table(
        catalog_name="<offline_catalog_name>",
        schema_name="<offline_schema_name>",
        table_name="<offline_table_name>",
        online_db_name="<online_schema_name>",
        online_table_name="<online_table_name>",
    )
except ValueError as e:
    # 业务参数错误，直接记录并返回
    logger.error("invalid params: %s", e)
except RuntimeError as e:
    # 资源 / 云 API 错误，错误信息中的 request_id 可用于联系支持
    logger.error("publish failed: %s", e)
except TencentCloudSDKException as e:
    logger.error("cloud api error code=%s msg=%s reqid=%s",
                 e.get_code(), e.get_message(), e.get_request_id())
except WedataEnvError as e:
    logger.error("env not ready: %s", e)
```

> **提示**
> 如遇文档未覆盖的接口或异常场景，请在工单系统提单，并附上 `request_id`、SDK 版本与错误堆栈，便于快速定位。

常见问题

Q: `score_batch` 提示找不到 `feature_spec.yaml` 怎么办?

`feature_spec.yaml` 是在 `fe.log_model` 时自动写入的。如果训练时使用的是原生 `mlflow.<flavor>.log_model` 而非 `fe.log_model`，则不会写入特征来源信息。请改用 `fe.log_model` 重新记录模型。

Q：发布模型为在线服务时报错存在未发布为在线表的离线特征表?

模型工件中引用到的所有离线特征表都必须先发布为在线特征表，在线服务才能创建。请回到对应的离线特征表，完成发布并等待同步任务首次执行成功后再重试（详见 界面操作 ）。

Q：在线特征数据看起来比离线表落后，是什么原因?

在线特征表的数据来自同步任务，时延取决于同步方式：**定时触发** 受调度周期限制；**持续运行** 实时性最高但仍存在任务执行间隔。如对实时性有较高要求，建议在持续运行模式下配置较小的执行间隔，并监控同步任务的运行状态与延迟。

相关文档

- 界面操作
- 特征工程 SDK

模型管理
什么是模型管理

最近更新时间：2026-09-11 20:42:31

模型管理（Models）是 DataBuddy 数据科学模块中机器学习模型的统一资产入口，基于 MLflow 与 Catalog 构建，承担模型从注册、版本控制、治理到部署的全生命周期管理职责，上承模型实验产生的训练结果，下接模型服务对外提供推理能力。
模型在 Catalog 中作为一类资产，与数据表（Table）、卷（Volume）共享同一套权限、标签、血缘体系，并以三段式命名 `catalog.schema.model_name` 唯一标识。

数据组织模型

模型与表、卷共享同一套四层数据组织模型：

层级	说明
Catalog	数据目录，跨工作空间可见，是模型注册的最外层容器，拥有明确的类型（Table / Model / Volume 等）。
Schema	Catalog 下的子命名空间，用于组织一组相关模型。
Model	已注册的模型资产，全名为 `catalog.schema.model_name`，包含一个或多个版本。
Model Version	模型的一个具体版本，对应一次训练产物，包含工件、指标、参数、血缘等完整元数据。

注册模型前，需先在 Catalog 中创建一个类型为 **Model** 的目录并在其下创建 Schema，详见 模型 Catalog 。

关键概念

理解模型管理，需要熟悉以下核心概念：

- **已注册模型（Registered Model）**：纳入 Catalog 治理的模型资产，拥有三段式全名与一条递增的版本序列。
- **模型版本（Model Version）**：模型的最小可部署单元，对应一次训练产物。每次注册版本号自动递增（1、2、3 …），不支持手动指定。
- **模型签名（Model Signature）**：模型输入、输出字段的定义。注册受管模型强制要求签名，缺失时注册会失败。
- **标签（Tags）**：作用于模型或版本的键值对，用于标记业务属性（如团队、业务线、框架）与生命周期阶段，并支持在列表中筛选。
- **血缘（Lineage）**：模型版本与上下游资产（特征表、Notebook、推理表、模型服务等）的关系，由 Catalog 统一记录。

相关文档

- 模型管理
- 模型实验
- 模型服务
- 模型 Catalog
- 术语表

模型列表

最近更新时间：2026-09-11 18:24:34

模型列表展示当前 Workspace 下所有已注册的机器学习模型，是模型资产的统一入口。您可以在此搜索、筛选模型，跳转到模型详情页。

前提条件

在进入模型列表并执行操作前，请确认：

- 已开通 DataBuddy，且当前账号已加入目标 Workspace。
- 当前账号对至少一个类型为 **Model** 的 Catalog 拥有 `USE CATALOG` 与 `USE SCHEMA` 权限。否则模型列表会展示对应的占位提示而无具体内容。
- 如需导入外部模型，账号需对目标 Catalog / Schema 拥有 `CREATE MODEL` 权限，并具备一个可用的作业集群。

进入模型列表

在 DataBuddy 控制台左侧导航栏点击 **数据科学** → **模型管理** ，即可进入模型列表页。

列表字段说明

字段	说明
名称	模型名称，格式为 `model_name` 。点击名称跳转到该模型在 Catalog 下的详情页。
Catalog	模型所属的 Catalog 名称。点击跳转到该 Catalog 详情页。
Schema	模型所属的 Schema 名称。点击跳转到对应 Schema 详情页。
修改时间	模型的最近一次元数据变更时间（含版本注册、标签变更等）。
负责人	模型创建者，可在 Catalog 中通过转交修改。
最新版本	模型的最新版本号
描述	模型描述，可在详情页进行修改

搜索与筛选

模型列表提供以下检索能力：

1.关键词搜索

在搜索框输入关键字，实时匹配模型名称，命中即展示该模型。搜索结果中，匹配关键词的部分高亮显示。

2.负责人筛选

搜索框右侧提供负责人下拉筛选器：

- 选项从当前用户可访问的模型中提取去重的负责人列表，默认选中全部。
- 选中某负责人后，列表仅显示该负责人的模型；关键词搜索与负责人筛选同时生效。

注册新模型

模型列表本身不提供**新建模型** 。模型通过 Autologging、MLflow API 或实验运行详情页 UI 注册后，自动出现在列表中。注册的三种方式、模型签名要求与版本递增规则详见 模型版本管理 。

导出模型

如需将受管模型导出到本地或外部环境，请在 Notebook 中通过 MLflow API 执行。在 DataBuddy Notebook 内核中，Registry 连接已自动配置，**无需手动设置 URI**：

```
import mlflow

# DataBuddy Notebook 内核已自动配置 Registry，直接下载即可
# model_name: 模型全限定名，格式为 <目录名>.<模式名>.<模型名>，如 catalog.schema.model
# model_version: 模型版本号（整数），如 1、2、3
mlflow.artifacts.download_artifacts(
    f"models:/{model_name}/{model_version}",
)
```

使用限制

模型列表仅展示当前 Workspace 下、当前用户拥有 `USE CATALOG` 与 `USE SCHEMA` 权限的模型；无权限的模型不展示。

常见问题

Q：在模型列表搜索不到刚刚注册的模型?
A：请确认：1. 您拥有该模型所属 Catalog / Schema 的 `USE` 权限；2. 模型注册流程已完整结束（注册操作是异步的，刷新列表查看）；3. 搜索是否落在错误的 Workspace。
Q：从列表删除模型与从 Catalog 删除模型有区别吗?
A：没有。模型在 DataBuddy 中始终是 Catalog 中的一项资产，删除操作直接作用于 Catalog 元数据，且会级联删除该模型所有版本，操作不可恢复。

相关文档

- 模型版本管理
- 模型标签与生命周期
- 模型实验
- 模型服务

模型版本管理

最近更新时间：2026-09-11 18:24:32

模型版本是模型资产的最小可部署单元，对应一次具体的训练产物。DataBuddy 模型版本管理在 Catalog 中以 `catalog.schema.model_name/<version>` 的方式组织，提供版本的注册、查看、复制、删除以及完整的元数据展示能力。

前提条件

- 已拥有目标 Catalog 与 Schema 的 `USE CATALOG`、`USE SCHEMA` 权限。
- 注册新模型需要 `CREATE MODEL` 权限；为已有模型注册新版本需要 `CREATE MODEL VERSION` 权限。
- 所注册的模型必须包含模型签名（Model Signature）：使用 `mlflow.<flavor>.log_model()` 时通过 `signature=` 参数显式传入；使用特征工程 SDK 的 `fe.log_model()` 时由 `training_set.feature_spec` 自动推断。请勿依赖 Autologging 的自动签名推断——它对部分 flavor（如自定义 PyFunc）或含 `decimal` 类型的训练数据**不保证生成签名**，会导致注册失败。
- 删除版本需要对应模型的 `DELETE` 权限。

> ⓘ **注意**
> 模型未携带签名时注册会失败并提示，且签名缺失会导致后续 `score_batch` 与在线推理的特征自动补全失败。请在训练或日志记录阶段确认签名已写入后再注册。

注册新版本

方式一：特征工程 SDK 注册

如果模型在训练时通过 `create_training_set` 拼接了特征表，必须使用特征工程 SDK 的 `fe.log_model()` 注册。它是唯一会写入 `feature_spec.yaml`、支持特征消费链路（`score_batch` 自动查找、在线推理自动补全）的方式：

```
from wedata.feature_engineering.client import FeatureEngineeringClient

fe = FeatureEngineeringClient(spark)

fe.log_model(
    model=model,
    artifact_path="model",
    flavor=mlflow.sklearn,
    training_set=training_set,                    # 携带特征来源，写入
feature_spec.yaml
```

```
    registered_model_name="catalog.schema.model_name",
    infer_input_example=True,
)
```

`fe.log_model()` 会在记录模型的同时完成版本注册，并把模型签名与 `input_example` 依据 `training_set.feature_spec` 自动推断。完整用法详见 特征工程 SDK 。

方式二：UI 注册

注册入口位于实验运行详情页：

1. 进入 **数据科学** → **模型实验** ，打开目标实验运行（Run）的详情页。
2. 在右上角点击 **注册模型** 。
3. 在弹窗中选择 **已有模型新版本** ，下拉选择目标模型；版本号会基于该模型当前最大版本自动递增。
4. 点击 **注册模型** 完成注册。

如需新建模型，请在弹窗中选择 **新模型** ，并按 `catalog.schema.modelname` 规范输入名称。模型重名时，会提示“当前模型已被注册，请注册该模型新版本”。

版本详情

进入模型详情页有两种入口：

- 在 **模型列表** 点击模型名称进入模型详情，再在版本列表点击具体版本号。
- 在 **数据目录（Catalog）** 中按 `catalog → schema → model → version` 路径进入。

1. 概览

在 **概览** Tab 顶部可对当前版本进行编辑：

- **添加描述** ：补充版本说明，便于团队识别版本用途。
- **添加负责人** ：为版本指定负责人。
- **添加标签** ：为版本打标，便于检索与生命周期管理（详见 模型标签与生命周期）。

概览 Tab 展示该版本来源实验运行记录的指标与训练参数。其中 **源运行** 可点击跳转到对应运行详情。**活动日志** 中可按时间倒序记录该版本的注册、复制、删除等关键事件，用于审计追溯；后续将扩展至模型部署、自动部署作业等场景。还可查询模型指标、参数、版本签名等。

2. 血缘

血缘展示模型版本与上下游资产的关系，支持两种视图：

- **列表模式** ：展示一层上下游列表。
  - 上游节点类型：数据表、特征表（ `Table` ），过程类型：Notebook、Studio。
  - 下游节点类型：数据表、推理表（ `Table` ），过程类型：模型服务（ `Process` ）。
- **血缘图模式** ：以图谱形式展示完整链路，节点支持点击跳转：
  - 模型节点支持切换不同版本；

- 模型服务节点展示模型名称、创建人、注册时间、版本创建时间，点击服务名称跳转到模型服务详情；
- 推理表节点信息与普通数据表保持一致，点击进入表详情。

血缘数据由 Catalog 统一查询接口提供，模型注册与服务部署时由各模块上报。

3. 模型工件

工件以树形结构展示模型相关文件（如 `MLmodel`、`conda.yaml`、`python_env.yaml`、模型权重等）：

- 点击文件夹展开下级；
- 点击单个文件，右侧显示文件内容预览；
- 文件支持下载到本地。

4. 复制版本

跨模型复制版本可用于将已有版本快速分发到 Catalog 或 Schema 下，适合多环境（开发、生产）或跨项目共享场景。

1. 在版本详情页右上角点击 **复制此版本** 。
2. 在弹窗中选择目标模型。可直接输入关键字过滤。目标模型可以是源模型自身（即在同一模型下复制出新版本），也可以是同 Catalog / Schema 下的其他模型。
3. 点击 **复制** 。

5. 删除版本

在版本详情页右上角点击 **删除此版本**， ** 点击 确认删除** 按钮即可删除。

> **警告**
> 删除版本不可恢复。如该版本已被在线推理服务引用，请先解除部署关系再删除，避免线上服务受影响。

通过代码管理版本

DataBuddy 模型版本管理接口与 MLflow 标准 API 兼容，常用接口：

API	说明
`mlflow.register_model(model_uri, name)`	注册模型或新版本
`mlflow.search_model_versions(filter_string)`	按过滤条件搜索版本
`mlflow.search_registered_models(filter_string)`	按过滤条件搜索已注册模型
`MlflowClient.delete_model_version(name, version)`	删除模型版本

> **注意**

> 删除模型版本需要导入 `MlflowClient`。

在 DataBuddy Notebook 中运行时，内核会自动注入临时密钥并完成 Tracking / Registry 的连接配置，无需手动设置 URI，直接调用上述接口即可：

```
import mlflow

# DataBuddy Notebook 内核已自动配置 Tracking / Registry，直接使用即可
mlflow.register_model(model_uri="runs:/<run_id>/model",
name="catalog.schema.model_name")
```

> **提示**
> 如需在 DataBuddy Notebook 之外的环境（如本地）使用，需手动配置指向 DataBuddy 的 Tracking / Registry URI 及认证信息。具体取值请参考平台连接说明。

使用限制

- 同一模型下版本号自动递增，不支持手动指定版本号。
- 跨 Catalog 复制版本仅在当前用户对源版本有读取权限、对目标模型有 `CREATE MODEL VERSION` 权限时可用。
- 删除已存在部署服务引用的版本前，请先在模型服务侧解绑该版本，避免运行时报错。
- 工件展示与下载依赖底层 Catalog 与对象存储能力，超大工件可能存在加载延迟。

常见问题

Q：版本注册成功但 指标 Tab 为空?
A：指标来源于源实验运行（Source Run）。请确认源 Run 是否记录了指标；如使用 `mlflow.register_model(model_uri=...)` 时 `model_uri` 来自外部，未关联实验 Run，指标会为空。
Q：血缘图中看不到下游模型服务?
A：模型服务下游血缘需要部署侧上报。请确认该版本已通过 模型服务 部署，并刷新血缘视图。
Q：复制版本后能否回滚到源版本?
A：可以。复制不会修改源版本，源版本与新版本完全独立；如需回滚，可在模型服务中将部署目标改回源版本。

相关文档

- 模型列表
- 模型标签与生命周期
- 模型实验
- 模型服务
- 数据血缘

模型服务
什么是模型服务

最近更新时间：2026-09-11 20:42:31

模型服务把一个模型版本封装为长期运行的在线推理端点。一次完整的服务链路包括：

- 输入：从 Catalog 的注册模型中选择已有的模型版本进行部署。
- 运行：平台按量计费提供算力，并按弹性策略自动调整副本数。
- 调用：业务系统通过 HTTP 接口、携带鉴权 Token 调用服务。
- 沉淀：请求与响应 payload 自动落库到 Catalog 中的推理表，运行指标与告警接入工作空间的通知渠道。

关键概念

理解模型服务，需要熟悉以下核心概念：

概念	说明
服务（Endpoint）	一个对外暴露的推理端点，对应稳定的服务名称与调用地址。一个服务下可挂载多个版本。
服务版本（Served Entity）	服务下的一次具体部署，对应「某个模型 + 某个版本 + 一组资源配置」。同一服务内可并存多个版本，按流量比例分发请求。
副本（Replica）	同一服务版本的运行实例。副本数越多吞吐越高、可用性越好，支持手动 / 定时 / 自动 / 组合调节。
计算资源	服务运行时占用的算力，支持平台按量计费。资源规格决定单副本的 CPU / 内存 / GPU 容量。
流量比	多版本并存时按比例（如 80 / 20）将请求分发到不同版本，便于灰度发布与 A/B 测试。
推理表（Inference Table）	由 Catalog 管理的 Delta 表，自动收集请求与响应 payload，作为数据漂移、效果与公平性监控的事实表。
鉴权 Token	服务启用鉴权后，调用方需在请求 Header 中携带的 Bearer Token，由平台生成与轮换。

相关文档

- 模型服务
- 模型管理
- 数据质量概述

- 术语表

资源配置与自动伸缩

最近更新时间：2026-09-11 18:24:32

模型服务的运行成本与可用性主要取决于资源配置。本文介绍如何为服务版本选择计算资源、设置副本数量，以及通过控制台或 SDK 调节服务的运行状态。

前提条件

- 已创建模型服务并完成基础信息与版本配置。新建服务的流程详见 部署在线推理服务。
- 当前账号在工作空间内具备模型服务的编辑权限。

选择计算资源

新建或编辑服务版本时，**资源来源** 选择 **平台按量计费**（Serverless）。该模式下：

- 平台按副本运行时长计费，无需预购机器资源组。
- 资源池由平台统一调度，可按请求量动态拉起或回收副本。
- 服务停止后立即停止计费。

资源规格 需要根据模型的算力需求选择。常见档位示例：

类型	典型规格	适用模型
CPU	多档 CPU 规格	传统机器学习模型（树模型、线性模型、轻量级推荐模型）。
GPU	含 GPU 卡的规格	深度学习模型（图像识别、NLP、向量检索）。

> **提示**
> 单副本规格与副本数量共同决定服务容量。建议先以 1 副本、中等规格上线压测，再依据 QPS / 延迟指标横向扩展。

设置副本数量

在版本配置或服务详情页输入目标**副本数量**，平台立即扩缩到指定数量。多副本可提升服务可用性，适合临时压测、紧急扩容与稳定生产服务。

更新服务

服务运行后，在服务详情页单击**编辑** 进入**更新服务**，调整资源与运行参数：

- **可修改**：服务描述、资源规格、请求限流、副本数量、副本调节、调度策略、自动停止等配置项。
- **不可修改**：服务名称（页面中置灰）。
- **更换模型版本**：
  - 控制台：**更新服务** 操作不修改已部署的模型，更换模型版本需通过服务详情页的**新增版本** 完成。

- SDK：`update_deployment` 支持通过 `model_uri` 参数更换模型版本。

页面右侧 **配置概览** 实时汇总本次配置。

> **提示**
> 修改资源规格后副本的具体重启行为以平台实际实现为准。

通过 SDK 调节资源与运行状态

除控制台外，也可通过 MLflow Deployments SDK 在 Notebook / Python 任务中调节副本数（`config={"replicas": N}`）与启停服务（`service_action` 取值 `STOP` / `RESUME` / `SCALE`）。具体调用方式与注意事项详见 部署在线推理服务 的更新、扩缩容与版本切换。

常见问题

Q：副本数为 0 时服务还可被调用吗?
A：不可以。副本数为 0 时服务进入已停止状态，调用地址返回失败。如需暂时停止服务并停止计费，请在服务详情页单击 **停止**，或通过 SDK 调用 `update_deployment(config={"service_action": "STOP"})`。

相关文档

- 模型服务
- 部署在线推理服务（REST API）
- 推理监控与日志

推理监控与日志

最近更新时间：2026-09-11 18:24:33

模型服务运行后需要持续观测调用量、延迟、错误率与模型效果。DataBuddy 提供\运行指标、实时日志、推理表\三类能力，覆盖从在线问题排查到离线模型质量监控的全链路。

前提条件

- 已创建并运行模型服务。基础流程详见 部署在线推理服务（REST API）。
- 启用推理表前，需对目标 Catalog / Schema 拥有 `Use Catalog` 、 `Use Schema` 、 `Create Table` 权限。

查看运行指标

平台提供两处监控视图，分别位于服务与版本两级页面：
调用监控（服务详情页 > 服务版本下的页签）：按时间范围（近 1 小时、近 24 小时、近 7 天，或自定义起止时间）按调用方式（如 http）展示调用统计（次）。

指标	说明
接收请求数	服务收到的请求总数。
成功请求数	正常返回的请求数。
失败请求数	返回错误的请求数。
被限制请求数	触发请求限流被拒绝的请求数。

监控（版本详情页 > 实例列表下的页签）：按时间范围与 **时间粒度**（全部 / 1 分钟等）、按实例查看流量信息。

类型	指标	说明
流量信息	网络流量（MBytes）	服务实例的网络数据传输下行（入口）流量。反映模型服务的通信负载，突发流量可能预示请求量激增或异常调用，可结合 QPS 一起分析。
流量信息	QPS（次 / 秒）	每秒处理的请求数（Query Per Second），衡量服务吞吐量的核心指标。
流量信息	QPS 限流（次 / 秒）	主动限制每秒最大请求数的机制，超过阈值的请求会被拒绝。
流量信息	并发请求数	同一时刻正在处理的请求数量（包括等待和计算中的请求）。

资源信息	CPU 使用率（%）	服务实例的 CPU 资源占用百分比。
资源信息	MEM 使用率（%）	服务实例的内存占用百分比。
资源信息	显存使用率（%）	服务实例的 GPU 显存使用占用百分比。
资源信息	GPU 使用率（%）	服务实例的 GPU 计算单元负载百分比。
实例信息	实例数量	当前配置的服务实例总数（包括启动中、运行中、异常等所有状态）。
实例信息	运行中实例数量	实际正常处理请求的实例数。

查看实例日志

服务创建时如已开启 **服务日志投递**（日志服务），即可在 **版本详情页 > 实例列表 > 日志** 页签查看实例日志。

1. 选择 **实例名称** 与 **时间范围**（如近 24 小时，或自定义起止时间）。
2. 可开启 **自动刷新** 持续拉取最新日志，开启 **日志换行** 优化长日志展示。
3. 在搜索框按 **完整单词** 检索关键字（如错误码、Trace ID）。

实例日志主要用于排查系统级问题（连接超时、模型加载失败、内存溢出）。如需基于请求、响应做特征偏移、效果监控等深度分析，请使用推理表能力。

> **提示**
> 日志服务默认存储 15 天，长期分析建议同时启用推理表，将明细持久化到 Catalog。

通过 SDK 查询实例日志与重启实例

除控制台外，也可通过 MLflow Deployments SDK 在 Notebook / Python 任务中按时间区间分页拉取实例日志、重启异常实例（SDK 接入方式详见 部署在线推理服务）。

按时间区间分页拉取实例日志：

```
import datetime as dt

end   = dt.datetime.now(dt.timezone(dt.timedelta(hours=8)))
start = end - dt.timedelta(hours=1)

ctx = None
```

```
while True:
    resp = client.get_pod_logs(
        service_id=service_id,
        pod_name=f"{service_id}-*",      # 支持通配符匹配该服务下的实例
        limit=200,                        # 单次条数建议 ≤ 500
        start_time=start.isoformat(),     # ISO 8601，务必带时区（如
+08:00）
        end_time=end.isoformat(),
        context=ctx,                      # 翻页上下文，来自上次响应
    )
    for entry in resp["logs"]:
        print(entry.get("timestamp"), entry.get("pod_name"),
entry.get("message"))
    ctx = resp["context"]
    if not ctx:                           # 空字符串表示已到末页
        break
```

重启（重建）指定实例，用于应对内存溢出、健康检查持续失败等异常：

```
resp = client.restart_pod(service_id=service_id, pod_name="ms-xxxxxx-
1abc-xxxx")
print(resp["request_id"])
```

> ⓘ **注意**
>
> - **日志启用前提：服务创建时已开启日志投递（`log_enable=True`）；否则 `get_pod_logs` 返回为空。**
> - **时间参数必须是 ISO 8601 含时区（如 `+08:00`），直接用本地无时区时间易因时区错位漏取日志。**
> - **`restart_pod` 不支持通配符，必须传具体实例名，可从 `get_pod_logs` 返回的 `pod_name` 字段获取。**

查看服务事件

进入 版本详情页 > 实例列表 > 事件 页签，可查看实例底层资源的运行事件，用于排查副本拉起失败、健康检查异常等问题。事件列表字段：

字段	说明
首次出现时间 / 最后出现时	该事件首次与最近一次发生的时间。

间	
级别	`Normal`（正常）或 `Warning`（告警）。
资源类型	触发事件的底层资源类型，可按类型筛选，包括 Pod、Deployment、ReplicaSet、Service、Ingress、Endpoints、HorizontalPodAutoscaler、DistDeployment、PyTorchJob 等。
资源名称	触发事件的具体资源名称（如实例名 `ms-xxxxxx-1-xxxx`）。
详细描述	事件详情，如 `Readiness probe failed`、`pod group is ready` 等。
出现次数	该事件累计发生的次数。

可开启 **自动刷新** 持续拉取最新事件。

> **提示**
> 事件仅保留最近 15 天内的记录，请尽快查阅。

事件用于排查\突然变化\类问题（副本异常退出、健康检查持续失败、扩缩容未生效等），是定位实例级故障的首选入口。

服务日志投递及推理表收集

支持将推理服务日志投递至指定的 CLS 中，并且支持开启推理表（Inference Table）收集，将服务的请求 payload 与响应 payload 实时写入 Catalog 中的 Delta 表，作为后续质量监控的事实数据。

在新建服务时启用

在新建服务的 **高级配置 > 服务日志投递** 处打开开关后，配置 CLS 投递：

参数	说明	是否必填	默认值
日志集	CLS 的日志集，用于投递服务日志。	是	—
日志主题	CLS 的日志主题，用于投递服务日志。	是	—

然后开启**推理表收集**，依次配置：

参数	说明	是否必填	默认值
Catalog	推理表所在的 Catalog。下拉中只展示当前账号有 `Use Catalog` 权限的 Catalog。	是	—

Schema	推理表所在的 Schema。下拉中只展示有 `Use Schema` 与 `Create Table` 权限的 Schema。	是	—
表名前缀	推理表名称的前缀。填写后表名为 `{前缀}`；未填写时表名默认为 `{服务名称}_payload`。	否	—

平台会自动在指定 Catalog / Schema 下创建 Delta 表，并维护字段结构与样例数据。

在已运行服务上二次启用、关闭

服务详情页 → **编辑高级配置**：

- 新建时**未启用** → 编辑时可二次开启。
- 新建时**已启用** → 编辑时可二次关闭。关闭后历史推理表保留在 Catalog 中，仅停止后续写入。

推理表常见字段

平台默认包含的字段：

- 请求时间戳。
- 请求唯一 ID（Trace ID）。
- 输入 payload（JSON）。
- 输出 payload（JSON）。
- 模型名称、模型版本、服务版本。

具体字段会根据模型类型（分类、回归、深度学习）自动适配。进入 Catalog 中对应表的 **概览** 页可查看完整 Schema、示例数据与表属性。

基于推理表发起质量监控

推理表落库后，可在推理表详情页或数据质量模块中，以推理表为事实表创建质量监控任务（数据漂移、模型效果、公平性等长周期监控）。具体指标与配置方式详见 数据质量。

常见问题

Q：为什么推理表里没有数据?
A：依次确认：① 服务高级配置中推理表开关已打开；② 服务有真实流量进入；③ 当前账号对推理表所在 Catalog / Schema 拥有读权限。如以上都满足仍无数据，进入版本详情页 **日志** 页签查看是否有 Catalog 写入失败的报错。
Q：日志服务和推理表的区别?
A：日志服务是**实时排查工具**，存放系统级请求、响应日志、保留期较短，不适合长周期分析；推理表是**结构化离线事实表**，存于 Catalog，可被 SQL 查询、可做漂移监控、可作为重训样本来源。

相关文档

- 模型服务

- 部署在线推理服务（REST API）
- 资源配置与自动伸缩
- 数据质量
- 设置通知渠道