大数据智能体工作台 DataBuddy

数据工程

腾讯云

文档目录

数据工程
Studio 概述
Studio 基础
项目与文件组织

最近更新时间：2026-09-11 20:57:30

Studio 是 DataBuddy 的统一开发工作台。本节介绍 Studio 总览页面的整体界面布局，以及文件组织、检索、收藏与回收站等基础操作。

概述

Studio 在同一个界面中支持 Notebook、SQL、Python、Markdown、CSV 等多种文件类型的开发，覆盖 Notebook、SQL、Python 等所有开发场景，统一提供目录、新建、检索、收藏夹与回收站等交互。Studio 目录按用户视角组织，文件之间通过 ACL 权限控制访问范围。

前提条件

- 已加入目标工作空间，且具备 Studio 模块的访问权限。
- 工作空间已绑定计算资源（运行 Notebook / Python / SQL 文件时使用）。
- 如果需要使用 Git 文件夹，工作空间已配置 Git 连接，且当前用户已完成个人 Git 配置。

Studio 目录结构

Studio 目录按以下四个区域组织：

区域	说明	权限
Workspace	默认展示，存放所有成员在工作空间内创建的开发文件。	默认仅创建人和工作空间管理员可访问，可通过 ACL 授权给其他用户。
GitFolder	工作空间配置 Git 连接后展示，目录内容与远程 Git 仓库同步。	默认仅创建人可访问，不支持单独授权。
收藏夹	当前用户收藏的文件夹与文件，跨区域聚合展示。	收藏夹归当前用户所有。
回收站	当前用户从其他区域删除的对象，支持恢复或彻底删除。	回收站归当前用户所有。

> **提示**

> Notebook、SQL、Python、其他文件均可放在同一个本地文件夹下。文件类型由扩展名识别（如 `.ipynb`、`.sql`、`.py`、`.sh`、`.csv`、`.json`、`.yaml`、`.md`、`.txt`）。

操作步骤

步骤 1：浏览 Studio 总览页面

1. 点击 Studio 功能模块，默认打开「Workspace」目录。
2. 用户可以在左侧目录「Workspace」「GitFolder」「收藏夹」「回收站」之间切换，按层级下钻查看子文件夹。
3. 单击文件夹名称，在右边可以看到文件夹下的子文件夹和子文件。

步骤 2：新建文件夹与文件

1. 在页面右上角单击 **+ 创建** 。
2. 在弹出的菜单中选择新建对象：
   - **文件夹**：用于组织文件。
   - **Notebook**：以 `.ipynb` 结尾的开发文件。
   - **SQL**：以 `.sql` 结尾的脚本文件。
   - **Python**：选择文件，扩展名以 `.py` 结尾。
   - **文件**：手动输入扩展名，支持 `.sh`、`.csv`、`.json`、`.yaml`、`.md`、`.txt` 等。
   - **仪表盘**：跳转到「仪表盘」模块，并自动创建新仪表盘。
   - **模型实验**：跳转到「模型实验」模块，并打开新建机器学习实验页面。
1. 输入名称并选择父文件夹，单击 **确定** 。

步骤 3：上传文件

1. 在目标文件夹–更多操作中，选择 **上传文件** 。
2. 选择本地文件，单击 **打开** 。
3. 上传后的文件出现在目标文件夹下。`.zip`、`.pdf` 等不支持在线打开的文件类型，单击后则会提示「无法预览」。

步骤 4：检索文件

1. 在页面右上角单击 **检索** 。
2. 在顶部检索框中输入文件名关键字。检索范围默认为当前目录。
3. 检索结果将显示在检索框下方，可以进行进一步选择。

步骤 5：文件操作

对象	支持操作

文件夹	新建文件夹 / Notebook / SQL / 文件、复制路径、重命名、移动、上传、权限配置、删除、收藏 / 取消收藏
文件	复制路径、重命名、移动、复制、下载、权限配置、收藏 / 取消收藏、删除

- **复制路径**：返回完整路径，如 `/workspace_1/your_username/folder_1/file_1.ipynb`，可粘贴进 Notebook 中通过 `%run` 或 `import` 引用。
- **删除**：进入回收站；可在「回收站」恢复或彻底删除。

步骤 6：管理收藏夹与回收站

- **收藏夹**：列表展示对象名称、类型、收藏时间、原始路径，支持按类型筛选，支持取消收藏与批量取消。
- **回收站**：列表展示对象名称、类型、删除时间、原始路径，支持按类型筛选与按删除时间排序，支持恢复、批量恢复、彻底删除、批量删除。
  - 备注：恢复时如源路径不存在，需重新选择父文件夹；如目标路径已有同名文件，需重命名。

步骤 7：进入 Studio IDE

进入 Studio IDE 后，界面分为三栏：

- 最左侧：功能侧栏，依次为 **Studio 目录** 、**数据目录** 、**Git 源代码管理** 。
- 中间：当前侧栏对应的目录或检索结果。
- 右侧：编辑器主区域，按 Tab 形式打开多个文件。

> ⓘ **注意**
> Studio 目录仅支持新建文件夹、Notebook、SQL、Python 和其他文件类型，不支持新建仪表盘和模型实验。其余操作与总览页面基本相同。

参数说明

新建对象命名规则

参数	说明	是否必填	默认值
名称	文件夹或文件名，支持中英文、数字、下划线、短横线、连字符、点、空格和冒号。	是	无
扩展名	仅在选择 **新建文件** 时需要，决定文件类型。	是	无
父文件夹	文件所在路径。Git 文件夹仅在工作空间配置 Git 连接后可选。	是	当前所在目录

资源配额

项目	限制
单文件大小（页面创建/上传）	500 MB
单文件大小（OpenAPI 创建/上传）	10 MB
开发目录深度	不超过 10 层
文件名称长度	不超过 255 字符
单层目录下文件数量	不超过 2000 个

使用限制

- 不同模块（Studio、Workflow）目录互相隔离；Studio 中的 Notebook 文件不能直接作为调度任务运行，需要在 Workflow 中创建任务并引用。
- Git 文件夹下的文件不支持单独授权；如需精细化控制，请放在本地文件夹下。
- `.zip`、`.pdf` 等二进制文件可上传与下载，但不支持在线打开。

常见问题

Q1：删除的文件能找回吗？

可以。删除的文件夹与文件都会进入个人回收站，可在回收站中选择 **恢复** 。彻底删除后不可恢复，请谨慎操作。

Q2：他人创建的文件我能看到吗？

- 在 **Workspace** 中，如果你是 Workspace Admins，则可以看到他人文件；如果你不是 Workspace Admins，但对方通过 ACL 授予了「查看、运行、编辑、管理」权限，也可在你的目录中看到对应文件。
- 在 **GitFolder** 中默认看不到他人文件，如登陆远程 Git 仓库，则所有工作空间成员均能看到所有文件。

Q3：单层目录下文件数量超过 2000 怎么办？

请按业务逻辑拆分子文件夹。建议按「场景、主题、日期」三层组织，避免在一个目录下堆放过多文件。

相关文档

- 文件权限
- Git 初始化配置 / GitFolder 创建、源代码管理操作
- 数据目录导航
- 设置 Git 连接

文件权限

最近更新时间：2026-09-11 20:42:32

Studio 中的文件与文件夹支持基于 ACL（Access Control List）的精细化授权。本节介绍文件权限的模型、授权方式与生效范围，便于您根据团队协作需要限制开发文件的访问与修改权限。

概述

Studio 中的文件权限由两层共同决定：

1. **角色权限点（项目级）**：通过工作空间角色控制是否能访问 Studio 模块、是否能创建文件等粗粒度操作。详见 成员与权限管理 。
2. **文件 ACL（对象级）**：在文件夹或文件层级，通过 ACL 授予具体用户或角色「管理、编辑、运行、查看」权限。本文重点介绍 ACL。

前提条件

- 您是文件、文件夹的创建人，或被授予了「管理」权限。
- 仅 Workspace 下的对象支持 ACL 授权，GitFolder 下的对象不支持 ACL 授权。
- 您所在的角色需具备访问 Studio 模块的权限点。

权限模型

权限项

权限	包含的能力
管理	查看、运行、编辑文件；管理 ACL（添加 / 删除授权）；删除文件、移动、重命名。
编辑	查看、运行、编辑文件；不能管理 ACL；不能删除文件。
运行	查看文件；运行 Notebook / SQL / Python 文件；不能编辑、不能删除。
查看	仅可打开文件查看代码与历史，不能运行、编辑或删除。
无	不可见。文件不会出现在该用户的开发目录中。

权限继承

- 文件夹 ACL 自动继承到其下的子文件夹与文件。
- 子对象可单独覆盖授权；覆盖后该子对象的 ACL 以自身为准，与父文件夹脱钩。

默认授权

- 创建人默认拥有「管理」权限。

- 工作空间管理员默认拥有所有 Workspace 下对象的「管理」权限。
- 其他用户默认无任何权限（不可见）。

操作步骤

步骤 1：进入文件权限配置

1. 在 Studio 开发目录中右键目标文件夹或文件。
2. 选择 **权限配置** 。

步骤 2：添加授权

1. 在权限配置抽屉中单击 **添加授权** 。
2. 在「主体」下拉中选择 **用户** 或 **角色** ，并搜索目标用户名、角色名。
3. 在「权限」下拉中选择 **管理、编辑、运行、查看** 。
4. 单击 **确定** 保存。

步骤 3：调整或撤销授权

- 在已授权列表中单击行末的 **编辑** ，可修改权限项。
- 单击 **删除** ，可移除该用户、角色的授权。删除后该主体回到「无权限」状态。

步骤 4：查看生效权限

- 抽屉底部「生效权限」列表显示当前文件、文件夹下所有有权访问的用户与权限项。
- 该列表合并了直接授权与继承授权，供您快速核对。

参数说明

参数	说明	是否必填	默认值
主体类型	选择「用户」或「用户组」。	是	无
主体	用户名或用户组名称。	是	无
权限	可管理 / 可编辑 / 可运行 / 可查看。	是	可管理

使用限制

- Git 文件夹下的对象不支持 ACL，所有项目成员均拥有管理权限。
- 工作空间管理员的「管理」权限不可撤销。
- 文件被删除后，ACL 一并随回收站记录保留；恢复后 ACL 自动还原。

常见问题

Q1：用户被授予了「编辑」权限，但仍然不能在 Workflow 中运行该文件，为什么？

运行 Workflow 中的任务需要文件的「运行」权限。如果该任务的运行账号没有运行权限，提交或调度时会返回权限校验失败。请同时给运行账号授予「运行」或更高权限。

Q2：撤销一个用户的「管理」权限，他原先创建的子文件夹会受影响吗？

不会。子对象的 ACL 独立保存。撤销「管理」权限后该用户无法继续管理新对象，但已存在子对象的 ACL 不会自动调整。

Q3：授权给「角色」与授权给「用户组」哪个优先？

两者并存。某用户的最终权限是其所属用户组权限与个人权限的并集。例如：用户组被授予「查看」、个人被授予「编辑」，则最终权限为「编辑」。

Q4：文件移动到另一个父文件夹时，ACL 是跟随子对象走，还是被新父文件夹覆盖？

两者并存。子对象本身的 ACL 权限项会保留，同时继承新父文件夹的权限，继承原父文件夹的权限项会被移除。

相关文档

- IDE 界面、项目与文件组织
- 成员与权限管理

Notebook开发
Notebook IDE 基础操作（多语法支持、多人协同）

最近更新时间：2026-09-11 20:57:30

DataBuddy Studio 中的 Notebook 文件以 `.ipynb` 为扩展名，按单元格（Cell）组织代码。本节介绍 Notebook 编辑器的单元格管理、多语法切换、多人协同和版本管理等基础能力。运行相关内容请参考 Notebook 运行 。

概述

Notebook IDE 提供与 Jupyter Notebook 类似的开发体验：

- 默认单元格语言为 Python；通过魔法命令切换 SQL、Markdown 等语法。
- 单元格支持新建、上移、下移、删除、隐藏运行结果等操作。
- 支持多人协作：支持多人编辑场景下的提示功能，可查看当前文件的编辑人列表。
- 文件支持手动保存与版本管理，最多保留 200 个保存版本。

前提条件

- 已加入工作空间，且对当前 Notebook 文件具备「编辑」或「管理」权限。
- 已创建或拥有可访问的 Notebook 文件。

操作步骤

步骤 1：打开或新建 Notebook 文件

1. 在 Studio 左侧 **开发目录** 中：
   - 单击已有的 `.ipynb` 文件直接打开。
   - 或单击 **+ 新建 > Notebook**，输入文件名并选择父文件夹后单击 **确定** 。
1. 文件以 Tab 形式打开在右侧编辑器。

步骤 2：管理单元格

操作栏 **+ 代码/文本** 支持新增单元格，类型包括：

类型	说明
代码	默认为 Python 语法类型，支持用户切换为 SQL 语法。
文本	用于撰写说明文字，支持完整 Markdown 语法。

每个单元格支持进行如下快捷操作：

操作	说明
执行单元格	支持执行当前单元格、执行上方所有的单元格、执行当前单元格及以下。
上移 / 下移	调整单元格顺序。
隐藏结果	折叠单元格运行结果展示区域。
删除	删除当前单元格。

步骤 3：通过魔法命令切换语法

在 Python 单元格首行使用魔法命令切换该单元格的语言：

魔法命令	切换后的语法
`%sql`	Spark SQL
`%md`	Markdown
`%scala`	Scala

```
%sql
SELECT * FROM your_catalog.your_schema.your_table LIMIT 10;
```

> **提示**
> 魔法命令只对当前单元格生效；切换语法后，该单元格的代码将被对应解释器执行。

步骤 4：使用代码补全与语法检测

- **自动补全**：输入数据库、表、视图、字段、函数名时，编辑器会基于已绑定的数据目录给出补全建议；输入 Python 库名（如 `pandas`、`numpy`）也会展示常用 API。
- **语法检测**：单元格在编辑过程中会高亮 Python 语法错误、缩进错误、未定义变量等问题，并在出错行左侧显示红色波浪线。
- **括号匹配**：自动配对 `()`、`[]`、`{}`、单引号、双引号。

步骤 5：保存与查看版本

- **自动缓存**：离开编辑器或定时触发自动保存。再次打开文件时自动恢复最新草稿。
- **手动保存**：单击操作栏 **保存** 或使用快捷键 `⌘ + S`，生成一条版本记录。
- **查看版本**：在右侧 **版本管理** 抽屉中查看历史版本，每个版本展示版本号、保存人、保存时间、描述。

- **版本对比、回滚：** 选择两个版本进行差异对比，或选择历史版本执行回滚。

> **提示**
> 最多保留 200 个手动版本；超出后从最早的版本开始覆盖。

步骤 6：多人协同

- 用户 A 打开文件进入编辑状态时，如果其他用户 B 正在编辑同一个 Cell，则会在 Cell 右上角显示当前编辑人列表，从而对用户进行提示，降低文件内容互相覆盖的风险。

参数说明

Notebook 快捷键（Mac 键位）

快捷键	操作
`Y`	切换为 Code Cell
`M`	切换为 Markdown Cell
`D, D`	删除当前 Cell
`⌘ + Enter`	运行当前 Cell
`⇧ + Enter`	运行当前 Cell 并选中下一个

Windows 用户请将 `⌘` 替换为 `Ctrl`，`⇧` 替换为 `Shift`。

支持版本管理的文件类型

`.ipynb` / `.py` / `.sql` / `.r` / `.scala` / `.sh` / `.json` / `.yaml` / `.yml` / `.md` / `.txt` / `.csv`

使用限制

- 单文件最多保留 200 个手动版本，自动缓存内容不计入版本。
- `%sql` 等非 Python 魔法命令的具体支持范围依赖底层引擎，详见 辅助工具（DLCUtils、魔法命令）。

相关文档

- Notebook 运行
- 辅助工具（DLCUtils、魔法命令）
- 文件权限

Notebook 运行（内核配置、环境管理、运行结果处理）

最近更新时间：2026-09-11 20:57:30

Notebook 文件运行依赖 **远程内核（Remote Kernel）**：每个 Notebook 文件对应一个 Kernel，Kernel 在 Serverless 计算资源上启动。本节介绍 Kernel 的连接、新建、切换、释放，以及环境管理与运行结果处理。

概述

DataBuddy 中 Notebook 的运行模型如下：

- 一个 Notebook 文件对应一个 **内核**（Kernel）。
- Kernel 运行在工作空间的 **作业集群类型计算资源** 上，按量计费、动态扩缩容。
- 每个 Kernel 拥有独立的运行环境（Python 解释器、已安装依赖库 + Spark 配置）。
- 同一个 Notebook 文件新建 Kernel 会自动释放之前的 Kernel，原会话中的变量与缓存被清空。

运行结果以单元格为单位展示在单元格下方，支持表格化交互（排序、筛选、下载）。

前提条件

- 当前工作空间已绑定可用的 **作业集群类型计算资源** 。详见 计算资源概述 。
- 您具备当前 Notebook 的「运行」「编辑」或「管理」权限。
- 如需访问 Catalog 中的库表，您需具备相应的数据访问权限。

操作步骤

步骤 1：连接 Kernel

1. 打开 Notebook 文件。
2. 在编辑器右上角单击 **未连接** ，选择一个计算资源，采用默认环境配置进行连接。
3. 如需修改环境配置参数，点击 **修改配置** ，在「环境」抽屉中填写参数（详见下方 新建 Kernel 参数）后进行连接。
4. 点击**连接** 后，会显示当前的 Kernel 连接进度。
5. 连接成功后，右上角状态为「计算资源名称（已连接）」。

步骤 2：运行单元格、全部运行

1. 单元格运行：单击单元格左侧的**运行** ，或使用快捷键 `⌘ + Enter` 。
2. 全部运行：单击操作栏的 **全部运行** ，按顺序执行所有单元格。
3. 部分运行：选中代码片段后使用 `Ctrl + Shift + Enter` 仅运行选中部分。
4. 运行上方、下方所有单元格：使用 `⌥ + Shift + ↑` / `⌥ + Shift + ↓` 。

5. 停止运行：单击操作栏**停止**，立即终止当前正在运行的单元格。

步骤 3：处理运行结果

3.1 表格结果交互

调用 `display(df)` 或运行 `pandas.DataFrame` / `pyspark.sql.DataFrame` 时，下方以表格展示数据，支持：

操作	说明
复制	选中单元格区域后右键复制，或使用 `⌘ + A` / `⌘ + C` 全选复制。
下载	单击右上角 **下载**，支持 CSV / Excel / TXT 三种格式。
排序	单击列名按升序 / 降序排序。
字段配置	选择显示 / 隐藏列、冻结列、调整列顺序。
数据检索	在结果区查找指定文本，高亮命中并支持上下条切换。
结果筛选	添加筛选条件（多个条件用 AND / OR 连接）。

3.2 隐藏与查看运行信息

- 单元格运行结束后，hover 输出区域可查看 **运行人、运行时间、运行耗时** 。
- 单击单元格折叠按钮，可隐藏运行结果。

步骤 4：管理环境

在 Notebook 文件左侧单击 **环境管理** 抽屉，可查看与调整运行环境：

区域	说明
资源模式	支持选择**分布式** 或**单节点** 。 • 如果选择分布式模式，则会分别启动一个 Driver 和多个 Executor 节点，适用于大数据的分布式处理。 • 如果选择单节点模式，则仅启动一个 Driver 作为单节点计算资源，适用于运行纯 Python 代码。注意：如果选择单节点模式，但代码中使用了 SQL 或者 Spark，则会自动转成分布式模式，分别启动一个所配规格的 Driver 和 Executor 节点。
基础环境	支持选择**默认** 或**自定义** 。 • 如果选择默认，则对应资源组内置镜像。 • 如果选择自定义，则支持选择一个声明依赖库列表的.yml/.yaml 文件，启动 Kernel 时会在内置镜像的基础上自动安装文件中的依赖库。

高级参数	支持自定义运行环境的规格参数（如 `Spark Executor CU 数（spark.executor.memory）`、`Spark Executor 个数（spark.executor.number）`等）。
依赖库	连接计算资源后，可以查看当前环境中所安装的依赖库列表，包括默认安装和自定义安装两种类型。
应用	修改后单击 **应用**，按新配置重建内核。会清空通过代码安装的依赖与定义的变量。
重置环境	单击 **更多 > 重置环境**，将运行环境恢复到基础环境状态，清空所有 `pip install` 的依赖与变量。
重置环境	单击 **更多 > 导出环境**，支持将当前运行环境导出为.yaml 文件并保存到 Studio 开发目录中，便于在不同文件复用环境配置。

> ⓘ **注意**
> 单击 **应用** 或 **重置环境** 后，会重建内核；通过 `pip install` 安装的依赖、`dlcutils.widgets.text(...)` 之外的变量都会被清空。

通过 `pip install` 添加依赖

```
%pip install pandas==2.2.0 jieba
```

依赖安装后立即在当前会话生效；新建会话或重置环境后会被清空，需要重新安装。

步骤 5：重启或释放内核

- **重启内核**：单击文件上方 **重启内核**，仅重启 Kernel，不会创建已创建的 Spark APP。
- **断开连接**：单击右上角资源组名称 → **断开连接**，弹窗确认后释放当前 Spark APP，状态回到「未连接」。
- **自动释放机制**：Kernel 在不活跃时间达到设定值（默认 10 分钟）后会自动释放，再次进入文件时需要重新连接。Spark APP 在不活跃时间达到设定值（默认 2 小时）后会自动释放，如果 Kernel 释放后，Spark APP 尚未释放，则下次连接时会直接连上已存在的 Spark APP。

使用限制

- Kernel 仅支持连接 **作业集群-Spark 类型** 的计算资源，其他类型暂不支持。
- 通过代码 `%pip install` 安装的依赖在新建内核或重置环境后丢失；如需持久化，请使用自定义基础环境功能。
- 运行结果仅支持展示前 1w 行，大小不超过 2M；结果集下载同样遵循此限制。

常见问题

Q1：运行时报「未连接 Kernel」怎么办？

单击右上角 **未连接 > 选择计算资源** 创建 Kernel 即可。如果当前没有可用的计算资源，请联系工作空间管理员在 创建计算资源 创建。

Q2：运行卡住了怎么强制停止？

单击操作栏 **停止** 终止当前单元格。如仍未释放，可在右上角资源组下拉中 **断开连接** ，再重新创建内核。

Q3：内存不足（OOM）怎么办？

单击 **环境管理 > 高级参数** ，调高 `spark.executor.memory` / `spark.driver.memory` 。

相关文档

- Notebook IDE 基础操作
- 辅助工具（DLCUtils、魔法命令）
- 计算资源概述
- 创建计算资源

辅助工具（DLCUtils、魔法命令）

最近更新时间：2026-09-11 20:57:30

DataBuddy 在 Notebook 中提供 **DLCUtils 函数库**与 **魔法命令（Magic Commands）**两类辅助工具，用于参数化、文件交互、密钥管理与多语法切换。本节给出常用模块、典型示例与使用规范。

概述

类型	说明	适用范围
DLCUtils 函数库	与 Notebook 配套的 Python 工具库，简化参数、密钥、任务通信等操作。	所有 Notebook 文件。
魔法命令	Jupyter 提供的特殊指令，用于切换语法、引用其他 Notebook、安装依赖等。	所有 Notebook 文件。

前提条件

- Notebook 已连接 Kernel。
- 您具备当前 Notebook 的「运行」「编辑」或「管理」权限。

DLCUtils 模块速查

常用模块概览

模块	主要函数	用途
`dlcutils.widgets`	`text(name, defaultValue, label)`、`get(name)`、`remove(name)`、`removeAll()`、`getAll()`	定义与获取动态参数。
`dlcutils.notebook`	`exit(value)`、`run(path, timeout, parameters)`	Notebook 间退出参数传递与跨文件执行。
`dlcutils.jobs.taskValues`	`set(key, value)`、`get(taskKey, key, debugValue)`	Workflow 任务间通过 taskValues 传递结果。
`dlcutils.secrets`	`get(secretName, secretVersion, region)`	从腾讯云 SSM 凭据管理服务获取加密凭据。

完整模块、函数清单详见 腾讯云官方文档：dlcutils 工具库及魔法指令使用指南 。

参数化示例

1. 在文件内定义并获取参数

```
# 定义文本参数
dlcutils.widgets.text(name='your_name', defaultValue='Ricky',
label='Your name')

# 获取参数
print(dlcutils.widgets.get('your_name'))
# 输出: Ricky
```

2. 在 Workflow 任务中获取上层参数（带容错）

```
# 获取任务参数
try:
    task_value = dlcutils.widgets.get('task_param')
    if not task_value:
        task_value = 'default_value'
except Exception:
    task_value = 'default_value'
print(task_value)

# 获取工作流参数
try:
    workflow_value = dlcutils.widgets.get('workflow_param')
    if not workflow_value:
        workflow_value = 'default_value'
except Exception:
    workflow_value = 'default_value'
print(workflow_value)
```

Notebook 间参数传递

上游 Notebook 退出并输出参数

```
# upstream.ipynb
dlcutils.notebook.exit('12345')
```

下游 Notebook 接收参数

```
# downstream.ipynb
result = dlcutils.notebook.run('/folder/upstream.ipynb', 60)
print(result)
# 输出：12345
```

Workflow 任务间通过 taskValues 传递

```
# 上游任务 notebook_upstream
dlcutils.jobs.taskValues.set(key='favorite_food', value='apple')
```

```
# 下游任务 notebook_downstream（已配置上游为 notebook_upstream）
value = dlcutils.jobs.taskValues.get(
    taskKey='notebook_upstream',
    key='favorite_food',
    debugValue='banana'
)
print(value)
# Studio 调试时输出：banana
# Workflow 调度时输出：apple
```

> **提示**
> `taskValues.get` 仅在 Workflow 调度运行场景下能取到上游 set 的值；在 Studio 调试运行时取的是 `debugValue`。

通过 SSM 获取密钥

避免在代码中明文写入 AK/SK、数据库密码：

1. 前往 腾讯云 SSM 控制台 创建凭据，记录凭据名称、版本与地域。
2. 在 Notebook 中获取：

```
secret = dlcutils.secrets.get('my_secret', 'v1', 'ap-guangzhou')
print(secret)
```

1. 使用密钥后续调用 SDK 或访问数据库。

魔法命令清单

单元格语法切换

命令	作用
`%sql`	将当前单元格切换为 Spark SQL 语法。
`%md`	将当前单元格切换为 Markdown，仅作展示用。
`%scala`	将当前单元格切换为 Scala 语法（依赖引擎）。

```
%sql
SELECT count(*) FROM your_catalog.your_schema.your_table;
```

跨文件运行

```
%run /path/to/another_notebook.ipynb
```

执行另一个 Notebook 的所有单元格，等价于将其内容内联到当前单元格。常用于公共变量与函数定义共享。

依赖安装

```
%pip install pandas==2.2.0 jieba
```

在当前 Spark 会话中安装依赖。新建会话或重置环境后失效，需要重新安装。

操作步骤

步骤 1：在 Notebook 中调用 DLCUtils

1. 确认当前 Notebook 已连接 Kernel。
2. 直接在 Python 单元格中调用 `dlcutils.widgets.text(...)`、`dlcutils.notebook.exit(...)` 等函数，无需额外 import。

步骤 2：使用魔法命令切换语法

1. 在 Python 单元格的首行输入 `%sql` / `%md` / `%scala` 。
2. 之后的代码按对应语法解释执行。

步骤 3：调试参数

1. 单击操作栏 **参数** 按钮，弹窗显示当前 Notebook 已识别的参数（含通过 `dlcutils.widgets.text` 定义的与可视化定义的）。

2. 在弹窗中调整参数值用于本次调试运行。

使用限制

- `dlcutils.notebook.run` / `dlcutils.jobs.taskValues` 在 Studio 调试运行时返回 `debugValue`；只有在 Workflow 调度运行场景下能取到真实上下游传递值。
- `%pip install` 安装的依赖不持久化；新建会话或重置环境后失效。
- Notebook 中获取参数请统一使用 `dlcutils.widgets.get`；不支持 `${param_key}` 占位符语法（该语法仅适用于 SQL）。

常见问题

Q1：调用 `dlcutils.widgets.get('xxx')` 返回 NameError?

通常是该参数名称未被定义，可以使用 `dlcutils.widgets.widgets()` 函数进行参数名和参数值的定义，或者在界面上点击**参数** 按钮，在弹窗中定义参数名称和取值。

Q2：`%run` 引用的 Notebook 路径怎么写?

- 使用绝对路径：`/workspace_1/your_username/folder/upstream.ipynb`。
- 使用相对路径：`folder/upstream.ipynb`（相对当前文件）。
- 在目录中右键 **复制路径** 直接粘贴。

Q3：SSM 密钥的地域参数如何确定?

SSM 凭据按地域隔离，需要传入凭据创建时所在地域，例如 `ap-guangzhou` / `ap-shanghai`。可在 SSM 控制台查看凭据归属地域。

相关文档

- Notebook IDE 基础操作
- Notebook 运行
- 参数化实现
- 腾讯云官方文档：使用 dlcutils 函数库和魔法命令

SQL 开发
SQL 脚本运行（数据源、资源组配置、运行日志处理、结果查看）

最近更新时间：2026-09-11 18:24:35

本节介绍 Studio 中 SQL 脚本的运行流程：从选择计算资源、设置默认 Catalog，到查看结果、日志、运行历史，以及结果导出。

概述

SQL 脚本的运行模型如下：

- 运行需要选择 **交互式集群类型** 计算资源。
- 运行结果在编辑器下方分 **结果、日志、代码信息** 三个 Tab 展示。
- 最近 30 天的运行历史记录可在 **运行记录** 中查看。

前提条件

- 当前工作空间已绑定 **交互式集群** 类型计算资源。详见 计算资源概述 。
- 您具备当前 SQL 文件的「运行」「编辑」或「管理」权限。
- 您具备目标 Catalog 中相关库表的访问权限。

操作步骤

步骤 1：选择计算资源

1. 打开 SQL 文件。
2. 在编辑器右上角单击 **计算资源** 下拉，选择当前工作空间内的 **交互式集群** 类型计算资源。

步骤 2：设置默认 Catalog 与 Schema（可选）

1. 在编辑器右上角设置 **默认 Catalog** 与 **默认 Schema** 。
2. 之后 SQL 中可直接写 `table_name`，系统按默认 Catalog / Schema 解析。
3. 也可通过 `catalog_name.schema_name.table_name` 三段式指定查询对象信息。

步骤 3：运行 SQL

可选三种方式运行：

方式	入口	行为

全部运行	操作栏 **全部运行**	顺序执行文件中所有以 `;` 分隔的 SQL。
运行选中	选中片段后单击 **运行选中** 或 `⌘ + Enter`	仅执行选中代码。
运行光标所在语句	光标置于语句中并按 `⌘ + Enter`（不选中任何文本）	自动识别当前 `;` 之间的语句执行。

步骤 4：填写参数（如有）

如果 SQL 中包含动态参数（如 `${param_key}`），运行时弹窗提示填写参数值。详见 参数化实现 。

步骤 5：查看运行结果

运行结束后，编辑器下方展示三类信息：

5.1 结果 Tab

展示当前 SQL 的查询结果表格，支持：

操作	说明
排序	单击列名升序 / 降序排序。
显示字段	选择显示 / 隐藏列、冻结列、调整列顺序。
数据检索	在结果区查找文本，高亮命中并支持上下条切换。
结果筛选	添加筛选条件（多条件用 AND / OR 连接）。
复制	选中区域右键复制，或 `⌘ + A` / `⌘ + C` 全选复制。
下载	单击 **下载**，支持 CSV / Excel / TXT。
可视化	单击 **创建图表**，按列生成柱状图、折线图、饼图等。

5.2 日志 Tab

展示 SQL 执行的日志，包括 Spark Session 输出、Spark 执行日志、错误堆栈等。

5.3 代码信息 Tab

展示当前 SQL 执行的代码段信息。

步骤 6：查看运行历史

1. 在编辑器下方单击 **运行记录**，可查看当前脚本近 30 天的所有运行记录，以及近 3 天的所有运行结果。
2. 支持的操作：

- **查看详情：** 进入单次运行的结果、日志、代码信息。
- **查看任务洞察：** 查看单次运行的任务洞察和 Spark UI 信息。

> **提示**
> 运行历史按文件 ACL 过滤，您只能看到您有权访问的文件对应的运行记录。

步骤 7：导出结果

格式	说明
CSV	默认编码 UTF-8，包含表头。
Excel（.xlsx）	保留列类型与展示格式。
TXT	制表符分隔。

参数说明

运行配置

参数	说明	是否必填	默认值
计算资源	交互式集群资源。	是	无
默认 Catalog	解析未带 Catalog 前缀的引用。	否	无
默认 Schema	解析未带 Schema 前缀的引用。	否	无

结果区配额

项目	限制
单次查询默认展示数据量	行数不超过 2w 行，大小不超过 10M
下载结果默认数据量	行数无限制，大小不超过 500M
查询记录保存时长	30 天
查询结果保存时长	3 天
查询记录条数	不超过 1024 条
子结果 tab 数	不超过 100 个

常见问题

Q1：报「Resource group not connected」？

- 工作空间未绑定交互式集群计算资源，或所选资源组状态不可用。请前往 创建计算资源 创建并绑定，或选择其他可用资源组。

Q2：查询很慢，如何排查？

- 切到 **代码信息 Tab** 查看 Spark 执行洞察（扫描行数、扫描字节数、shuffle 数据量等）。
- 单击 **查询 ID > 查看 Spark UI**（路标），获取算子级执行详情。
- 检查 SQL 是否触发了全表扫描；考虑添加分区过滤或调整 Catalog 物化视图。

Q3：如何中止一个跑了很久的查询？

- 当前文件正在运行：单击操作栏 **停止** 。

相关文档

- SQL IDE 基础操作
- 参数化实现
- 计算资源概述
- 创建计算资源

参数化实现

最近更新时间：2026-09-11 20:57:31

DataBuddy SQL 脚本支持通过参数化实现「同一段 SQL 在不同运行环境取不同值」的需求，例如开发与生产环境使用不同的库表前缀、按调度时间动态过滤数据等。本节聚焦 Studio 中 SQL 脚本的参数化使用，工作流和任务上的参数管理请参考 参数管理 。

概述

DataBuddy SQL 中的参数分为以下几类，统一以 `${param_key}` 形式引用：

参数类型	定义入口	取值优先级
文件级参数	SQL 文件操作栏的 **参数** 弹窗	调试运行时使用
任务级参数	Workflow 任务配置	调度运行时优先级最高
工作流级参数	Workflow 通用配置	调度运行时次优先
系统内置参数	DataBuddy 自带	仅在调度运行场景生效

参数优先级（从高到低）：**任务 > 工作流 > 文件** 。

前提条件

- 您具备当前 SQL 文件的「编辑」权限。
- 调度运行场景下，需在 Workflow 中创建 SQL 任务并引用本文件。

操作步骤

步骤 1：在 SQL 中引用参数

在 SQL 中使用 `${param_key}` 占位符：

```
SELECT *
FROM your_catalog.your_schema.orders
WHERE dt = '${dp_data_dt}'
LIMIT 100;
```

步骤 2：定义文件级参数

1. 单击操作栏 **参数** 按钮。
2. 在弹窗中单击 **+ 添加参数** 。

3. 填写：
   - **参数名**：与 SQL 中 `${...}` 内的标识符一致，如 `dp_data_dt`。
   - **参数值**：调试运行时使用的值，如 `2026-06-05`。
1. 单击 **确定** 保存。

步骤 3：调试运行

- 单击 **运行**、**全部运行** 时，系统将文件级参数值替换到 SQL 中执行。
- 如果 SQL 中引用了未定义的参数，运行时弹窗提示填写。

步骤 4：发布到 Workflow 调度

1. 单击操作栏 **快速创建任务**，选择目标工作流。
2. 进入 Workflow 任务配置，工作流和任务参数会按以下规则识别：
   - SQL 中已使用的 `${...}` 参数会自动识别为任务参数候选项。
   - 您可在工作流参数、任务参数中给出实际取值（自定义值或引用系统内置参数）。
1. 调度运行时取值规则：**任务参数 > 工作流参数 > 文件参数**。

步骤 5：使用系统内置参数

可将系统内置参数赋给自定义参数，再在 SQL 中引用：

1. 在工作流配置中定义参数 `dp_data_dt = {{workflow.start_time.day}}`。
2. SQL 中使用 `WHERE dt = '${dp_data_dt}'`。
3. 调度运行时 `${dp_data_dt}` 自动替换为工作流的实例数据时间。

常用系统内置参数：

参数	含义
`{{workflow.id}}`	工作流 ID
`{{workflow.name}}`	工作流名称
`{{workflow.run_id}}`	工作流运行 ID
`{{workflow.start_time.year}}` / `month` / `day` / `hour` / `minute` / `second`	实例数据时间各分量
`{{workflow.start_time.iso_date}}`	实例数据时间 ISO 8601 日期格式 `YYYY-MM-DD`
`{{workflow.start_time.iso_datetime}}`	实例数据时间 ISO 8601 日期时间格式

`{{workflow.start_time.is_weekday}}`	是否工作日（true / false）
`{{workflow.trigger.time.day}}`	计划调度时间日
`{{workspace.id}}`	工作空间 ID
`{{task.name}}`	任务名称
`{{task.run_id}}`	任务运行 ID
`{{tasks.<上游任务名>.values.<参数 key>}}`	上游 Notebook 任务通过 `dlcutils.jobs.taskValues.set` 输出的参数
`{{tasks.<上游任务名>.output.first_row}}`	上游 SQL 任务输出的第一行
`{{tasks.<上游任务名>.output.first_row.<列名>}}`	上游 SQL 任务输出第一行指定列
`{{tasks.<上游任务名>.output.rows}}`	上游 SQL 任务输出全部行（JSON 数组）

完整内置参数清单详见 系统内置参数 。

参数说明

参数命名规范

项目	规范
字符集	中英文、数字、下划线 `_` 、短横线 `-`
长度	不超过 128 字符
大小写	区分大小写

参数取值规范

项目	规范
长度	不超过 2048 字符
类型	字符串。SQL 中按字符串拼接到位，需要数值时手动加引号或类型转换。

使用限制

- 参数仅支持 Key-Value 形式；不支持嵌套结构（数组、JSON 对象）。

- 文件级参数仅在 Studio 调试运行时使用；调度运行以工作流和任务参数为准。
- `dlcutils.widgets.get` 不能直接在 SQL 中使用，仅适用于 Notebook。
- 参数值为字符串类型，使用时请注意 SQL 中是否需要单引号包裹（如日期字段）。

常见问题

Q1：调试运行时 `${dp_data_dt}` 没有被替换？

- 检查参数名是否完全一致，包括大小写。
- 检查文件级参数中是否定义了同名参数。
- 调度运行时如果工作流和任务都没有同名参数，会按文件级参数兜底；调度场景请确保至少在工作流参数中定义。

Q2：如何在 SQL 中获取「上一周第一天」？

推荐在工作流参数中定义自定义参数 `dp_start_dt`，取值引用系统内置参数并做偏移：

```
dp_start_dt = {{workflow.start_time.iso_date}}    -- 当天
```

如需更复杂的日期偏移，可在工作流参数取值中使用 `iso_date` 系列内置参数与配套偏移表达式（详见 系统内置参数），SQL 中通过 `${dp_start_dt}` 引用。

Q3：参数值包含单引号或特殊字符怎么办？

建议在参数值中避免使用特殊字符。如需特殊字符，请在 SQL 中使用反斜杠或拼接函数（如 `concat`）转义。

相关文档

- SQL IDE 基础操作
- SQL 脚本运行
- 辅助工具（DLCUtils、魔法命令）

Python 开发
Python IDE 基础操作

最近更新时间：2026-09-11 20:57:31

DataBuddy Studio 支持以独立 Python 文件（ `.py` ）的形式进行开发，与 Notebook 互补：Python 文件适合工程化场景（模块化代码、脚本化作业、可复用函数库）；Notebook 适合交互式探索。本节介绍 Python 文件编辑器的基础能力，运行相关请参考 Python 文件运行 。

概述

Python IDE 与 Notebook、SQL 共用同一套 Studio 工作台，特点如下：

- **完整 Python 语法**： 支持完整 Python 3 语法，不强制单元格化。
- **元数据感知**： 可在编辑器中调用统一 Catalog API 操作库表。
- **跨文件引用**： 支持在同目录下通过 `import` 引用其他 `.py` 模块。
- **AI 助手**： 内置 Buddy 协助代码生成、补全、纠错。
- **版本管理**： 每次手动保存生成一个版本，最多 200 个。

前提条件

- 已加入工作空间，且对当前 Python 文件具备「编辑」或「管理」权限。
- 已创建或拥有可访问的 Python 文件。

操作步骤

步骤 1：打开或新建 Python 文件

1. 在 Studio 左侧 **开发目录** 中：
   - 单击已有的 `.py` 文件直接打开。
   - 或单击 **+ 新建 > Python** ，输入文件名并选择父文件夹后单击**确定** 。
2. 文件以 Tab 形式打开在右侧编辑器。

步骤 2：编写 Python 代码

Python 编辑器具备以下能力：

能力	说明
语法高亮	关键字、字符串、数字、注释、装饰器分别着色。

自动补全	输入标准库（`os`、`json`）、常用三方库（`pandas`、`numpy`）、当前文件已定义的函数与变量时实时联想。
错误检测	语法错误、未定义变量、类型错误实时高亮。
括号匹配	自动配对 `()`、`[]`、`{}`、单引号、双引号。

步骤 3：跨文件引用

支持通过标准 `import` 引用同工作空间内的其他 `.py` 文件：

3.1 同目录引用

```
your_folder/
├── utils.py
└── main.py
```

```
# main.py
from utils import add_numbers

result = add_numbers(1, 2)
print(result)
```

3.2 子目录引用

子目录需要包含 `__init__.py`，或使用相对路径写法：

```
from sub_folder import helper
```

3.3 注意事项

- 本地文件夹和 Git 文件夹下的文件均可被引用。
- 引用范围以当前用户在 Studio 内拥有可见权限的文件为准。

步骤 4：与 Notebook 文件系统交互

Python 文件运行时可读写工作空间下的文件：

```
import pandas as pd
```

```
# 读取 CSV
df = pd.read_csv('your_folder/data.csv')

# 写出
df.to_csv('your_folder/result.csv', index=False)
```

详见 Python 文件运行 的「文件系统交互」章节。

步骤 5：使用 AI 助手

- 选中 Python 片段后右键 **Buddy > 解释**： AI 解释当前代码含义。
- 选中错误片段右键 **Buddy > 纠错**： AI 修复语法错误并给出建议。
- 输入自然语言注释后右键 **Buddy > 续写**： AI 基于注释生成代码。
- 详细 AI 助手用法请参考 代码辅助 。

步骤 6：保存与版本管理

- **自动保存**： 编辑过程中自动缓存草稿。
- **手动保存**： 单击操作栏 **保存** 或 `⌘ + S`，生成一个版本记录。
- **版本对比、回滚**： 在右侧 **版本管理** 抽屉查看历史版本，支持对比与回滚。

参数说明

编辑器顶部操作栏

操作	说明
全部运行	执行整个 Python 文件。
运行选中	选中片段后单击 **运行选中** 仅执行选中代码（在交互式 REPL 中执行）。
停止	终止当前运行。
保存	生成一个手动版本。
格式化	按 PEP 8 规范统一缩进与空白。

常用快捷键

快捷键	操作
`⌘ + Enter`	运行光标所在行 / 选中代码

`Ctrl + ⇧ + Enter`	运行全部
`⌘ + S`	保存
`⌘ + ⇧ + F`	格式化
`⌘ + /`	注释 / 取消注释

使用限制

- `dlcutils` 函数库可在 Python 文件中使用，行为与 Notebook 一致。
- 跨文件引用仅适用于工作空间内的文件，无法引用本地磁盘上的 `.py` 。
- Python 解释器版本与所选计算资源镜像一致；如需特定版本，请在新建 Kernel 时选择对应版本的计算资源。

常见问题

Q1：`import utils` 报 `ModuleNotFoundError`？

- 检查 `utils.py` 是否与当前文件在同一目录。
- 跨目录引用需要确保父目录是 Python 包（包含 `__init__.py` ）。
- 检查文件名是否为合法 Python 标识符（不含中文、横线）。

Q2：Python 文件能直接发布到 Workflow 吗?

可以。Workflow 中的 Python File 任务支持引用 Studio 中的 `.py` 文件。详见 Python File 任务 。

Q3：Python 文件支持 Debug 模式吗?

支持。在编辑器中设置断点后单击 **Debug 运行** ，可逐行调试并查看变量。

相关文档

- Python 文件运行
- 辅助工具（DLCUtils、魔法命令）
- Git 初始化配置 / GitFolder 创建、源代码管理操作

Python 文件运行（内核配置、环境管理、运行结果处理）

最近更新时间：2026-09-11 20:57:31

本节介绍 Python 文件（`.py`）在 Studio 中的运行流程：从连接 Kernel、运行整文件，到与文件系统交互、查看结果与日志。Python 文件运行模型与 Notebook 高度一致，差异在于运行粒度（整文件 vs 单元格）。

概述

Python 文件运行模型：

- 一个 Python 文件对应一个 **内核（Kernel）**，与 Notebook 共用相同的 Kernel 模型。
- 运行整文件即按顺序执行所有代码；选中片段则在交互式 REPL 中执行选中部分。
- 同样支持 `dlcutils` 函数库与 `pip install` 安装依赖。

前提条件

- 当前工作空间已绑定 **作业集群型** 计算资源（DLC Spark MLlib）。详见 计算资源概述 。
- 您具备当前 Python 文件的「运行」「编辑」或「管理」权限。

操作步骤

步骤 1：连接 Kernel

1. 打开 Python 文件。
2. 编辑器右上角单击 **未连接**，进入新建 Kernel 弹窗。
3. 填写参数（参数项与 Notebook 一致，详见 Notebook 运行）。
4. 单击 **确定** 。状态变为「计算资源名称（已连接）」。

步骤 2：运行 Python 文件

方式	入口	行为
全部运行	操作栏 **全部运行**	按文件从头到尾顺序执行所有代码，等价命令行 `python file.py` 。
运行选中	选中代码后 `⌘ + Enter`	在交互式 REPL 中执行选中片段，变量保留供后续选中运行使用。
停止	操作栏 **停止**	终止当前运行。

> **提示**

> 「全部运行」每次都是新的进程，文件中变量在运行结束后不保留。「运行选中」共享同一 REPL，变量持续可用。

步骤 3：与文件系统交互

3.1 增删改查文件

```
import os
import shutil

# 创建文件夹
os.makedirs('your_folder/sub', exist_ok=True)

# 复制文件
shutil.copy('your_folder/data.csv', 'your_folder/sub/data_copy.csv')

# 删除文件
os.remove('your_folder/sub/data_copy.csv')
```

3.2 读取 CSV / JSON

```
import pandas as pd

# CSV
df = pd.read_csv('your_folder/data.csv')

# JSON
import json
with open('your_folder/config.json', 'r') as f:
    config = json.load(f)
```

文件路径以当前文件所在目录为基准，或使用绝对路径 `/workspace_1/your_username/your_folder/data.csv`。

3.3 跨文件引用

支持 `import` 同工作空间下其他 `.py` 模块：

```
from utils import add_numbers
```

```
result = add_numbers(1, 2)
```

详见 Python IDE 基础操作 > 跨文件引用 。

步骤 4：处理运行结果

运行结果显示在编辑器下方，分为以下区域：

区域	说明
输出	`print(...)` 、异常栈、 `display(df)` 表格结果。
日志	Spark 执行日志、错误堆栈、详细运行信息。
运行信息	运行人、运行时间、运行耗时、所用资源。

表格结果交互

调用 `display(df)` 后下方以表格展示，支持的操作与 Notebook 一致：复制、下载（CSV / Excel / TXT）、排序、筛选、字段显示控制等。详见 Notebook 运行 > 处理运行结果 。

步骤 5：管理环境

环境管理交互与 Notebook 完全一致，详见 Notebook 运行 > 管理环境 。Python 文件中的 `pip install` 写法：

```
%pip install pandas==2.2.0 jieba
```

或在 Shell 单元格中：

```
%sh
pip install -r requirements.txt
```

步骤 6：切换或释放 Kernel

操作与 Notebook 一致。详见 Notebook 运行 。

使用限制

- Python 文件运行仅支持使用 **作业集群类型** 计算资源。
- 「全部运行」每次启动新进程，变量与 `pip install` 安装的依赖在运行结束后丢失（依赖在会话级保留）。
- Python 文件不支持单元格交互；如需逐段调试，请使用 Notebook 文件。

常见问题

Q1：选中代码运行后，变量为什么在「全部运行」时取不到?

「全部运行」是独立进程，与「运行选中」的 REPL 不共享变量。请将变量定义放在文件中，或在「全部运行」前先在 REPL 中执行初始化。

Q2：报「ModuleNotFoundError: utils」?

- 检查 `utils.py` 是否与当前文件在同一目录。
- 跨目录引用需要确保父目录是 Python 包（包含 `__init__.py`）。
- 在文件目录中右键 **复制相对路径** 确认实际路径。

Q3：如何调度 Python 文件?

- 方式 1：在 Workflow 中创建 Python File 任务，引用 Studio 中的 `.py` 文件。详见 Python File 任务 。
- 方式 2：使用 `dlcutils.notebook.run` 在 Notebook 中跨文件触发（适用于嵌入式调用）。

相关文档

- Python IDE 基础操作
- Notebook 运行
- 辅助工具（DLCUtils、魔法命令）
- 计算资源概述

数据目录导航

最近更新时间：2026-09-11 18:24:35

Studio 集成数据目录（Catalog）侧栏，让您在编写 SQL / Notebook / Python 的过程中无需切换页面，即可查看库表元数据、复制表名、插入字段名并收藏常用表。数据目录侧栏的元数据源与 数据目录 保持一致，仅在展现形态上做了针对开发场景的优化。

概述

Studio 数据目录侧栏按 **数据源** → **Catalog** → **Schema** → **Table** → **Column** 的层级展示，并在表与字段名后提供「插入到编辑器」「复制」「收藏」等快捷操作。可见的库表范围以当前工作空间内您拥有的 Catalog 权限为准。

场景	收益
编写 SQL 时不确定表结构	侧栏直接浏览字段列表，一键插入到编辑器
团队协作中查找他人开发的表	全局检索表名、备注、字段名
常用表频繁访问	收藏后置顶展示，减少查找成本

前提条件

- 已加入工作空间，且具备 Studio 模块访问权限。
- 工作空间已绑定一个或多个数据源，并完成相应的 Catalog 接入。

操作步骤

步骤 1：打开数据目录侧栏

1. 在 Studio 左侧切换到 **数据目录** 侧栏。
2. 下方按 Catalog → Schema → Table → Column 树状展开。

步骤 2：查看元数据

- **hover 表名**：弹窗展示表名、创建人、描述、表大小、最近更新时间等元数据。
- **hover 字段名**：弹窗展示字段名、字段类型、是否分区列、字段描述。
- **单击表名**：展开字段列表。
- **单击字段名**：展开字段详细信息。

步骤 3：检索元数据

1. 单击侧栏顶部 **检索** 图标，顶部出现独立检索框。

2. 输入关键字，可同时匹配 Catalog 名称、Schema 名称、表名、字段名。
3. 检索结果以列表展示，单击任一结果在树状目录中定位。
4. 检索范围：当前选中的数据源下所有 Catalog。

步骤 4：插入与复制

4.1 插入表名、字段名到编辑器

- 在表名、字段名后单击 **插入** 图标，当前光标位置自动插入完整三段名 `catalog.schema.table` 或 `column_name` 。

4.2 复制名称

- hover 表名、字段名，弹窗中单击 **复制** ，将名称复制到剪贴板。
- 用于粘贴到 SQL 编辑器或 Notebook 单元格。

参数说明

检索范围与命中规则

项目	说明
检索字段	Catalog 名 / Schema 名 / Table 名 / Column 名 / 表描述 / 字段描述
命中规则	包含匹配（不区分大小写）
检索范围	当前所选数据源下，您有权限的库表
单次返回上限	详见控制台

使用限制

- 数据目录中可见的库表受 Catalog 权限控制；详见 数据安全 。
- 检索仅返回您有权限的对象，无权限对象不会出现在结果中。
- 「插入」「快速生成 SELECT」仅在 Notebook、SQL、Python 编辑器内可用。

常见问题

Q1：在数据目录中能看到一张表，但 SQL 查询时报「table not found」？

- 可能是默认 Catalog / Schema 未设置，导致查询时表名解析失败。请使用三段名 `catalog.schema.table` 引用，或在 SQL 编辑器右上角设置默认 Catalog。
- 检查表所在 Catalog 是否已绑定到当前工作空间。

Q2：检索结果不全?

- 检索仅返回有权限的对象。如果您是新加入的成员，请联系工作空间管理员授予 Catalog 访问权限。

- 单击侧栏 **刷新** 重新加载元数据索引。

Q3：刚创建的表为什么在数据目录中看不到？

元数据同步存在短延迟（通常 1 分钟内）。如长时间不可见，单击 **刷新** 或重新选择数据源。

相关文档

- Studio 概述
- SQL IDE 基础操作
- SQL 脚本运行
- 数据目录概述
- 数据发现概述

Workflow
工作流概述

最近更新时间：2026-09-11 20:57:31

概述

工作流（Workflow）是 Workflow 模块中负责编排与调度的最小独立单元。一个工作流由若干个 **任务节点** 与节点之间的 **依赖关系** 组成，描述「什么任务在什么条件下执行」。
工作流的核心能力：

- **承接 Studio**：Studio 完成开发与调试，Workflow 完成发布、调度与运维。
- **统一编排**：所有任务类型（Notebook / SQL / Python File / Ray / 离线数据接入 / IF ELSE / For Each / 嵌套工作流、质量监控）在同一画布上编排。
- **完整运维链路**：覆盖运行记录、运行详情（图模式、列表模式、甘特图）、智能诊断、告警与通知。

核心概念

概念	说明
工作流（Workflow）	一组通过依赖关系连接起来的任务集合，是调度的最小单元。
任务（Task）	工作流中的执行单元，对应一段代码（Notebook / SQL / Python）或一个内置节点（IF ELSE / For Each / 嵌套）。
工作流运行（Workflow Run）	一次工作流执行的实例，记录开始时间、结束时间、状态、参数等运行时信息。
任务运行（Task Run）	一次任务执行的实例，归属于某个工作流运行。
调度触发器（Trigger）	工作流的执行入口：定时 / Cron / 文件到达 / 持续运行 / 手动触发。

前提条件

- 已加入工作空间，且具备 Workflow 模块访问权限。
- 工作空间已绑定计算资源；任务运行依赖对应的资源组。

工作流状态

状态	说明

等待中	所有任务都处于等待状态。
运行中	至少一个任务在运行。
成功	所有任务都成功或被排除运行。
失败	所有任务到达终态，且至少一个失败。
跳过运行	所有任务被跳过（并发限制）。
终止中	工作流正在终止。

任务状态

状态	说明
等待工作流并发	工作流并发达上限，等待其他工作流释放。
等待上游	上游任务未到达终态。
等待任务并发	项目级任务并发上限达 2000。
等待资源	等待调度 / 引擎资源。
运行中	任务执行中。
终止中	任务正在终止。
失败重试	任务等待失败后重试。
跳过运行	因并发限制跳过。
成功	任务运行成功。
失败	任务运行失败。
上游失败	上游任务为失败状态导致下游失败。
排除运行	因不满足运行条件或上游条件分支被排除。

使用限制

- 单工作空间最多 10000 个工作流；单工作流最多 1000 个任务节点。
- 平台级最大并发任务数：2000（每个工作空间）。

相关文档

- 配置工作流

- 调度与触发器
- 其他配置
- 工作流与任务的运行监控
- 工作流与任务的运维操作

配置工作流
调度与触发器

最近更新时间：2026-09-11 20:47:30

调度与触发器（Trigger）决定工作流何时开始运行。本节是 **调度的中心文档**：所有任务类型的调度均通过工作流统一配置，任务级无需单独设置调度周期。涵盖定时触发、持续运行两种触发方式。

概述

DataBuddy 工作流支持以下触发方式：

触发方式	适用场景
定时触发	周期性任务（每天 / 每小时 / 每周 / 每月 / 每年）
持续运行	一次实例完成后立即触发下一次，常用于流式处理
手动触发	不配置调度，仅手动 / API 触发；不需要单独选项，未配置调度即视为手动触发

调度状态分为 **启动** 和 **暂停**：暂停状态下工作流不会按调度触发，但仍可通过手动触发运行。

前提条件

- 已是工作流的所有者或被授予「管理」权限。
- 工作流内至少包含一个任务节点。

操作步骤

步骤 1：进入调度配置

1. 登录 DataBuddy 控制台，进入目标工作空间。
2. 在左侧导航栏选择 **数据工程** > **Workflow**，在工作流列表中单击目标工作流名称，进入工作流详情页。
3. 在右侧侧栏单击 **调度配置**。
4. 单击 **添加调度**，打开调度配置弹窗。

步骤 2：选择触发方式

在弹窗中选择 **触发方式**：定时触发、持续运行。两种方式的配置项不同，参考下方分组说明。

定时触发

通用配置

参数	说明	是否必填	默认值
调度状态	启动 / 暂停	是	启动
调度时区	工作流的执行时区，影响调度时间计算。	是	当地时区
调度时间范围	调度生效的开始时间和结束时间，精确到分钟。	是	开始时间=当前时间；结束时间=2099-12-31
配置方式	常规 / Cron 表达式	是	常规
调度周期	仅常规模式：天 / 小时 / 分钟 / 周 / 月 / 年	是	天

提示

若选择观察夏令时的时区，每小时执行的任务在夏令时切换时可能跳过 1 小时或延迟 1 小时。如需绝对时间，请选择 UTC 时区。

常规模式按周期细分

周期	配置项
天	执行时间（00:00 - 23:59，到分钟）
小时（指定间隔）	每 1-12 小时；从 [开始时间] 到 [结束时间]。间隔不跨天，每天重新从起始小时计算。
小时（指定时间）	选择多个具体小时 + 分钟
分钟	每 1-30 分钟；从 [开始时间] 到 [结束时间]。间隔不跨小时，每小时重新从起始分钟计算。
周	执行日期（多选：星期日-星期六）+ 执行时间
月	执行日期（多选：1-31 号 或 月末，两者不可同时选择）+ 执行时间
年	月份（多选：1-12 月）+ 日期（多选：1-31 号、月末）+ 执行时间

Cron 模式

参数	说明

Cron 表达式	采用 7 段格式「秒 分 时 日 月 周 年」，如 `0 0 0 * * ? *` 表示每天 00:00 执行。

> **注意**
> Cron 最小调度间隔为 1 分钟。若 Cron 配置间隔小于 1 分钟，系统会按每分钟一次起调。

结果预览

填写完成后单击 **结果预览**，下方展示最近 5 次预计运行时间。

持续运行

参数	说明	是否必填	默认值
调度状态	启动 / 暂停	是	启动
任务重试模式	失败 / 从不	是	失败

持续运行规则

- 工作流前一次运行到达终态后，间隔 60 秒触发下一次。
- **不支持任务间依赖**：所有任务并行运行。
- **不支持任务级重试策略**：通过任务重试模式控制（失败、从不）。
- **同一时间只能运行 1 个实例**：重新启动会终止前一次未完成的运行。
- **不支持高级配置（排队、最大并发）**：持续运行模式下不适用。

> **提示**
> 启动持续运行后，工作流操作栏的「运行」按钮变为「重新启动运行」。单击会终止当前运行并重新启动，该次运行记录的触发方式为手动触发；持续运行 trigger 仍正常触发后续运行。

步骤 3：保存调度配置

1. 完成参数填写。
2. 单击 **确定** 保存。
3. 调度配置区域显示当前配置摘要，例如：

```
启动 - 每天 00:00 执行 (UTC+08:00 — Beijing, Chongqing)
```

步骤 4：暂停、启动、删除调度

在调度配置区域：

- **暂停调度 / 启动调度**：切换调度状态。
- **删除调度**：删除调度配置，工作流回到「未配置调度」状态。

步骤 5：手动触发或高级运行

无论是否配置调度，您都可以在工作流详情页操作栏：

- **运行**：使用已保存的参数立即触发一次。
- **高级运行**：自定义本次运行的参数与执行任务范围。

使用限制

- Cron 最小间隔为 1 分钟；小于 1 分钟会按每分钟起调。
- 持续运行不支持任务依赖、任务级重试与高级配置（排队、最大并发）。
- 调度时间范围结束后，工作流自动停止生成实例；如需延长，请编辑调度配置。

常见问题

Q1：调度状态是「启动」但没看到周期实例?

- 检查调度时间范围是否覆盖当前时间。
- 检查工作流是否有可运行的任务（任务为空时不会生成实例）。
- 检查工作流并发是否达到上限（详见 其他配置 的「高级设置」）。

Q2：持续运行的工作流如何发送告警?

告警配置依旧通过工作流告警面板设置，详见 工作流与任务的告警 。常见做法是配置「失败」告警，避免任务连续失败被遗漏。

相关文档

- 工作流概述
- 其他配置
- 系统内置参数
- 工作流与任务的运行监控

其他配置（基本信息、标签、监控指标、告警、计算资源、Git 配置、权限）

最近更新时间：2026-09-11 20:47:30

工作流详情页右侧抽屉提供工作流级的通用配置项。除调度与参数外，本节统一介绍：基本信息、标签、监控指标、告警配置、计算资源、Git 配置、权限。

概述

工作流配置面板默认从上到下依次展示：

区域	说明
基本信息	工作流 ID、创建人、运行账号、描述
调度配置	详见 调度与触发器
参数	详见 参数配置
标签	工作流标签管理
监控指标	添加监控指标
告警配置	详见 工作流与任务的告警
计算资源	工作流下任务关联的计算资源汇总，支持批量切换
Git 配置	Git 仓库配置
权限	工作流级 ACL：管理 / 运行 / 查看 / 无

前提条件

- 已是工作流所有者或具备「管理」权限。
- 修改运行账号、权限等敏感配置需具备相应权限点。

操作步骤

步骤 1：基本信息

参数	说明	是否必填	默认值
工作流 ID	系统生成，唯一标识，支持复制。	—	系统生成

创建者	创建工作流的用户。	—	当前用户
运行账号（Run As）	工作流执行时使用的身份。	是	创建人
描述	工作流说明，最多 10000 字符。	否	空

修改运行账号：

1. 单击 **运行账号** 行的编辑图标。
2. 在下拉中搜索目标用户（支持姓名、邮箱模糊匹配）。
3. 单击 **保存** 。

> ⓘ **注意**
> 切换运行账号后，工作流以新账号身份执行，资源访问权限随之变化。请确保新账号具备所有任务引用的文件、数据源、计算资源的运行权限。

步骤 2：标签

标签用于工作流的多维分类管理，例如按业务域、生命周期、责任人组织工作流。

添加标签

1. 单击 **标签** 行的 **添加标签** ，弹出标签管理弹窗。
2. 单击 **+ 添加** 增加一行：
   - **标签名**：必填，所有语言、大小写、数字、`_`、`-`，最长 128 字符。
   - **标签值**：选填，规范同标签名。
1. 单击 **确定** 保存。

标签规则

规则	说明
唯一性	同一工作流内标签名不可重复。
上限	单工作流最多 1000 个标签。

步骤 3：监控指标

单击 **添加指标** 可为工作流配置监控指标，用于告警判断或运维看板展示。

步骤 4：告警配置

工作流级告警配置详细规则集中在 工作流与任务的告警 。本节仅说明入口与差异点。

项	说明

入口	工作流配置抽屉 > 告警配置 > 立即添加
监控场景	启动 / 成功 / 失败 / 监控指标（运行时长、等待时长、完成时间）
接收渠道	Email / Webhook / Teams / Slack
免打扰	手动终止时免打扰（工作流级仅支持此项）

工作流告警与任务告警相互独立，互不干扰。

步骤 5：计算资源

工作流的计算资源不能单独配置，由工作流下所有任务关联的资源汇总形成：

- **未创建任务**：计算资源行展示 **未配置** 。
- **已创建任务**：列出去重后的计算资源，含状态点、名称、配额、可用配额。

批量切换计算资源

1. 单击某资源行末的 **切换**。
2. 选择新的计算资源。
3. 系统弹窗提示「将批量修改 N 个任务的计算资源」并展示受影响任务列表。
4. 单击 **更新** 完成切换。

步骤 6：Git 配置

单击 **添加配置** 可为工作流关联 Git 仓库，用于版本管理与协作开发。

步骤 7：权限

工作流支持四类权限：

权限	查看	复制	运行	编辑	删除	权限设置
无权限	不支持	不支持	不支持	不支持	不支持	不支持
查看权限	支持	支持	不支持	不支持	不支持	不支持
运行权限	支持	支持	支持	不支持	不支持	不支持
管理权限	支持	支持	支持	支持	支持	支持

配置权限

1. 单击 **权限** 行的 **管理权限** ，打开权限管理弹窗。
2. 已授权列表展示所有用户、用户组的权限。
3. 单击 **添加授权**：

- **主体：** 单选用户或用户组（搜索支持名称、邮箱模糊匹配）。
- **权限：** 选择查看、运行、管理。

1. 单击 **确定** 。

权限规则

- 工作流创建者默认拥有「管理」权限。
- 工作空间管理员默认拥有所有工作流的「管理」权限，且不可删除。
- 「管理权限」可设置其他成员的所有四类权限。

参数说明

Run As 与 ACL 联动

任务运行时使用 Run As 账号校验文件、数据源、计算资源的 ACL 权限。详细规则参考 成员与权限管理：

- 文件 ACL：详见 文件权限 。
- 工作流 ACL：本文档「步骤 7：权限」。
- 计算资源 ACL：详见 计算资源概述 。

任务运行时若 Run As 账号缺失任意权限，任务直接置失败，错误信息记入运行日志。

使用限制

- 工作流名称同空间唯一；同名 API 创建会被后端拦截。
- 工作流复制不会带过权限配置，复制后需要重新授权。
- 删除调度需要管理权限；运行账号修改受 Run As 可编辑权限约束。
- 「计算资源」配置仅展示去重后的列表；本身不承载独立的资源选项。

常见问题

Q1：把工作流授予用户「运行权限」，他能在 Workflow 列表看到吗?

可以。「无权限」用户在列表中不会看到该工作流；「查看、运行、管理」用户都能看到。

相关文档

- 调度与触发器
- 参数配置
- 工作流与任务的告警
- 成员与权限管理
- 计算资源概述

配置任务
任务通用配置

最近更新时间：2026-09-11 20:42:32

任务（Task）是工作流中的执行单元。DataBuddy 提供 9 种任务类型，本文档作为 **任务配置中心文档**：列出所有任务类型，并集中说明跨任务类型共用的配置项（任务名称、上游依赖、参数、监控指标、告警、失败与超时策略）。各任务类型的差异化配置见对应子文档。

任务类型清单

类型	适用场景	详细文档
Notebook 任务	引用 Studio Notebook 文件，运行 Python + Spark 代码。	Notebook 任务
Python File 任务	引用 Studio Python 文件，运行单文件 Python 代码。	Python File 任务
Ray 任务	提交 Ray 作业（zip 包）到 Ray 集群。	Ray 任务
SQL 任务	引用 Studio SQL 文件，运行 Spark SQL。	SQL 任务
离线数据接入任务	引用数据接入模块的离线同步任务。	离线数据同步任务
IF ELSE 条件任务	按条件分支控制下游执行链路。	IF ELSE 条件任务
For Each 循环任务	按数据集循环执行内部任务。	For Each 循环任务
嵌套工作流任务	在当前工作流中调用另一个工作流。	嵌套工作流
质量监控任务	引用数据质量任务，执行规则检查。	质量监控任务

通用配置项

无论选择哪种任务类型，以下配置项的交互与规则一致。各任务类型文档不再重复展开，仅说明本类型独有的字段。

任务名称

项目	规则
字符集	中英文、数字、下划线 `_` 、短横线 `-`

长度	1-512 字符
唯一性	同工作流内任务名不可重复
校验	输入空格自动转下划线；含非法字符或重名时输入框下方红字提示

上游依赖任务（多任务场景）

字段	说明	是否必填
上游依赖任务	多选下拉，选项为当前工作流内的其他任务。新建任务时自动选中画布上当前选中的任务。	否（仅多任务场景显示）
运行条件（触发规则）	单选 / 高级模式	是

运行条件（触发规则）

控制任务在何种上游状态组合下触发。简单模式下拉选项：

选项	触发条件	上游不满足时下游状态
全部成功（默认）	所有上游均成功	上游失败 / 排除运行
全部失败	所有上游均失败或上游失败	排除运行
全部完成	所有上游到达终态（成功 / 失败 / 上游失败 / 排除运行）	等待上游
全部完成且至少一个成功	所有上游完成 + 至少一个成功	排除运行
全部完成且至少一个失败	所有上游完成 + 至少一个失败	排除运行
全部排除运行	所有上游均被排除运行	排除运行
全部完成且没有失败	所有上游完成且无失败	上游失败
没有失败且至少一个成功	无失败且至少一个成功	排除运行
没有排除运行	所有上游都已实际运行（成功 / 失败 / 上游失败）	排除运行

至少一个失败	任一上游失败立即触发	直到失败前等待上游；全部完成无失败 → 排除运行
至少一个成功	任一上游成功立即触发	直到成功前等待上游；全部完成无成功 → 排除运行
至少一个完成	任一上游到达终态立即触发	等待上游

高级模式

支持组合多个 `{上游任务} = {状态集合}` 表达式，用 `任意一个 / 全部` 来判断。例如：

```
task_A = 成功、失败 且 task_B = 成功
```

- 上游任务：单选下拉，选项为「上游依赖任务」中已选的任务，每个上游任务在配置中只能出现一次。
- 运行条件：多选下拉，可选状态为 `成功 / 失败 / 上游失败 / 排除运行`。
- 条件连接：下拉选择 `且 / 或`，默认 `或`。

任务运行状态：排除运行

不满足运行条件的任务进入「**排除运行**」终态，与「跳过运行」（工作流并发不足整体被跳过）不同：

状态	说明	终态
排除运行（Excluded）	因不满足任务运行条件，或上游为分支节点且本节点不在分支命中的链路上	是
跳过运行（Skipped）	工作流整体因并发限制被跳过	是

排除运行的错误码为 `ExecutionConditionsNotMet`，运行详情页提示形如「当前任务运行条件为 ${具体运行条件}，不满足运行条件，因此排除运行。」

分支节点与运行条件的优先级

当上游存在分支节点（IF ELSE / For Each 等）时，**分支判断优先级高于运行条件**：

- 若不命中分支：本任务直接置「排除运行」，不再判断运行条件。
- 若命中分支：按本任务配置的运行条件继续判断是否触发。

描述

- 支持任意符号。
- 字符超过自动停止输入。
- 选填。

参数

任务参数详细规则见 任务参数配置 。每个任务的参数面板包括：

来源	行为
工作流下发参数	工作流参数自动下发到任务，**参数名置灰、参数值不可改**，需在工作流参数中修改。
任务参数	在任务上新增参数。当与工作流参数同名时，**任务参数优先级更高**（任务参数 > 工作流参数）。

参数命名、取值规则：

项目	规则
参数名	所有语言、大小写、数字、`_`、`-`，最长 128 字符
参数值	字符串，最长 2048 字符

监控指标

监控指标详细规则见 工作流与任务的告警 的「步骤 4：配置监控指标」。简表：

指标	用途	默认值
运行时长	任务运行持续时长	00h00m00s（不生效）
等待时长	等待上游 / 资源的时长	00h00m00s（不生效）
完成时间	完成时间点（仅周期运行）	空（不生效）

告警配置

告警规则集中维护在 工作流与任务的告警 。本节仅说明入口：任务配置面板 → **告警配置 > 配置告警**。

失败与超时策略（重试）

控制任务在执行失败或超时后的处理方式。

参数	说明	默认值

执行失败后重试	是否在失败时自动重试。	勾选
超时后行为	直接置失败 / 触发重试	直接置失败
最大重试次数	1-999	4 次（共 5 次执行）
重试间隔	数值 + 单位（秒 / 分钟 / 小时），单位最大值 9999。	5 分钟

提示：保存后，任务配置面板用自然语言展示策略，例如：「执行失败后重试，最多重试次数 4 次，重试之间等待 5 分钟。」

任务保存

操作	说明
创建任务	新建任务时按钮文案。
保存任务	编辑已存在任务时按钮文案。
离开未保存	弹窗提示：取消 / 放弃更改 / 保存并继续。

任务版本

任务保存时自动生成版本记录，每个任务最多保留 1000 个版本。重跑或运行均使用 **最新代码、最新非依赖关系配置**；如运行过程中修改了依赖关系，按修改前的依赖关系继续执行。详见 工作流与任务的运维操作 的「重跑与运行逻辑」。

共享数据接入与计算资源

资源类型	说明
计算资源	由各任务类型独立选择；通常为 Spark / Ray 资源组。详见 计算资源概述 。

相关文档

- 工作流概述
- 参数配置
- 工作流与任务的告警

Notebook 任务

最近更新时间：2026-09-11 20:57:31

Notebook 任务用于在 Workflow 中调度运行 Studio 中开发好的 `.ipynb` 文件。Notebook 任务支持引用 Studio 工作空间或 Git 文件夹中的 Notebook，运行在用户选择的计算资源上。

概述

Notebook 任务是 DataBuddy 中最常用的任务类型，覆盖 ETL、机器学习训练、特征工程等场景。本文档专注 Notebook 任务的差异化配置，**通用配置**（任务名称、上游依赖、运行条件、参数、监控指标、告警、失败与超时策略）请参考 配置任务 。

前提条件

- 工作空间已绑定可用的计算资源（支持 CPU / GPU 计算类型，分布式、单节点资源模式）。详见 计算资源概述 。
- Studio 中已存在目标 Notebook 文件，且当前用户（或 Run As 账号）具备文件的「运行」权限及以上。详见 文件权限 。
- 如使用 Git 引用，工作空间已配置 Git 连接。详见 设置 Git 连接 。

操作步骤

步骤 1：创建任务

1. 在工作流编辑器画布中单击 **+ 添加任务** 。
2. 任务类型选择 **Notebook** 。
3. 进入任务配置面板。

步骤 2：配置基本信息

参数	说明	是否必填
任务名称	同工作流内唯一。	是
任务类型	Notebook。可在下拉中切换为其他任务类型（Python File / SQL / Ray / 离线数据接入等），切换后参数区会清空。	是
任务来源	工作空间 / Git。	是
路径	引用的 Notebook 文件路径。	是

步骤 3：选择 Notebook 文件

任务来源：工作空间

1. 单击「路径」输入框，弹出文件选择器：
   - **工作空间**：按目录浏览，仅展示文件夹与 `.ipynb` 文件。
   - **最近打开**：列出当前用户最近 90 天内浏览过的 Notebook（按时间倒序）。
1. 选中后路径填入输入框，例如 `/Workspace/your_username/data_etl.ipynb`。
2. 路径输入框右侧操作：
   - **跳转图标**：新开页面到 Studio 中打开文件。
   - **复制图标**：复制完整路径。

任务来源：Git

1. 选择 Git 后下拉显示当前工作空间已配置的 Git 分支。
2. 在「路径」输入框中填入相对路径（基于远程仓库根目录），例如 `notebooks/etl/daily.ipynb`。
3. 系统不在配置时刻校验路径是否存在；运行时若文件不存在会以错误码 `Git 路径无效` 失败。

> **提示**
> 工作空间未配置 Git 时，「Git」选项置灰。单击「去配置」跳转到 设置 Git 连接 完成配置。

步骤 4：配置计算资源

参数	说明	是否必填	默认值
计算资源	下拉展示当前工作空间已绑定且可用的计算资源。	是	无（用户自行选择）
资源模式	分布式 / 单节点。	是	首次配置文件时该文件在 Studio 中的资源模式；从 Git 引用时默认「分布式」
资源配置	默认配置 / 自定义配置。	是	默认配置

资源模式：分布式（CPU / GPU）

配置项	说明	默认值
Executor 资源	small (1CU) / medium (2CU) / large (4CU) / xlarge (8CU)	跟随计算资源默认值
Executor 个数	动态分配（最小-最大）/ 固定分配（具体数量）	计算资源默认值

Driver 资源	同 Executor 规格选项	计算资源默认值
Executor GPU 数	仅 GPU 类型计算资源展示。支持小数。	计算资源默认值
Driver GPU 数	仅 GPU 类型计算资源展示。支持小数。	计算资源默认值

> ⓘ **注意**
> 自定义配置的资源总量（Executor 资源 × 个数 + Driver 资源）不能超过计算资源最大可用配额。超出时确认按钮置灰，并提示「超出当前计算资源最大可用数（{计算资源最大 CU/GPU}）」。

资源模式：单节点

配置项	说明
资源规格	下拉选择，默认 `large(4CU)`。仅显示不超过计算资源最大规格的选项。
GPU 配置	仅 GPU 类型计算资源展示。支持小数。

自定义配置 – JSON 模式

UI 模式与 JSON 模式可双向切换。JSON 内容遵循腾讯云 API 规范：仅可修改 Value，新增、修改 Key 会校验失败。

步骤 5：上游依赖与运行条件

参考 配置任务 的「上游依赖任务」与「运行条件」。运行条件支持简单模式（12 种触发选项，覆盖「全部成功、全部失败、全部完成、至少一个成功、至少一个失败、没有失败且至少一个成功」等场景）和高级模式（按上游逐个配置状态集合，用 `且` / `或` 连接）。不满足运行条件的任务终态为「**排除运行**」，错误码 `ExecutionConditionsNotMet`。

步骤 6：参数

参考 任务参数配置。Notebook 任务在代码中获取参数：

```
value = dlcutils.widgets.get('your_param')
```

详见 辅助工具（DLCUtils、魔法命令）。

步骤 7：监控指标、告警、失败与超时策略

集中维护，本任务直接使用：

能力	中心文档

监控指标 / 告警	工作流与任务的告警
失败与超时策略	配置任务 的「失败与超时策略」

步骤 8：保存

单击 **创建任务**（新建场景）或 **保存任务**（编辑场景）。

参数说明

路径示例

来源	示例
工作空间	`/Workspace/your_username/data_etl.ipynb`
工作空间（Git Folder）	`/GitFolder/your_username/etl.ipynb`
Git 远程仓库	`notebooks/etl/daily.ipynb`

任务运行说明

- 使用 Serverless 资源组的任务，会在运行前自动安装 Notebook 文件中的环境配置（基础环境 + `pip install` 安装的依赖），无需在任务运行时单独安装。
- 调度运行使用「文件最新代码、任务最新配置」。如运行过程中文件被修改，下次运行使用新代码。

使用限制

- 计算资源由用户在任务配置时选择，需具备运行权限。
- 引用文件被移动、删除会导致任务运行失败，错误码 `xxx 路径文件不存在`。
- Git 路径不在配置时刻校验，存在路径错误风险；建议保存后立即调试运行验证。
- 自定义资源配置不能超过所选计算资源的最大配额。

常见问题

Q1：Notebook 任务可以使用哪些计算资源?

Notebook 任务在配置时可下拉选择当前工作空间已绑定的计算资源；下拉列表展示资源名称、配额、可用配额。具体支持的资源类型与配额管理详见 计算资源概述。

Q2：从 Git 引用 Notebook 时，每次运行都要拉取最新代码吗?

是的。Notebook 任务每次运行（含调度运行和调试运行）都会从远程 Git 仓库获取最新代码。

- 调试运行：使用当前运行人的 Git 配置。
- 调度运行：使用任务 Run As 账号的 Git 配置。

Q3：参数值可以引用上游 SQL 任务的 output 行吗？

可以。在任务参数中写：

```
parameter = {{tasks.<上游 SQL 任务名>.output.first_row.<列名>}}
```

详见 参数传递 。

相关文档

- 配置任务
- Notebook 运行
- 辅助工具（DLCUtils、魔法命令）
- 参数传递
- 工作流与任务的告警
- 计算资源概述

Python File 任务

最近更新时间：2026-09-11 20:57:31

Python File 任务用于在 Workflow 中调度运行 Studio 中的 `.py` 文件。运行链路与 Notebook 任务类似，但文件粒度为单 Python 脚本，适合工程化场景（模块化代码、脚本作业、Python 包入口）。

概述

维度	Notebook 任务	Python File 任务
文件类型	`.ipynb`	`.py`
运行方式	按单元格组织、按顺序执行	整文件运行
适用场景	交互式开发、ML 实验	工程化脚本、生产作业
数据交互能力	DLCUtils 全模块支持	DLCUtils 全模块支持
资源模式	分布式 / 单节点	分布式 / 单节点

通用配置（任务名称、上游依赖、运行条件、参数、监控指标、告警、失败与超时策略）请参考 配置任务 。

前提条件

- 工作空间已绑定可用的计算资源（CPU 计算类型，分布式、单节点资源模式）。详见 计算资源概述 。
- Studio 中已存在目标 Python 文件；当前用户（或 Run As 账号）具备「运行」权限以上。
- 如使用 Git 引用，工作空间已配置 Git 连接。

操作步骤

步骤 1：创建任务

1. 在工作流画布单击 **+ 添加任务** 。
2. 任务类型选择 **Python File** 。

步骤 2：配置基本信息

参数	说明	是否必填
任务名称	同工作流内唯一。	是
任务类型	Python File。可在下拉中切换为其他任务类型，切换后参数区会清空。	是
任务来源	工作空间 / Git。	是

路径	引用的 `.py` 文件路径。	是

步骤 3：选择 Python 文件

操作与 Notebook 任务一致，详见 Notebook 任务 的「步骤 3：选择 Notebook 文件」。文件选择器仅展示 `.py` 文件。

步骤 4：配置计算资源

参数	说明	是否必填	默认值
计算资源	下拉展示当前工作空间已绑定且可用的计算资源。	是	无（用户自行选择）
资源模式	分布式 / 单节点。	是	首次配置文件时该文件在 Studio 中的资源模式；从 Git 引用时默认「分布式」
资源配置	默认配置 / 自定义配置（CPU 模式）。Python File 任务的 GPU 配置在本期不开放。	是	默认配置

具体配置项与 Notebook 任务的「步骤 4：配置计算资源」相同，区别如下：

差异	说明
GPU 配置	Python File 任务**不支持** GPU 资源配置。
基础环境	工作流前端无此配置项；如 Studio 中已配置则后台直接复用，否则使用默认配置。
依赖库	工作流前端无此配置项；如 Studio 中已配置则复用，否则提交为空。

步骤 5：上游依赖、参数、监控指标、告警、失败与超时策略

参考 配置任务 。

步骤 6：保存

单击 **创建任务** 或 **保存任务** 。

参数说明

路径示例

来源	示例

工作空间	`/Workspace/your_username/etl_pipeline.py`
工作空间（Git Folder）	`/GitFolder/your_username/etl.py`
Git 远程仓库	`scripts/etl/main.py`

运行模型

- 使用 Serverless 资源组的任务，运行前自动安装 Python 文件在 Studio 中配置的基础环境与依赖。
- 整文件按 `python file.py` 等价方式执行。变量在运行结束后不保留。
- `dlcutils` 函数库在 Python File 任务中可用，行为与 Notebook 一致。

使用限制

- 计算资源由用户在任务配置时选择，需具备运行权限。
- Python File 任务**不支持** GPU 资源配置（本期限制）。
- 文件运行不支持单元格交互；如需逐段调试，请使用 Notebook 文件。
- Git 路径不在配置时刻校验，需要保存后通过调试运行验证。

常见问题

Q1：为什么 Studio 里能选 GPU 镜像，但 Workflow 中 Python File 任务不能配 GPU?
本期 Python File 任务的 GPU 配置未开放。如需 GPU 训练，请改用 Notebook 任务或 Ray 任务。
Q2：Python 文件中能调用 `dlcutils.notebook.run` 触发 Notebook 吗?
可以。在 Python 文件中使用：

```
result = dlcutils.notebook.run('/path/to/another.ipynb', 60)
print(result)
```

详见 辅助工具（DLCUtils、魔法命令）。
Q3：跨文件 import 是否支持?
支持。同目录或同 Git 文件夹下的 `.py` 文件可通过标准 `import` 引用。详见 Python IDE 基础操作 的「步骤 3：跨文件引用」。

相关文档

- 配置任务
- Notebook 任务
- Python IDE 基础操作
- Python 文件运行
- 辅助工具（DLCUtils、魔法命令）

- 计算资源概述

Ray 任务

最近更新时间：2026-09-11 18:24:35

Ray 任务用于在 Workflow 中调度提交 Ray 作业到 Ray 集群，适用于分布式机器学习训练、强化学习、大规模数据预处理等场景。

概述

Ray 是一个高性能分布式计算框架，覆盖：

- **分布式机器学习训练**： 大规模模型训练与超参数调优。
- **强化学习**： 基于 RLlib 的 RL 训练。
- **大规模数据预处理**： 特征工程、数据清洗。
- **模型服务**： 基于 Ray Serve 的在线推理。

通用配置（任务名称、上游依赖、运行条件、参数、监控指标、告警、失败与超时策略）请参考 配置任务。

前提条件

- 工作空间已绑定 **作业集群 – Ray 集群** 类型计算资源。详见 创建计算资源 。
- 已准备好 Ray 作业的 zip 包：包含入口脚本与依赖。

操作步骤

步骤 1：创建任务

1. 在工作流画布单击 **+ 添加任务** 。
2. 任务类型选择 **Ray 作业** 。

步骤 2：配置基本信息

参数	说明	是否必填
任务名称	同工作流内唯一。	是
任务类型	Ray 作业（不可改）。	是
任务来源	上传 zip 文件。	是

步骤 3：上传 zip 包

参数	说明	是否必填
zip 文件	包含 Ray 作业入口脚本与依赖的压缩包。	是

上传后 zip 包存储在工作空间内，供本任务调度使用。

步骤 4：配置计算资源

参数	说明	是否必填
计算资源	选择 **作业集群 – Ray 类型** 的计算资源。	是
入口指令	启动 Ray 作业的命令，例如 `python -c 'import ray; ray.init(); ...'` 。	是
存储配置	DLC 存储配置，决定 Ray 作业读写的存储位置。	是
计算环境	DLC 计算环境，对应 Ray 内核版本。	是

> **提示**
> 存储配置和计算环境的具体选项与 DLC 控制台保持一致，详细字段以控制台为准。

步骤 5：上游依赖与运行条件

参考 配置任务。Ray 任务可与其他任务类型混编，按 DAG 依赖执行。

步骤 6：监控指标、告警、失败与超时策略

参考 配置任务 与 工作流与任务的告警 。

步骤 7：保存

单击 **创建任务** 或 **保存任务** 。

参数说明

Ray 任务参数

Ray 任务在本期不支持参数传递（与其他任务类型不同）。如需在 Ray 作业中传递动态值，请通过 zip 包内置配置或环境变量自管理。

任务运行详情

Ray 任务的运行详情页 Tab：

Tab	说明

运行结果	本期暂无（DLC 限制）。
运行日志	本期暂无。
任务洞察	本期暂无。
事件日志	DLC 提供的事件日志，本任务展示。
Pod 详情	DLC Pod 列表与状态。
Spark UI	对应 Ray Dashboard，单击按钮跳转到 Ray UI。

使用限制

- 计算资源类型必须为 **作业集群 – Ray 集群**；不支持其他类型。
- Ray 任务不支持参数传递（本期）。
- 任务来源**本期仅支持本地上传 zip 包**；引用 Studio workspace、Volume、远程 Git 仓库中的文件或 zip 包暂不支持。
- 由于 DLC 当前接口能力限制，运行结果、运行日志、任务洞察 Tab 暂不展示。

常见问题

Q1：怎么准备 Ray 作业的 zip 包?

推荐结构：

```
your_ray_job.zip
├── main.py              # 入口脚本
├── requirements.txt     # Python 依赖
└── modules/             # 自定义模块
    └── trainer.py
```

入口指令通过 `python main.py` 触发，main.py 中初始化 Ray 集群并提交作业。

Q2：Ray 任务能用于实时推理吗?

本任务用于离线 Ray 作业调度。实时推理建议使用 Ray Serve 部署，详见 部署在线推理服务（REST API）（待生成）。

Q3：Ray UI 能看到任务的执行细节吗?

可以。任务运行详情页单击 **Spark UI** 按钮跳转到 Ray Dashboard，查看作业拓扑、任务排队、资源使用情况。

相关文档

- 工作流与任务的告警
- 计算资源概述

- 创建计算资源

SQL 任务

最近更新时间：2026-09-11 20:57:31

SQL 任务用于在 Workflow 中调度运行 Studio 中的 `.sql` 文件，统一使用 Spark SQL 语法。SQL 任务支持引用工作空间或 Git 文件夹中的 SQL 文件，运行在数据分析型、数据计算型 Serverless 资源组上。

概述

项目	说明
文件类型	`.sql`
语法	Spark SQL（基于 Kyuubi）
计算资源	数据分析型 / 数据计算型 Serverless 资源组
多语句	支持文件内多条 SQL（以 `;` 分隔），按顺序执行
输出能力	可作为下游任务的参数来源（`{{tasks.<name>.output.rows}}`）

通用配置（任务名称、上游依赖、运行条件、参数、监控指标、告警、失败与超时策略）请参考 配置任务。

前提条件

- 工作空间已绑定 **数据分析型** 或 **数据计算型** Serverless 资源组。详见 计算资源概述 。
- Studio 中已存在目标 SQL 文件，且当前用户（或 Run As 账号）具备「运行」权限以上。
- 已具备目标 Catalog 中相关库表的访问权限。

操作步骤

步骤 1：创建任务

1. 在工作流画布单击 + **添加任务** 。
2. 任务类型选择 **SQL** 。

步骤 2：配置基本信息

参数	说明	是否必填
任务名称	同工作流内唯一。	是
任务类型	SQL。可在下拉中切换为其他任务类型，切换后参数区会清空。	是
任务来源	工作空间 / Git。	是

路径	引用的 `.sql` 文件路径。	是

步骤 3：选择 SQL 文件

任务来源：工作空间

1. 单击「路径」输入框，弹出下拉列表：
   - 仅展示当前用户具有「查看」及以上权限、**且已发布过** 的 SQL 文件。
   - 列表显示文件名、文件路径，支持模糊搜索。
1. 选中后路径填入输入框，例如 `/Workspace/your_username/daily_summary.sql` 。
2. 选中文件后右侧操作：
   - **预览图标**：弹窗预览 SQL 代码内容。
   - **跳转图标**：新开页面到 Studio 中打开文件。

任务来源：Git

操作与 Notebook 任务一致，详见 步骤 3：选择 Notebook 文件 – 任务来源：Git 。

> ⓘ **注意**
> 若用户选中仅有「查看」权限的文件，输入框下方提示「当前文件仅有查看权限，无法用于任务运行」，**创建、保存按钮置灰** 。运行需要至少「运行」权限。

步骤 4：配置计算资源

参数	说明	是否必填	默认值
计算资源	数据分析型或数据计算型 Serverless 资源组。下拉仅展示这两类。	是	引用文件中选中的资源

允许在任务级别覆盖默认资源。**任务中调整不会回写到 Studio 中的 SQL 文件** 。

步骤 5：参数

参考 任务参数配置 。SQL 任务的特殊行为：

工作空间 SQL 文件

- 任务配置面板自动从 Studio SQL 文件的「参数」弹窗中拉取参数（仅参数名置灰，参数值可改）。
- 在任务级别可新增任务专属参数，参数名不能与从文件下发的参数重复。

Git 远程仓库 SQL 文件

- 不自动解析文件中的参数；需要手动添加参数名与参数值。

SQL 中引用参数

```
SELECT *
FROM your_catalog.your_schema.orders
WHERE dt = '${dp_data_dt}'
LIMIT 100;
```

参数优先级（高 → 低）：**任务参数 > 工作流参数 > 文件参数** 。当任务参数与工作流参数同名时，任务参数优先生效。详见 参数传递 。

步骤 6：上游依赖、监控指标、告警、失败与超时策略

参考 配置任务。SQL 任务的失败与超时策略默认为「执行失败后重试 = 关闭」。

步骤 7：保存

单击 **创建任务** 或 **保存任务** 。

参数说明

SQL 输出作为下游参数

SQL 任务的输出可被下游任务通过参数引用：

引用	含义	限制
`{{tasks.<name>.output.rows}}`	全部行（JSON 数组）	最多 1000 行 / 48 KB
`{{tasks.<name>.output.first_row}}`	第一行（JSON 对象）	同上
`{{tasks.<name>.output.first_row.<col>}}`	第一行指定列值	同上

> **提示**
> SQL 文件中包含多条 SQL 时，只返回 **第一个 SELECT 语句** 的结果。INSERT、UPDATE 等不返回行的语句不计入输出。

运行结果 tab 展示上限

SQL 任务运行详情页的「运行结果」tab 直接展示 SELECT 语句返回的结果集，与下游参数传递的场景使用不同的上限：

场景	行数上限	大小上限
运行结果 tab 展示	1w 行	5 M
作为下游参数源（`output.rows` / `output.first_row` 等）	1000 行	48 KB

超出部分不展示、不计入参数传递。如需获取完整结果集，请在 SQL 中使用 `INSERT INTO` 写入目标表后再消费。

默认 Catalog / Schema

任务运行时按 Studio SQL 文件中保存的「默认 Catalog」「默认 Schema」解析未带前缀的表名。如 SQL 中只写 `table_name`，会自动补全为 `<默认 Catalog>.<默认 Schema>.table_name`。详细规则参考 SQL IDE 基础操作。

使用限制

- 计算资源仅支持数据分析型、数据计算型 Serverless 资源组。
- SQL 任务文件必须是 **已发布状态**（即已在 Studio 中保存为正式版本）；草稿态文件不能被任务引用。
- 多语句按顺序执行，遇错中断；不支持并行。
- 运行结果 tab 最多展示 1w 行 / 5M；作为下游参数源时最多 1000 行 / 48 KB（详见上文「运行结果 tab 展示上限」）。
- SQL 输出最多 1000 行 / 48 KB；超出部分不参与下游传递。
- 任务运行需要 Run As 具备文件 ACL「运行」及以上权限，否则直接置失败。
- 当 Studio 中引用的 SQL 文件被删除后，工作流任务运行直接置失败。

常见问题

Q1：SQL 文件中包含 `INSERT INTO` 和 `SELECT`，下游能拿到 SELECT 结果吗?

取决于语句顺序。系统按出现的第一个 SELECT 返回结果；如果 SELECT 在 INSERT 之后，可正常被引用。

Q2：从 Git 引用 SQL 文件，参数怎么写?

Git 引用不自动解析参数。需要：

1. 在 SQL 中使用 `${param_name}` 占位符。
2. 在任务配置面板手动添加任务参数 `param_name`，并填入值。

Q3：发布 Studio SQL 文件时会同步更新所有引用的任务参数吗?

- **新增参数**：自动同步到所有引用任务。
- **任务中已有参数、工作流参数**：不覆盖任务中的取值。

Q4：单文件多条 SQL 中只想运行其中一条怎么办?

SQL 任务执行整文件。如需精细化控制，请将不同语句拆为独立 SQL 文件、独立任务。

相关文档

- SQL IDE 基础操作
- SQL 脚本运行
- 参数化实现
- 参数传递
- 计算资源概述

离线数据接入任务

最近更新时间：2026-09-11 20:57:31

离线数据接入任务用于在 Workflow 中调度运行数据接入模块的离线同步任务。任务通过引用接入任务 ID 的方式建立联动，所有接入配置仍在 数据接入 模块中维护。

概述

离线数据接入任务的特点：

- **不再嵌入接入配置**：仅作为「引用执行入口」，详细的同步配置（源端、目标端、字段映射、同步模式）在数据接入模块完成。
- **使用数据接入型计算资源**：与 Notebook、SQL 任务的资源类型不同。
- **运行详情显示接入结果**：任务运行详情页直接展示数据接入运行的进度、行数、失败原因。

通用配置（任务名称、上游依赖、运行条件、参数、监控指标、告警、失败与超时策略）请参考 配置任务 。

前提条件

- 数据接入模块已创建并保存目标的离线同步任务。详见 离线数据同步 系列文档。
- 工作空间已绑定 **数据接入型** 计算资源。详见 计算资源概述 。

操作步骤

步骤 1：创建任务

1. 在工作流画布单击 **+ 添加任务** 。
2. 任务类型选择 **离线数据接入** 。

步骤 2：配置基本信息

参数	说明	是否必填
任务名称	同工作流内唯一。	是
任务类型	离线数据接入（不可改）。	是
任务来源	单击下拉选择数据接入模块中的离线同步任务。	是

步骤 3：选择接入任务

1. 单击「任务来源」下拉，列表展示当前工作空间内的离线接入任务。
2. 支持模糊搜索与精确搜索。
3. 选中后任务名称展示在配置面板，单击可跳转到接入任务的执行页查看详情。

步骤 4：配置计算资源

参数	说明
计算资源	仅展示 **数据接入型** 计算资源。下拉选择即可。

步骤 5：上游依赖、参数、监控指标、告警

参考 配置任务 。

参数处理特殊说明

参数处理方式与 SQL 任务一致：在任务上配置参数 Key-Value，运行时替换到接入任务执行命令中。详见 任务参数配置 。

监控指标

支持运行时长、等待时长、完成时间三类指标。详见 工作流与任务的告警 。

步骤 6：失败与超时策略

参考 配置任务 。

步骤 7：保存

单击 **创建任务** 或 **保存任务** 。

任务运行详情

离线数据接入任务的运行详情页：

Tab	说明
运行结果	数据接入任务的运行进度与执行情况，与数据接入模块中查看一致。
运行日志	同上。

详细字段说明请参考数据接入模块文档： 离线同步任务运维监控 。

使用限制

- 仅支持引用 **离线** 数据接入任务，不支持实时数据接入任务。
- 计算资源类型必须为数据接入型。
- 接入任务被删除后，工作流任务运行直接失败。
- 任务来源选定后，详细的同步配置（源端、目标端、映射等）只能在数据接入模块编辑。

常见问题

Q1：实时数据接入也能在 Workflow 中编排吗?

不支持。实时数据接入任务自带常驻调度，不通过 Workflow 触发。建议在数据接入模块中维护实时任务，参考 实时数据同步 系列文档。

Q2：上游 Notebook 任务能传递参数到离线数据接入任务吗?

可以。参数传递规则与其他任务一致：

```
target_dt = {{tasks.<上游 Notebook 任务名>.values.target_dt}}
```

详见 参数传递 。

Q3：接入任务运行失败后能在 Workflow 中重跑吗?

可以。在工作流运行详情页选中任务节点单击 **重跑** 即可，重跑会触发新的接入任务执行。详见 工作流与任务的运维操作 。

相关文档

- 配置任务
- 表到表同步
- 离线同步任务运维监控
- 计算资源概述

IF ELSE 条件任务

最近更新时间：2026-09-11 20:57:31

IF ELSE 条件任务（条件节点）用于在工作流中实现分支逻辑：根据条件计算结果，决定下游任务链路是否执行。

概述

条件节点常用于：

- 按时间维度触发不同业务流程（如季度末走结算流程，月末走对账流程）。
- 按上游运行结果选择不同处理路径（如成功走 A 链路，失败走告警链路）。
- 按外部参数动态决定执行内容。

通用配置（任务名称、上游依赖、参数、监控指标、告警、失败与超时策略）请参考 配置任务 。

前提条件

- 已是工作流所有者或具备「管理」权限。
- 工作流中已存在或将创建用于条件结果分支的下游任务。

操作步骤

步骤 1：创建条件节点

1. 在工作流画布单击 **+ 添加任务** 。
2. 任务类型选择 **IF ELSE** 。

步骤 2：配置条件

条件节点支持配置多组条件，每组条件关联不同的下游任务。

添加条件

1. 任务配置面板的 **条件配置** 区域单击 **添加按钮** ，弹出条件配置弹窗。
2. 选择 **简单模式** 或 **高级模式** 。
3. 配置条件后单击 **确定** 。

简单模式

参数	说明	是否必填
参数	输入框，支持常数和系统内置参数（如 `{{workflow.start_time.iso_date}}` 、`{{tasks.<name>.values.<key>}}` ）。	是

值 1		
运算符	单选下拉：`==` / `!=` / `>` / `<` / `>=` / `<=` 。	是
参数值 2	输入框，规则同参数值 1。	是

支持的运算符：

运算符	名称	说明
`==`	等于	字符串比较
`!=`	不等于	字符串比较
`>`	大于	数值比较（浮点）
`<`	小于	数值比较（浮点）
`>=`	大于等于	数值比较
`<=`	小于等于	数值比较

> **提示**
> 使用 `==` / `!=` 时参数值按字符串处理；使用 `>` / `<` / `>=` / `<=` 时按数值（浮点）处理。

高级模式

输入框中以 Python 风格表达式描述条件，最长 512 字符，例如：

```
{{tasks.upstream.values.month}} == 12 and {{workflow.start_time.day}} ==
31
```

关联下游任务

参数	说明	是否必填
条件关联任务	多选下拉，选项为当前工作流下除当前任务外的所有任务。	是

条件描述	选填，最长 512 字符。	否

> **注意**
> 关联任务不能选择条件节点的直接或间接上游，否则提交时会被拦截并提示「任务图不能包含循环依赖」。
> 同一关联任务不能在多个条件中重复使用。

步骤 3：在画布连线建立条件分支

除任务配置面板外，画布上为条件节点连接下游时也会触发条件配置：

1. 从条件节点连接到下游任务。
2. 系统自动弹出条件配置弹窗，**条件关联任务** 默认选中刚连接的下游。
3. 完成条件配置后单击 **确定** 。

步骤 4：管理条件列表

条件配置区域以列表展示所有已配置条件：

字段	说明
条件编号	系统生成
条件	简单模式或高级模式表达式
条件关联任务	关联的下游任务名称
条件描述	用户填写的描述
操作	编辑 / 删除

最多配置 50 条条件，超出后添加按钮置灰。

步骤 5：上游依赖、参数、监控指标、告警、失败与超时策略

参考 配置任务 。条件节点的告警支持启动、成功、失败、持续时间告警。

步骤 6：保存

单击 **创建任务** 或 **保存任务** 。

> **提示**
> 保存任务时，对于在条件配置中关联的下游任务，如果该任务未将条件节点配置为上游，系统会自动添加上游依赖。

运行行为

条件节点本身

- 满足条件：节点状态为「成功」，按下游任务的运行条件继续触发执行。
- 不满足条件：节点状态为「成功」（条件计算正常），但 **不满足条件分支下的下游节点（含间接下游）会自动置为「排除运行」状态** 。

下游任务运行详情

不满足条件被排除运行的任务在运行详情页展示：
当前节点因上游条件节点 "task_name"、"task_name_2" 不满足条件，自动排除运行

单独运行、重跑

条件节点支持单独运行或重跑，运行结果正常展示。条件单独运行不会触发下游执行。

条件节点运行详情

条件节点运行详情页展示条件列表：

列	说明
条件编号	与配置时一致
条件	原始条件
条件解析	参数替换后的实际条件，例如 `12 == 12`
条件关联任务	关联任务名（点击跳转配置页）
条件结果	true / false（运行成功后展示）

如条件节点本身失败：

```
条件节点运行失败，失败原因：{关键错误日志}
```

使用限制

- 单条件节点最多配置 50 条条件。
- 关联任务不能选择直接或间接上游（避免循环依赖）。
- 同一下游任务不能在多个条件中重复关联。
- 条件高级模式表达式最长 512 字符。

常见问题

Q1：条件节点的上游有 SQL 任务，如何在条件中使用 SQL 输出?

通过参数引用：

```
{{tasks.upstream_sql.output.first_row.month}} == 12
```

详见 系统内置参数 。

Q2：条件不满足时下游被「排除运行」，更下游的任务能继续执行吗?

- 如果更下游的任务上游均被排除运行，按下游任务的「运行条件」决定。例如运行条件设为「全部完成」会触发执行；设为「全部成功」会进入「上游失败」状态。
- 通常建议在分支汇聚点使用「没有失败且至少一个成功」运行条件，避免排除运行向下传播。

Q3：高级模式中支持哪些语法?

高级模式支持 Python 条件表达式语法，包括：

- 比较运算符：`==` 、`!=` 、`<` 、`>` 、`<=` 、`>=`
- 逻辑运算符：`and` 、`or` 、`not`
- 括号 `()` 用于分组
- Python 内置函数（如 `len()` 、`str()` 、`int()` 等）

建议尽量保持简洁，复杂逻辑可拆为多条简单条件。

相关文档

- 配置任务
- 参数传递
- 系统内置参数
- 工作流与任务的运行监控

For Each 循环任务

最近更新时间：2026-09-11 21:09:30

For Each 循环任务用于按数据集（数组或对象数组）循环执行内部任务，常用于按分区并行处理、多库多表同步、大规模批处理等场景。

概述

For Each 节点由 **外部循环节点** 和 **内部循环节点** 组成：

- **外部节点**：定义循环参数与并发数，决定循环次数与并行度。
- **内部节点**：实际执行的任务，每次循环按不同的输入参数执行一次。

典型应用场景：

场景	描述
按分区并行处理	上游查询出多个日期分区，并行启动多个补数据任务。
多库多表同步	从元数据库读取表清单，动态生成多个同步任务。
大规模模型推理	将海量数据切分为多个 Batch，并行启动多个推理节点。

通用配置（任务名称、上游依赖、运行条件、参数、监控指标、告警、失败与超时策略）请参考 配置任务 。

前提条件

- 已是工作流所有者或具备「管理」权限。
- 已准备好循环参数（数组、对象数组）或可从上游任务的输出引用。

操作步骤

步骤 1：创建 For Each 节点

1. 在工作流画布单击 + **添加任务** 。
2. 任务类型选择 **For Each** 。

步骤 2：配置外部 For Each 节点

参数	说明	是否必填
任务名称	同工作流内唯一。	是
任务类型	For Each（不可改）。	是
循环参数	循环依赖的数组或对象数组。最长 5000 字符。	是

最大并发数	同时运行的循环次数，1-100 之间正整数。	否（默认 1）

循环参数支持的来源

来源	写法
静态数组	`["Beijing", "Shanghai", "Guangzhou"]`
静态对象数组	`[{"db": "inventory", "region": "shanghai"}, {"db": "orders", "region": "beijing"}]`
上游 Notebook	`{{tasks.<name>.values.<key>}}`（上游通过 `dlcutils.jobs.taskValues.set` 设置）
上游 SQL	`{{tasks.<name>.output.rows}}`

上游依赖与运行条件

参考 配置任务 。

监控指标、告警

参考 工作流与任务的告警 。For Each 任务支持运行时长、等待时长、完成时间监控。

步骤 3：添加内部循环节点

1. 配置完外部节点后单击 **添加循环节点** 。
2. 进入内部循环节点的配置面板。
3. 选择内部任务类型：**Notebook / Python File / SQL / 离线数据接入、嵌套工作流、数据质量任务** 。
4. 内部节点的配置项与对应任务类型一致，参数中可使用循环变量。

循环变量（仅在内部节点使用）

变量	说明	示例
`{{loopDataArray}}`	整个循环参数数组，每次循环都返回完整数据。	`[{...}, {...}]`
`{{loopDataCurrent}}`	本次循环对应的元素。	`{"db": "inventory", "region": "shanghai"}`
`{{loopDataCurrent.<key>}}`	本次循环元素中指定 key 的值。	`shanghai`

`{{loopTimes}}`	当前是第几次循环（从 1 开始）。	`1`

如内部节点配置参数时未引用循环变量，参数下方提示：
您可以通过配置参数值为 `{{loopDataCurrent}}` 获取每次迭代的输入数据。

步骤 4：保存

单击 **创建任务** 或 **保存任务** 。保存后画布上 For Each 节点呈双层框结构：

- **外框**： For Each 节点本身。
- **内框**： 内部循环节点。

画布交互

操作	说明
单击外框	选中 For Each 节点并打开外部配置面板。
单击内框	选中内部循环节点并打开其配置面板。
拖拽外框	整体移动 For Each 节点（含内部节点）。
与其他节点连线	仅外框可连线，内部循环节点不能与外部节点直接连接。

运行行为

整体状态

场景	状态
所有循环成功	For Each 节点状态：成功
部分循环失败	For Each 节点状态：失败（共 N 次循环，X 次失败）
至少一次循环正在运行	状态：运行中（共 N 次循环，X 次失败）
循环参数为空	状态：成功；运行详情提示「无循环执行」。
循环参数格式不正确	状态：失败；运行详情提示「输入字段格式不正确。循环参数必须是 JSON 格式的基础类型数组或对象数组。」

查看循环执行详情

运行结束后，单击 For Each 节点进入运行详情页，可以：

1. **查看循环执行列表**： 每次循环独立一行，展示循环次数、运行状态、运行时长、参数值等信息。

2. **筛选失败循环**：勾选「仅展示失败循环」快速定位失败项。
3. **跳转到单次循环详情**：单击某次循环的运行 ID，进入内部节点运行详情页，查看具体的执行日志和参数。
4. **查看循环配置**：右侧信息栏展示循环参数原始值和最大并发数。

> **提示**
> 重跑策略配置在内部循环节点上，重跑时仅重跑该次循环中失败的内部节点。

使用限制

- For Each 仅支持配置 **一个** 内部循环节点；如需多步骤，请将多步骤封装为嵌套工作流后引用。
- 内部循环节点不能与外部其他任务连线。
- 循环参数（常量）最长 5000 字符；上游传递的引用值（变量）最大 48 KB，超出后任务运行失败。
- 最大循环次数由循环参数的数组长度决定，无独立上限配置。
- 单 For Each 最大并发数 100。
- 循环中有任意一次失败，For Each 节点整体状态为失败。

常见问题

Q1：循环参数从 SQL 任务的 output.rows 引用，循环行为是什么？

SQL 输出的每一行作为一次循环。例如：

```
{{tasks.test_sql_up.output.rows}}
```

返回 `[{"id": 1, "name": "alice"}, {"id": 2, "name": "bob"}]`，则会执行 2 次循环。`{{loopDataCurrent.name}}` 分别为 `alice` 和 `bob`。

Q2：某次循环失败了如何重跑？

重跑策略配置在内部循环节点上。重跑时只会重跑该次循环中失败的内部节点，不会重跑已成功的循环。

Q3：循环参数是空数组 `[]` 时会发生什么？

For Each 节点视为「无循环执行」，状态置为成功；下游任务按运行条件继续执行。

Q4：循环参数格式校验在何时？

在运行时校验。配置时不强制校验，建议保存后通过调试运行验证格式合法性。

相关文档

- 配置任务
- 参数传递
- 系统内置参数
- 嵌套工作流

嵌套工作流任务

最近更新时间：2026-09-11 21:09:30

嵌套工作流任务（Run Job）用于在当前工作流中调用另一个工作流，实现工作流的模块化复用与跨模块编排。

概述

嵌套工作流的核心特点：

- **可复用**：将通用的子流程（如数据校验、维度生成、模型评估）封装为独立工作流，多个主流程通过嵌套引用。
- **解耦**：子工作流可由不同团队维护，主工作流仅引用其结果。
- **多层嵌套**：支持最多 **3 层** 嵌套；超过会被拦截。

通用配置（任务名称、上游依赖、运行条件、参数、监控指标、告警、失败与超时策略）请参考 配置任务 。

前提条件

- 已是工作流所有者或具备「管理」权限。
- 被嵌套的工作流（子工作流）已创建并可运行；当前用户（或 Run As 账号）对子工作流具备「运行」或「管理」权限。

操作步骤

步骤 1：创建任务

1. 在工作流画布单击 **+ 添加任务** 。
2. 任务类型选择 **嵌套工作流** 。

步骤 2：配置基本信息

参数	说明	是否必填
任务名称	同工作流内唯一。	是
任务类型	嵌套工作流。可在下拉中切换为其他任务类型，切换后参数区会清空。	是
选择工作流	下拉选择被嵌套的工作流。	是

步骤 3：选择被嵌套工作流

1. 单击「选择工作流」下拉，列表展示当前用户具有「运行」或「管理」权限的工作流。
2. 支持模糊搜索与精确搜索。
3. 选中工作流后右侧出现跳转 Icon，单击跳转到所选工作流的工作流编排 Tab。

步骤 4：参数

嵌套工作流任务的参数有三类来源，按优先级（高 → 低）：

优先级	来源	行为
1	当前任务参数	任务级新增参数。最高优先级。
2	当前工作流参数	当前工作流参数自动下发到任务，参数名置灰、不可改。
3	被嵌套工作流的任务参数	显示在任务面板，参数名不可改、参数值可改；被改后传给子工作流。
4	被嵌套工作流的工作流参数	同上。

同名冲突规则

嵌套工作流的参数同名优先级遵循「**外层 > 内层**」是大原则，同层级内再按「**任务参数 > 工作流参数**」这两个独立维度共同决定：

- **维度一（跨层级）外层 > 内层**：当前工作流的任意参数 > 被嵌套工作流的任意参数。
- **维度二（同层级内）任务参数 > 工作流参数**：在每一层内部，任务级参数高于工作流级参数。

两者结合即得到上方表格的 4 级优先级：当前任务参数 (1) > 当前工作流参数 (2) > 被嵌套工作流的任务参数 (3) > 被嵌套工作流的工作流参数 (4)。

下游配置面板中出现同名时：

- 被嵌套工作流参数会自动置灰，参数值显示父级取值。
- 如果当前工作流参数与被嵌套工作流参数同名，则被嵌套参数显示提示「此参数为被嵌套的工作流参数；若与工作流参数名称一致，外层工作流参数优先级更高，其值会覆盖被嵌套工作流参数值」。

步骤 5：上游依赖、运行条件、监控指标、告警、失败与超时策略

参考 配置任务 与 工作流与任务的告警 。

步骤 6：保存

单击 **创建任务** 或 **保存任务** 。

运行行为

任务运行触发子工作流

嵌套工作流任务运行时，会触发对应的子工作流运行：

- 子工作流的运行列表中新增一条记录。
- 子工作流运行的「触发方式」展示为：父任务名称（点击跳转到嵌套工作流任务运行详情）。

任务运行详情

嵌套工作流任务运行详情页仅展示「运行结果」Tab：

内容	显示
运行结果	「请到嵌套的工作流运行详情页面查看运行结果：/嵌套工作流 ID/」（蓝色字体可跳转）
文件路径	显示嵌套工作流名称，单击跳转到工作流详情页

工作流运行详情跳转

- **单任务工作流（仅一个嵌套节点）**：单击工作流运行 ID 直接跳转到子工作流运行详情。
- **多任务工作流**：单击工作流运行进入工作流详情页，再单击嵌套节点跳转到子工作流运行详情。

参数说明

参数传递规则

场景	行为
上游传递的参数	在嵌套节点跟随外部工作流运行时可传递生效，作用在 run_job 节点下的每个任务中。
Run Job 节点单独运行	无法获取上游传递参数。
Run Job 节点的参数传递给下游	下游可使用其中定义的参数。
内外部参数同名	在外部运行：以外部为准；在 Run Job 内部运行：以内部为准。

详细参数规则参考 参数传递 。

使用限制

- 嵌套层级最多 **3 层** 。例如 `Workflow A → Workflow B → Workflow C` 是 3 层；再嵌套 D 即超出限制。
- **不支持循环嵌套**（如 A 嵌套 B、B 嵌套 A）。
- 调度后台会**逐层校验** 运行账号（Run As / 当前操作人）对所引用的每一层子工作流的运行权限：任意一层缺少「可运行」及以上权限，整体置失败。
- 嵌套工作流任务被删除后，对应的子工作流仍存在，仅父工作流的引用关系断开。

常见问题

Q1：嵌套工作流的运行账号使用谁的身份？

- 调度运行：使用当前任务的 Run As 账号；该账号需要对子工作流具备运行权限。
- 调试运行：使用当前操作人账号。

Q2：嵌套工作流任务执行失败时，可能出现哪些错误提示？

错误提示	含义
工作流不存在	被嵌套的子工作流已被删除。
工作流内任务不存在	子工作流中没有任何任务。
`{账号}` 无 `{工作流名称}` 的运行权限	Run As 账号对子工作流无可运行及以上权限（任意一层校验未通过）。

详细错误信息会展示在任务运行日志顶部。

Q3：能在 For Each 节点中嵌套工作流吗?

可以。For Each 内部节点支持嵌套工作流类型。多次循环对应嵌套工作流的多次实例。详见 For Each 循环任务 的「内部循环节点配置」。

Q4：嵌套工作流的实例被删除后再查看任务详情会怎样?

跳转到嵌套工作流页面时提示：「无此页面，页面可能被删除」。

相关文档

- 配置任务
- 参数传递
- For Each 循环任务
- 工作流与任务的运行监控

质量监控任务

最近更新时间：2026-09-11 18:24:36

质量监控任务用于在 Workflow 中调度执行数据质量模块定义的质量任务，对监控对象（表、字段）执行规则检查，结果存储到运行记录中。

概述

质量监控任务的核心特点：

- **不在 Workflow 中编辑规则**：所有质量规则在 数据质量 模块中维护。
- **作为工作流节点编排**：与其他任务类型混编，在 DAG 中按依赖关系运行。
- **运行结果反映规则检测**：每条规则的检测结果（正常、异常、失败）在任务运行详情页中展示。

通用配置（任务名称、上游依赖、运行条件、参数、监控指标、告警、失败与超时策略）请参考 任务通用配置。

前提条件

- 数据质量模块中已创建并 **发布** 目标质量任务。详见 创建质量规则。
- 工作空间已绑定 **数据计算型** 或 **数据分析型** Serverless 资源组。

操作步骤

步骤 1：创建任务

1. 工作流列表中点击工作流名称进入工作流配置，选择任务编排，工作流画布单击 **+ 创建任务** 。
2. 任务类型选择 **数据质量** 。

步骤 2：配置基本信息

参数	说明	是否必填
任务名称	同工作流内唯一。	是
任务类型	数据质量。可在下拉中切换为其他任务类型，切换后参数区会清空。	是
任务来源	引用数据质量模块中已发布的规则集。	是

步骤 3：选择质量任务

1. 单击**任务来源**，下拉选择需要执行的质量任务。
2. 仅展示已发布的规则集。
3. 支持模糊搜索。
4. 选中后任务名称展示在配置面板，单击可跳转到对应质量任务详情页。

步骤 4：配置计算资源

参数	说明	是否必填
计算资源	数据计算型 或 数据分析型 Serverless 资源组。	是

> **提示**
> **编排场景 vs 试运行场景**：在工作流编排（即本节配置）中可选择 **数据计算型、数据分析型** 两类资源；而在数据质量模块的「试运行」入口下，下拉框仅展示 **数据分析型** 资源。

步骤 5：上游依赖、参数、监控指标、告警、失败与超时策略

参考 任务通用配置。
与工作流的两种联动方式如下：

方式	配置位置
在工作流中创建质量任务节点（本节）	Workflow 编排画布
质量任务关联工作流（与某个已发布任务同频调度）	创建质量规则

第二种方式下，质量任务出现在被关联工作流的画布中（不可编辑），起到「在线检查」作用。

步骤 6：保存任务

单击 **创建任务** 或 **保存任务** 。

任务运行详情

质量监控任务运行详情页：

区域	显示
整体执行状态	等待运行 / 执行中 / 成功 / 失败 / 终止超时
检测状态	正常 / 异常 / 检测中 / 未检测
正常 / 总规则数	例如 `8 / 10`
规则执行明细	每条规则一行：规则名称 / 模板类型 / 监控对象 / 检测结果 / 实际值 / 操作（日志、执行历史）

详细字段说明参考 质量任务运维（待生成）。

使用限制

- 仅支持引用 **已发布** 的质量任务；草稿状态的质量任务不在下拉列表中。
- 一个质量任务同时只存在一个发布态版本；编辑后会生成新的草稿版本，**原发布态版本继续生效，工作流调度始终使用最新发布态**，不会因为存在草稿而失败。
- 质量任务在 Workflow 中调度时，使用工作流任务的 Run As 账号；该账号需要对监控对象（表、字段）具备访问权限。
- 质量监控任务的告警与监控指标只针对任务执行（运行时长、等待时长等），规则触发的业务告警通过 质量规则告警配置（待生成）处理。

常见问题

Q1：质量任务运行失败和质量规则检测异常的区别?

状态	说明
任务运行失败	任务无法正常执行（如代码错误、资源不足）。
检测异常	任务运行成功，但规则检测结果不满足业务约束。

可在告警配置中分别处理两种场景：失败 → 任务失败告警；检测异常 → 通过规则告警配置发送。

Q2：能在工作流中编辑质量规则吗?

不能。质量规则在数据质量模块维护，Workflow 只是调度执行入口。如需修改规则，请前往数据质量模块。

Q3：上游任务输出表后，下游质量任务能立即检测吗?

可以。在 Workflow 中将质量任务设置为表写入任务的下游：

```
[Notebook 任务：写表] → [质量任务：检测表]
```

设置上游依赖后，质量任务会等待写表任务成功后再开始检测。

相关文档

- 任务通用配置
- 创建质量规则
- 质量任务运维
- 工作流与任务的告警

参数管理
参数配置

最近更新时间：2026-09-11 20:57:31

DataBuddy 工作流支持两个层级的参数配置：**工作流参数** 和**任务参数** 。工作流参数用于定义全局共享的 Key-Value，工作流内所有任务自动继承；任务参数用于定义任务级专属的 Key-Value，仅作用于当前任务。

概述

层级	定义位置	作用范围	优先级
工作流参数	工作流配置抽屉	工作流内所有任务自动继承	低
任务参数	任务配置面板	仅当前任务	高

前提条件

已是工作流所有者或具备**管理权限** 。

配置工作流参数

入口

在数据开发页面，打开目标工作流 > 右侧配置抽屉中选择 **参数 > 参数管理** 。

步骤 1：进入参数配置

1. 打开工作流详情页。
2. 在右侧抽屉单击 **参数 > 参数管理** 。

步骤 2：UI 模式 – 添加参数

1. 默认弹窗为 UI 模式。
2. 单击 + 添加一行参数：
   - **参数名**：必填，支持所有语言、大小写、数字、`_`、`-`，最长 128 字符。
   - **参数值**：必填，字符串，最长 2048 字符。
1. 单击 **确定** 保存。

步骤 3：JSON 模式 – 批量编辑

1. 单击弹窗右上角 **JSON 模式** ，切换到 JSON 编辑器。
2. 输入符合 JSON 规范的对象，例如：

```
{
  "dp_data_dt": "{{workflow.start_time.iso_date}}",
  "env": "production",
  "batch_size": "1000"
}
```

1. 编辑过程中实时校验语法；存在异常时下方显示错误提示，**确定按钮置灰** 。
2. 单击**确定** 保存。

> ⓘ **注意**
> JSON 模式不支持嵌套结构（仅支持一级键值对）；所有值必须是字符串。

步骤 4：删除参数

- UI 模式：单击参数行末的删除图标。
- JSON 模式：直接编辑 JSON 删除对应键值对。

步骤 5：保存与查看

保存后参数列表展示在工作流配置抽屉的 **参数** 区域，每行展示一个参数，最多展示 20 个，超出滚动。

配置任务参数

入口

在数据开发页面，打开目标工作流 > 点击任务编排 > 单击画布中的目标任务节点 > 下方任务配置中参数配置部分。

步骤 1：查看继承工作流参数

任务面板「参数」区域顶部展示继承自工作流的参数，每行包含：

列	说明
参数名	工作流参数名，置灰不可编辑。
参数值	工作流参数值，不可编辑。

步骤 2：添加任务参数

1. 单击 **+ 添加** 添加一行任务参数。
2. 填写：

- **参数名**： 支持所有语言、大小写、数字、`_` 、`-` ，最长 128 字符。不能与已存在的继承工作流参数或其他任务参数重名。

- **参数值**：字符串，最长 2048 字符。

1. 单击 **保存任务** 。

步骤 3：参数与工作流参数同名时

如果任务参数名与继承工作流参数同名：

- 系统会在该参数下方提示：「此任务参数与工作流参数同名。任务参数优先级更高，将覆盖工作流参数的值。」
- 任务仍可保存运行；但运行时使用工作流参数值。

如需保留两份独立参数，请避免同名。

步骤 4：参数值的格式校验

系统内置参数引用格式不正确

异常	示例
双引号不完整	`{{job.id`
`{{}}` 内容为空	`{{}}`

提示文案：「系统参数引用格式错误，将会导致参数引用失效」
不阻塞保存 。运行时该部分按字符串处理，不进行参数替换。

引用了不存在的系统内置参数

异常	示例
拼错参数名	`{{job.id1}}`

提示：「没有对应的系统内置参数。」
不阻塞保存 。运行时该部分按字符串处理，不进行参数替换。

步骤 5：删除参数

单击参数行末的删除图标。继承工作流参数不能在任务面板删除，需要在右侧工作流配置面板的参数配置中删除。

步骤 6：保存

单击 **保存任务** 。

参数取值支持的形式

形式	示例	适用层级
静态字符串	`production`	工作流参数 /

		任务参数
系统内置参数	`{{workflow.start_time.iso_date}}`	工作流参数 / 任务参数
多个内置参数拼接	`{{workflow.start_time.year}}-{{workflow.start_time.month}}-{{workflow.start_time.day}}`	工作流参数 / 任务参数
上游 Notebook 输出	`{{tasks.<name>.values.<key>}}`	仅任务参数
上游 SQL 行输出	`{{tasks.<name>.output.first_row.<col>}}`	仅任务参数

完整内置参数列表参考 系统内置参数 。上游输出引用规则参考 参数传递 。

参数说明

命名规范

项目	规范
字符集	所有语言、大小写、数字、`_` 、`-`
长度	≤ 128 字符
唯一性	同一工作流内参数名不可重复；同一任务内参数名不可重复

取值规范

项目	规范
长度	≤ 2048 字符
类型	字符串。SQL / Notebook 中按字符串拼接到位
系统内置参数	用 `{{...}}` 包围，详见 系统内置参数

引用文件时的参数行为

对于 Notebook / SQL 任务，在任务中引用文件时，文件中配置的参数名和参数值会自动填充到任务参数中。填充后：

- 参数值可修改，修改后按任务参数保存。
- 后续文件中的参数变更**不会联动** 到已引用的任务参数。

- 如果切换了引用的文件，任务参数区域会被清空，并从新文件中重新填充参数。请在切换文件前确认参数配置已经迁移或备份。

任务版本与参数

任务参数变更会随任务版本一同保存。重跑或运行均使用最新版本的任务配置（包括参数）。详见 工作流与任务的运维操作 。

使用限制

- 工作流参数被所有任务自动继承，不能在任务级修改；如需任务级专属取值，请在任务参数中单独定义。
- 在工作流运行场景下，任务参数与继承工作流参数同名时，任务参数优先。
- 参数仅支持字符串类型；如需数值，请在代码中显式转换。
- 参数值最长 2048 字符；超长部分被拒绝。
- 工作流参数在启动时一次性解析，无法引用任务输出（ `{{tasks.<name>...}}` ）。
- 引用上游任务参数时，要求上游任务必须已运行（详见 参数传递 ）。
- 参数值变更后，本次正在运行的实例不受影响；下一次运行使用新值。

常见问题

Q1：在工作流参数和任务参数中都定义了 `dp_data_dt` ，运行时会用哪个？

任务参数。DataBuddy 中任务参数优先级高于工作流参数。

Q2：可以在工作流参数中引用任务输出吗？

不可以。工作流参数在工作流启动时一次性解析；任务输出参数（ `{{tasks.<name>.values.<key>}}` ）只能在任务参数中使用。详见 参数传递 。

Q3：JSON 模式下出现「语法错误」无法保存，怎么排查？

- 确保 JSON 整体被 `{}` 包围。
- 确保所有键和值都是字符串（用双引号）。
- 确保不存在尾部多余的逗号。
- 单引号、回车在 JSON 中不允许；如需多行，请保持单行字符串或用 `\n` 转义。

Q4：工作流和任务都没有定义某个参数，但 SQL 中引用了 `${unknown_param}` 会怎样？

- 调试运行：弹窗提示填写参数值。
- 调度运行：参数值替换为空字符串，可能导致 SQL 语法错误。建议至少在工作流参数中定义。

相关文档

- 参数使用
- 参数传递
- 系统内置参数
- 工作流与任务的运维操作

参数使用

最近更新时间：2026-09-11 20:57:30

本文档介绍如何在 Notebook 和 SQL 任务的代码中引用已配置的参数。参数配置方法参考 参数配置 。

概述

参数配置完成后，需要在任务代码中通过特定语法引用参数值。不同任务类型使用不同的引用方式：

任务类型	引用语法	说明
Notebook	`dlcutils.widgets.get('param_name')`	返回字符串类型的参数值
SQL	`${param_name}`	运行时直接替换为参数值

在 Notebook 中使用参数

引用语法

```
value = dlcutils.widgets.get('param_name')
```

- `param_name`：参数名，对应工作流参数或任务参数中定义的 Key。
- 返回值为字符串类型。如需数值类型，请在代码中显式转换。

示例

引用工作流参数

假设在工作流参数中定义了：`dp_data_dt = {{workflow.start_time.iso_date}}`

```
# 获取参数值
dt = dlcutils.widgets.get('dp_data_dt')
print(dt)
# 输出：2026-06-11

# 如需数值，显式转换
batch_size = int(dlcutils.widgets.get('batch_size'))
```

引用任务参数

假设在任务参数中定义了：`task_param = {{tasks.upstream_sql.output.first_row.col_a}}`

```
col_a_value = dlcutils.widgets.get('task_param')
print(col_a_value)
# 输出：上游 SQL 第一行 col_a 的值
```

调试运行与调度运行的区别

场景	行为
Studio 调试运行	如果参数未定义，弹窗提示用户手动填写参数值。
Workflow 调度运行	系统自动解析参数值，未定义的参数返回空字符串。

在 SQL 中使用参数

引用语法

```
${param_name}
```

运行时 `${param_name}` 会被直接替换为参数值（字符串替换）。

示例

引用工作流参数

假设在工作流参数中定义了：`dp_data_dt = {{workflow.start_time.iso_date}}`

```
SELECT *
FROM my_table
WHERE dt = '${dp_data_dt}';
-- 运行时替换为：WHERE dt = '2026-06-11'
```

引用任务参数

假设在任务参数中定义了：`target_db = my_database`

```
USE ${target_db};
SELECT COUNT(*) FROM orders;
-- 运行时替换为：USE my_database;
```

多参数组合

```
SELECT *
FROM ${target_db}.${target_table}
WHERE dt = '${dp_data_dt}'
  AND env = '${env}';
```

调试运行与调度运行的区别

场景	行为
Studio 调试运行	如果参数未定义，弹窗提示用户手动填写参数值。
Workflow 调度运行	系统自动解析；未定义的参数替换为空字符串，可能导致 SQL 语法错误。

在其他任务类型中使用参数

任务类型	引用语法
离线数据接入	`${param_name}`
条件节点	`${param_name}`
For Each 循环	`${param_name}`
嵌套工作流（Run Job）	`${param_name}`

最佳实践

1. 在工作流参数中封装系统内置参数

推荐将系统内置参数封装为语义明确的业务参数，代码中只引用业务参数名。

```
# 工作流参数定义
dp_data_dt = {{workflow.start_time.iso_date}}
```

```
-- SQL 中引用（推荐）
WHERE dt = '${dp_data_dt}'

-- 不推荐直接在代码中写内置参数
```

```
# Notebook 中引用（推荐）
dt = dlcutils.widgets.get('dp_data_dt')
```

好处：

- 参数名语义明确，代码可读性高。
- 统一在工作流参数中管理，修改一处即可生效。
- 日志中参数值可追溯。

2. 基于 trigger 时间做偏移计算

常见场景：需要用 trigger 时间 −1 天作为分区时间（T+1 场景）。

> ⓘ **注意**
> 系统内置参数 `{{workflow.trigger.time.iso_date}}` 不支持直接在模板语法中做偏移运算，需要在代码中自行计算。

前置步骤：在工作流参数中定义 trigger 日期

```
trigger_date = {{workflow.trigger.time.iso_date}}
```

方式一：在 SQL 中使用函数计算

```
-- 工作流参数已定义：trigger_date = {{workflow.trigger.time.iso_date}}
-- 使用 DATE_SUB 函数将 trigger 日期减 1 天作为分区时间
SELECT *
FROM dw.user_daily
WHERE dt = DATE_SUB('${trigger_date}', INTERVAL 1 DAY);
-- 如果 trigger 时间为 2026-06-12，则等价于：WHERE dt = '2026-06-11'
```

如果引擎不支持 `DATE_SUB`，也可以使用 `DATE_ADD`：

```
WHERE dt = DATE_ADD('${trigger_date}', INTERVAL -1 DAY);
```

方式二：在 Notebook 代码中计算偏移

```
from datetime import datetime, timedelta

# 获取工作流参数中的 trigger 日期
```

```
trigger_date = dlcutils.widgets.get('trigger_date')
# trigger_date 值示例：'2026-06-12'

# 计算前一天作为分区时间
dt_obj = datetime.strptime(trigger_date, '%Y-%m-%d')
partition_date = (dt_obj - timedelta(days=1)).strftime('%Y-%m-%d')
print(partition_date)
# 输出：2026-06-11

# 后续使用 partition_date 进行数据读写
spark.sql(f"SELECT * FROM dw.user_daily WHERE dt = '{partition_date}'")
```

3. 注意字符串类型

参数值始终为字符串类型。在 SQL 中引用时需注意是否需要加引号：

```
-- 字符串类型字段：加引号
WHERE dt = '${dp_data_dt}'

-- 数值类型字段：不加引号（参数值本身是数字字符串）
WHERE id = ${target_id}
```

在 Notebook 中需显式类型转换：

```
# 字符串参数
env = dlcutils.widgets.get('env')

# 需要数值时
batch_size = int(dlcutils.widgets.get('batch_size'))
threshold = float(dlcutils.widgets.get('threshold'))
```

4. 设置合理的默认值

在开发调试阶段，建议为参数提供默认值，避免调试运行时反复手动输入：

```
import os

# 在 Notebook 中，可以先尝试获取参数，获取不到时使用默认值
dt = dlcutils.widgets.get('dp_data_dt')
```

```
if not dt:
    dt = '2026-06-11'  # 开发用默认值
```

使用限制

- 参数值始终为字符串类型，代码中如需其他类型请显式转换。
- `${param_name}` 在 SQL 中是纯文本替换，不会自动加引号。
- 参数名区分大小写：`dp_data_dt` 和 `DP_DATA_DT` 是两个不同的参数。
- 未定义的参数在调度运行时替换为空字符串，可能导致 SQL 语法错误。

常见问题

Q1：Notebook 中 `dlcutils.widgets.get` 返回 None 或空字符串?

- 检查参数名是否拼写正确（区分大小写）。
- 检查是否在工作流参数或任务参数中定义了该参数。
- 如果是调试运行，检查是否在弹窗中填写了参数值。

Q2：SQL 中 `${param}` 没有被替换?

- 确认参数名拼写正确。
- 确认该参数已在工作流参数或任务参数中定义。
- 如果是在 Studio 中调试，需在弹窗中填写参数值。

Q3：如何在 SQL 中引用上游任务的输出?

不能直接在 SQL 代码中使用 `{{tasks.<name>.values.<key>}}`。正确做法：

1. 在任务参数中定义：`my_param = {{tasks.<name>.values.<key>}}`
2. 在 SQL 中引用：`${my_param}`

详见 参数传递 。

相关文档

- 参数配置
- 参数传递
- 系统内置参数

参数传递

最近更新时间：2026-09-11 20:57:30

参数传递（Parameter Passing）允许下游任务引用上游任务的输出，实现任务之间的数据流转。目前支持 Notebook 和 SQL 任务之间的参数传递。

概述

DataBuddy 中的参数传递有三种典型场景：

场景	上游	下游	引用方式
Notebook → Notebook	Notebook	Notebook	`dlcutils.jobs.taskValues.set/get`
Notebook → SQL	Notebook	SQL	`{{tasks.<name>.values.<key>}}` 在任务参数中引用
SQL → Notebook	SQL	Notebook	`{{tasks.<name>.output.rows / first_row / first_row.<col>}}` 在任务参数中引用

前提条件

- 上游任务已在工作流中创建并配置了输出。
- 下游任务已配置上游依赖。否则下游运行时会报错「无法解析引用」。

操作场景

场景 1：Notebook 任务 → Notebook 任务

上游 Notebook 设置参数

```
my_value = "example_value"
dlcutils.jobs.taskValues.set(key='my_value', value=my_value)
```

下游 Notebook 获取参数

下游 Notebook 任务需要在工作流中将上游设为依赖。然后：

```
value = dlcutils.jobs.taskValues.get(
    taskKey='upstream_task_name',
    key='my_value',
```

```
    debugValue='debugValue'
)
print(value)
# 在 Studio 调试运行时输出 debugValue
# 在 Workflow 调度运行时输出 example_value
```

> **提示**
> `dlcutils.jobs.taskValues.get` 仅在 Workflow 调度运行场景下能取到上游 set 的值；Studio 调试运行返回 `debugValue`。

场景 2：Notebook → SQL 任务

上游 Notebook 设置参数

```
dlcutils.jobs.taskValues.set(key='favorite_food', value='apple')
```

下游 SQL 任务参数配置

在下游 SQL 任务的参数面板配置：

```
task_param = {{tasks.<上游 Notebook 任务名>.values.favorite_food}}
```

下游 SQL 中使用参数

```
SELECT '${task_param}' AS food;
-- 结果: apple
```

场景 3：SQL → Notebook 任务

上游 SQL 任务

```
SELECT 1 AS col_a, 2 AS col_b, 3 AS col_c
UNION
SELECT 4, 5, 6
UNION
SELECT 7, 8, 9;
```

下游任务参数配置

下游任务参数面板可以引用三种粒度：

引用	含义	示例返回
`{{tasks.<name>.output.rows}}`	全部行（JSON 数组）	`[{"col_a":1,"col_b":2,"col_c":3}, ...]`
`{{tasks.<name>.output.first_row}}`	第一行（JSON 对象）	`{"col_a":1,"col_b":2,"col_c":3}`
`{{tasks.<name>.output.first_row.<col>}}`	第一行指定列	`1`

> **提示**
> SQL 输出的限制：最多 1000 行，最大 48 KB。SQL 文件中存在多个 SELECT 语句时，仅第一个 SELECT 的结果作为 output。

下游 Notebook 中获取

```
param_value = dlcutils.widgets.get('task_param')
print(param_value)
# [{"col_a":1,"col_b":2,"col_c":3}, ...]
```

场景 4：For Each 中循环参数传递

For Each 节点的循环参数支持引用上游任务输出：

```
{{tasks.<上游 SQL>.output.rows}}
{{tasks.<上游 Notebook>.values.<key>}}
```

详见 For Each 循环任务 。

运行时校验

上游未运行时

如果只重跑或运行下游任务，**没有同时选中上游任务** ，会发生：

阶段	行为
调度判断	直接失败。

错误信息	执行失败，失败原因展示：无法解析引用：由于任务"upstream_name"未在运行范围内，因此无法访问其输出。
错误码	获取上游传递参数失败 。

重跑场景

重跑范围	行为
仅下游	使用上次成功运行结果中的参数。
包含上游	上游重新运行后，下游使用最新结果。

使用限制

- `dlcutils.jobs.taskValues` 仅在 Notebook 任务中使用；SQL 任务通过 `output.rows` 输出。
- SQL 任务的 `output.rows` 最多 1000 行 / 48 KB；超出部分被截断。
- 参数引用的上游必须是 **直接或间接的上游** ，不能跨链路引用同级或下游任务。
- 不支持跨工作流参数传递，参数传递仅限于同一工作流内的任务。
- 单参数取值长度 ⩽ 2048 字符；上游传递的引用值最大 48 KB（针对 SQL output）或 10 KB（针对 job parameters）。

常见问题

Q1：跨工作流能传递参数吗?

不支持。参数传递仅限于同一工作流内的上下游任务之间。跨工作流（包括嵌套工作流/Run Job 场景）暂不支持参数传递。

Q2：上游 Notebook 中 `dlcutils.jobs.taskValues.set` 设置了参数，但下游报错获取不到?

- 检查上游是否在 Workflow 中而不是 Studio 中运行。Studio 调试时不会写入 taskValues。
- 检查下游是否将上游配置为依赖。
- 检查 `dlcutils.jobs.taskValues.set` 中 key 与下游引用的 key 大小写一致。

Q3：如何让上游 SQL 输出多行数据被下游 For Each 循环使用?

配置 For Each 节点的「循环参数」为 `{{tasks.<上游 SQL>.output.rows}}` 。SQL 任务输出的每一行作为一次循环。详见 For Each 循环任务 。

相关文档

- 参数配置
- 参数使用
- 系统内置参数

- 嵌套工作流
- For Each 循环任务

系统内置参数

最近更新时间：2026-09-11 20:57:31

系统内置参数（Built-in Parameters）由 DataBuddy 自动维护，可在工作流参数、任务参数、调度参数中引用，常用于按运行时间动态过滤数据、记录运行 ID、追溯上下游执行情况等。本文档作为 **系统内置参数参考手册**。

引用语法

```
{{<scope>.<field>[.<sub_field>]}}
```

例如：

引用	含义
`{{workflow.id}}`	工作流 ID
`{{workflow.start_time.iso_date}}`	工作流实例数据时间的 ISO 8601 日期格式
`{{tasks.upstream_sql.output.first_row.col_a}}`	上游 SQL 任务输出第一行 `col_a` 列的值

工作流（Workflow）参数

标识

参数	含义
`{{workflow.id}}`	工作流 ID
`{{workflow.name}}`	工作流名称
`{{workflow.run_id}}`	工作流运行 ID（每次运行唯一）
`{{workflow.repair_count}}`	第几次重跑

工作流开始时间（start_time）

开始时间为工作流运行的开始时间，UTC+0。

参数	含义

`{{workflow.start_time.year}}`	年份（如 `2026`）
`{{workflow.start_time.month}}`	月（如 `06`）
`{{workflow.start_time.day}}`	日（如 `15`）
`{{workflow.start_time.hour}}`	小时
`{{workflow.start_time.minute}}`	分钟
`{{workflow.start_time.second}}`	秒
`{{workflow.start_time.is_weekday}}`	是否工作日（`true` / `false`）
`{{workflow.start_time.iso_date}}`	ISO 日期 `YYYY-MM-DD`
`{{workflow.start_time.iso_datetime}}`	ISO 日期时间 `YYYY-MM-DDTHH:mm:ss`
`{{workflow.start_time.iso_weekday}}`	ISO 星期几（`1`-`7`，1 表示周一）
`{{workflow.start_time.timestamp_ms}}`	毫秒级时间戳

工作流计划调度时间（trigger.time）

计划调度时间为调度配置的下次执行时间，UTC+0。

参数	含义
`{{workflow.trigger.time.year}}`	年份
`{{workflow.trigger.time.month}}`	月
`{{workflow.trigger.time.day}}`	日
`{{workflow.trigger.time.hour}}`	小时
`{{workflow.trigger.time.minute}}`	分钟
`{{workflow.trigger.time.second}}`	秒
`{{workflow.trigger.time.is_weekday}}`	是否工作日
`{{workflow.trigger.time.iso_date}}`	ISO 日期
`{{workflow.trigger.time.iso_datetime}}`	ISO 日期时间

{{workflow.trigger.time.iso_weekday}}	ISO 星期几
{{workflow.trigger.time.timestamp_ms}}	毫秒级时间戳

工作空间（Workspace）参数

参数	含义
{{workspace.id}}	工作空间 ID
{{workspace.url}}	工作空间 URL

任务（Task）参数

当前任务

参数	含义
{{task.name}}	任务名称
{{task.run_id}}	任务运行 ID
{{task.execution_count}}	当前任务运行的次数（含重试和重跑）
{{task.path}}	当前 Notebook / Python File 任务引用的文件路径

上游任务（指定 task_name）

> **注意**
> 「上游」必须是当前任务的直接或间接上游，且已配置依赖关系。

> **提示**
> 嵌套多层工作流时，最外层上游传递的参数在最内层可以使用。

参数	含义
{{tasks.<task_name>.run_id}}	上游任务运行 ID
{{tasks.<task_name>.result_state}}	运行状态：success / failed / excluded / canceled / evicted / timeout / upstream_canceled / upstream_evicted / upstream_failed

`{{tasks.<task_name>.error_code}}`	任务错误码（成功时为空字符串）
`{{tasks.<task_name>.execution_count}}`	上游任务运行次数
`{{tasks.<task_name>.path}}`	上游 Notebook 任务的文件路径
`{{tasks.<task_name>.values.<value_name>}}`	上游 Notebook 通过 `dlcutils.jobs.taskValues.set` 设置的值
`{{tasks.<task_name>.output.rows}}`	上游 SQL 任务的全部输出行（JSON 数组）
`{{tasks.<task_name>.output.first_row}}`	上游 SQL 任务输出的第一行
`{{tasks.<task_name>.output.first_row.<column_alias>}}`	上游 SQL 输出第一行的指定列
`{{tasks.<task_name>.output.alert_state}}`	上游 SQL 告警任务的状态：`UNKNOWN` / `OK` / `TRIGGERED`

For Each 循环参数（仅在 For Each 内部节点使用）

参数	含义
`{{loopDataArray}}`	整个循环参数数组
`{{loopDataCurrent}}`	本次循环的元素
`{{loopDataCurrent.<key>}}`	本次循环元素中指定 key 的值
`{{loopTimes}}`	当前循环次数（从 1 开始）

详见 For Each 循环任务 。

时间格式说明

所有以 UTC+0 为基准的时间字段含义：

字段	取值范围
`year`	4 位数字（如 `2026`）
`month`	1-12
`day`	1-31
`hour`	0-23
`minute`	0-59
`second`	0-59
`is_weekday`	`true` / `false`
`iso_date`	`YYYY-MM-DD`
`iso_datetime`	`YYYY-MM-DDTHH:mm:ss`
`iso_weekday`	1-7（1 表示周一，7 表示周日）
`timestamp_ms`	13 位毫秒级时间戳

引用建议

1. 在工作流参数中定义、SQL/Notebook 中引用

推荐做法：在工作流参数中将系统内置参数封装成业务参数，代码中只引用业务参数。便于后续调整与日志可读性。

```
# 工作流参数
dp_data_dt = {{workflow.start_time.iso_date}}

# SQL 中引用
WHERE dt = '${dp_data_dt}'

# Notebook 中引用
dt = dlcutils.widgets.get('dp_data_dt')
```

2. 在 SQL output 与下游传递

```
# 下游任务参数
task_param = {{tasks.upstream_sql.output.first_row.dt}}
```

```
# 下游 SQL 中
WHERE dt = '${task_param}'
```

时区说明

- `start_time` / `trigger.time` 系列字段以 UTC+0 为基准。
- 如需其他时区，请通过工作流参数显式拼接（推荐使用 ISO 字段、时区偏移）。
- 「完成时间」告警按调度时区计算，详见 工作流与任务的告警 。

相关文档

- 参数配置
- 参数使用
- 参数传递
- 调度与触发器

运维与监控
工作流与任务的运行监控

最近更新时间：2026-09-11 20:57:31

工作流与任务的运行监控用于查看历次运行的状态、时长、参数与异常情况，是运维过程中最高频的入口。本文档覆盖工作流运行列表、工作流运行详情、任务运行详情三层视图。

概述

DataBuddy 提供以下运行监控视图：

视图	入口	主要用途
工作流运行列表	工作流-工作流运行	全局视角查看所有工作流的运行情况
工作流详情页 – 运行记录	工作流-工作流	查看单工作流的运行历史与时长趋势
工作流运行详情	列表运行 ID/工作流运行甘特图跳转	查看单次运行的 DAG 图、列表、甘特图
任务运行详情	DAG 节点/列表名称/甘特图名称跳转	查看单次任务的运行结果、运行日志、任务洞察、Spark UI

前提条件

- 已加入工作空间，且对目标工作流具备「查看」及以上权限。
- 工作流已至少运行过一次。

操作步骤

步骤 1：查看工作流运行列表（全局）

1. 在左侧导航选择 **工作流 > 工作流运行** ，进入运行列表页。
2. 顶部展示筛选框，支持同时对图和列表进行筛选。
3. 顶部图表展示：
   - **运行数量变化趋势图**： 堆积图，按时间展示不同状态的运行数量。
   - **Top 5 错误码**： 条形图，单击错误码可快速筛选当前列表。
1. 下方展示空间下有查看权限的运行实例列表。

筛选项

筛选项	说明
工作流名称	输入工作流名称，支持模糊搜索
开始时间/结束时间	时间选择器，精确到分钟，可选范围近 60 天，默认近 3 天
运行账号	根据运行账号筛选，支持多选

列表字段

字段	说明
工作流名称	单击跳转到工作流详情页
运行流 ID	单击跳转到运行详情页
开始时间 / 结束时间	格式 `YYYY-MM-DD HH:MM:SS`
运行状态	见下方「运行状态」
运行时长	格式 `6m3s`
触发方式	手动触发 / 调度触发 / 持续运行
错误码	失败时显示
运行参数	格式 `name:sys;age:20`
操作	重跑 / 终止 / 删除（详见 工作流与任务的运维操作 ）

> **提示**
> 工作流运行列表数据仅保留 60 天。

步骤 2：查看运行记录（单工作流）

1. 打开目标工作流详情页。
2. 切换到 **运行记录** Tab。
3. 顶部展示筛选框，支持同时对图和列表进行筛选。
4. 上方显示「运行时间变化趋势图」，下方显示运行实例列表。

筛选项

筛选项	说明
开始时间/结束时间	时间选择器，精确到分钟，可选范围近 60 天，默认近 3 天

运行账号	根据运行账号筛选，支持多选

列表字段

字段	说明
运行流 ID	单击跳转到运行详情页
开始时间 / 结束时间	格式 `YYYY-MM-DD HH:MM:SS`
运行状态	见下方「运行状态」
运行时长	格式 `6m3s`
触发方式	手动触发 / 调度触发 / 持续运行
错误码	失败时显示
运行参数	格式 `name:sys;age:20`
操作	重跑 / 终止 / 删除（详见 工作流与任务的运维操作 ）

> **提示**
> 列表展示字段可根据需求勾选或取消勾选调整，默认展示所有字段，操作按钮在列表右上角。

步骤 3：查看工作流运行详情

工作流运行列表中单击 **运行流 ID /工作流运行甘特图** 跳转到运行详情页。运行详情提供 3 种视图：

3.1 图模式（DAG）

DAG 图展示工作流下所有任务实例和上下游依赖关系。每个节点的颜色表示状态：

颜色	状态
绿色	成功
红色	失败 / 上游失败
蓝色	运行中
灰色	等待中 / 排除运行
黄色	失败重试

操作	说明

节点 hover	显示任务详细信息（运行 ID、运行账号、状态、时长、错误码等）
节点单击	跳转到任务运行详情页
工具栏	放大 / 缩小 / 重置缩放到 100% / 搜索 / 全屏 / 刷新

3.2 列表模式

以表格形式显示当前运行下所有任务实例，支持按照任务名称搜索。列字段：

字段	说明
任务名称	单击跳转到任务运行详情
运行 ID	任务运行的唯一标识
任务类型	Notebook / SQL / Python / 离线接入 / IF ELSE / For Each / 嵌套工作流 / 数据质量
开始时间 / 结束时间	实际执行时间
运行状态	见下方「任务运行状态」
运行时长	格式 `6m3s`
触发方式	手动触发 / 调度触发 / 持续运行
重试次数	当前任务的重试次数
计算资源	实际使用的资源组
源文件路径	单击跳转到对应文件

3.3 甘特图

横向时间轴展示每个任务的开始–结束区间：

颜色	状态
绿色	成功
红色	失败
灰色	排除运行

3.4 运行信息（右侧侧栏）

区域	字段
基础信息	工作流 ID / 工作流运行 ID / 触发方式 / 状态 / 开始时间 / 结束时间 / 等待时长 / 运行时长
计算资源	工作流下任务关联的资源组
参数配置	本次运行的参数

步骤 4：查看任务运行详情

工作流运行详情页 DAG 节点、列表名称、甘特图名称跳转到任务运行详情。任务详情页 Tab：

Tab	说明
运行结果	展示任务代码与结果
运行日志	错误码 + 错误信息 + 日志
任务洞察	累计 CPU 时、扫描数据大小、Shuffle、输出行数等
Spark UI	单击按钮跳转到 Spark UI

运行日志区操作

操作	说明
刷新	选择刷新频率（如手动刷新、每 10 秒自动刷新）
自动换行	长行自动换行
搜索	搜索日志关键字，支持快速跳转到匹配的日志
下载日志	下载完整日志到本地

任务运行详情公共信息（右侧侧栏）

区域	字段
基础信息	工作流 ID / 工作流运行 ID / 任务运行 ID /运行账号/ 触发方式 / 状态 / 开始时间 / 结束时间 / 等待时长 / 运行时长
文件	引用的文件路径，单击跳转到引用文件位置
计算资源	实际使用的资源组与配置

参数配置	本次任务的参数

步骤 5：执行选择（同任务多次运行）

如同一任务在一次运行中执行了多次，任务运行详情页顶部支持选择查看任意一次执行：

- 默认选中最新一次执行。
- 按运行开始时间倒序排列。

状态说明

工作流运行状态

状态	说明
等待中	所有任务都处于等待状态。
运行中	至少一个任务在运行。
成功	所有任务都成功或被排除运行。
失败	所有任务到达终态，且至少一个失败。
跳过运行	工作流整体因并发限制被跳过。
终止中	工作流正在终止。

任务运行状态

状态	说明	是否终态
等待工作流并发	工作流并发达上限，等待其他工作流释放	否
等待上游	上游任务未到达终态	否
等待任务并发	项目级任务并发上限达 2000	否
等待资源	等待调度 / 引擎资源	否
运行中	任务执行中	否
终止中	任务正在终止	否
失败重试	任务等待失败后重试	否
跳过运行	因并发限制跳过	是

成功	任务运行成功	是
失败	任务运行失败	是
上游失败	上游任务为失败状态导致下游失败	是
排除运行	因不满足运行条件或上游条件分支被排除	是

使用限制

- 运行记录数据仅保留 60 天。
- 列配置（展示哪些列、列顺序）会保留到下次进入。
- 任务运行详情 Spark UI 通过新开浏览器 Tab 跳转。

常见问题

Q1：DAG 中某节点为灰色「排除运行」是什么意思?

该任务因上游条件节点的判断结果不满足，自动跳过执行。详见 IF ELSE 条件任务 。

Q2：任务长时间停在「等待资源」怎么办?

- 在任务运行详情页查看 任务的智能诊断 ，确认是排队、资源不足、配额超限还是其他原因。
- 进入 计算资源概述 查看资源组负载。

Q3：日志查询过大不显示完整内容?

单击 **下载日志** 获取完整日志。

相关文档

- 工作流与任务的运维操作
- 工作流与任务的告警
- 任务的智能诊断
- IF ELSE 条件任务
- 计算资源概述

工作流与任务的运维操作

最近更新时间：2026-09-11 20:57:31

运维操作是排障与日常运维的核心动作。本文档统一介绍工作流和任务在不同状态下支持的运维操作。

概述

运维操作分为以下几类：

操作	适用对象	行为
运行	工作流	立即触发一次工作流运行
高级运行	工作流	自定义参数和执行任务范围
重跑	工作流运行 / 任务运行	重新执行已完成的实例
终止	工作流运行 / 任务运行	终止正在执行或等待中的实例
删除	工作流运行 / 工作流	删除运行记录或工作流
全部终止	工作流所有运行	终止当前工作流的所有运行实例
终止全部等待中运行	工作流所有等待中运行	仅终止处于等待状态的运行

前提条件

- 对目标工作流具备「运行」（重跑、终止）或「管理」（删除）权限。
- 操作账号具备工作流 Run As 账号对应的资源访问权限。

操作步骤

步骤 1：运行、高级运行

运行

入口	行为
工作流列表行尾「运行」	直接触发一次运行，使用已保存的参数
工作流详情页操作栏「运行」	同上
工作流详情页操作栏「重新启动运行」（持续运行模式）	终止当前运行并重新启动；后续仍按持续运行调度

成功后展示提示： 运行工作流"{工作流名称}"成功 查看详情 。

高级运行

入口	行为
工作流列表行尾「高级运行」	直接触发一次运行，使用已保存的参数
工作流详情页操作栏「高级运行」	同上

1. 单击**高级运行** 打开弹窗。
2. **左侧 DAG 图**：单击节点选中或取消，决定本次运行执行哪些任务。默认全部选中。
3. **右侧参数配置**：
   - 默认展示工作流已配置的参数。Key 不可改、Value 可改。
   - 可单击 **+ 添加参数** 增加临时参数。
   - 临时参数仅本次运行生效。
1. 单击**运行** 触发运行。

步骤 2：重跑

重跑用于重新执行某次运行（含任务粒度）。重跑对象决定行为：

重跑工作流运行

1. 在工作流运行列表或运行详情页单击 **重跑** 。
2. 弹窗与高级运行类似：
   - DAG 图中默认选中失败的节点。
   - 参数默认填充本次运行使用的参数，可调整。
1. 单击运行后新起执行。

重跑任务运行

1. 在任务运行详情页单击 **重跑** 。
2. 弹窗与重跑工作流类似，但仅选中当前任务。

重跑限制

场景	行为
工作流运行未到终态	单个任务不允许重跑（按钮禁用）
工作流依赖关系发生变更	重跑按钮禁用，hover 提示「工作流中任务依赖关系发生变更，不支持重跑操作」
For Each 内部循环节	不支持单次循环重跑，需要重跑整个 For Each 节点

点	

重跑参数与代码

- **代码**：使用最新版本（即任务最近一次保存的代码）。
- **配置**（除依赖关系外）：使用最新配置。
- **依赖关系**：依赖关系变更重跑禁用。

若任务配置了告警，重跑也会触发告警。

步骤 3：终止

终止用于停止正在执行或等待中的实例。

终止规则

状态	是否可终止
等待中 / 运行中 / 失败重试	✓
已成功 / 已失败 / 跳过运行	✗
终止中	✗（已发起终止）

工作流级终止

操作	说明
全部终止	终止当前工作流的所有运行（含运行中和等待中）
终止全部等待中运行	仅终止处于等待中的运行

二次确认弹窗：`确定终止工作流"{工作流名称}"的所有执行/所有等待中运行。`

任务级终止

操作	说明
终止	终止当前任务的运行（含运行中和等待中）

二次确认弹窗：`是否确认终止当前任务?`

步骤 4：删除

删除工作流运行

1. 仅运行到达终态后才能删除。非终态时按钮禁用。

2. 二次确认弹窗：确定删除工作流运行 {运行 ID}，删除后工作流运行记录将无法找回。

删除工作流

1. 工作流详情页、列表行尾单击 **删除** 。
2. 二次确认弹窗：确定删除工作流"{工作流名称}"，删除后工作流及运行记录将无法找回。

> **警告**
> 删除不可恢复。删除工作流将同步删除该工作流下的所有运行记录。

步骤 5：常见操作组合

场景	推荐操作
任务失败后修改代码再跑	修改代码 → 在工作流运行详情中重跑失败任务
配置错误想从头重跑	在工作流运行列表中单击「重跑」并选中所有任务
任务长时间不释放资源	终止 → 排查资源问题 → 重跑
调度产生过多无效运行	暂停调度 → 删除运行记录 → 修复后启动调度

状态变化矩阵

工作流运行操作矩阵

状态	重跑	终止	删除
等待中	✗	✓	✗
运行中	✗	✓	✗
成功 / 失败 / 跳过运行	✓	✗	✓
终止中	✗	✗	✗

任务运行操作矩阵

状态	重跑	终止
等待工作流并发 / 等待上游 / 等待任务并发 / 等待资源	✗	✓
运行中	✗	✓
失败重试	✗	✓

终止中	✕	✕
成功 / 失败 / 上游失败 / 跳过运行 / 排除运行	✓	✕

失败与超时策略联动

任务运行的「重试」与「超时」策略详见 配置任务 的「失败与超时策略」。在重跑时，重试策略仍生效；超时阈值的计算从重跑开始时间重新计算。

使用限制

- 删除工作流必须先终止所有运行；正在运行、等待中的工作流不能删除。
- For Each 内部循环节点不支持单次重跑。

常见问题

Q1：重跑时如何选中「失败任务及其上游」？

- 在重跑弹窗的 DAG 图中，按住 Shift 单击节点选择多个，或使用框选工具一次选中链路。
- 如重跑选项支持「重跑实例范围 = 当前实例及上游实例」（部分场景），可在弹窗中切换。

Q2：工作流删除后能恢复吗？

不能。工作流及其运行记录在删除后立即不可恢复。请确认无业务依赖再操作。

Q3：重跑时上游任务参数被修改，下游引用是否使用新参数？

- 不重跑上游：下游使用上次成功运行结果中的参数。
- 重跑包含上游：下游使用上游最新结果中的参数。

详见 参数传递 的「重跑场景」。

Q4：「全部终止」与「终止」的区别？

- **终止**：在某次运行的列表行操作，仅终止该次运行。
- **全部终止**：在工作流详情页操作，终止该工作流下所有运行（含运行中和等待中）。

相关文档

- 工作流与任务的运行监控
- 工作流与任务的告警
- 任务的智能诊断
- 参数传递

工作流与任务的告警

最近更新时间：2026-09-11 20:57:31

工作流与任务的告警是 DataBuddy 中的统一通知能力。**本文档是告警的中心文档**：所有任务类型的告警配置规则一致，本文档详细展开；其他任务文档（Notebook / Python / SQL / 离线接入、嵌套工作流、质量监控等）只做简表、跳转链接。

概述

工作流与任务可以独立配置告警，互不干扰。告警由 **告警条件、接收渠道、免打扰规则** 三部分组成。

分类	工作流告警	任务告警
启动通知	✓	✓
成功通知	✓	✓
失败通知	✓	✓
监控指标告警	✓（运行时长 / 完成时间）	✓（运行时长 / 等待时长 / 完成时间）
接收渠道	Email / Webhook / Teams / Slack	Email / Webhook / Teams / Slack
免打扰：手动终止	✓	✓
免打扰：最后一次重试前	–	✓

前提条件

- 已是工作流所有者或具备「管理」权限。
- 工作空间已配置接收渠道（除直接填写邮箱外，Webhook / Teams / Slack 需提前在 设置通知渠道 中配置）。

操作步骤

步骤 1：进入告警配置

工作流告警

1. 打开工作流详情页。
2. 在右侧抽屉单击 **告警配置 > 添加告警** 。

任务告警

1. 在工作流编辑器画布中单击目标任务。
2. 任务配置面板中找到 **告警配置 > 添加告警** 。

步骤 2：添加通知渠道

渠道	配置方式
Email	选择个人邮箱渠道，在输入框处手动输入 Email 地址，不支持选择已配置的 Email 渠道（工作空间 > 通知管理中添加的 Email 不用于告警渠道引用）。
Webhook	选择已配置的 Webhook 渠道。
Teams	选择已配置的 Teams 渠道。
Slack	选择已配置的 Slack 渠道。

添加系统渠道

如下拉中没有合适选项：

1. 单击 **+ 添加发送方式** 。
2. 跳转到 **工作空间设置 > 通知管理** 配置新渠道。
3. 完成后返回告警配置，下拉中出现新渠道。

步骤 3：选择告警条件

至少勾选一项：

条件	说明
启动	工作流 / 任务开始运行时发送通知。
成功	工作流 / 任务运行成功时发送通知。
失败	工作流 / 任务运行失败时发送通知。**默认勾选** 。
监控指标	当监控指标触发警告或超时时发送通知。需要先配置监控指标阈值才能勾选。

> ⓘ **注意**
> 未先配置监控指标阈值时，「监控指标」选项不可勾选。请先在「监控指标」中设置时间阈值。

步骤 4：配置监控指标

在任务、工作流的 **监控指标** 区域单击 **添加** ，弹出指标配置弹窗。每种指标只能配置一条规则；任意指标触发即视为告警条件命中。

指标	说明	工作流	任务
运行时长	任务运行的持续时长。	✓	✓
等待时长	工作流触发后，等待上游或资源的时长。	–	✓
完成时间	任务运行完成的具体时间点（仅周期运行）。	✓	✓

每条指标配置：

阈值	含义
警告阈值	超过此值触发告警（关联告警条件 **监控指标** ）。
超时阈值	超过此值任务失败/重试，需要联动配置失败与超时策略。

> ⓘ **注意**
> 超时阈值必须 ≥ 警告阈值，否则无法保存。

阈值默认值与不生效场景

指标	默认值	不生效场景
运行时长	`00h00m00s`	默认值不生效，必须手动改为大于 0 的值才会触发监控。
等待时长	`00h00m00s`	同上。
完成时间	空	空值不生效，必须填写具体时间点才会触发监控。

步骤 5：勾选免打扰

免打扰	说明	工作流	任务
手动终止时免打扰	任务被手动终止时不发送通知。	✓	✓
最后一次重试前免打扰	重试期间不发送通知，仅在失败与超时策略中配置的最后一次重试失败时通知。	–	✓

步骤 6：保存

单击 **保存** ，告警配置生效。配置完成后，工作流、任务详情页 **告警配置** 区域展示：

- 接收渠道（用户组名称、渠道图标）。
- 告警策略。

参数说明

告警条件触发逻辑

条件	触发时机
启动	工作流 / 任务从「等待中」进入「运行中」时一次性触发。
成功	工作流 / 任务到达终态「成功」时触发。
失败	工作流 / 任务到达终态「失败」时触发；包含被超时阈值置失败的场景。
监控指标	监控指标超过警告阈值时一次性触发。

接收渠道附加规则

项	规则
单告警弹窗内不可重复配置同一邮箱	配置时校验，重复时提示「此电子邮箱已存在」。
单告警弹窗内不可重复配置同一系统渠道	已被配置的渠道在下拉中隐藏。
单工作流 / 任务告警条数上限	50 条。

完成时间告警的时区

完成时间告警的触发时间点与工作流的调度时区保持一致。例如调度时区设置为美国东部时间，完成时间设置为 09:00，告警按美国东部时间 09:00 触发。

使用限制

- 仅 Email 渠道支持直接填写个人渠道；系统渠道需在 设置通知渠道 配置。
- 监控指标默认值（00h00m00s 或空）会被系统过滤，告警实际不生效。请填写真实阈值。
- 告警条件中的 **监控指标** 选项需先配置监控指标阈值。
- 告警上限为 50 条、工作流（或任务）；超出后无法添加。

常见问题

Q1：为什么任务失败但没收到告警?

- 检查告警配置中是否勾选 **失败** 。
- 检查是否被 **手动终止时免打扰 / 最后一次重试前免打扰** 拦截。
- 检查接收渠道是否仍可达（如 Webhook URL 是否过期、Email 是否进入垃圾邮件）。

Q2：监控指标 = 完成时间，调度时区切换后告警时间会变吗?

会。完成时间告警按调度时区计算。如调度时区从中国切到美国，9:00 的告警会按美国时间 9:00 触发。

Q3：是否支持告警去重?

监控指标在每次任务运行期间到达警告阈值时只触发一次告警。不同次运行之间互相独立，不做跨运行去重。

Q4：可以临时关闭某个工作流的所有告警吗?

可以。在告警配置中删除所有告警条目，工作流不会发送任何通知。建议改为暂停个别条目而非删除，便于后续恢复。

相关文档

- 其他配置
- 工作流与任务的运行监控
- 任务的智能诊断
- 设置通知渠道

任务的智能诊断

最近更新时间：2026-09-11 20:42:32

任务的智能诊断（AI Diagnose）基于 Buddy AI 助手，对失败、等待资源、上游失败状态的任务自动分析失败原因、影响范围、修复建议，帮助用户快速定位问题。

概述

智能诊断的覆盖范围：

状态	是否诊断	诊断重点
失败	✓	错误分类、根因、影响范围
等待资源	✓	资源排队 / 不足 / 配额超限 / 并发达上限 / 资源组配置异常 / 任务请求过大
上游失败	✓	失败链路追踪、根因任务定位、影响范围
其他状态（成功、运行中、跳过运行等）	✗	—

前提条件

- 对该任务具备「查看」及以上权限即可触发诊断。
- 工作空间已开通 Buddy。

操作步骤

步骤 1：找到 AI 诊断入口

智能诊断入口分布在多个位置，根据当前查看视图选择合适的入口：

入口位置	触发方式
工作流运行详情 – DAG 图	hover 失败 / 等待资源 / 上游失败状态的节点，弹窗中单击 **AI 诊断**
工作流运行详情 – 列表模式	列表行失败 / 等待资源 / 上游失败状态列后的 **AI 诊断**
工作流运行详情 – 甘特图	甘特图节点 hover 失败 / 等待资源 / 上游失败状态节点，弹窗中的 **AI 诊断**
工作流详情页列表 – 运行记录图	hover 失败 / 等待资源 / 上游失败状态的节点，弹窗中单击 **AI 诊断**

任务运行详情页	顶部操作栏 **AI 诊断** 按钮

步骤 2：触发诊断

1. 单击 **AI 诊断**，自动唤起 Buddy 助手并开始诊断。

> **说明：**
> 当一个任务有多次运行时：
> - 从工作流列表 / DAG / 列表、甘特图入口触发：诊断 **最后一次** 任务运行。
> - 从任务运行详情页入口触发：诊断 **当前所选** 的任务运行。

步骤 3：查看诊断结果

诊断完成后，Buddy 中展示：

区域	内容
实例信息	任务名称、任务运行 ID、当前状态等
问题摘要	问题分类
根因分析	根因结论 + 详细分析过程
影响范围	直接下游任务个数及列表（支持展开）
修复建议	AI 给出的修复方案

步骤 4：诊断反馈

诊断结果右侧支持：

操作	说明
有帮助	标记诊断有用，反馈数据用于模型监督学习。
不准确	标记诊断不准确。
重新诊断	重新运行诊断。

反馈数据有助于 Buddy 持续优化诊断效果。

权限说明

权限	触发诊断	执行修复操作

查看	✓	✗（仅查看建议）
运行 / 管理	✓	✓

如建议涉及修改任务配置或重跑，且当前用户仅有「查看」权限，需「请联系管理员处理」。

使用限制

- 智能诊断仅对「失败、等待资源、上游失败」三种状态生效。
- 诊断结果由 AI 生成，仅供参考；建议人工复核。
- 历史失败任务的日志若已被清理（超过保留期）会导致诊断信息不完整。

常见问题

Q1：诊断给出「资源排队」建议但实际资源充足，为什么？

可能资源组负载已经在诊断完成后下降。建议：

- 单击 **重新诊断** 获取最新状态。
- 进入 计算资源概述 直接确认资源组负载。

Q2：上游失败诊断中能直接重跑根因任务吗？

Buddy 中可以基于诊断结果发起重跑。重跑后续行为与手动重跑一致，详见 工作流与任务的运维操作 。

Q3：智能诊断对国际站、海外用户支持哪些语言？

Buddy 支持中英文输入和输出。任务名、错误码等技术字段保持原样，便于排错。

Q4：诊断不出结果怎么办？

- 检查 Buddy 是否正常加载（有时需要重新登录）。
- 检查任务日志是否完整：日志被截断或保留过期时诊断准确率会下降。
- 可单击 **重新诊断** 重试；多次失败时联系工作空间管理员。

相关文档

- 工作流与任务的运行监控
- 工作流与任务的运维操作
- 工作流与任务的告警
- 工作流排错与诊断

CI/CD
CD 概述

最近更新时间：2026-09-11 20:57:31

什么是 CI/CD

CI/CD（持续集成、持续交付）是一种通过自动化流水线实现短周期、高频率交付软件的开发实践。在 DataBuddy 中，CI/CD 使数据工程师能够像管理软件代码一样管理数据工作流：通过 Bundle 将工作流配置文件化、版本化，结合 Git 实现多环境的自动化部署。

核心能力

能力	说明
Bundle 包管理	将工作流、任务、参数等资源定义为 YAML 配置文件，实现 Infrastructure as Code（基础设施即代码）
多环境部署	通过 Targets 配置（dev / staging / prod）实现一套配置多环境部署，每个环境可使用独立的 Workspace 和变量
CLI 命令驱动	提供 `databuddy bundle` 系列命令，支持初始化、验证、部署、运行、销毁等完整生命周期操作
变量覆盖	支持通过 `--var` 参数在命令行中动态覆盖变量值，无需修改配置文件
Git 联动	支持将 Bundle 项目托管到 Git 仓库，结合 CI/CD 流水线（GitHub Actions / GitLab CI）实现自动化部署
版本追溯	每次部署自动记录部署状态和 Git 元信息，支持回溯和审计

关键概念

理解 CI/CD 模块，需要熟悉以下核心概念：

- **Bundle**：DataBuddy 的资产包，包含工作流配置（YAML）、代码文件（Notebook / SQL / Python）和部署元信息。等同于 Databricks 的 Asset Bundles。
- **Target**：部署目标环境，如 dev（开发）、staging（预发布）、prod（生产）。每个 Target 可独立配置 Workspace、变量和权限。
- **Resource**：Bundle 中定义的资源实体，当前支持 Workflow 类型，工作流中可包含 Notebook、SQL、Python、嵌套工作流、数据质量等多种任务类型。
- **Profile**：本地身份认证配置，存储在 `~/.databuddycfg` 中，包含 Workspace 地址和认证凭据。
- **databuddy CLI**：DataBuddy 提供的命令行工具，用于 Bundle 的全生命周期管理。

适用场景

场景	适用方式	推荐功能
多人协作开发	每人使用独立 dev Target，代码合并后统一部署到生产	使用 Bundle 开发
多环境隔离部署	开发 / 测试 / 生产三套环境，通过 Target 变量切换	Bundle 包定义
Git 自动化发布	Push 到特定分支自动触发部署，代码审查通过后自动上线	Git 联动开发
工作流配置版本化	将工作流配置纳入 Git 版本管理，可追溯、可回滚	Bundle 命令

架构概览

DataBuddy CI/CD 的整体工作流程如下：

阶段	操作	说明
1. 安装 CLI	`curl ... | sudo sh`	安装 databuddy 命令行客户端
2. 登录认证	`databuddy auth login`	跳转到 DataBuddy 登录页完成身份认证
3. 初始化 Bundle	`databuddy bundle init`	选择模板（SQL / Notebook / Minimal）创建 Bundle 项目
4. 定义资源	编辑 YAML + 代码	在 `resources/workflow/` 中定义工作流，在 `src/` 中编写代码
5. 验证配置	`databuddy bundle validate`	校验 YAML 语法和资源引用
6. 部署	`databuddy bundle deploy -t dev`	将 Bundle 内容部署到目标环境
7. 运行	`databuddy bundle run workflow <key>`	在目标环境中运行工作流

子功能导航

子功能	说明	文档
实践教程	端到端演示 Bundle 开发和 Git 联动的完整流程	–
Bundle 包定义	YAML 配置文件的结构、字段、多环境配置详解	Bundle 包定义

Bundle 命令	CLI 命令完整参考，含安装、认证、部署、运行等	Bundle 命令

推荐学习路径

如果您是首次使用 CI/CD 功能，建议按以下顺序阅读：

1. **了解核心概念：** 本文
2. **动手实践：** 使用 Bundle 开发（端到端完成一次部署）
3. **掌握配置语法：** Bundle 包定义
4. **查阅命令参考：** Bundle 命令
5. **集成 Git 流水线：** Git 联动开发

相关文档

- Bundle 包定义
- Bundle 命令
- 工作流概述

实践教程

教程：使用 Bundle 开发

最近更新时间：2026-09-11 20:57:31

> **提示**
> 预计用时 15 分钟。学习目标：完成从 Bundle 初始化到工作流部署运行的端到端流程。

场景介绍

Bundle 是 DataBuddy CI/CD 的核心载体，允许您将工作流配置和代码文件组织为一个可版本化、可部署的项目包。本教程演示如何从零开始使用 Bundle 完成一次完整的开发部署流程。

在本教程中，您将：

1. 安装 databuddy CLI 并完成身份认证
2. 初始化一个 Bundle 项目
3. 验证并部署 Bundle 到开发环境
4. 在开发环境运行工作流

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 Workspace 中拥有 **管理** 权限。
- 本地已安装 `curl` 命令（macOS / Linux 默认自带）。

步骤 1：安装 databuddy CLI

根据您的操作系统选择对应的安装命令：

macOS / Linux：

```
curl https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.sh | sudo sh
```

Windows：

```
curl.exe -fsSL https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.bat -o install.bat &&
install.bat
```

安装成功后，验证安装：

```
databuddy -v
```

预期输出：`DataBuddy CLI <版本号>`

步骤 2：登录认证

登录前请先获取 Workspace 地址：登录 DataBuddy 控制台并进入目标 Workspace，直接复制浏览器地址栏的完整 URL，形如 `https://databuddy.cloud.tencent.com/workbench?o=<workspace_id>&r=<region_id>`。其中 `o` 为 Workspace ID，`r` 为地域 ID，均由控制台自动生成，无需手动拼接。

> ⓘ **注意：**
> Workspace 地址必须包含 `o=` 参数（Workspace ID），否则登录失败并提示 `workspace id is empty, 'o' param not found in URL`。

执行登录命令：

```
databuddy auth login
```

终端会依次提示：

1. 显示当前 Profile（默认为 `DEFAULT`）。
2. 输入新的配置名称，直接回车表示沿用当前配置。
3. 输入上面获取的 DataBuddy Workspace 地址。

完成登录后，终端显示：

```
Verification Success
```

登录信息保存在本地配置文件 `~/.databuddycfg` 中，其中的配置名称即后续命令 `-p` 参数要用的 Profile 名称。

> ⓘ **说明：**
> 登录状态保留 7 天。也可以在登录时直接指定 Workspace 地址：`databuddy auth login --host "<workspace-url>"`。地址必须用双引号包裹——使用单引号会导致 URL 解析失败，不加引号会导致身份验证失败。

步骤 3：初始化 Bundle 项目

在本地创建 Bundle 项目：

```
databuddy bundle init -p <profile>
```

其中 `-p <profile>` 指定步骤 2 中登录使用的 Profile 名称，后续所有命令均需携带此参数。

> ⓘ **注意：**
> Profile 名称须与 `~/.databuddycfg` 中的配置名称完全一致（区分大小写），默认配置名为 `DEFAULT`。名称不匹配会提示 `profile <name> not found in file`。

系统会引导您完成交互式配置：

1. **选择模板**：根据开发场景选择模板：
   - `SQL`：适用于 SQL 数据开发任务。
   - `NoteBook`：适用于交互式数据分析与探索。
   - `Minimal`：空项目骨架，适合自定义场景。
1. **输入项目名称**：例如 `yiwu_etl`。
2. **选择计算资源**：输入任务使用的计算资源组名称。

完成后，系统在当前目录创建项目文件夹。以 SQL 模板为例，生成结构如下：

```
yiwu_etl/
├── databuddy.yml            # Bundle 主配置文件
├── src/                     # 代码文件
│   └── sql_example.sql
├── resources/               # 资源定义 YAML
│   └── workflow/
│       └── sql_example.yml
└── README.md                # 项目说明
```

> ⓘ **说明：**
> 不同模板生成的初始目录结构不同：
> - **SQL**：`src/` 下包含 `.sql` 示例文件，`resources/workflow/` 下包含对应的任务定义 `.yml` 文件。
> - **NoteBook**：`src/` 下包含 `.ipynb` 示例文件，`resources/workflow/` 下包含对应的任务定义 `.yml` 文件。
> - **Minimal**：仅包含空的 `src/`、`resources/` 目录和 `databuddy.yml`，不预置任何示例代码。

步骤 4：验证 Bundle 配置

进入项目目录并验证配置：

```
cd yiwu_etl
databuddy bundle validate -t dev -p <profile>
```

验证成功输出：

```
Name: yiwu_etl
Target: dev
Workspace:
  url: <项目链接>
  User: your_name@example.com
  Path: /Workspace/Users/your_name@example.com/.bundle/yiwu_etl/dev

Validation OK!
```

> ⓘ **注意：**
> 如果验证失败，系统会列出错误详情（资源类型、资源名、错误原因）。请根据提示修改 YAML 配置后重新验证。

步骤 5：部署到开发环境

```
databuddy bundle deploy -t dev -p <profile>
```

部署成功输出：

```
Uploading bundle files to
/Workspace/Users/your_name@example.com/.bundle/yiwu_etl/dev/files...
Deploying resources...
Updating deployment state...
Deployment complete!
```

部署完成后，您可以在 DataBuddy 控制台的 **Workflow** 中看到已部署的工作流。

步骤 6：运行工作流

```
databuddy bundle run workflow example_workflow -t dev -p <profile>
```

运行结果会展示各任务的执行状态：

```
task name          | status   | time consuming
extract_data       | success | 12s
transform_data     | success | 8s

Run Summary:
  Number of running tasks: 2
  Number of successes: 2
  Number of failures: 0
```

验证结果

完成所有步骤后：

1. 在 DataBuddy 控制台 **数据工程 > Workflow** 列表中可以看到已部署的工作流，名称带有 `dev_<用户名>_` 前缀。
2. 在工作流详情页可查看运行记录和任务执行日志。
3. 本地 Bundle 项目中的代码文件已同步到 DataBuddy Studio 的 `.bundle/<项目名>/dev/files/` 目录下。

总结

在本教程中，您完成了：

- 安装 databuddy CLI 并完成身份认证
- 初始化 Bundle 项目并理解其目录结构
- 验证、部署 Bundle 到开发环境
- 在开发环境运行工作流并查看结果

下一步

- 进阶：Git 联动开发
- 参考：Bundle 包定义
- 参考：Bundle 命令

教程：Git 联动开发

最近更新时间：2026-09-11 18:24:35

> **提示**
> 预计用时 20 分钟。学习目标：将 Bundle 项目接入 Git 仓库，配置 CI/CD 流水线实现自动化部署。

场景介绍

在团队协作开发中，通常需要通过代码审查（Code Review）和自动化流水线来保证部署质量。本教程演示如何将 Bundle 项目与 Git 仓库联动，实现 Push 到不同分支时自动部署到对应环境。

在本教程中，您将：

1. 将 Bundle 项目初始化为 Git 仓库并配置 `.gitignore`
2. 配置多分支策略（dev → prod）
3. 在 CI/CD 平台配置密钥认证
4. 编写流水线配置文件（以 GitLab CI 和 GitHub Actions 为例）
5. 验证自动化部署流程

前提条件

- 已完成 使用 Bundle 开发 教程，具备 Bundle 基础操作能力。
- 已有 Git 仓库（GitHub / GitLab / 其他 Git 平台）。
- 已获取腾讯云 API 密钥（SecretId + SecretKey），用于 CI/CD 流水线中的非交互式认证。
- 已开通 DataBuddy Studio 功能。

> **提示**
> 如需获取 API 密钥，请登录 腾讯云 CAM 控制台 创建或查看。

使用限制

- CI/CD 流水线中使用密钥认证方式（非交互式），不支持浏览器登录方式。
- 流水线 Runner 环境需为 Linux（macOS / Windows 下安装命令不同，本教程以 Linux 为例）。
- `databuddy bundle deploy -s` 会将变更直接提交到调度（生产环境），请确保已在流水线中配置审批保护。

步骤 1：规划分支策略

本教程以 `dev` 和 `prod` 两个分支为例进行演示，分别对应开发环境和生产环境的 Target：

分支	对应 Target	用途

`dev`	`dev`	开发集成，日常开发合入点
`prod`	`prod`	生产环境，只接受 MR/PR 合入

> **说明：**
> 以上分支策略仅为演示目的。实际使用中，团队可根据自身情况灵活设计。例如：每位开发者使用自己的个人分支进行开发，通过 MR/PR 合入 `dev` 或 `prod` 分支触发对应环境的部署。

步骤 2：初始化 Git 仓库

1. 进入 Bundle 项目目录，初始化 Git 仓库：

```
cd my_bundle
git init
```

2. 创建 `.gitignore` 文件，排除本地状态目录：

```
# DataBuddy CLI 状态目录（自动生成，包含绑定关系和本地状态，不应提交）
.bundle/
```

3. 完成首次提交：

```
git add .
git commit -m "chore: init bundle project"
git remote add origin <你的仓库地址>
git push -u origin dev
```

> **注意：**
> `.bundle/` 目录包含本地绑定状态和认证缓存，**必须** 加入 `.gitignore`，不应提交到 Git 仓库。

步骤 3：配置密钥认证

CI/CD 流水线中无法使用浏览器交互式登录，需要通过密钥方式认证。DataBuddy CLI 通过读取 `~/.databuddycfg` 配置文件获取认证信息。

GitLab CI

在 GitLab 项目中配置 CI/CD 变量：**Settings → CI/CD → Variables**，添加以下 4 个变量：

变量名	说明	选项
`WEDATA_HOST`	DataBuddy Workspace 地址（完整 URL）	Masked
`WEDATA_SECRET_ID`	腾讯云 API 密钥 ID	Masked
`WEDATA_SECRET_KEY`	腾讯云 API 密钥 Key	Masked
`WEDATA_REGION`	地域，如 `ap-beijing-fsi`	—

GitHub Actions

在 GitHub 仓库中配置 Secrets：**Settings** → **Secrets and variables** → **Actions**，添加以下 4 个 Secret：

Secret 名称	说明
`WEDATA_HOST`	DataBuddy Workspace 地址（完整 URL）
`WEDATA_SECRET_ID`	腾讯云 API 密钥 ID
`WEDATA_SECRET_KEY`	腾讯云 API 密钥 Key
`WEDATA_REGION`	地域，如 `ap-guangzhou`

> **警告：**
> 密钥绝不能明文写入配置文件或提交到 Git 仓库。务必通过 CI/CD Variables 或 Secrets 注入。

步骤 4：编写 CI/CD 流水线配置

GitLab CI

在仓库根目录创建 `.gitlab-ci.yml`：

```
stages:
  - validate
  - deploy

# ---- 全局配置 ----
variables:
  BUNDLE_DIR: my_bundle    # 修改为实际的 bundle 目录名

before_script:
```

```
  # 安装 DataBuddy CLI（Go 二进制版本）
  - curl 'https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.sh' | sudo sh || true
  - databuddy --version
  # 写入认证配置文件（密钥从 CI/CD Variables 注入）
  - |
    cat << EOF > "$HOME/.databuddycfg"
    host = ${WEDATA_HOST}
    secret_id = ${WEDATA_SECRET_ID}
    secret_key = ${WEDATA_SECRET_KEY}
    region = ${WEDATA_REGION}
    EOF
  - cd ${BUNDLE_DIR}

# ========== dev 分支：部署到开发环境 ==========

validate-dev:
  stage: validate
  script:
    - databuddy bundle validate -t dev
  rules:
    - if: '$CI_COMMIT_BRANCH == "dev"'

deploy-dev:
  stage: deploy
  script:
    - databuddy bundle deploy -t dev
  rules:
    - if: '$CI_COMMIT_BRANCH == "dev"'
  environment:
    name: development

# ========== prod 分支：部署到生产环境 ==========

validate-prod:
  stage: validate
  script:
    - databuddy bundle validate -t prod
  rules:
    - if: '$CI_COMMIT_BRANCH == "prod"'
```

```
deploy-prod:
  stage: deploy
  script:
    - databuddy bundle deploy -t prod -s
  rules:
    - if: '$CI_COMMIT_BRANCH == "prod"'
  environment:
    name: production
```

GitHub Actions

在仓库中创建 `.github/workflows/deploy.yml` **：**

```
name: DataBuddy Bundle CI/CD

on:
  push:
    branches: [dev, prod]
  pull_request:
    branches: [prod]

env:
  BUNDLE_DIR: my_bundle    # 修改为实际的 bundle 目录名

jobs:
  # ========== dev 分支：部署到开发环境 ==========
  deploy-dev:
    if: github.ref == 'refs/heads/dev'
    runs-on: ubuntu-latest
    environment: development
    steps:
      - uses: actions/checkout@v4

      - name: Install DataBuddy CLI
        run: |
          curl 'https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.sh' | sudo sh || true
          databuddy --version
```

```
      - name: Configure credentials
        run: |
          cat << EOF > "$HOME/.databuddycfg"
          host = ${{ secrets.WEDATA_HOST }}
          secret_id = ${{ secrets.WEDATA_SECRET_ID }}
          secret_key = ${{ secrets.WEDATA_SECRET_KEY }}
          region = ${{ secrets.WEDATA_REGION }}
          EOF

      - name: Validate
        run: |
          cd ${{ env.BUNDLE_DIR }}
          databuddy bundle validate -t dev

      - name: Deploy
        run: |
          cd ${{ env.BUNDLE_DIR }}
          databuddy bundle deploy -t dev

  # ========== prod 分支：部署到生产环境 ==========
  deploy-prod:
    if: github.ref == 'refs/heads/prod'
    runs-on: ubuntu-latest
    environment: production
    steps:
      - uses: actions/checkout@v4

      - name: Install DataBuddy CLI
        run: |
          curl 'https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.sh' | sudo sh || true
          databuddy --version

      - name: Configure credentials
        run: |
          cat << EOF > "$HOME/.databuddycfg"
          host = ${{ secrets.WEDATA_HOST }}
          secret_id = ${{ secrets.WEDATA_SECRET_ID }}
          secret_key = ${{ secrets.WEDATA_SECRET_KEY }}
          region = ${{ secrets.WEDATA_REGION }}
```

```
          EOF

      - name: Validate
        run: |
          cd ${{ env.BUNDLE_DIR }}
          databuddy bundle validate -t prod

      - name: Deploy
        run: |
          cd ${{ env.BUNDLE_DIR }}
          databuddy bundle deploy -t prod -s
```

> **说明：**
> 建议在 GitHub 上为 `production` 环境配置 Required Reviewers，确保生产部署需要人工审批。

步骤 5：日常开发协作流程

完成流水线配置后，团队成员日常按以下流程协作：

1. 从 dev 分支创建 feature 分支进行开发：

```
git checkout dev
git pull origin dev
git checkout -b feature/add-report-task
```

2. 编辑 Bundle 配置和代码文件，本地校验通过后提交：

```
# 本地校验
databuddy bundle validate -t dev -p <profile>

# 提交代码
git add .
git commit -m "feat(tasks): add daily report task"
git push origin feature/add-report-task
```

3. 在 Git 平台创建 Merge Request / Pull Request，合入 `dev` 分支。流水线自动执行 validate 和 deploy 到开发环境。
4. 在开发环境验证无误后，创建 MR 从 `dev` 合入 `prod`，经 Code Review 通过后合入。流水线自动部署到生产环境。

步骤 6：验证自动化流程

完成以上配置后，执行一次端到端验证：

1. 在 feature 分支修改 Bundle 配置，推送后创建 MR 合入 `dev` 。
2. 检查 CI/CD 流水线是否自动执行 validate + deploy-dev。
3. 在 DataBuddy 控制台的开发环境中确认工作流已更新。
4. 创建 MR 从 `dev` 合入 `prod` ，确认 deploy-prod 流水线执行成功。
5. 在 DataBuddy 控制台的生产环境中确认工作流已部署且处于调度状态。

常见问题

流水线中 validate 失败

检查以下几点：

- 确认 `BUNDLE_DIR` 变量与实际 Bundle 目录名一致。
- 确认 `~/.databuddycfg` 中的 `host` 、 `secret_id` 、 `secret_key` 、 `region` 四个字段均正确注入。
- 确认 CI Runner 网络可访问 DataBuddy API 端点。

流水线中 deploy 失败

- 确认目标 Target（dev / prod）已在 `databuddy.yml` 中正确定义。
- 确认 API 密钥对应的账号在目标 Workspace 中拥有足够权限。

如何回滚

如果生产部署出现问题：

1. 使用 `git revert` 撤回有问题的 commit。
2. 将 revert commit 合入 `prod` 分支。
3. 流水线自动执行 deploy，将远端资源恢复到上一状态。

相关文档

- 使用 Bundle 开发
- Bundle 包定义
- Bundle 命令
- CI/CD 概述

Bundle 包定义

最近更新时间：2026-09-11 18:24:35

概述

Bundle 包是 DataBuddy CI/CD 的核心配置单元，通过 YAML 文件定义工作流、任务、环境变量和部署策略。本文档详细说明 Bundle 的目录结构、配置文件语法和各字段含义。

目录结构

一个标准的 Bundle 项目包含以下目录和文件：

```
my_bundle/
├── databuddy.yml            # Bundle 主配置文件（必须）
├── resources/               # 资源定义目录
│   └── workflow/            # 工作流资源子目录
│       └── workflow_a.yml
├── src/                     # 代码文件目录
│   ├── task_a.sql
│   └── task_b.sql
└── README.md                # 项目说明
```

主配置文件（databuddy.yml）

`databuddy.yml` 是 Bundle 的入口配置文件，定义 Bundle 基础信息、环境配置和变量。

完整示例

```
# Bundle 基本配置
bundle:
  name: my_bundle
  uuid: a97c467e-xxxx-xxxx-xxxx-525400694f7c

# 引入资源文件
include:
  - resources/workflow/*.yml

# 全局变量定义
variables:
  scheduling_resource_group:
```

```
    description: resource_group_name

# 多环境配置
targets:
  dev:
    mode: development
    default: true
    variables:
      scheduling_resource_group: dev_resource_group
    workspace:
      host: https://databuddy.cloud.tencent.com/workbench?o=
<workspace_id>&r=<region_id>

  prod:
    mode: production
    variables:
      scheduling_resource_group: prod_resource_group
    workspace:
      host: https://databuddy.cloud.tencent.com/workbench?o=
<workspace_id>&r=<region_id>

  staging:
    mode: production
    variables:
      scheduling_resource_group: staging_resource_group
    workspace:
      host: https://databuddy.cloud.tencent.com/workbench?o=
<workspace_id>&r=<region_id>
```

配置字段详解

bundle（必填）

参数	说明	是否必填	默认值
`name`	Bundle 项目名称，用于标识项目	是	—
`uuid`	Bundle 唯一标识符，`databuddy bundle init` 时自动生成	是（自动）	自动生成

`wedata_cli_version`	所需的 databuddy CLI 最低版本	否	不限制

include

指定要包含在 Bundle 中的资源配置文件路径，支持通配符：

```
include:
  - resources/workflow/*.yml     # 匹配 resources/workflow/ 下的所有 yml 文件
```

variables（全局变量）

定义可在整个 Bundle 中引用的变量，使用 `${var.<变量名>}` 引用。

参数	说明	是否必填
`description`	变量描述	否
`default`	默认值	否
`type`	变量类型（string）	否

变量可在 Target 中覆盖，实现不同环境使用不同的值。

targets（多环境配置）

每个 Target 代表一个部署环境。

参数	说明	是否必填	默认值
`mode`	部署模式：`development` 或 `production`	否	`development`
`default`	是否为默认 Target（不指定 `-t` 时使用）	否	`false`
`variables`	环境级变量覆盖	否	—
`workspace.host`	目标 Workspace 地址	是	—
`workspace.root_path`	Bundle 文件在 Studio 中的存储路径	否	`.bundle/<name>/<target>`

`permissions`	部署后资源的权限配置	否	Profile 用户为 CAN_MANAGE

development 模式与 production 模式的区别：

行为	development	production
工作流名称前缀	自动添加 `dev_<用户名>_` 前缀	无前缀
文件路径	指向 `.bundle/<name>/dev/files/`	指向任务配置中的引用路径
`root_path` 限制	必须在个人文件夹下（`~/` 开头）	无限制，建议指定独立路径

> ⓘ **注意**
> development 模式下 `root_path` 必须以 `~/` 开头或包含当前用户名，否则 deploy 会报错：`[Error] root path must start with '~/' or contain the current username to ensure uniqueness when using 'mode: development'`

permissions

参数	说明	取值
`user_name`	用户账号	邮箱格式
`level`	权限级别	`CAN_VIEW` / `CAN_RUN` / `CAN_MANAGE` / `IS_OWNER`

资源配置（resources）

资源配置文件放在 `resources/workflow/` 目录下，定义 Bundle 中的工作流及其包含的任务。

Workflow 资源示例

以下是一个包含两个 SQL 任务的工作流配置示例（`resources/workflow/workflow_a.yml`）。不同任务类型（如 SQL、Shell、Python 等）的 `task_type` 及 `task_type_property_list` 字段会有所不同，此处仅以 SQL 任务为例：

```
resources:
  workflows:
    workflow_a:
      name: workflow_a
      execute_user: <your_username>
      trigger:
```

```
    - scheduler_status: ACTIVE
      trigger_mode: TIME_TRIGGER
      scheduler_time_zone: Asia/Shanghai
      start_time: "2026-06-01 00:00:00"
      end_time: "2099-12-31 23:59:59"
      config_mode: COMMON
      cycle_type: DAY_CYCLE
      crontab_expression: 0 0 2 * * ? *
  advance_config:
    queuing_mode: "OFF"
    max_concurrent_num: 1
  task_list:
    - task_name: task_a
      file_path: ../../src/task/task_a.sql
      remote_dir_path: /Workspace/task_a.sql
      resource_group_name: ${var.scheduling_resource_group}
      depend_on_run_condition: ALL_SUCCESS
      task_type:
        task_type_name: SQL
        task_type_property_list:
          - property_key: Source
            property_value: WorkSpace
          - property_key: SqlPath
            property_value: /Workspace/task_a.sql
          - property_key: CodeFileName
            property_value: task_a.sql
      param_list:
        - param_key: bizdate
          param_value: '{{workflow.trigger.time.iso_date}}'
      task_retry_strategy:
        max_retry_time: 3
        retry_between_wait_time: 5
        retry_between_wait_time_unit: SECOND
        task_run_failure_retry_switch: true

    - task_name: task_b
      file_path: ../../src/task/task_b.sql
      remote_dir_path: /Workspace/task_b.sql
      resource_group_name: ${var.scheduling_resource_group}
      depend_on_run_condition: ALL_SUCCESS
```

```
    task_type:
      task_type_name: SQL
      task_type_property_list:
        - property_key: Source
          property_value: WorkSpace
        - property_key: SqlPath
          property_value: /Workspace/task_b.sql
        - property_key: CodeFileName
          property_value: task_b.sql
    param_list:
      - param_key: bizdate
        param_value: "1"
    depend_on_list:
      - task_name: task_a
    task_retry_strategy:
      max_retry_time: 3
      retry_between_wait_time: 5
      retry_between_wait_time_unit: SECOND
      task_run_failure_retry_switch: true
```

对应的 SQL 代码文件放在 `src/task/` 目录中，例如 `src/task/task_a.sql`：

```
INSERT OVERWRITE TABLE target_schema.table_a PARTITION(dt =
'${bizdate}')
SELECT  col_1,
        col_2,
        COALESCE(col_3, 0) AS col_3,
        col_4
FROM    source_schema.table_b
WHERE   dt = '${bizdate}'
AND     col_1 IS NOT NULL
```

Workflow 字段说明

参数	说明	是否必填	默认值
`name`	工作流名称	是	—
`execute_user`	执行用户	否	当前 Profile 用户

`trigger`	触发器配置（调度规则）	否	不启用调度
`advance_config`	高级配置（排队模式、并发数）	否	—
`task_list`	任务列表	是	—

Task 字段说明

参数	说明	是否必填	默认值
`task_name`	任务名称	是	—
`file_path`	本地代码文件的相对路径	是	—
`remote_dir_path`	远程存储路径	是	—
`resource_group_name`	执行资源组名称，支持变量引用	是	—
`depend_on_run_condition`	依赖运行条件（`ALL_SUCCESS` 等）	否	`ALL_SUCCESS`
`task_type`	任务类型配置	是	—
`task_type.task_type_name`	任务类型名称（`SQL` / `Python` / `Notebook` 等）	是	—
`task_type.task_type_property_list`	任务类型属性列表	否	—
`param_list`	参数列表	否	—
`depend_on_list`	依赖的上游任务列表	否	—
`task_retry_strategy`	重试策略	否	—

Trigger 字段说明

参数	说明	是否必填	默认值
`scheduler_status`	调度状态（`ACTIVE` / `INACTIVE`）	否	`INACTIVE`
`trigger_mode`	触发模式（`TIME_TRIGGER`）	否	—

`scheduler_time_zone`	调度时区	否	`Asia/Shanghai`
`start_time`	调度生效开始时间	否	—
`end_time`	调度生效结束时间	否	—
`config_mode`	配置模式（`COMMON`）	否	—
`cycle_type`	周期类型（`DAY_CYCLE` / `HOUR_CYCLE` 等）	否	—
`crontab_expression`	Cron 表达式	否	—

变量引用

在资源配置中可以通过 `${var.<变量名>}` 引用 `databuddy.yml` 中定义的全局变量，例如：

```
resource_group_name: ${var.scheduling_resource_group}
```

不同 Target 下变量值会自动替换为对应环境的配置。

使用限制

- Bundle 中的文件总数不能超过 5,000 个。
- 主配置文件必须命名为 `databuddy.yml`，否则 CLI 无法识别 Bundle 根目录。
- `include` 路径支持通配符 `*`，但不支持递归通配 `**`。
- development 模式下 `root_path` 必须为个人路径。
- 工作流名称在 development 模式下会自动添加前缀，部署后名称为 `dev_<用户名>_<原名>`。

常见问题

Q1：databuddy.yml 中未定义 uuid 怎么办?

`uuid` 在 `databuddy bundle init` 时自动生成。如果手动创建的项目缺少 uuid，执行 `databuddy bundle validate` 时会报错 `[ERROR] bundle id not defined`。建议通过 `init` 命令创建项目。

Q2：如何在不同环境使用不同的资源组?

在 `variables` 中定义变量，然后在不同的 Target 中覆盖：

```
variables:
  scheduling_resource_group:
    description: resource_group_name
```

```
targets:
  dev:
    variables:
      scheduling_resource_group: dev_resource_group
  prod:
    variables:
      scheduling_resource_group: prod_resource_group
```

Q3：resources 目录下没有 yml 文件会怎样?

执行 `databuddy bundle validate` 或 `deploy` 时会报错：`[ERROR] There is no resource yml file defined in bundle.` 请确保 `resources/workflow/` 等子目录中至少包含一个 yml 文件。

相关文档

- CI/CD 概述
- Bundle 命令
- 使用 Bundle 开发
- 调度与触发器

Bundle 命令

最近更新时间：2026-09-11 20:57:31

概述

databuddy CLI 提供了一套完整的命令行工具，覆盖 Bundle 从安装到销毁的全生命周期。本文档作为命令参考手册，列出所有支持的命令及参数说明。

安装、卸载、升级

安装客户端

macOS / Linux：

```
curl https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.sh | sudo sh
```

Windows：

```
curl.exe -fsSL https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.bat -o install.bat &&
install.bat
```

卸载客户端

macOS / Linux：

```
curl https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.sh | sudo sh -s -- --uninstall
```

Windows：

```
curl.exe -fsSL https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.bat -o install.bat &&
install.bat --uninstall
```

升级客户端

macOS / Linux：

```
curl https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.sh | sudo sh -s -- --upgrade
```

Windows：

```
curl.exe -fsSL https://wedata3-platform-bundle-cq-1257305158.cos.ap-
chongqing.myqcloud.com/bundle_sh/install.bat -o install.bat &&
install.bat --upgrade
```

查看版本

```
databuddy -v
databuddy --version
```

身份认证

登录

```
databuddy auth login
databuddy auth login --host "<workspace-url>"
```

参数	说明	是否必填
`--host`	目标 Workspace 地址	否（使用默认 Profile 中的地址）

`<workspace-url>` 为 DataBuddy 控制台中目标 Workspace 的完整地址，登录控制台并进入目标 Workspace 后，从浏览器地址栏复制即可，形如 `https://databuddy.cloud.tencent.com/workbench?o=<workspace_id>&r=<region_id>` 。其中 `o` 为 Workspace ID，`r` 为地域 ID，均由控制台自动生成，无需手动拼接。

> ⓘ **注意：**
> `<workspace-url>` 必须用双引号包裹。使用单引号会导致 URL 解析失败，不加引号会导致身份验证失败。地址中必须包含 `o=` 参数，否则提示 `workspace id is empty, 'o' param not found in URL` 。

直接执行 `databuddy auth login` 时，终端会依次提示输入配置名称（直接回车沿用当前配置）与 Workspace 地址。登录成功后终端显示 `Verification Success` ，登录状态保留 7 天。

使用密钥登录

适用于 CI/CD 流水线等非交互式场景。编辑 `~/.databuddycfg` 配置文件：

```
[default]
host       = https://databuddy.cloud.tencent.com/workbench?o=
<workspace_id>&r=<region_id>
secret_id  = <your_secret_id>
secret_key = <your_secret_key>
project_id = <project_id>
user_name  = <user_name>
user_id    = <user_id>
region     = ap-singapore
is_intl    = false
```

> ⓘ **注意：**
> 密钥文件包含敏感信息，请确保文件权限设置为 `600`（仅当前用户可读写）。

查看认证信息

```
databuddy auth describe
databuddy auth describe -p <profile_name>
databuddy auth describe --host <sub_account>
databuddy auth describe --sensitive
```

参数	说明
`-p`	查看指定 Profile 的认证信息
`--host`	查看指定子账号对应的 Profile 信息
`--sensitive`	输出敏感信息（secret_id 和 secret_key）；未指定时显示为 `******`

Bundle 操作

通用参数

以下参数适用于大部分 `databuddy bundle` 子命令：

参数	说明

`-t, --target <string>`	指定目标 Target（如 dev / staging / prod）
`-p, --profile <string>`	指定使用的 Profile
`--var "<key>=<value>"`	覆盖 Bundle 变量（支持 validate / deploy / summary / destroy）
`--help`	查看命令帮助

初始化 Bundle

```
databuddy bundle init
```

交互式引导创建 Bundle 项目模板。系统依次引导：

1. 选择模板

系统展示内置模板列表，支持上下键选择或输入模板名称：

模板	说明	适用人群
SQL	SQL 加工任务项目骨架，含示例 SQL 文件和工作流配置	数据开发工程师，写 SQL 做 ETL
Notebook	Notebook 任务项目骨架，含示例 ipynb 文件	用 Python 做复杂数据处理
Minimal	最小骨架，仅含 `databuddy.yml` 和空目录结构	想完全自定义的高级用户

2. 配置项目

选择模板后，系统依次询问项目名称和计算资源。`[]` 中为默认值，直接按回车将使用默认配置。Compute 需要手动输入计算资源名称。

3. 完成创建

生成的目录结构示例（SQL 模板）：

```
yiwu_etl/
├── databuddy.yml
├── src/
│   └── sql_example.sql
├── resources/
│   └── workflow/
│       └── sql_example.yml
└── README.md
```

> **说明：**
> `init` 生成的项目无需修改配置即可直接执行 `databuddy bundle deploy`。调度默认为每天 0 点执行（UTC+8），计算资源和运行账号已在初始化时配置完成。

拉取资源（generate）

```
databuddy bundle generate workflow --existing-workflow-id <workflow_id>
```

将 DataBuddy 中已有的工作流配置拉取到本地 Bundle。`workflow` 为资源类型，`--existing-workflow-id` 指定要拉取的工作流 ID。

Flag	说明
`-t, --target`	指定目标 Target
`-p, --profile`	指定 Profile

绑定资源（bind）

```
databuddy bundle deployment bind workflow <resource_key> <workflow_id>
```

将线上已有的工作流与本地 Bundle 中的资源定义绑定。`<resource_key>` 为本地资源标识，`<workflow_id>` 为线上工作流 ID。绑定后执行 `deploy` 可将本地配置同步到该工作流。

`<resource_key>` 取自本地 `resources/workflow/*.yml` 中 `resources` → `workflows` 下的键名。以下配置中的 `<resource_key>` 即为 `workflow_a`：

```
resources:
  workflows:
    workflow_a:           # 该键名即 <resource_key>
      name: workflow_a
```

`<workflow_id>` 可在 DataBuddy 控制台的 **数据工程 > Workflow** 中打开目标工作流，从详情页地址栏或工作流属性中获取。

Flag	说明
`-t, --target`	指定目标 Target
`-p, --profile`	指定 Profile

解除绑定（unbind）

```
databuddy bundle deployment unbind workflow <resource_key>
```

解除本地资源定义与线上工作流的绑定关系。`<resource_key>` 为本地资源标识。

Flag	说明
`-t, --target`	指定目标 Target
`-p, --profile`	指定 Profile

验证 Bundle（validate）

```
databuddy bundle validate
```

校验 Bundle 配置的语法正确性和资源引用完整性。

Flag	说明
`-t, --target`	指定目标 Target
`-p, --profile`	指定 Profile
`--var`	覆盖 Bundle 变量（详见本文「变量覆盖（--var）」）

部署 Bundle（deploy）

```
databuddy bundle deploy
databuddy bundle deploy -t dev
databuddy bundle deploy -t prod -p admin_profile
databuddy bundle deploy --var="database=prod_ods"
```

将 Bundle 中定义的资源部署到目标环境。部署过程依次执行：配置校验 → 文件上传 → 资源部署。如果校验发现 Warning，系统会暂停并询问是否继续。若某个资源部署失败，后续资源仍会继续，最终统一展示失败项。

Flag	说明
`-t, --target`	指定目标 Target
`-p, --profile`	指定 Profile

`--var`	覆盖 Bundle 变量（详见本文「变量覆盖（--var）」）

运行（run）

```
databuddy bundle run workflow <resource_key> -t dev
databuddy bundle run workflow <resource_key> --params="key1=value1" --
params="key2=value2"
```

触发指定工作流的一次性运行。`<resource_key>` 为要运行的工作流资源标识。

Flag	说明
`-t, --target`	指定目标 Target
`-p, --profile`	指定 Profile
`--params`	指定运行时参数值，格式为 `key=value`，支持多次使用

销毁（destroy）

```
databuddy bundle destroy
databuddy bundle destroy -t dev
```

删除目标环境中由该 Bundle 部署的所有资源（工作流 + Studio 文件）。本地 Bundle 文件不受影响。

Flag	说明
`-t, --target`	指定目标 Target
`-p, --profile`	指定 Profile
`--var`	覆盖 Bundle 变量

> **警告：**
> `destroy` 操作不可逆。执行前系统会列出待删除资源并要求确认。

查看摘要（summary）

```
databuddy bundle summary
databuddy bundle summary -t dev
```

```
databuddy bundle summary --var="database=prod_ods"
```

展示当前 Bundle 的部署状态和资源信息。

Flag	说明
`-t, --target`	指定目标 Target
`-p, --profile`	指定 Profile
`--var`	覆盖 Bundle 变量

查看帮助

```
databuddy bundle --help
```

变量覆盖（--var）

在命令行中使用 `--var` 参数可覆盖 `databuddy.yml` 中定义的变量值，无需修改配置文件。

基本语法

```
databuddy bundle <command> --var="<key>=<value>" --var="<key>=<value>"
databuddy bundle <command> --var="<key>=<value>,<key>=<value>"
```

支持的命令

命令	示例
`validate`	`databuddy bundle validate --var="database=prod_ods"`
`deploy`	`databuddy bundle deploy --var="database=prod_ods"`
`summary`	`databuddy bundle summary --var="database=prod_ods"`
`destroy`	`databuddy bundle destroy --var="database=prod_ods"`

传参方式

```
# 单个变量
databuddy bundle deploy --var="database=prod_ods"
```

```
# 多个变量（重复使用 --var）
databuddy bundle deploy --var="database=prod_ods" --var="engine=presto"

# 多个变量（逗号分隔）
databuddy bundle deploy --var="database=prod_ods,engine=presto"
```

优先级

变量值的优先级从高到低：

1. 命令行 `--var` 指定的值（最高）
2. `resources/` 中资源配置里的变量值
3. `databuddy.yml` → `targets` → 当前 Target 中定义的变量值
4. `databuddy.yml` → `variables` 中定义的全局默认值（最低）

命令速查表

命令	用途
`databuddy -v`	查看 CLI 版本
`databuddy auth login`	登录认证
`databuddy auth describe`	查看认证配置
`databuddy bundle init`	初始化 Bundle 项目（支持模板选择）
`databuddy bundle generate workflow --existing-workflow-id <id>`	拉取工作流到本地
`databuddy bundle deployment bind workflow <key> <id>`	绑定工作流
`databuddy bundle deployment unbind workflow <key>`	解绑工作流
`databuddy bundle validate -t <target>`	验证 Bundle 配置
`databuddy bundle deploy -t <target>`	部署 Bundle
`databuddy bundle deploy --var="key=value"`	部署时覆盖变量
`databuddy bundle run workflow <key> -t <target>`	运行工作流

`databuddy bundle run workflow <key> --params="key=value"`	带参数运行工作流
`databuddy bundle destroy -t <target>`	销毁部署
`databuddy bundle summary -t <target>`	查看部署摘要
`databuddy bundle --help`	查看帮助

相关文档

- CI/CD 概述
- Bundle 包定义
- 使用 Bundle 开发
- Git 联动开发