包学包会

DataFlux 入门系列

摘要

从界面操作，到代码编写
全面介绍 DataFlux.f(x)的使用方式

周 逸灵

zyl@jiagouyun.com

目录

1. 前言 4

2. 登录 5

3. 脚本编辑器 6

3.1. 脚本库 7

3.1.1. 添加/配置脚本集 9

3.1.2. 添加/配置脚本 10

3.1.3. 代码编辑器 11

3.1.4. 从帮助按钮获取帮助 15

3.1.5. 代码快速查看面板 16

3.2. 数据源 17

3.2.1. 添加/配置数据源 17

3.2.2. 数据源简易调试面板 20

3.3. 环境变量 23

3.3.1. 添加/配置环境变量 23

3.4. 参考 23

4. 管理 24

4.1. 授权链接 24

4.1.1. 添加/配置授权链接 26

4.2. 自动触发配置 26

4.2.1. 添加/配置自动触发配置 26

4.3. 脚本包导出 27

4.3.1. 执行脚本包导出 27

4.4. 脚本包导入 27

4.4.1. 执行脚本包导入 28

4.5. 脚本库还原 28

4.5.1. 创建脚本库还原点 29

4.6. 官方脚本库 29

5. 函数文档 30

6. 授权链接函数文档 31

7. 设置 33

7.1. 清除缓存 33
7.2. 代码编辑器配置 33
7.3. 实验性功能 33
7.4. 关于 33
8. 脚本编写 34
8.1. 最简单的函数 34
8.2. 编写函数的要求 35
8.2.1. 允许 import 的 Python 内置库 36
8.2.2. 允许 import 的第三方 Python 库 36
8.2.3. 函数名称 37
8.2.4. 函数定义 37
8.2.5. 函数标题 37
8.2.6. 函数文档 37
8.2.7. 函数参数 39
8.2.8. 函数返回值 41
8.3. 使用内置功能 43
8.3.1. 将函数对外暴露（DFF.API） 43
8.3.2. 操作数据源 43
8.3.3. 获取环境变量 53
8.3.4. 输出日志 53
8.4. 附带分类的函数 54
8.4.1. DataFlux 标准数据结构 54
8.4.2. 预测函数 55
8.4.3. 转换函数 55
8.4.4. 行为函数 56
8.4.5. 命令函数 57
8.4.6. 查询函数 57
8.4.7. 检测函数 59
8.5. 调用另一个脚本中的函数 60
8.6. 脚本集、脚本、函数编排设计 62
8.6.1. 按照用途、类型合理划分脚本集和脚本 62
8.6.2. 单个脚本的代码量及依赖链 62
8.7. 常见问题处理方式 63
8.7.1. 使用列表生成式生成列表 63
8.7.2. 使用内置 map 函数处理列表 64
8.7.3. 获取 JSON 数据中多层嵌套中的值 65
8.7.4. 使用 arrow 库正确处理时间 66
8.7.5. 向外部发送 HTTP 请求 67
8.7.6. 发送钉钉机器人消息 68

8.7.7. 避免 SQL 注入.....69

9. FAQ.....70

附录.....75

本文档基于 20200228 版本编写第一版，

新版本更新后会同步更新本文档，并会标注新增内容属于哪个版本

由于本系统会持续更新升级，因此文档与实际可能会略有出入

1. 前言

DataFlux 是驻云 CloudCare 品牌下的实时大数据分析平台，它包含 DataKit 采集器、DataWay 数据网关、DataFlux Studio 实时数据洞察平台，DataFlux Admin Console 管理后台， DataFlux.f(x) 实时数据处理开发平台五大功能模块。DataFlux 面向企业提供全场景的数据洞察分析能力， 具有实时性、灵活性、易扩展、易部署等特点，支持云端 SaaS 和本地部署模式。

DataFlux.f(x)是 DataFlux 的一个组成部分，读作「DataFlux Function」。一方面可以扩展 DataFlux 图表分析、图表数据查询、基线告警等功能；另一方面本身也是一个类似 Server Less 的平台，可以直接对外提供 HTTP API 提供数据查询、计算能力。因此，可以将 DataFlux.f(x)看作是一个多功能函数计算平台。

DataFlux.f(x)从具体功能上来说，可以响应客户端通过 HTTP 请求，并调用脚本中的函数。

基于上述基本业务逻辑，DataFlux.f(x) 包含以下基本概念：

- 脚本库：DataFlux.f(x) 所有脚本内容
- 脚本集：多个脚本的集合
  （注：脚本集并不是 Python 模块，两者无任何关系）
- 脚本：即 Python 脚本文件，一个脚本集包含多个脚本
- 顶层函数：脚本中最外层的函数，这些函数可通过内建装饰器暴露为 HTTP API，但仅在集群内可访问
- 函数导出装饰器：将「顶层函数」暴露为 HTTP API 的装饰器
- 数据源：脚本中可调用的数据源，可以为 DataFlux 所用的时序数据库、DataFlux DataWay，或用户自行添加的数据库等。
- 环境变量：脚本中可调用的环境变量，可用作配置、常量等
- 脚本编辑器：即脚本编辑页面
- 授权链接：用于控制对外暴露的函数的 URL 别名，可将函数暴露到集群外
- 自动触发：基于 Crontab 语法的自动执行函数的功能
- 脚本包：若干个脚本集导出后的文件，可用于分发、升级等目的
- 还原：可将整个脚本库还原到特定历史状态的功能
- 函数文档：包含所有暴露的函数信息的在线动态文档
- 授权链接函数文档：包含所有授权链接信息的在线动态文档

2. 登录

访问 DataFlux.f(x) 首页后，如果没有登录，则自动跳转到登录界面。
登录界面如下图所示：

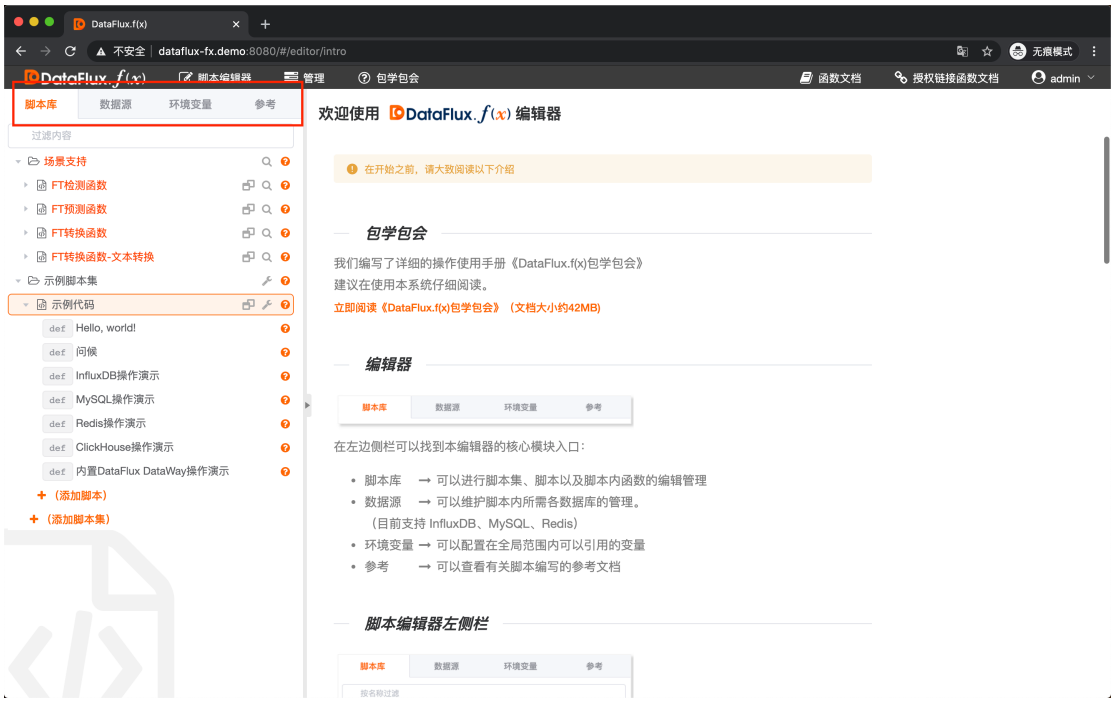
DataFlux.f(x) 属于 DataFlux 的后台管理系统之一，需要使用具有相应权限/角色的 DataFlux 后台管理账号登录（具体请参考 DataFlux 相关文档）。
如无法登录或没有账号，可联系后台管理员开设账号。

3. 脚本编辑器

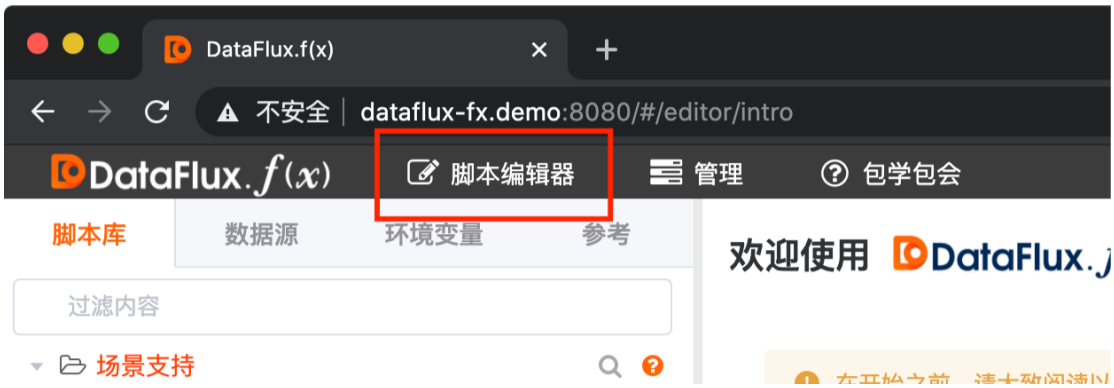
登录成功后，默认进入简介画面。
简介画面提供了各画面的进入方式和简单使用方式的快速手册。

画面左边有包含脚本编辑器与脚本编写有关的所有相关项目的侧栏，包含以下内容：

- 脚本库：所有的脚本集、脚本，以及脚本中已发布的导出函数列表
- 数据源：所有脚本需要连接的数据库列表
- 环境变量：所有脚本需要的环境变量列表
- 参考：有关脚本编写的快速手册



当位于其他模块时，点击顶部的「脚本编辑器」即可回到脚本编辑器



3.1. 脚本库

脚本库以树状菜单显示，最顶层到最底层依次是：

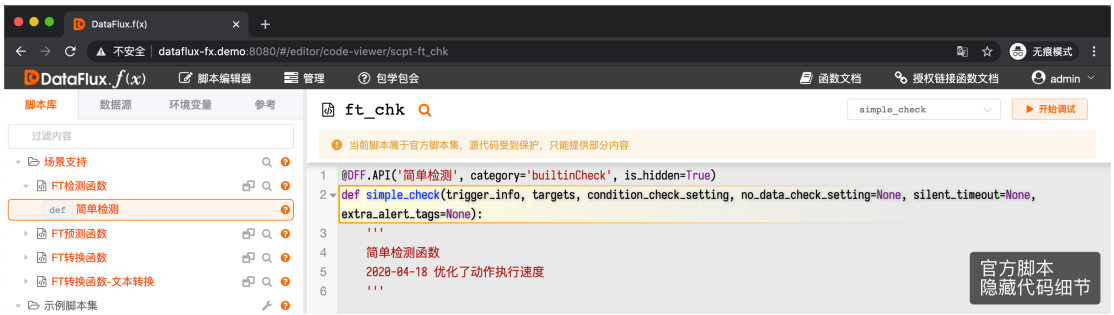
1. 脚本集
2. 脚本
3. 脚本中已发布的导出函数

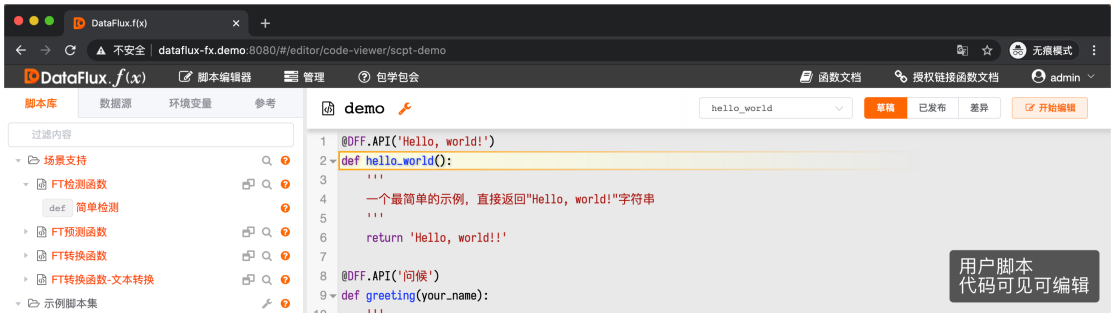
其中，黑色文字表示普通用户编写的脚本集。橙色文字表示官方发布的脚本集。

官方发布的脚本集无法修改，具体代码内容也会隐藏。

> *驻云本公司内部部署版作为官方脚本集的制作方，允许编辑所有脚本（以后不再重复说明）*

用户/官方的区分在脚本集层面指定，如当一个脚本集为用户脚本时，则下属的所有脚本都是用户脚本，不会出现同一个脚本集下同时存在用户脚本和官方脚本的情况。





点击「开始编辑」即可进入在线代码编辑器，对代码进行修改。（详见 3.2.3. 使用在线编辑器）

对于官方脚本，按钮为「开始调试」，仅允许在线调用函数查看返回值，不允许进行编辑。

3.1.1. 添加/配置脚本集

在脚本集列表最后，点击「（添加脚本集）」可进入新脚本集添加界面。
点击列表中脚本集右侧的扳手按钮，可进入既存脚本集的配置界面。

官方脚本集无法编辑，因此不会有扳手按钮，而是放大镜按钮，可以查看配置（下同）

添加、配置脚本集时，需要填写如下字段：

- 标题：单纯展示用字段，填写后作为新建的脚本集名称显示在左侧栏中
- 描述：单纯展示用字段，填写后作为新建的脚本集详情展示在左侧栏帮助提示中
- 引用名：关键字段，用于后续编写 Python 脚本时和外部调用时的引用名

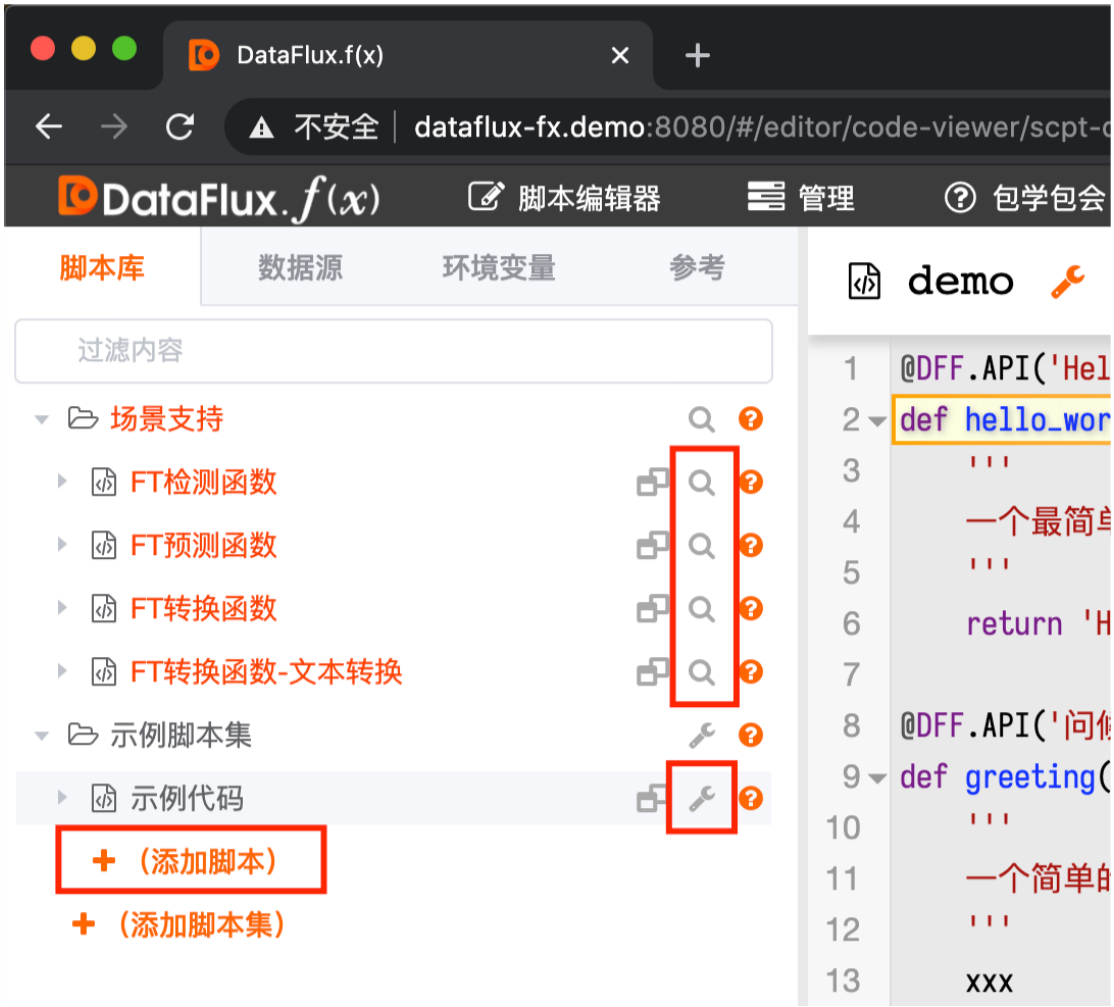
配置脚本集页面除了可以修改脚本集，也可以将脚本集删除。

官方脚本集无法配置或删除，所有字段都为只读。

3.1.2. 添加/配置脚本

每个脚本集下，在脚本列表最后，点击「（添加脚本）」可进入新脚本添加界面，添加的脚本归入所属的脚本集。

点击列表中脚本右侧的扳手按钮，可进入既存脚本的配置界面。



仅用户脚本集可以添加脚本

添加脚本时，需要填写如下字段：

- 标题：单纯展示用字段，填写后作为新建的脚本名称显示在左侧栏中
- 描述：单纯展示用字段，填写后作为新建的脚本详情展示在左侧栏帮助提示中
- 引用名：关键字段，用于后续编写 Python 脚本时和外部调用时的引用名

配置脚本页面除了可以修改脚本，也可以将脚本删除。

官方脚本无法配置或删除，所有字段都为只读。

3.1.3. 代码编辑器

点击任意脚本，或脚本下属的函数，右侧主面板会进入对应脚本的在线代码查看器。

此时，页面为只读状态，底色为灰色。用户可以切换查看草稿、已发布的代码和草稿与已发布代码之间的差异。对于存在编辑但未发布的脚本，左侧栏会出现相应标记。

当需要修改脚本时，点击「开始编辑」，即可进入在线脚本编辑器。

> *在线脚本编辑器具有「草稿」状态，单纯保存仅仅为保存草稿，只有发布之后，脚本内容才会生效。（见下文）*

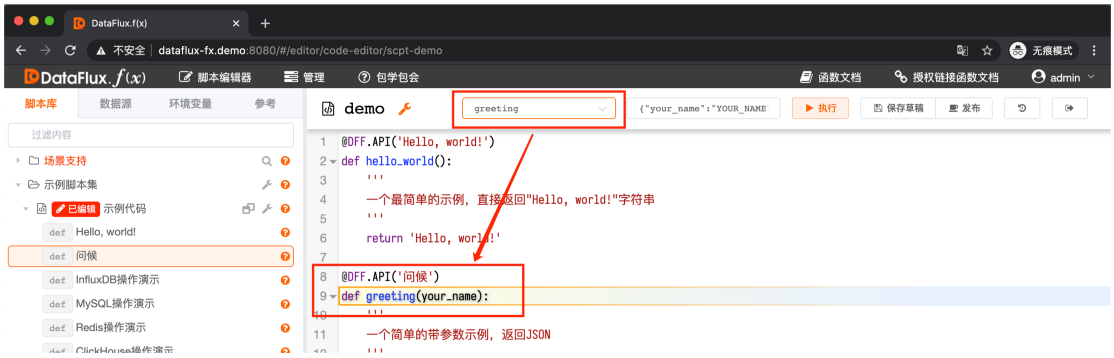
在线脚本编辑器左上方为为当前脚本引用名。

右上方为操作区，从左到右依次为：

1. 当前脚本所有函数列表（包括未导出的函数）
2. 函数调试执行参数：按 JSON 格式填写，建议在参数非常简单时使用，否则应编写测试函数来调用）
3. 执行按钮：执行所选的函数、以填写的执行参数运行，并输出返回值（见下文）
4. 保存按钮：保存当前脚本为草稿
5. 发布按钮：保存并发布当前脚本
6. 复原按钮：将草稿恢复到上次发布时的状态
7. 退出编辑：进入脚本查看页面

所有函数选择列表中会列出本脚本中的所有非下划线开头的顶层函数，无论是否导出。

因此，在在线脚本编辑器中，可以直接运行任何顶层脚本，方便测试或调试。
选择某个函数后，编辑器会跳转到函数所在位置，如下图：



编辑器随时可以指定函数执行，确认脚本状态。
附带参数的，需要按 JSON 格式输入参数列表，如：

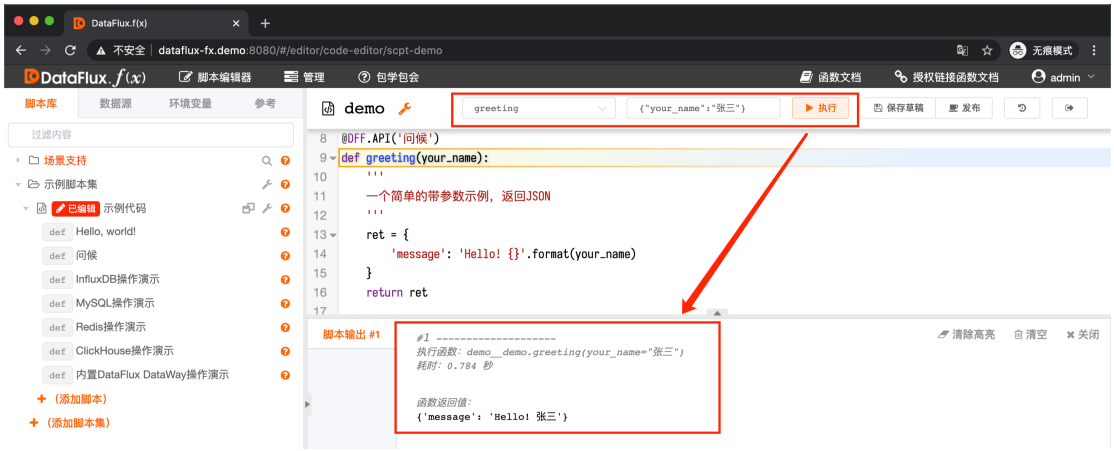
```
{"msg": "你好"}
```

虽然可以通过选择函数和填入参数运行任意函数
但从便捷角度，另外编写专用的测试函数更为方便，如：

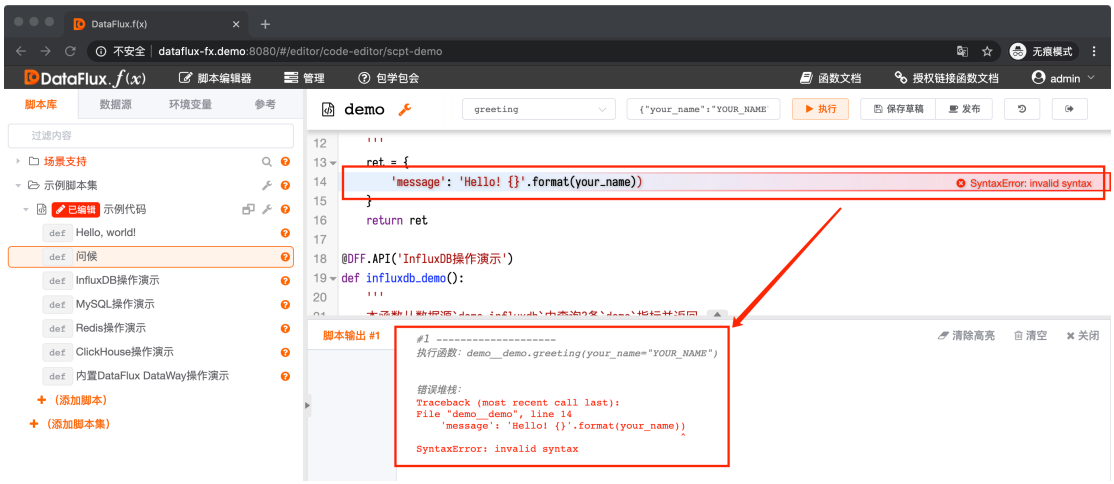
```
def my_func(a, b):
    return a + b


# 测试函数中可以一次运行多个测试用例
# 且选择函数后不用另外填写复杂的参数，直接运行即可
def test_my_func():
    print(my_func(1, 2))
    print(my_func(1, None))
```

函数执行后，编辑器底部会弹出输出窗格，包含函数执行时间、print 输出以及函数返回值，如下图：



如脚本执行出错，编辑器会标注错误行，并在底部输出窗格展示详细错误堆栈，如下图：

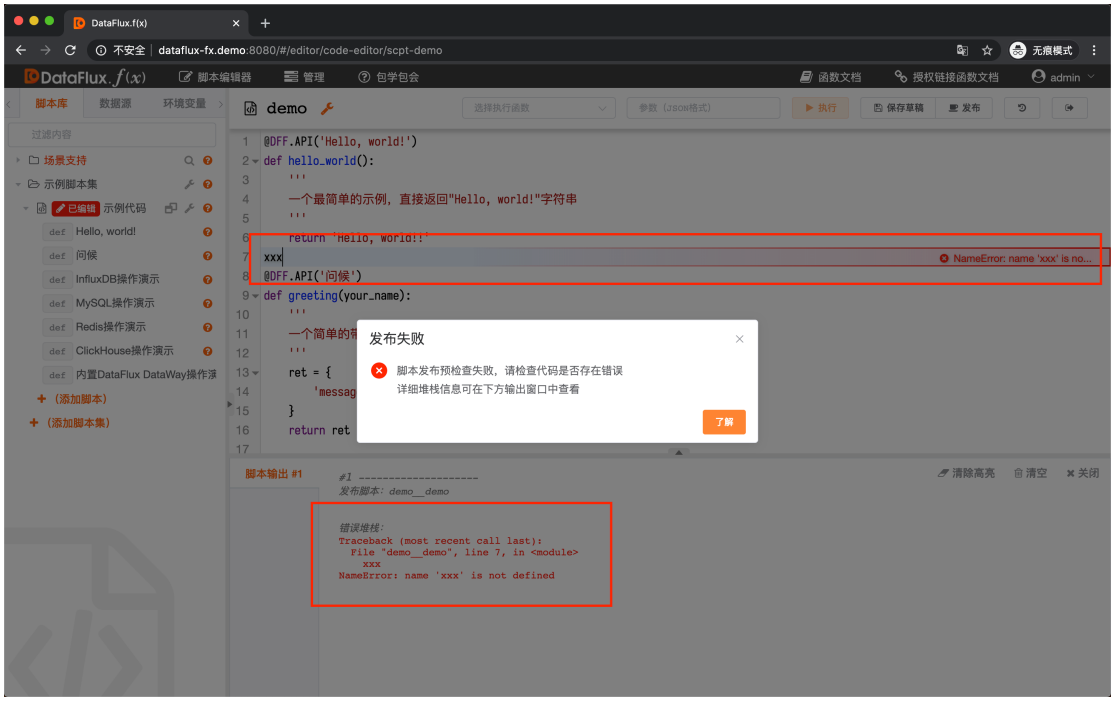


注意！在编辑脚本时，务必随时保存，防止误关浏览器导致内容丢失！

单纯保存脚本并不会影响系统对外的函数接口。
当确认脚本编辑完毕后，点击发布按钮，即可更新系统对外的函数接口。

> *在编辑页面，可以使用 Command + S（Mac）或 Ctrl + S（Windows）快捷键执行保存*
>
> *由于缓存策略，发布后可能有一小段时间的延迟才会实际体现*

脚本发布时，会进行简单的基础检查。
如果存在问题，发布会失败。同时会标注第一个错误行，并在最底下弹出的输出窗格展示详细错误信息。



当脚本编辑完毕后，请勿继续停留在编辑页面，应当点击退出编辑按钮。

*同一个脚本仅允许单个登录账号处于编辑状态，其他登录账号只能阅读，因此**请勿多人共享账号***

在线代码编辑器快捷键

在线代码编辑器提供了一些类 Sublime Text 的快捷键。

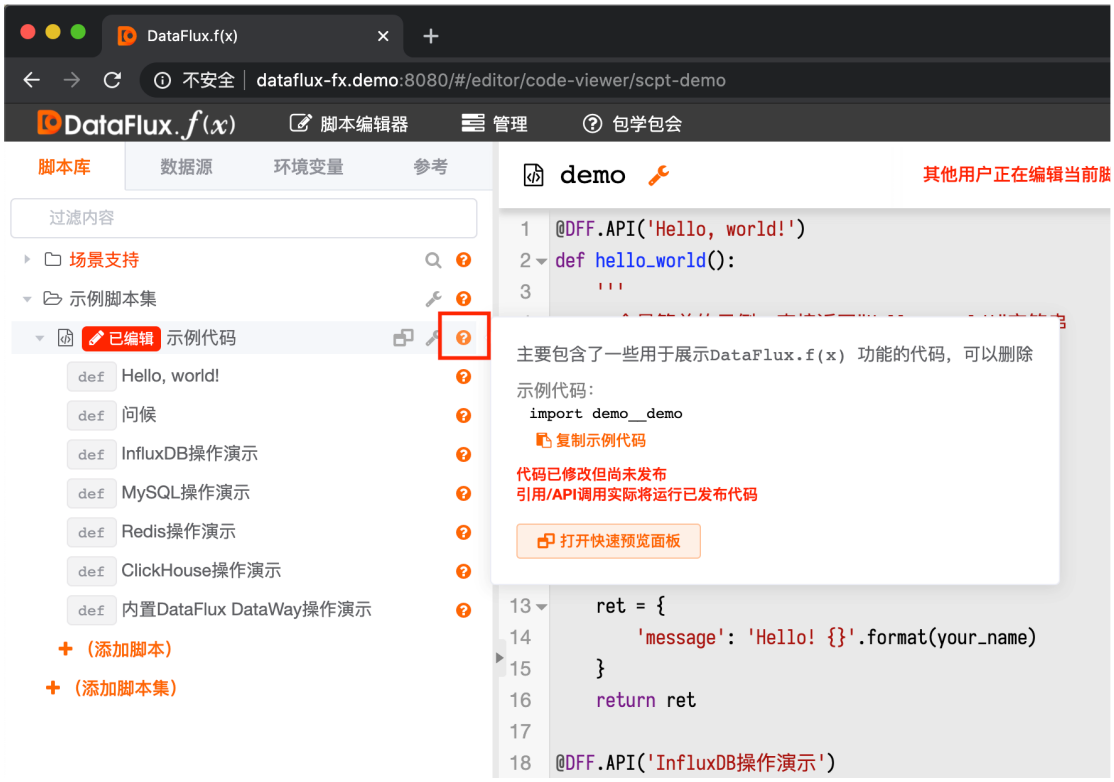
随键输入出现提示后 **TAB** 或 **ENTER**	提示上屏
COMMAND + **/** （Mac） **CTRL** + **/** （Windows）	注释/取消注释 支持选中多行进行操作
光标未选中字符时 **COMMAND** + **X** （Mac） 或 **CTRL** + **X** （Windows）	剪切整行
光标未选中字符时 **COMMAND** + **C** （Mac） 或 **CTRL** + **C** （Windows）	复制整行
剪切或复制整行后 **COMMAND** + **V** （Mac） 或 **CTRL** + **V** （Windows）	粘贴整行

选中多行后 **TAB**	增加多行缩进
选中多行后 **SHIFT** + **TAB**	减少多行缩进
COMMAND + **]** （Mac） 或 **CTRL** + **]** （Windows）	增加缩进 支持选中多行进行操作
COMMAND + **[** （Mac） 或 **CTRL** + **[** （Windows）	减少缩进 支持选中多行进行操作
COMMAND + **D** （Mac） 或 **CTRL** + **D** （Windows）	选中整个单词
选中整个单词后 连续按下 **COMMAND** + **D** （Mac） 或连续按下 **CTRL** + **D** （Windows）	继续选中相同的单词 并支持多点同时编辑

3.1.4. 从帮助按钮获取帮助

左侧栏中的脚本集、脚本、数据源、环境变量右侧都有一个问号图标。
将鼠标指向这个问号图标，即可显示相关的信息：

- 脚本集：描述信息
- 脚本：描述信息、示例代码
- 函数：描述信息、示例代码
- 数据源：描述信息、示例代码、简易调试面板入口（见下文）
- 环境变量：描述信息、示例代码



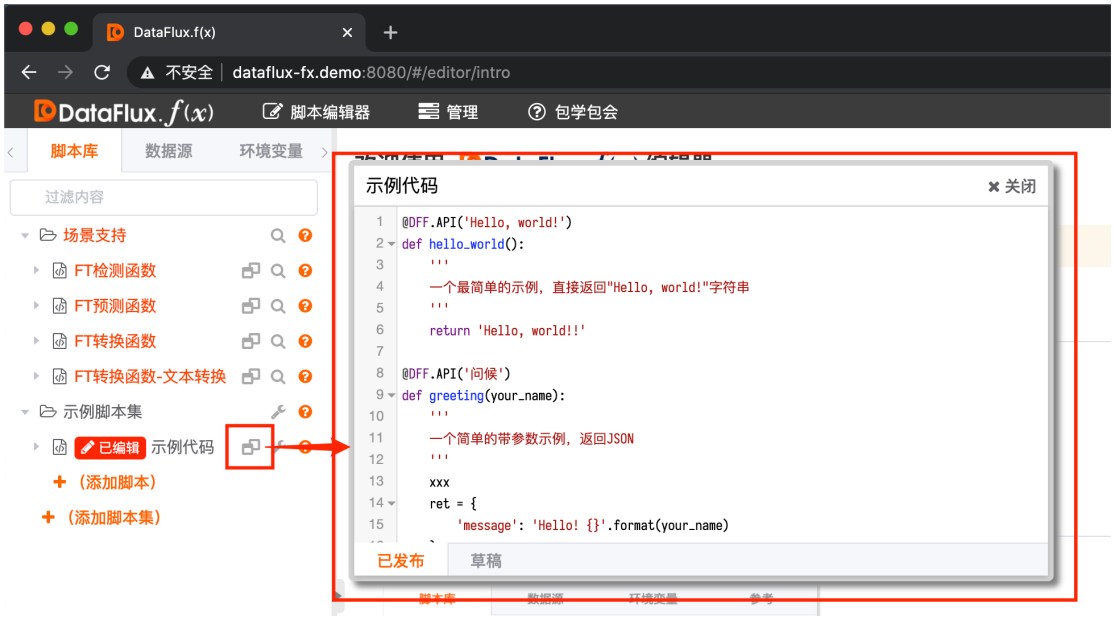
3.1.5. 代码快速查看面板

2022-04-21 新增功能

脚本库中的脚本，可以使用快速查看面板随时查看内容，且支持切换查看已发布的代码和草稿代码。点击脚本项目右侧的窗口图标，即可显示快速查看面板。

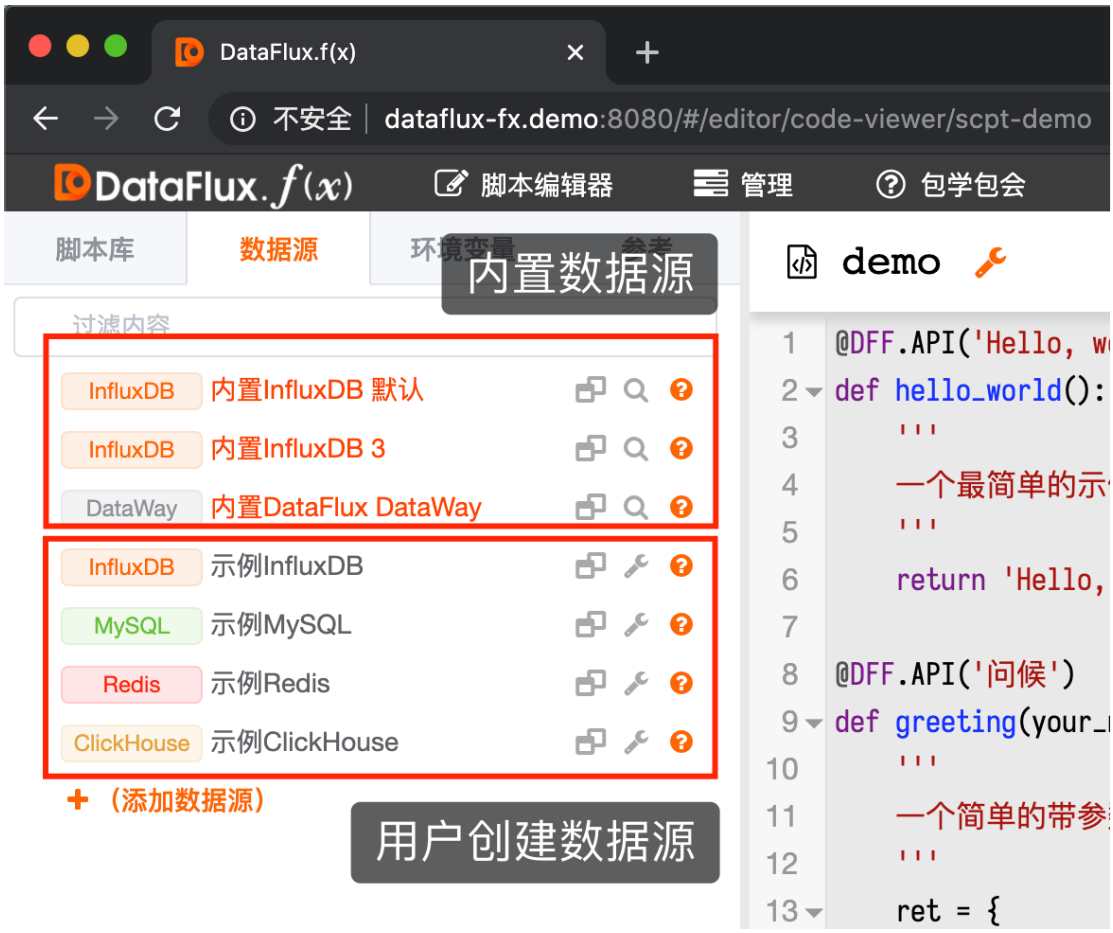
快速查看面板可以拖动到页面的任意位置。

注：快速查看面板仅在左侧栏选中脚本库时才会显示



3.2. 数据源

数据源以列表方式展示，包含了所有脚本可以访问的数据库、缓存数据库、DataFlux DataWay 等资源。
其中，黑色文字表示用户自行添加的数据源。橙色文字表示 DataFlux 系统内置的数据源。



3.2.1. 添加/配置数据源

在数据源列表最后，点击「（添加数据源）」可进入新数据源添加界面。
点击列表中数据源右侧的扳手按钮，可进入既存数据源的配置界面。

仅用户自行添加的脚本集可以修改

添加数据源时，需要填写如下字段：

- 类型：关键字段，数据源所属类型（见下文）
- 标题：单纯展示用字段，填写后作为新建的数据源名称显示在左侧栏中
- 描述：单纯展示用字段，填写后作为新建的数据源详情展示在左侧栏帮助提示中
- 引用名：关键字段，用于后续编写 Python 脚本时和外部调用时的引用名
- 其他数据源连接字段，如访问地址、端口等信息

在选择数据源类型后，会出现额外需要填写的数据源相关字段。不同种类的数据源所需填写的字段会有不同。
配置数据源页面除了可以修改数据源，也可以将数据源删除。
注意：数据源一旦创建，则不允许修改类型

添加数据源时，系统会自动对填写的配置做简单的连通性检查，只有能够连通的数据源才能够被添加成功。
此外，修改数据源配置时，密码栏不会显示之前填写的密码，且必须重新填写正确的密码才能够进行修改。

添加/配置 InfluxDB 数据源

InfluxDB 数据源的连接字段包括：

- 主机
- 端口
- 协议
- 数据库
- 用户名
- 密码

添加/配置 MySQL 数据源

MySQL 数据源的连接字段包括：

- 主机
- 端口
- 数据库
- 用户名
- 密码
- 编码

添加/配置 Redis 数据源

Redis 数据源的连接字段包括：

- 主机
- 端口
- 数据库
- 密码

添加/配置 Memcached 数据源

2020-05-21 新增功能

Memcached 数据源的连接字段包括：

- 服务器列表

Memcached 服务器存在多个时，使用英文逗号分隔，如：

host1:11211,host2:11211,host3:11211

添加/配置 DataFlux DataWay 数据源

DataFlux DataWay 数据源的连接字段包括：

- 主机
- 端口
- 协议

添加/配置 ClickHouse (TCP) 数据源

ClickHouse (TCP) 数据源的连接字段包括：

- 主机
- 端口
- 数据库
- 用户名
- 密码

ClickHouse 会暴露 2 个端口，一个用于原生 TCP 接口（默认为 9000），另一个用于 HTTP 接口（默认为 8123）。此处的端口需要填写用于原生 TCP 接口的端口。

对于使用阿里云提供的 ClickHouse 时，默认 TCP 端口可能为 3306。具体请以控制台配置为准。
此外，各个云平台提供的数据库产品一般都会有 IP 白名单限制。添加数据源之前，请确保连通性。

添加/配置 Oracle 数据源

2020-05-21 新增功能

Oracle 数据源的连接字段包括：

- 主机
- 端口
- 数据库（服务名）
- 用户名
- 密码

添加/配置 SQL Server 数据源

2020-05-21 新增功能

SQL Server 数据源的连接字段包括：

- 主机
- 端口
- 数据库
- 用户名
- 密码

添加/配置 PostgreSQL 数据源

2020-05-21 新增功能

PostgreSQL 数据源的连接字段包括：

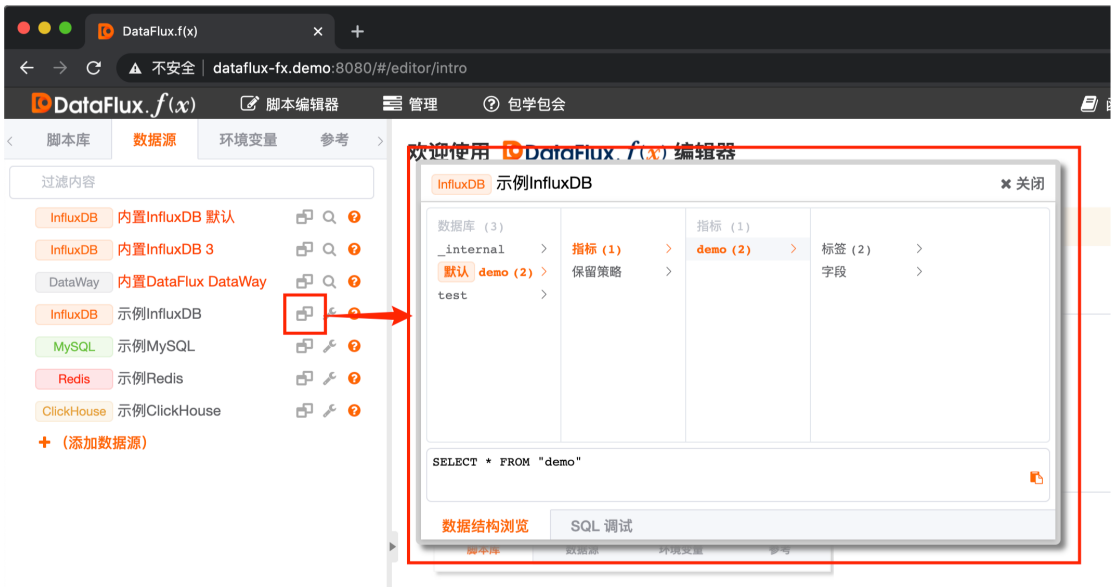
- 主机
- 端口
- 数据库
- 用户名
- 密码

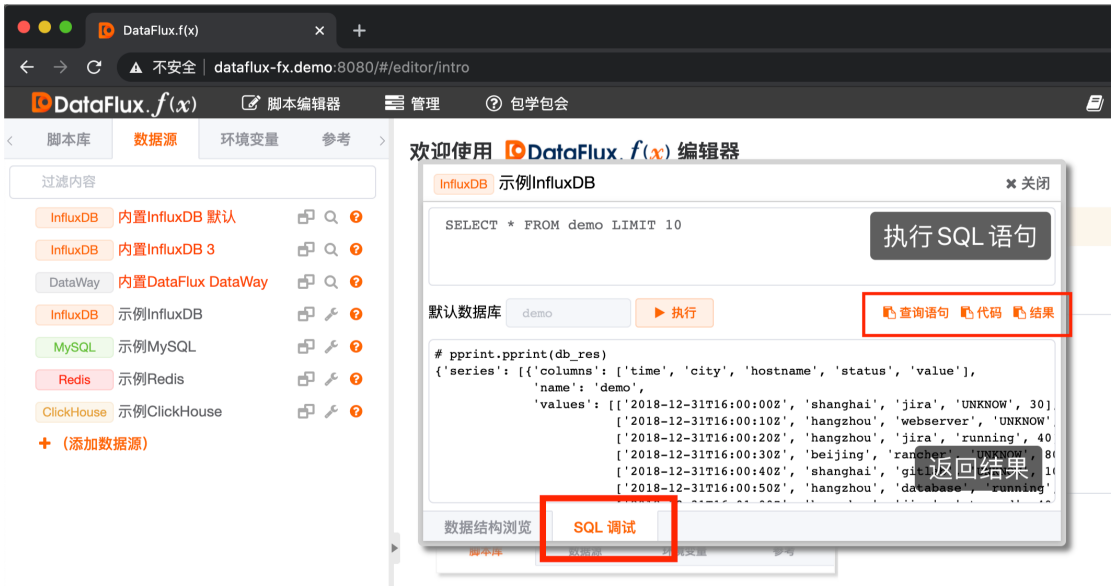
3.2.2. 数据源简易调试面板

已添加的数据源，可以使用简易调试面板执行查询语句，且支持执行 SQL 语句或命令，以及浏览数据结构。点击数据源项目右侧的窗口图表，即可显示简易调试面板。

简易调试面板可以拖动到页面的任意位置。

注：简易调试面板仅在左侧栏选中数据源时才会显示





此外，在 SQL 调试（Redis 为命令调试）中，可以直接执行 SQL 语句并展示结果。如查询执行成功，面板右侧还会出现 3 个方便按钮，分别是：

- 复制查询语句
- 复制示例代码
- 复制查询结果

用户可以在确定查询语句后，直接使用上述复制功能方便后续工作。

InfluxDB 查询语句

InfluxDB 查询使用 InfluxDB Query Language（Influx QL）语法，如：

```
SELECT * FROM some_measurement LIMIT 10
```

详细请参考 InfluxQL 文档：
https://docs.influxdata.com/influxdb/v1.7/query_language/

MySQL 查询语句

MySQL 查询使用标准 SQL 语法，如：

```
SELECT * FROM some_table LIMIT 10
```

详细请参考 MySQL SQL 文档：
https://dev.mysql.com/doc/refman/5.7/en/sql-statements.html

Redis 查询语句

Redis 查询直接使用命令行格式语法，如：

```
GET some_key
```

详细请参考 Redis 命令文档：
https://redis.io/commands

Memcached 查询语句

2020-05-21 新增功能

Memcached 查询直接使用命令行格式语法，如：

```
get some_key
```

详细请参考 Memcached 命令文档：
https://github.com/memcached/memcached/wiki/Commands

ClickHouse 查询语句

ClickHouse 查询使用 SQL 语法，如：

```
SELECT * FROM some_table LIMIT 10
```

详细请参考 ClickHouse 查询语句文档：
https://clickhouse.tech/docs/zh/query_language/select/

Oracle 查询语句

2020-05-21 新增功能

Oracle 查询使用 SQL 语法，如：

```
SELECT * FROM some_table WHERE ROWNUM <= 10
```

详细请参考 Oracle 查询语句文档：
https://www.oracle.com/technetwork/cn/database/database-technologies/sql/documentation/index.html

SQL Server 查询语句

2020-05-21 新增功能

SQL Server 查询使用 SQL 语法，如：

```
SELECT TOP 10 * FROM some_table
```

详细请参考 SQL Server 相关文档：
https://docs.microsoft.com/zh-cn/sql/sql-server

PostgreSQL 查询语句

2020-05-21 新增功能

PostgreSQL 查询使用 SQL 语法，如：

```
SELECT * FROM some_table LIMIT 10
```

详细请参考 PostgreSQL 相关文档：
https://www.postgresql.org/docs/

3.3. 环境变量

环境变量以列表方式展示，包含了所有脚本可以访问的环境变量。

3.3.1. 添加/配置环境变量

在环境变量列表最后，点击「（添加环境变量）」可进入新环境变量添加界面。

点击列表中环境变量右侧的扳手按钮，可进入既存环境变量的配置界面。

添加环境变量时，需要填写如下字段：

- 标题：单纯展示用字段，填写后作为新建的环境变量名称显示在左侧栏中
- 描述：单纯展示用字段，填写后作为新建的环境变量详情展示在左侧栏帮助提示中
- 引用名：关键字段，用于后续编写 Python 脚本时和外部调用时的引用名
- 值：关键字段，环境变量内容

注：环境变量值都是字符串

配置环境变量页面除了可以修改环境变量，也可以将环境变量删除。

3.4. 参考

参考以文档展示，包含了有关脚本编写的快速手册。

在阅读完整《包学包会》文档之后，一些需要经常查询的内容可在此查看。

4. 管理

点击顶部的「管理」即可进入管理模块

管理模块包含了有关本系统所有的管理项目，包含：

- 授权链接：允许指定某些函数可以在公网访问
- 自动触发配置：允许指定某些函数以特定频率自动被调用
- 脚本包导出：允许导出指定的脚本集，脚本包导出历史也在这里
- 脚本包导入：允许导入指定的脚本集，脚本包导入历史也在这里
- 脚本库还原：允许将整个脚本库（所有脚本集和所有脚本）还原至特定的时间点

2020-04-21 新增功能

- 官方脚本库：允许从驻云官方获取并安装最新版的通用脚本，包括且不限于触发器所需的检测脚本，Studio 图表分析模式中的预测、转换函数等

注意：由于本系统会持续更新，管理模块会不断增加新的管理项目

4.1. 授权链接

点击「管理」模块左侧栏的「授权链接」，即可打开授权链接列表。列表包含了当前已经创建的所有授权链接以及相关信息。

点击「API 调用示例」，可以查看对应授权链接的可用参数以及调用方式示例。

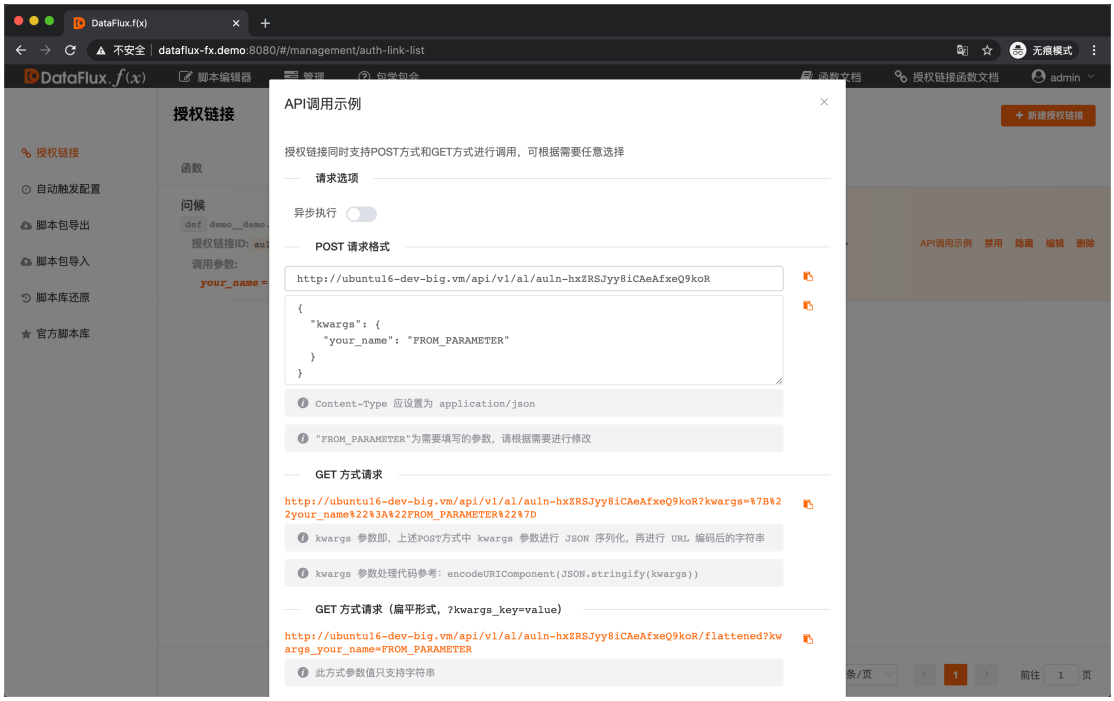
授权链接支持多种调用方式，包括：

- POST 方式：参数、选项包含在 body 中
- GET 方式：参数、选项按照 JSON 序列化后包含在 URL 中

2020-04-30 新增功能

- GET 方式（扁平形式）：参数、选项以 kwargs_*或 options_*包含在 URL 中
- GET 方式（简化形式）：参数包含在 URL 中

API 调用示例如下：

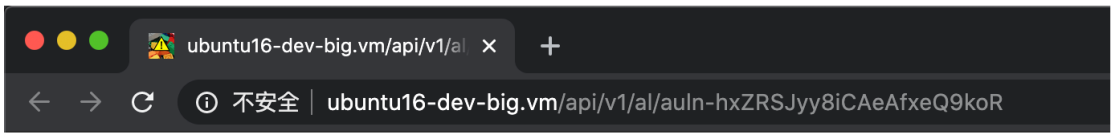


各种调用方式存在一些各自的限制：

调用形式	kwargs 参数	kwargs 参数可体现类型	options 参数
POST 方式	支持	支持	支持
GET 方式	支持	支持	支持
GET 方式（扁平形式）	支持	不支持	支持
GET 方式（简化形式）	支持	不支持	不支持

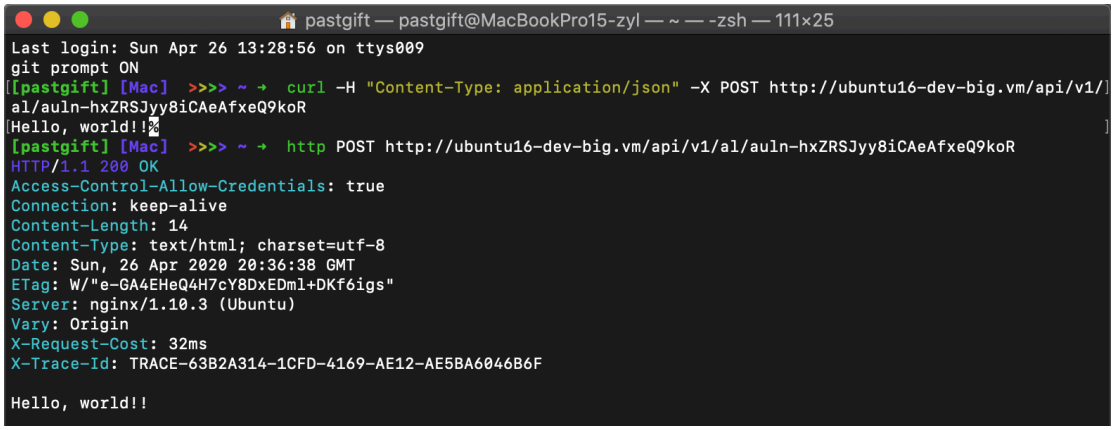
任何同步调用方式都能得到相同的返回结果，可根据需要自行选择。

GET 方式：



Hello, world!!

POST 方式：



注意：使用异步方式调用时，系统不会立即返回函数结果，而是返回结果查询 URL 地址。调用方需要后续根据此 URL 地址查询结果。
大多数情况下，使用同步方式即可

4.1.1. 添加/配置授权链接

在授权链接列表，点击「新建授权链接」，即可进入添加授权链接页面。
新建授权链接需要填写以下内容：

- 执行函数
- 调用参数（参数会在选择函数后自动生成模板）
- 是否启用
- 是否显示在「授权链接函数文档」中（见下文）
- 有效期
- 限流策略（允许按年、月、日、时、分、秒限制调用次数）
- 备注

4.2. 自动触发配置

点击「管理」模块左侧栏的「自动触发配置」，即可打开自动触发配置列表。
列表包含了当前已经创建的所有自动触发配置以及相关信息。

4.2.1. 添加/配置自动触发配置

在自动触发配置列表，点击「新建自动触发配置」，即可进入添加自动触发配置页面。
新建自动触发配置需要填写以下内容：

- 执行函数
- 调用参数（参数会在选择函数后自动生成模板）
- 重复规则（Crontab 语法格式）
- 备注

注意：自动触发配置的重复规则的最小间隔为 1 分钟。但如遇到前一次触发尚未运行结束，则本次的触发会被自动跳过。

4.3. 脚本包导出

点击「管理」模块左侧的「脚本包导出」，即可打开脚本包导出历史列表。列表包含了已执行过的脚本包导出历史。

4.3.1. 执行脚本包导出

在脚本包导出历史列表，点击「导出脚本包」，即可进入脚本包导出页面。

导出脚本包需要填写以下内容：

- 脚本集
- 操作人（仅用于记录说明）
- 操作备注（仅用于记录说明）

成功导出后的脚本包文件会自动下载，文件名为固定格式如下：

```
dataflux-fx.<导出年月日> _<导出时分秒>.user
```

此外，系统会自动生成区分大小写的「导入令牌」显示在页面。此令牌需要在导入本脚本包时填写才能正常导入，用于定向发布脚本时防止代码中途泄漏。（见下文）

用户导出的都是用户脚本包（扩展名为.user）。除此之外，还有由驻云官方制作的官方脚本包（扩展名为.official）

4.4. 脚本包导入

点击「管理」模块左侧的「脚本包导入」，即可打开脚本包导入历史列表。列表包含了已执行过的脚本包导入历史。

4.4.1. 执行脚本包导入

在脚本包导入历史列表，点击「导入脚本包」，即可进入脚本包导入页面。
导入脚本包需要填写以下内容：

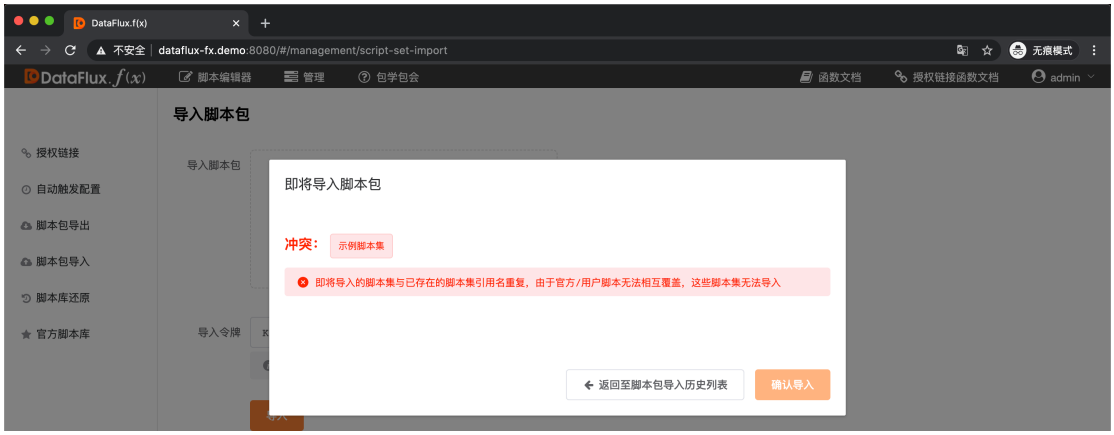
- 脚本包文件（扩展名为.user 或.official）
- 导入令牌（通用官方脚本包不需要导入令牌时留空即可）

脚本包上传并解析成功后，会出现确认导入对话框。
对话框中会展示即将导入的脚本集以及处理方式，主要有以下几种情况：

- 导入**官方**、*用户*脚本包时，如不存在相同引用名的脚本集，则添加
- 导入**官方**脚本包时，如已存在相同引用名的**官方**脚本集，则替换
- 导入*用户*脚本包时，如已存在相同引用名的*用户*脚本集，则替换
- 导入**官方**脚本包时，如已存在相同引用名的*用户*脚本集，则无法导入
- 导入*用户*脚本包时，如已存在相同引用名的**官方**脚本集，则无法导入

简而言之，脚本包的导入以每个脚本集的引用名为基准，且官方脚本集和用户脚本集不允许相互替换。

如恰巧用户创建的脚本集与官方脚本集引用名冲突，可以先修改脚本库中的用户脚本集引用名，然后再次尝试导入脚本包。



脚本包导入成功后立刻生效，不需要额外进行发布操作

4.5. 脚本库还原

点击「管理」模块左侧的「脚本库还原」，即可打开还原点列表。列表包含了所有可以恢复的脚本库还原点。需要还原脚本库到某个时间点时，可点击列表右侧「还原至此状态」。

脚本库还原会立刻生效，不需要额外进行发布操作

还原成功后，还原操作本身也会产生还原点。因此即使误操作执行了还原，也能够还原到未还原之前的状态。

4.5.1. 创建脚本库还原点

在脚本库还原点列表，点击「创建还原点」，即可进入创建脚本库还原点页面。

创建还原点需要填写以下内容：

- 说明（用于对即将创建的还原点添加描述）

4.6. 官方脚本库

点击「管理」模块左侧的「官方脚本库」，即可打开可安装的官方脚本包列表。需要安装某个官方脚本时，点击「安装此版本」即可。

官方脚本主要提供了一些 Studio 所需的预测、转换函数，以及基本检测所需的检测函数。

官方脚本库安装后会立刻生效，不需要额外进行发布操作

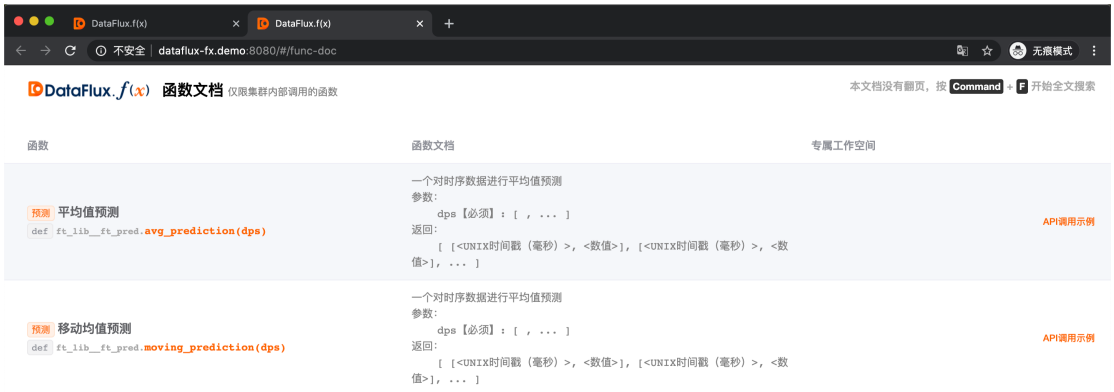
安装成功后，脚本库也会创建还原点，供必要时还原。

如 DataFlux 集群可以访问公网，那么可以直接安装

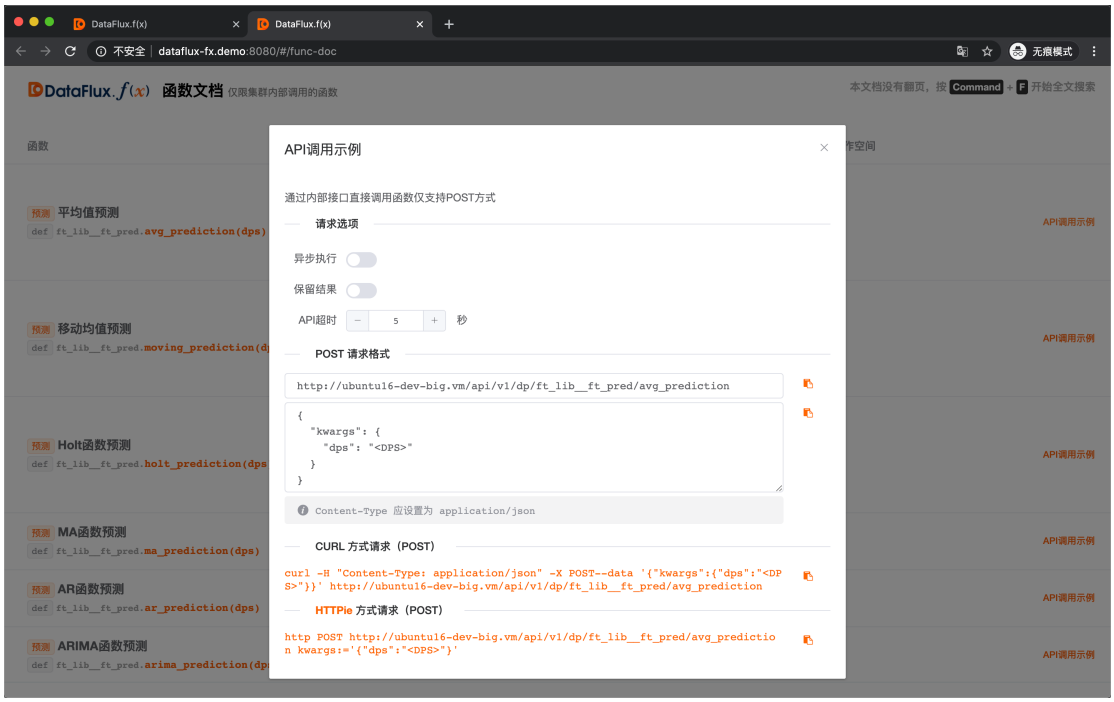
如 DataFlux 集群无法访问公网，那么安装时需要保证操作者计算机可以同时访问公网和 DataFlux.f(x)。系统会自动通过浏览器进行安装

5. 函数文档

点击导航栏的「函数文档」，会在新页面打开函数文档。文档包含了所有当前已发布脚本中，通过 DFF.API() 装饰器暴露的所有函数。



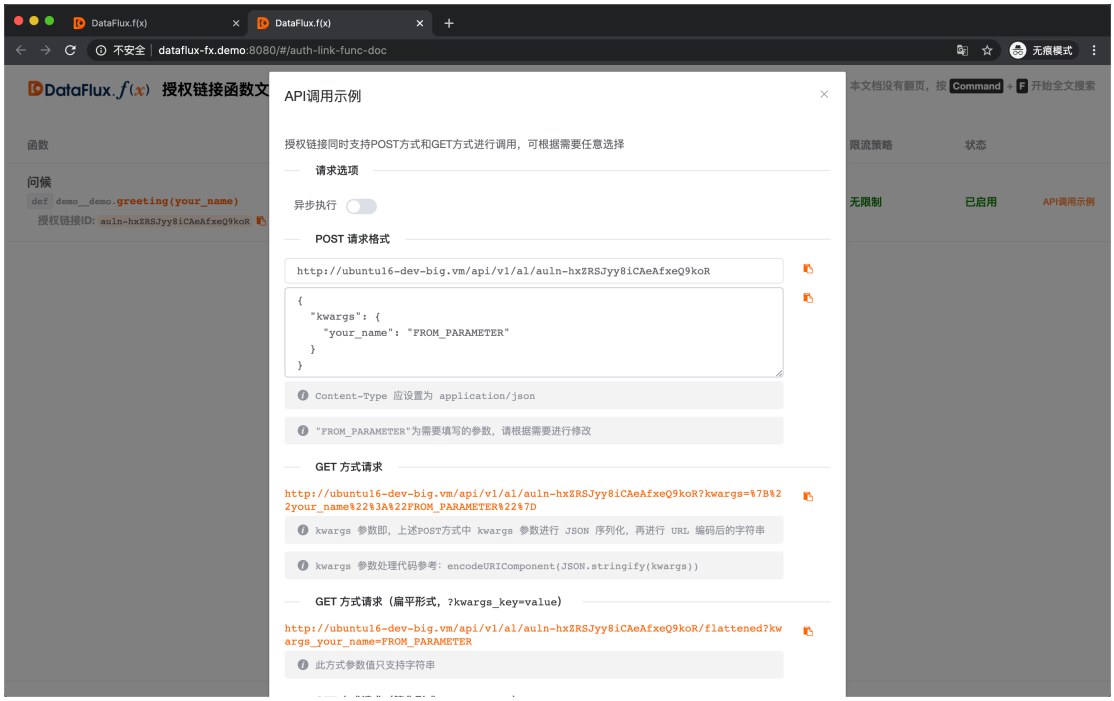
点击「API 调用示例」，可以查看对应函数的可用参数以及调用方式示例。



6. 授权链接函数文档

点击导航栏的「授权链接函数文档」，会在新页面打开函数文档。文档包含了所有标记为「显示」的授权链接对应的函数。

点击「API 调用示例」，可以查看对应授权链接的可用参数以及调用方式示例。



注意：只有开启「显示于文档」的授权链接才会出现在授权链接函数文档中。

授权链接函数文档包含 JSON 版，供第三方应用程序获取，访问以下地址即可：

```
/api/v1/auth-link-func-list
```

此外，授权链接函数文档 JSON 版支持查询参数，如：

```
/api/v1/auth-link-func-list?scriptSetRefName=xxx&scriptRefName=yyy
```

完整参数如下：

参数	说明	可选值
scriptSetRefName	脚本集引用名	DataFlux.f(x)中指定的脚本集引用名
scriptRefName	脚本引用名	DataFlux.f(x)中指定的脚本引用名
funcRefName 或 refName	函数引用名 （即函数名）	代码中的函数名
funcCategory 或 category	函数分类 （即 category）	代码中@DFF.API()装饰器指定的 category 参数值 可以使用英文逗号方式传递多个，表示过滤包含任意指定分类的函数 如：category=a,b 可以匹配分类为 a 和分类为 b 的函数，但无法匹配分类为 c 的函数
funcTags 或 tags	函数标签 （即 tags）	*2020-04-21 新增功能* 代码中@DFF.API()装饰器指定的 tags 参数 可以使用英文逗号方式传递多个，表示过滤包含所有指定标签的函数 如： tags=a,b 可以匹配具有标签 a、b、c 的函数，但无法匹配具有标签 a 的函数
scriptSetType	脚本集类型	官方脚本集 official 用户脚本集 user

7. 设置

点击右上角当前登录用户信息下拉框，选择「设置」，即可进入设置页面。
在设置页面，可以进行以下操作：

- 清除缓存
- 配置代码编辑器

2020-04-21 新增功能

- 实验性功能
- 关于

注意：所有设置仅保存在当前浏览器中，更换电脑、浏览器或清除浏览器缓存后所做的修改将会丢失而恢复默认。
此外，同一浏览器的不同标签页之间共享设置，在任一标签页修改的设置会同时影响其他所有标签页。

7.1. 清除缓存

点击「设置」模块左侧的「清除缓存」，即可进入缓存清除页面。一般来说，只有在页面操作时发生无法恢复的问题时，才需要尝试清除缓存。

7.2. 代码编辑器配置

点击「设置」模块左侧的「代码编辑器配置」，即可进入代码编辑器配置页面。用户可在此页面调整代码编辑器的着色主题，文字大小、行距等。右侧的预览可以看到实际效果。

7.3. 实验性功能

2020-04-21 新增功能

点击「设置」模块左侧的「实验性功能」，即可进入实验性功能开关配置页面。用户可以在此体验一些尚未提升为正式发布的前瞻性功能。具体功能详细见实际页面说明

注意：实验性功能可能存在一些问题，不保证一定可用。因此请勿过分依赖实验性功能。此外，实验性功能也会不定期新增、修改、删除，驻云官方不会另行告知。

7.4. 关于

2020-04-21 新增功能

点击「设置」模块左侧的「关于」，即可进入关于页面。用户可以在本页面确认与系统有关的信息。

8. 脚本编写

DataFlux.f(x)使用 Python 3.7.5 作为运行环境，并对脚本的执行上下文进行了一些定制，与原版的 Python 有一定区别。
根据 DataFlux.f(x)要求，本系统中的 Python 脚本都应该使用 4 个空格作为缩进单位，请勿使用 tab 等其他缩进方式。
此外，由于脚本中的函数返回值最终都会通过 HTTP 协议，以 JSON 数据返回给调用方。因此，函数返回值要求能够被序列化为 JSON 数据，一些复杂对象实例无法直接返回。

8.1. 最简单的函数

一个最简单的函数如下：

```
def hello_world():
    return 'Hello, World!'
```

可以看到，这个函数和原版 Python 中的函数写法并没有任何区别。
大部分情况下，在 DataFlux.f(x)中编写函数也与和原版 Python 没有什么区别。

此时，hello_world()函数已经可以使用，但仅限 DataFlux.f(x)内部被其他函数调用。如果希望函数能够在 DataFlux.f(x)外部被调用，则需要添加 @DFF.API() 装饰器。

装饰器的参数列表如下：

参数	是否必须	说明
title	必须	被修饰函数的中文标题。主要用于函数文档、UI 界面的展示，本身并没有具体功能
category	可选，默认 None	被修饰函数的分类，主要用于与 DataFlux Studio 对接（详见下文「附带分类的函数」）
tags	可选，默认 None	被修饰函数的标签，字符串数组，主要用于函数文档 API 的查询过滤
fixed_crontab	可选，默认 None	被修饰函数以自动触发方式运行时，以此固定的 crontab 表达式运行

典型示例如下：

```
@DFF.API('一个最简单的函数', category='action', tags=['hello', 'world'], fixed_crontab='*/15 * * * *')
def hello_world():
    return 'Hello, World!'
```

除了直接返回字符串，也可以返回能够序列化为 JSON 的对象。函数本身不需要手工进行序列化操作，如：

```
@DFF.API('一个最简单的函数')
def hello_world():
    ret = { 'message': 'Hello, World!' }
    return ret
```

如果确实没有返回值需要返回，返回 None 即可（Python 没有 return 语句时，同样返回 None），如：

```
@DFF.API('一个没有返回值的函数')
def hello_world():
    return None
```

函数可以附带参数、函数文档，同样按照正常 Python 编写即可，如：

```
@DFF.API('问候')
def hello_world(you_name=None):
    '''
    本函数返回一句问候语
    参数:
        your_name 可选，字符串。问候者姓名
    返回:
        字符串
    '''
    if you_name is None:
        your_name = 'My friend'
    return 'Hello, World! ' + your_name
```

在发布脚本后，DataFlux.f(x)会提取函数的标题、名称、参数列表、函数文档等，供 DataFlux 使用或通过函数文档对外展示。因此请务必正确填写。

8.2. 编写函数的要求

在 DataFlux.f(x)中编写函数，存在一些限制和要求。有些是出于安全考虑，有些是为了保证和 DataFlux 能够正常交互，还有一些主要是考虑到用户实际操作的便利性。

在编写函数时，请务必遵守这些规定，否则可能会带来意想不到的问题

8.2.1. 允许 import 的 Python 内置库

出于安全性方面的考虑，在 DataFlux.f(x)中只允许 import 部分内置库：

```
base64
collections
datetime
dns
hashlib
hmac
itertools
json
math
random
re
time
uuid
pprint
functools
gzip
```

8.2.2. 允许 import 的第三方 Python 库

除了 Python 的内置库，DataFlux.f(x)预装了部分常用的第三方库：

```
six        # v1.12.0
arrow      # v0.13.0
requests   # v2.21.0
simplejson # v3.16.0
ujson      # v1.35
xmltodict  # v0.9.2
PyYAML     # v4.2b2

numpy  # v1.18.1
pandas # v0.25.3
scipy  # v1.3.3

scikit-learn   # v0.22.1
statsmodels    # v0.11.0
fbprophet      # v0.5
eas_prediction # v0.12  (2020-04-21 新增)
```

8.2.3. 函数名称

函数名称即 def 后面的函数名称，在 DataFlux.f(x)中也被称为引用名。
除了需要满足 Python 本身的要求外，在 DataFlux.f(x)中额外要求长度不允许超过 256 个字符。

8.2.4. 函数定义

由于代码解析上的限制，函数定义**无论多长都必须写在一行里**，如：

```
def hello_world(arg1=None, arg2=None, arg3=None, args4=None):
    pass
```

以下为**错误示范**：

```
def hello_world(arg1=None, arg2=None,
    arg3=None, args4=None):
    pass
```

8.2.5. 函数标题

函数标题为@DFF.API()装饰器的第一个参数 title，在 DataFlux.f(x)中要求长度不超过 256 个字符。

8.2.6. 函数文档

函数文档为函数定义行（def 行）之后，第一个代码块，如：

```
@DFF.API('问候')
def hello_world(you_name=None):
    '''
    本函数返回一句问候语
    参数:
        your_name 可选，字符串。问候者姓名
    返回:
        字符串
    '''
    if you_name is None:
        your_name = 'My friend'
    return 'Hello, World! ' + your_name
```

编写时，缩进请勿小于三引号，并保证三引号都各自单独占有一行。

以下为**错误示范**：

```
def hello_world():
    '''这是函数文档的错误示范'''
    pass
```

```
def hello_world():
    '''
    这是函数文档的错误示范'''
    pass
```

```
def hello_world():
    '''这是函数文档的错误示范
    '''
    pass
```

```
def hello_world():
    '''
这是函数文档的错误示范
    '''
    pass
```

如内容比较多，需要分节展示，可以使用悬挂缩进方式：

```
def hello_world():
    '''
    内容比较多需要分节时，可以悬挂缩进
    第一节标题
        第一节内容
    第二节标题
        第二节内容
    '''
    pass
```

标准模板：

```
def hello_world(you_name=None):
    '''
    本函数返回一句问候语
    参数:
        your_name 可选，字符串。问候者姓名
    返回:
        字符串
    '''
    pass
```

8.2.7. 函数参数

函数参数本质上与原版 Python 并没有什么区别，但是由于 DataFlux.f(x)中暴露的函数需要与 DataFlux Studio 进行交互，必须考虑用户在 Web 页面上操作的便利性。因此，对于被@DFF.API()修饰的函数，需要按照一定规范设计参数列表。

不要使用 Python 参数解包功能（*args, **kwargs）

虽然 Python 参数解包功能非常便利，但这样的参数无法在 Web 页面供用户填写参数，也无法进行参数提示，因此要避免这种设计。

```
@DFF.API('错误示范')
def hello_world(*args, **kwargs):
    pass
```

```
@DFF.API('正确示范')
def hello_world(message=None):
    pass
```

如指定在@DFF.API()中指定了分类 category，要根据分类要求设置固定参数

详情见下文「附带分类的函数」

除固定参数外，接受到的参数值都是字符串，需要进行类型转换

由于很多调用都是用户在 Web 端进行，参数也是在 Web 输入框中输入，所以接收到的参数值都是字符串类型，需要函数内进行判断、转换类型。

```
@DFF.API('错误示范')
def get_sqrt(number=None):
    '''
    本函数执行可能会由于 number 参数为字符串而失败
    '''
    return number * number
```

```
@DFF.API('正确示范')
def get_sqrt(number=None):
    '''
    应当进行判断/类型转换
    '''
    try:
        number = float(number)
    except Exception as e:
        raise Exception('Parameter `number` is not a number')

    return number * number
```

除固定参数外，参数都要指定 None 为默认值

为了方便用户调用，函数参数都应该提供默认值，且默认值不要直接写在参数定义中，而是在代码中后续赋值，并在函数文档中进行说明。

```
@DFF.API('错误示范 1')
def hello_world(message):
    pass

@DFF.API('错误示范 2')
def hello_world(message='你好'):
    pass
```

```
@DFF.API('正确示范')
def hello_world(message=None):
    '''
    参数:
        message 可选，字符串。默认值为"你好"
    '''
    if message is None:
        message = '你好'
```

有关固定参数的说明，详见下文「附带分类的函数」

以上限制**仅限于对外暴露的函数**（即被@DFF.API()修饰的函数），普通函数由于不对外暴露，可以不遵循此规范。

8.2.8. 函数返回值

函数返回值本质上与原版 Python 并没有什么区别，但是由于 DataFlux.f(x)中暴露的函数需要与 DataFlux Studio 进行交互，必须考虑返回值可以 DataFlux Studio 正常解析。因此，对于被@DFF.API()修饰的函数，需要按照一定规范设计返回值。

总是返回能够被序列化为 JSON 的数据

在 Python，可以被正常序列化的数据为 dict、list、tuple、int、float、str、unicode、bool、None 等基本内置数据类型。
此外，对于第三方库特有的对象实例，如果实现了对应的序列化方法（即虽然不是内置数据类型，但依然可以序列化为 JSON）也可以返回。

```python
@DFF.API('正确示范')
def hello_world():
    '''
    返回可序列化为 JSON 的数据
    '''
    ret = {
        'list': [1, 1.1, True, False, None],
        'dict': {'int': 1, 'float': 1.1, 'bool': True, 'null': None},
        'int': 1,
        'float': 1.1,
        'bool': True,
        'null': None,
    }
```

如指定在@DFF.API()中指定了分类 category，要根据分类要求返回固定格式的返回值

详情见下文「附带分类的函数」

不要返回多个返回值

由于函数返回值在 JSON 序列化后会包裹在 result 字段中，方便程序获取，而返回多个返回值会导致 result 变成数组。

```
@DFF.API('错误示范')
def hello_world():
    '''
    请勿返回多个返回值
    '''
    return True, [1, 2, 3]
```

```
@DFF.API('正确示范')
def hello_world():
    '''
    一次只返回一个返回值
    '''
    ret = {
        'ok': True,
        'data': [1, 2, 3]
    }
    return ret
```

错误应当使用 raise 语句抛出，而不是 return

由于 DataFlux.f(x)本身需要监控函数执行成功与否。因此，当函数执行发生错误时，应当使用 raise 语句将问题抛出，而不是 return。否则 DataFlux.f(x)无法获知函数是否正确执行。

```
@DFF.API('错误示范')
def hello_world():
    '''
    请勿使用 return 代替 raise
    '''
    return False
```

```
@DFF.API('正确示范')
def hello_world():
    '''
    出错应当使用 raise 语句
    '''
    raise Exception('Some error message')
```

8.3. 使用内置功能

为了方便脚本编写，以及在脚本中使用各种 DataFlux.f(x)提供的功能。
DataFlux.f(x)在脚本运行上下文注入了一些额外功能。
使用这些功能不需要额外 import，直接使用即可。
这些功能都封装在 DFF 对象中（如上文出现的@DFF.API）

8.3.1. 将函数对外暴露（DFF.API）

一个装饰器，用于将被修饰的函数对外暴露，允许使用 API 方式调用。如：

```
@DFF.API('问候')
def hello_world(you_name=None):
    '''
    本函数返回一句问候语
    参数:
        your_name 可选，字符串。问候者姓名
    返回:
        字符串
    '''
    if you_name is None:
        your_name = 'My friend'
    return 'Hello, World! ' + your_name
```

此外，DFF.API 还可以额外指定分类 category，标示特定用途，如：

```
@DFF.API('一个预测函数', category='prediction')
def demo_prediction(dps):
    '''
    一个预测函数
    '''
    pass
```

有关分类 category 的内容，详见下文「附带分类的函数」

8.3.2. 操作数据源

在脚本编辑器左侧边栏配置的、内置的所有数据源，都可以在脚本中使用配置的「引用名」获取对应的数据源操作对象。需要查询数据时，使用此操作对象即可，如：

```
@DFF.API('InfluxDB 操作演示')
def influxdb_demo():
    '''
    本函数从数据源`demo_influxdb`中查询 3 条`demo`指标并返回
    '''
    db = DFF.SRC('demo_influxdb')
    return db.query('SELECT * FROM demo LIMIT 3')
```

如数据源配置了默认数据库，则查询会在默认数据库进行。
如数据源没有配置默认数据库，或在查询时需要查询不同的数据库，可在获取数据源操作对象时，指定数据库 database 参数，如：

```
db = DFF.SRC('demo_influxdb', database='my_database')
```

2020-04-21 新增功能
对于 DataFlux DataWay，可以获取数据源操作对象时指定 DataWay 的 rp 和 token 参数，如：

```
dataway = DFF.SRC('df_dataway', token='xxxxx', rp='rp0')
```

由于数据源具有不同类型，各个数据源操作对象略有区别，详情见下文。

InfluxDB 数据源操作对象

InfluxDB 数据源操作对象为 Python 第三方包 influxdb（版本 5.2.3）的封装，主要提供一些查询方法。

InfluxDBHelper.query()

query()方法用于执行 InfluxQL 语句，参数如下：

参数	类型	是否必须	默认值	说明
sql	str	必须		InfluxQL 语句，可包含绑定参数占位符形式为 $var_name
bind_params	dict	可选	None	绑定参数
database	str	可选	None	本次查询指定数据库
dict_output	dict	可选	False	返回数据自动转换为{列名:值}形式

示例如下：

```
sql = 'SELECT * FROM demo WHERE city = $city LIMIT 5'
bind_params = {'city': 'hangzhou'}
db_res = db.query(sql, bind_params=bind_params, database='demo')
# {'series': [{'columns': ['time', 'city', 'hostname', 'status', 'value'], 'name': 'demo',
'values': [['2018-12-31T16:00:10Z', 'hangzhou', 'webserver', 'UNKNOW', 90], ['2018-12-
31T16:00:20Z', 'hangzhou', 'jira', 'running', 40], ['2018-12-31T16:00:50Z', 'hangzhou',
'database', 'running', 50], ['2018-12-31T16:01:00Z', 'hangzhou', 'jira', 'stopped', 40],
['2018-12-31T16:02:00Z', 'hangzhou', 'rancher', 'UNKNOW', 90]]}], 'statement_id': 0}
```

```
sql = 'SELECT * FROM demo WHERE city = $city LIMIT 5'
bind_params = {'city': 'hangzhou'}
db_res = db.query(sql, bind_params=bind_params, database='demo', dict_output=True)
# {'series': [[{'city': 'hangzhou', 'hostname': 'webserver', 'status': 'UNKNOW', 'time':
'2018-12-31T16:00:10Z', 'value': 90}, {'city': 'hangzhou', 'hostname': 'jira', 'status':
'running', 'time': '2018-12-31T16:00:20Z', 'value': 40}, {'city': 'hangzhou', 'hostname':
'database', 'status': 'running', 'time': '2018-12-31T16:00:50Z', 'value': 50}, {'city':
'hangzhou', 'hostname': 'jira', 'status': 'stopped', 'time': '2018-12-31T16:01:00Z',
'value': 40}, {'city': 'hangzhou', 'hostname': 'rancher', 'status': 'UNKNOW', 'time': '2018-
12-31T16:02:00Z', 'value': 90}]], 'statement_id': 0}
```

InfluxDBHelper.query2()

Query2()方法同样用于执行 InfluxQL 语句，但参数占位符不同，使用问号?作为参数占位符。参数列表如下：

参数	类型	是否必须	默认值	说明
sql	str	必须		InfluxQL 语句，可包含参数占位符 ? 表示需要转义的参数 ?? 表示不需要转义的参数
sql_params	list	可选	None	参数
database	str	可选	None	本次查询指定数据库
dict_output	dict	可选	False	返回数据自动转换为{列名:值}形式

示例如下：

```
sql = 'SELECT * FROM ?? WHERE city = ? LIMIT 5'
sql_params = ['demo', 'hangzhou']
db_res = db.query2(sql, sql_params=sql_params, dict_output=True)
# {'series': [[{'city': 'hangzhou', 'hostname': 'webserver', 'status': 'UNKNOW', 'time':
'2018-12-31T16:00:10Z', 'value': 90}, {'city': 'hangzhou', 'hostname': 'jira', 'status':
'running', 'time': '2018-12-31T16:00:20Z', 'value': 40}, {'city': 'hangzhou', 'hostname':
'database', 'status': 'running', 'time': '2018-12-31T16:00:50Z', 'value': 50}, {'city':
'hangzhou', 'hostname': 'jira', 'status': 'stopped', 'time': '2018-12-31T16:01:00Z',
'value': 40}, {'city': 'hangzhou', 'hostname': 'rancher', 'status': 'UNKNOW', 'time': '2018-
12-31T16:02:00Z', 'value': 90}]], 'statement_id': 0}
```

MySQL 数据源操作对象

MySQL 数据源操作对象主要提供一些数据方法。

MySQLHelper.query()

query()方法用于执行 SQL 语句，参数如下：

参数	类型	是否必须	默认值	说明
sql	str	必须		SQL 语句，可包含参数占位符 ? 表示需要转义的参数 ?? 表示不需要转义的参数
sql_params	list	可选	None	参数

示例如下：

```
sql = 'SELECT * FROM ?? WHERE seq > ?'
sql_params = ['demo', 1]
db_res = db.query(sql, sql_params=sql_params)
# [{'createTime': datetime.datetime(2019, 12, 9, 15, 32, 45), 'id': 'id-002', 'seq': 2,
'updateTime': datetime.datetime(2019, 12, 9, 15, 32, 49), 'value': 'value-002'}]
```

Redis 数据源操作对象

Redis 数据源操作对象主要提供 Redis 的操作方法。

RedisHelper.query()

query()方法用于执行 Redis 命令，参数如下：

参数	类型	是否必须	默认值	说明
*args	[str]	必须		Redis 命令及参数

示例如下：

```
db.query('SET', 'myKey', 'myValue', 'nx')
db_res = db.query('GET', 'myKey')
# b'myValue'
```

注意：Redis 返回的值类型为 bytes。实际操作时，可以根据需要进行类型转换，如：

```
db_res = db.query('GET', 'intValue')
print(int(db_res))

db_res = db.query('GET', 'strValue')
print(str(db_res))

db_res = db.query('GET', 'jsonValue')
print(json.loads(db_res))
```

ClickHouse 数据源操作对象

ClickHouse 数据源操作对象主要提供一些数据方法。

ClickHouseHelper.query()

query()方法用于执行 SQL 语句，参数如下：

参数	类型	是否必须	默认值	说明
sql	str	必须		SQL 语句，可包含参数占位符 ? 表示需要转义的参数 ?? 表示不需要转义的参数
sql_params	list	可选	None	参数

示例如下：

```
sql = 'SELECT * FROM ?? WHERE age > ?'
sql_params = ['demo_table', 50]
db_res = helper.query(sql, sql_params=sql_params)
# [{'age': 99, 'demoDate': datetime.date(1970, 1, 1), 'name': 'oldman'}, {'age': 99,
'demoDate': datetime.date(1970, 1, 1), 'name': 'oldman'}]
```

DataFlux DataWay 数据源操作对象

DataFlux DataWay 数据源操作对象主要提供数据写入方法。

DFDatawayHelper.write_point()

write_point()方法用于向 DataWay 写入数据点，参数如下：

参数	类型	是否必须	默认值	说明
measurement	str	必须		指标集名称
tags	dict	可选	None	标签。键名和键值必须都为字符串
fields	dict	可选	None	指标。键名必须为字符串，键值可以为字符串/整数/浮点数/布尔值之一
timestamp	int/long/float	可选	当前时间	时间戳，支持秒/毫秒/微秒/纳秒。

示例如下：

```
dw.write_point(
    measurement='主机监控',
    tags={'host': 'web-01'},
    fields={'cpu': 10})
```

DFDatawayHelper.write_points()

write_point()的批量版本。

示例如下：

```
points = [
    {
        'measurement': '主机监控',
        'tags'       : {'host': 'web-01'},
        'fields'     : {'value': 10},
    },
    {
        'measurement': '主机监控',
        'tags'       : {'host': 'web-02'},
        'fields'     : {'value': 20},
    }
]
dw.write_points(points)
```

DFDatawayHelper.write_keyevent()

write_keyevent()方法用于向 DataWay 写入关键事件，参数如下：

参数	类型	是否必须	默认值	说明
title	str	必须		标题
timestamp	int/long/float	必须		时间戳，支持秒/毫秒/微秒/纳秒。
des	str	可选	None	描述
link	str	可选	None	关联的完整外部链接地址，如："http://some-domain/some-path"
source	str	可选	None	来源
tags	dict	可选	None	标签。键名和键值必须都为字符串

示例代码：

```
dw.write_keyevent(
    title='用户活跃比例超过 50%',
    timestamp=1577808000,
    des='用户活跃比例超过 50%，请考虑增加机器数量',
    link='http://my-app.com',
    source='用户监控',
    tags={'监控点': '001'})
```

DFDatawayHelper.write_keyevents()

write_keyevent()的批量版本。

示例代码：

```
keyevents = [
    {
        title    : '用户活跃比例超过 50%',
        timestamp: 1577808000,
        des      : '用户活跃比例超过 50%，请考虑增加机器数量',
        link     : 'http://my-app.com',
        source   : '用户监控',
        tags     : {'监控点': '001'},
    },
    {
        title    : '发帖量超过 10 万',
        timestamp: 1577808001,
        des      : '发帖量超过 10 万，请考虑增加缓存服务器',
        link     : 'http://my-bbs.com',
        source   : '论坛监控',
        tags     : {'监控点': '003'},
    },
]
dw.write_keyevents(keyevents)
```

DFDatawayHelper.write_flow()

write_flow()方法用于向 DataWay 写入流程行为，参数如下：

参数	类型	是否必须	默认值	说明
app	str	必须		应用名
trace_id	str	必须		标示一个流程单的唯一 ID
name	str	必须		节点名称
timestamp	int/long/float	必须		时间戳，支持秒/毫秒/微秒/纳秒。
duration	int/long	二选一		在当前节点滞留时间或持续时间（秒）
duration_ms	int/long	二选一		在当前节点滞留时间或持续时间（毫秒）
parent	str	可选	None	上一个节点的名称。第一个节点不用上报
tags	dict	可选	None	标签。键名和键值必须都为字符串
fields	dict	可选	None	指标。键名必须为字符串，键值可以为字符串/整数/浮点数/布尔值之一

示例代码：

```
dw.write_flow(
        app='OA',
        trace_id='TRACE-001',
        name='提交请假审批',
        duration=100,
        tags={'提交人': '张三'},
        fields={'时长': '72'},
        timestamp=1577808000)
```

DFDatawayHelper.write_flows()

write_flows()的批量版本。

示例代码：

```
flows = [
    {
        'app'      : 'OA',
        'trace_id' : 'TRACE-001',
        'name'     : '提交请假审批',
        'duration' : 100,
        'tags'     : {'提交人': '张三'},
        'fields'   : {'时长': '72'},
        'timestamp': 1577808000,
    },
    {
        'app'        : 'OA',
        'trace_id'   : 'TRACE-002',
        'name'       : '请假主管审批',
        'duration_ms': 100000,
        'parent'     : '提交请假审批',
        'tags'       : {'审批人': '李四'},
        'fields'     : {'意见': '同意'},
        'timestamp'  : 1577808001,
    },
]
dw.write_flows(flows)
```

DFDatawayHelper.write_alert()

2020-04-21 新增功能

write_alert()方法用于向 DataWay 写入告警，参数如下：

参数	类型	是否必须	默认值	说明
level	str	必须		告警级别，必须为 critical、warning、info、ok 之一
alert_id	str	必须		告警 ID
check_value	json/str(json)	必须		检测值 JSON 或 JSON 字符串
timestamp	int/long/float	必须		时间戳，支持秒/毫秒/微秒/纳秒。
duration	int/long	二选一		在当前节点滞留时间或持续时间（秒）
duration_ms	int/long	二选一		在当前节点滞留时间或持续时间（毫秒）
rule_id	str	可选		规则 ID
rule_name	str	可选		规则名
no_data	bool	可选		是否为无数据告警
action_type	str	可选		动作类型
action_content	json/str(json)	可选		动作数据 JSON 或 JSON 字符串
alert_item_tags	dict	可选	None	告警对象标签。键名和键值必须都为字符串
tags	dict	可选	None	额外标签。键名和键值必须都为字符串

示例如下：

```
dw.write_alert(
    level='critical',
    alert_id='ALERT-001',
    rule_id='RULE-001',
    rule_name='R1',
    no_data=True,
    duration=0,
    check_value={'M1': 90, 'M2': 90},
    action_type='mail',
    action_content={'to': 'someone@somemail.com', 'title': 'Test Alert Title', 'content':
'Test Alert Value'},
    alert_item_tags={'T1': 'X', 'T2': 'Y'},
    tags={'T1': 'X'},
    timestamp=1577808000)
```

DFDatawayHelper.write_alerts()

2020-04-21 新增功能

write_alert()的批量版本。

示例如下：

```
alerts = [
    {
        'level'          : 'critical',
        'alert_id'       : 'ALERT-001',
        'rule_id'        : 'RULE-001',
        'rule_name'      : 'R1',
        'no_data'        : True,
        'duration'       : 0,
        'check_value'    : {'M1': 90, 'M2': 90},
        'action_type'    : 'mail',
        'action_content' : {'to': 'someone@somemail.com', 'title': 'Test Alert Title',
'content': 'Test Alert Value'},
        'alert_item_tags': {'T1': 'X', 'T2': 'Y'},
        'tags'           : {'T1': 'X'},
        'timestamp'      : 1577808000,
    },
    {
        'level'          : 'ok',
        'alert_id'       : 'ALERT-001',
        'rule_id'        : 'RULE-001',
        'rule_name'      : 'R1',
        'no_data'        : False,
        'duration_ms'    : 10000,
        'check_value'    : {'M1': 10, 'M2': 10},
        'action_type'    : 'mail',
        'action_content' : {'to': 'someone@somemail.com', 'title': 'Test Alert Title',
'content': 'Test Alert Value'},
        'alert_item_tags': {'T1': 'X', 'T2': 'Y'},
        'tags'           : {'T1': 'X'},
        'timestamp'      : 1577808001,
    },
]
dw.write_alerts(alserts)
```

8.3.3. 获取环境变量

在脚本编辑器左侧边栏配置的所有环境变量，都可以在脚本中使用配置的「引用名」获取对应的环境变量值。

示例代码如下：

```
company_name = DFF.ENV('companyName')
# '上海驻云信息科技有限公司'
```

由于环境变量都是在 Web 中进行配置，因此所有的环境变量值都是字符串类型。使用时需要按照需要进行类型转换，如：

```
try:
    page_size = int(DFF.ENV('pageSize'))
except Exception as e:
    pass
```

8.3.4. 输出日志

由于脚本编辑器是 Web 应用程序，不同于一般意义上的 IDE，无法进行单步调试等操作。因此，调试主要依靠运行时输出日志进行。

为了让日志输出能够被 DataFlux.f(x)搜集并显示到脚本编辑器中，DataFlux.f(x)提供了专门的 DFF.log()日志输出函数。

```
DFF.log('Some log message')
```

8.4. 附带分类的函数

在使用@DFF.API()装饰器时，可以额外指定分类 category 参数。category 参数是字符串枚举值，包含以下选项：

- `prediction`：预测函数
- `transformation`：转换函数
- `action`：动作函数
- `command`：命令函数
- `query`：查询函数

2020-04-21 新增功能

- `check`：检测函数

指定了分类参数的函数，会被 DataFlux 获取，并可以直接在 DataFlux Studio 特定的页面中使用。因此不同的分类函数具有额外的设计规范。

8.4.1. DataFlux 标准数据结构

由于需要与 DataFlux Studio 进行交互，DataFlux 规定了一些标准的数据结构。这些结构在附带分类的函数中非常常见，且**作为参数时，参数名必须固定为其英文名**。

注意：函数的其他参数不要与固定参数名相同。

dps 时序数据

dps 时序数据为 DataFlux Studio 图表中的系列数据。一个 dps 时序数据在 DataFlux Studio 的图表中为一个数据系列，如一条折线、一排柱状图等。

dps 时序数据的固定参数名为 **`dps`**，结构描述如下：

[[UNIX 毫秒时间戳，值]，[UNIX 毫秒时间戳，值]，...]

具体示例如下：

```
[
  [ 1577808000000, 100 ],
  [ 1577808060000, 120 ],
  ...
]
```

start_time、end_time 起止时间

start_time、end_time 起止时间一般为图表操作时指定的时间范围端点，为 **UNIX 毫秒时间戳**。可能由用户选择输入，也有可能为 DataFlux Studio 全局时间范围填入。

start_time、end_time 起止时间的固定参数名为 **`start_time`**、**`end_time`**。

具体示例如下：

```
1577808000000
```

8.4.2. 预测函数

预测函数主要用于 DataFlux Studio 中图表分析模式，接收图表中的 dps 时序数据，返回预测后的 dps 时序数据。规范如下：

- 分类 category 为 **prediction**
- 第 1 个参数为固定参数，必须为 dps 时序数据。值为图表中单个数据系列的时序数据
- 可以不附带或附带一个或多个可选参数
- 返回值同样为 dps 时序数据。内容为预测所得的时序数据

示例如下：

```
@DFF.API('预测函数 1', category='prediction')
def my_prediction_1(dps):
    '''
    一个预测函数，只需要固定参数 dps
    '''
    # 进行数据预测，结果保存在 result 变量中
    return result

@DFF.API('预测函数 2', category='prediction')
def my_prediction_2(dps, param1=None, param2=None):
    '''
    一个预测函数，需要固定参数 dps 以及额外参数
    '''
    # 进行数据预测，结果保存在 result 变量中
    return result
```

8.4.3. 转换函数

转换函数主要用于 DataFlux Studio 中图表分析模式，接收图表中的 dps 时序数据，返回转换后的 dps 时序数据，规范如下：

- 分类 category 为 **transformation**。
- 第 1 个参数为固定参数，必须为 dps 时序数据。值为图表中单个数据系列的时序数据
- 可以不附带或附带一个或多个可选参数
- 返回值同样为 dps 时序数据。内容为转换所得的时序数据

示例如下：

```python
@DFF.API('转换函数 1', category='transformation')
def my_transformation_1(dps):
    '''
    一个转换函数，只需要固定参数 dps
    '''
    # 进行数据转换，结果保存在 result 变量中
    return result

@DFF.API('转换函数 2', category='transformation')
def my_transformation_2(dps, param1=None, param2=None):
    '''
    一个转换函数，需要固定参数 dps 以及额外参数
    '''
    # 进行数据转换，结果保存在 result 变量中
    return result
```

8.4.4. 行为函数

行为函数主要用于 DataFlux Studio 中基线告警的触发执行。没有特定的参数和返回值要求，规范如下：

- 分类 category 为 **action**
- 可以不附带或附带一个或多个可选参数
- 不需要返回值

示例如下：

```python
@DFF.API('行为函数 1', category='action')
def my_action_1():
    '''
    一个行为函数，没有参数和返回值要求
    '''
    pass

@DFF.API('行为函数 2', category='action')
def my_action_2(param1=None, param2=None):
    '''
    一个行为函数，没有参数和返回值要求
    '''
    pass
```

注意：「行为函数」有时候会被称为「动作函数」，是相同意思

8.4.5. 命令函数

命令函数主要用于 DataFlux Studio 中的点击自定义按钮的触发执行。没有特定的参数和返回值要求，规范如下：

- 分类 category 为 **command**
- 可以不附带或附带一个或多个可选参数
- 不需要返回值

示例如下：

```
@DFF.API('命令函数 1', category='command')
def my_command_1():
    '''
    一个命令函数，没有参数和返回值要求
    '''
    pass

@DFF.API('命令函数 2', category='command')
def my_command_2(param1=None, param2=None):
    '''
    一个命令函数，没有参数和返回值要求
    '''
    pass
```

8.4.6. 查询函数

查询函数主要用于 DataFlux Studio 中自定义查询展示图表的功能，可以接收图表界面指定的 start_time、end_time 起止时间（也可以不接收），返回查询获得的 dps 时序数据，规范如下：

- 分类 category 为 **query**
- 如存在参数 start_time、end_time 起止时间，则必须为第 1、第 2 个参数，值为 UNIX 毫秒时间戳
- 可以不附带或附带一个或多个可选参数
- 返回值为 dps 时序数据。内容为查询所得的时序数据

示例如下：

```python
@DFF.API('查询函数 1', category='query')
def my_query_1():
    '''
    一个查询函数，没有参数
    '''
    # 执行查询，结果保存在 result 中
    return result

@DFF.API('查询函数 2', category='query')
def my_query_2(start_time, end_time):
    '''
    一个查询函数，包含起止时间参数
    '''
    # 执行查询，结果保存在 result 中
    return result

@DFF.API('查询函数 3', category='query')
def my_query_3(start_time):
    '''
    一个查询函数，只包含开始时间或结束时间
    '''
    # 执行查询，结果保存在 result 中
    return result

@DFF.API('查询函数 4', category='query')
def my_query_4(end_time, start_time=None):
    '''
    一个查询函数，包含起止时间参数，且其中一项可选（也可以两者都可选）
    根据 Python 要求，可选参数必须在必选参数后面，所以此处 start_time 防止在 end_time 之后
    '''
    # 执行查询，结果保存在 result 中
    return result

@DFF.API('查询函数 5', category='query')
def my_query_5(param1=None, param2=None):
    '''
    一个查询函数，没有起止时间，但附带自定义参数
    '''
    # 执行查询，结果保存在 result 中
    return result
```

```
@DFF.API('查询函数 6', category='query')
def my_query_6(start_time, end_time, param1=None, param2=None):
    '''
    一个查询函数，包含起止时间，并附带自定义参数
    起止时间同样允许只存在其一，允许其中之一或两者可选
    '''
    # 执行查询，结果保存在 result 中
    return result
```

8.4.7. 检测函数

2020-04-21 新增功能

检测函数主要用于 DataFlux Studio 中的高级检测功能。没有特定的参数和返回值要求，规范如下：

- 分类 category 为 **check**
- 可以不附带或附带一个或多个可选参数
- 不需要返回值

示例如下：

```
@DFF.API('检测函数 1', category='check')
def my_check_1():
    '''
    一个检测函数，没有参数和返回值要求
    '''
    pass

@DFF.API('检测函数 2', category='check')
def my_check_2(param1=None, param2=None):
    '''
    一个检测函数，没有参数和返回值要求
    '''
    pass
```

8.5. 调用另一个脚本中的函数

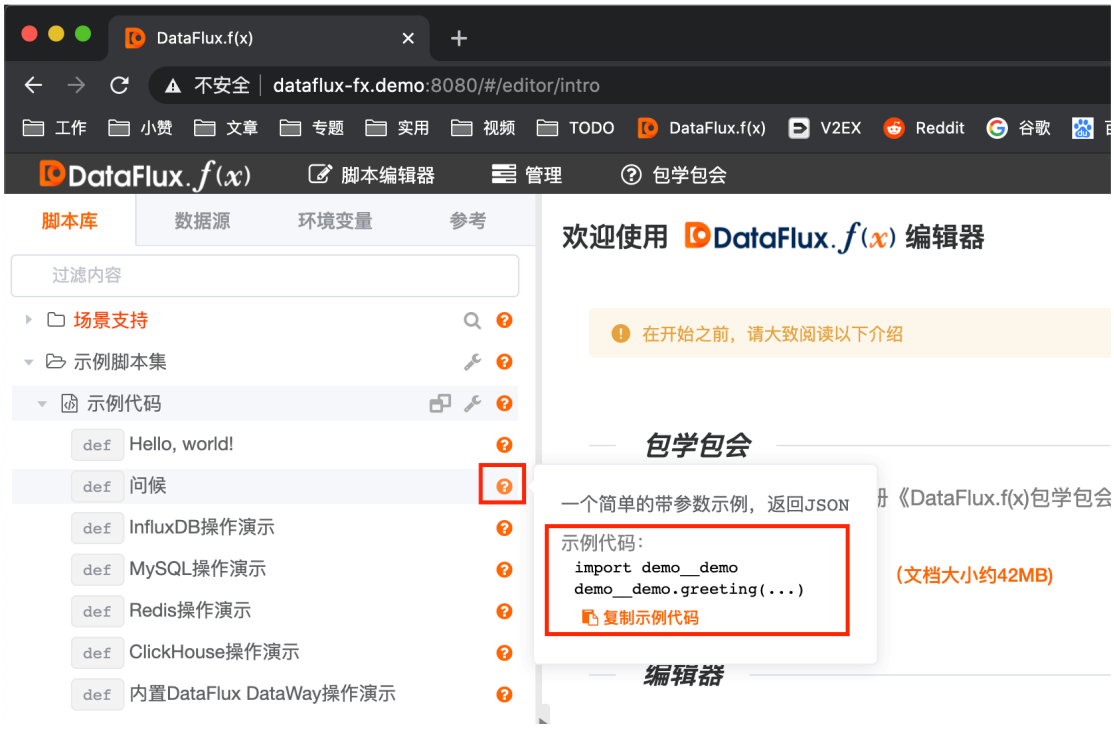
Python 代码可以根据功能、用途等不同需求划分到不同的脚本或脚本集中。位于不同脚本的代码是允许相互调用的。
需要调用另一个脚本中的函数时，只需要导入对应脚本即可。

导入另一个脚本时，必须按照固定写法：

```
# import <脚本集引用名>__<脚本引用名>
import demo__demo2

# 或 import <脚本集引用名>__<脚本引用名> as 别名
import demo__demo2 as demo2
```

也可以在脚本编辑器中，可以将鼠标指向脚本项右侧的问号，直接复制导入语句。



脚本或脚本集并不是 Python 中的模块本身并不能通过 import 语句导入。但 DataFlux.f(x)内部实现了动态加载机制，并允许使用 import 语句加载动态代码，因此，**以下写法都是错误的**。

```
# 错误写法 1：将脚本集当作模块导入
import demo

# 错误写法 2：将脚本当作模块导入
import demo.demo2
from demo import demo2

# 错误写法 3：只导入某个函数
from demo.demo2 import some_func
from demo__demo2 import some_func
```

此外，在导入脚本时，应当注意不要产生循环引用，如：

```
# 脚本集 demo 下的脚本 demo1
import demo__demo2

# 脚本集 demo 下的脚本 demo2
import demo__demo3

# 脚本集 demo 下的脚本 demo3
import demo__demo1
```

注意，在同时编辑多个脚本时，如果当前导入了另一个脚本，那么被引用的脚本实际会以已发布的版本执行。任何时候都不会导入草稿！

8.6. 脚本集、脚本、函数编排设计

脚本集、脚本、函数应当按照一定逻辑编排组织，而不是将代码随意堆砌在一起。合理编排脚本集和脚本有利于代码的维护以及系统运行效率。

8.6.1. 按照用途、类型合理划分脚本集和脚本

一般来说，推荐选用以下几种代码组织方式：

- 为通用处理脚本建立单独的脚本集，取引用名为 utils
- 将通用处理函数按照用途建立单独的脚本，如：时间处理类 time，字符串处理类 string，单位转换类 converter 等
- 为业务处理脚本按照行业、项目、组织等方式建立单独的脚本集，如：eShop、IoT、monitor、sales、marketing 等
- 特定业务专用的通用函数可以在业务脚本集中建立单独的脚本，取引用名 utils
- 为不同分类（category）的函数建立单独的脚本，将同类型的函数放置在相同脚本中，如 general、prediction、transformation、action、command、query、check
- 代码量过多时，根据使用频率划分低频使用的脚本和高频使用的脚本，如 prediction、advanced_prediction

8.6.2. 单个脚本的代码量及依赖链

由于 DataFlux.f(x)运行脚本时，采用动态加载需要的脚本执行。此外，如果其中导入了另一个脚本，被导入的脚本也会被动态加载。

因此，如果某个脚本中的某些函数会被特别频繁地调用，建议将其单独提取为独立的脚本，减少加载消耗。单个脚本大小建议控制在 1000 行以内。

此外，也要尽量避免过长的依赖链，导致无意义的性能损耗。如：

- 脚本 1 依赖脚本 2
- 脚本 2 依赖脚本 3
- 脚本 3 依赖脚本 4
- 脚本 4 依赖脚本 5
- …

Python 内置模块和第三方模块不受此限制影响

8.7. 常见问题处理方式

Python 是一个相当便捷的语言，特定的问题都有比较套路化的处理方式。以下是一些常见问题的处理方式

8.7.1. 使用列表生成式生成列表

Python 的列表生成器语法可以快速生成列表，在逻辑相对简单时，非常适合。但要注意的是，过分复杂的处理逻辑会使代码难以阅读。因此复杂逻辑建议直接使用 for 循环处理。

示例如下：

```
# 按时间生成 dps 数据
import time
dps = [ [(int(time.time()) + i) * 1000, i ** 2] for i in range(5) ]
dps = list(dps)
print(dps)
# [[1583172382000, 0], [1583172383000, 1], [1583172384000, 4], [1583172385000, 9],
[1583172386000, 16]]
```

```
# 将输入的 dps 乘方
dps = [
    [1583172563000, 0],
    [1583172564000, 1],
    [1583172565000, 2],
    [1583172566000, 3],
    [1583172567000, 4]
]
dps = [ [d[0], d[1] ** 2] for d in dps ]
dps = list(dps)
print(dps)
# [[1583172563000, 0], [1583172564000, 1], [1583172565000, 4], [1583172566000, 9],
[1583172567000, 16]]
```

8.7.2. 使用内置 map 函数处理列表

Python 内置的 map 函数是另一个方便处理列表的函数。逻辑简单时可以使用 lambda 表达式；逻辑复杂时，可以先定义函数后作为参数传入。

示例如下：

```
# 使用 lambda 表达式将输入的 dps 乘方
dps = [
    [1583172563000, 0],
    [1583172564000, 1],
    [1583172565000, 2],
    [1583172566000, 3],
    [1583172567000, 4]
]
dps = map(lambda x: [x[0], x[1] ** 2], dps)
dps = list(dps)
print(dps)
# [[1583172563000, 0], [1583172564000, 1], [1583172565000, 4], [1583172566000, 9],
[1583172567000, 16]]
```

```
# 使用传入函数将输入的 dps 乘方
dps = [
    [1583172563000, 0],
    [1583172564000, 1],
    [1583172565000, 2],
    [1583172566000, 3],
    [1583172567000, 4]
]
def get_pow(x):
    timestamp = x[0]
    value     = x[1] ** 2
    return [timestamp, value]

dps = map(get_pow, dps)
dps = list(dps)
print(dps)
# [[1583172563000, 0], [1583172564000, 1], [1583172565000, 4], [1583172566000, 9],
[1583172567000, 16]]
```

8.7.3. 获取 JSON 数据中多层嵌套中的值

有时函数会获取到一个嵌套层数较多的 JSON，同时需要获取某个层级下的值。可以使用 try 语法快速获取。

```
# 获取 level3 的值
input_data = {
    'level1': {
        'level2': {
            'level3': 'value'
        }
    }
}
value = None
try:
    value = input_data['level1']['level2']['level3']
except Exception as e:
    pass

print(value)
# value
```

注意：使用此方法时，try 代码块中不要添加其他任何无关代码，否则可能导致应当抛出的异常被跳过

8.7.4. 使用 arrow 库正确处理时间

Python 内置的日期处理模块在使用上有一定复杂度，并且对时区的支持并不好。在处理时间时，建议使用第三方 arrow 模块。

示例如下：

```
import arrow

# 获取当前 Unix 时间戳
print(arrow.utcnow().timestamp)
# 1583174345

# 获取当前北京时间字符串
print(arrow.now('Asia/Shanghai').format('YYYY-MM-DD HH:mm:ss'))
# 2020-03-03 02:39:05

# 从 Unix 时间戳解析，并输出北京时间字符串
print(arrow.get(1577808000).to('Asia/Shanghai').format('YYYY-MM-DD HH:mm:ss'))
# 2020-01-01 00:00:00

# 从 ISO8601 时间字符串解析，并输出北京时间字符串
print(arrow.get('2019-12-31T16:00:00Z').to('Asia/Shanghai').format('YYYY-MM-DD HH:mm:ss'))
# 2020-01-01 00:00:00

# 从非标准时间字符串解析，并作为北京时间字符串
print(arrow.get('2020-01-01 00:00:00', 'YYYY-MM-DD
HH:mm:ss').replace(tzinfo='Asia/Shanghai').format('YYYY-MM-DD HH:mm:ss'))
# 2020-01-01 00:00:00

# 时间运算：获取前一天，并输出北京时间字符串
print(arrow.get('2019-12-31T16:00:00Z').shift(days=-1).to('Asia/Shanghai').format('YYYY-MM-
DD HH:mm:ss'))
# 2019-12-31 00:00:00
```

更详细内容，请参考 arrow 官方文档：
https://arrow.readthedocs.io/

8.7.5. 向外部发送 HTTP 请求

Python 内置的 http 处理模块在使用上有一定复杂度。在发送 http 请求时，建议使用第三方 requests 模块。

示例如下：

```
import requests

# 发送 GET 请求
r = requests.get('https://api.github.com/')
print(r.status_code)
print(r.json())

# form 方式发送 POST 请求
r = requests.post('https://httpbin.org/post', data={'key':'value'})
print(r.status_code)
print(r.json())

# json 方式发送 POST 请求
r = requests.post('https://httpbin.org/post', json={'key':'value'})
print(r.status_code)
print(r.json())
```

更详细内容，请参考 requests 官方文档：
https://requests.readthedocs.io/

8.7.6. 发送钉钉机器人消息

钉钉自定义机器人可以简单的通过 POST 请求将消息推送到群中，是作为触发提示的不二选择。
由于只需要发送 HTTP 请求即可，因此选择使用 requests 模块。

示例如下：

```
import requests

url = 'https://oapi.dingtalk.com/robot/send?access_token=xxxxx'
body = {
    'msgtype': 'text',
    'text': {
        'content': '钉钉机器人提示信息 @180xxxx0000'
    },
    'at': {
        'atMobiles': [
            '180xxxx0000'
        ]
    }
}
requests.post(url, json=body)
```

更详细内容，请参考钉钉机器人官方文档：
https://ding-doc.dingtalk.com/doc#/serverapi2/qf2nxq

8.7.7. 避免 SQL 注入

当需要将函数的传入参数作为 SQL 语句的参数时，需要注意避免 SQL 注入问题。应当始终使用内置的数据源操作对象提供的 sql_params 参数，而不要直接拼接 SQL 语句字符串。

以下为正确示例：

```
helper.query('SELECT * FROM table WHERE id = ?', sql_params=[target_id])
```

以下错误示范：

```
helper.query("SELECT * FROM table WHERE id = '{}'".format(target_id))
helper.query("SELECT * FROM table WHERE id = '%s'" % target_id)
```

> *实际各类型的数据源操作对象使用方法略有不同，请参考「9.3.2 操作数据源」章节*

百度百科「SQL 注入」词条：
https://baike.baidu.com/item/sql%E6%B3%A8%E5%85%A5

W3Cschool「MySQL 及 SQL 注入」章节：
https://www.w3cschool.cn/mysql/mysql-sql-injection.html

9. FAQ

为什么使用 Python 作为脚本语言？

首先，Python 这门语言较为简单易上手。其次，DataFlux 主要处理的数据为监控等时序数据，同时 Python 生态已经大量存在了专业的数学、机器学习等第三方库，可以方便地满足业务需要。

为什么选用 Python 3.7 版本？

首先，Python 2.x 版本已于 2020 年 1 月 1 日停止维护，Python 3.x 是大势所趋。其次，Python 3.7 版本为目前的稳定版本，各第三方库对 3.7 的支持也最全面。

为什么只允许使用特定的 Python 库？

主要出于安全性的考虑，一些涉及系统的库的不当使用可能导致损害系统，因此使用白名单制限制可用的 Python 库

我能选择不同的 Python 版本来运行脚本吗？

不可以，DataFlux.f(x)是服务端应用，且为了性能，并没有使用虚拟化技术来运行用户脚本，因此用户脚本都运行在 Python 3.7 环境下

我能够修改官方提供的脚本吗？

不可以，官方脚本受到保护，不允许修改。如需要，可以编写新的函数调用官方脚本中的函数。

如何保证官方脚本未被篡改？

首先官方脚本包加入了数字签名，只有驻云官方发布的官方脚本包才能够被认证通过并导入到 DataFlux.f(x)中。其次，已经在脚本库中的官方脚本也不允许修改。

我可以使用哪些 Python 内置库或第三方库？

可用的 Python 内置库有：

- base64
- collections
- datetime
- dns
- hashlib
- itertools
- json
- math
- random
- re
- time
- uuid
- pprint
- wraps

可用的第三方库有：

- six（v1.12.0）
- arrow（v0.13.0）
- requests（v2.21.0）
- simplejson（v3.16.0）
- ujson（v1.35）
- xmltodict（v0.9.2）
- PyYAML（v4.2b2）
- numpy（v1.18.1）
- pandas（v0.25.3）
- scipy（v1.3.3）
- scikit-learn（v0.22.1）
- statsmodels（v0.11.0）
- fbprophet（v0.5）

2020-04-21 新增功能

- eas-prediction（v0.12）

我需要的 Python 库不在可用列表中，怎么办？

由于可用的 Python 库采用白名单制，额外增加可用库只能联系 DataFlux 官方说明需要的库，并在下一个版本更新后可用。

为什么 InfluxDB 无法使用 JOIN 功能？

与传统的关系型数据库不同，InfluxDB 是时序数据库，并不是关系型数据库，因此不具备 JOIN 功能。而大多数情况下对时序数据库的查询也不应该存在 JOIN 操作，必要时请重新评审需求是否合理。

此外，特定情况下，您可以在脚本代码中自行实现类似 JOIN 的联合查询。

为什么 InfluxDB 无法在某些字段上进行 COUNT()处理？

与 MySQL 不同，InfluxDB 的 COUNT()函数处理对象是字段 field 而非标签 tag

DataFlux.f(x)的性能如何？

很大程度上与函数处理内容有关，过分复杂的处理必然拖慢系统。

按照 DataFlux 官方基准测试，单个 Pod 执行 Hello World 函数的性能大概在 200 次/秒左右。

可以根据需要，调整 DataFlux.f(x)部署的 Pod 数量。

函数执行时间过长会怎么样？

按照规范，一个会在 DataFlux Studio 被调用的函数建议执行时间为 3 秒以内，以确保 UI 界面操作的连贯性。

但 DataFlux.f(x)最长允许一个函数执行 30 秒。超过此时长，执行进程会被强制停止。

脚本调试阶段执行了慢查询，是否会对系统有影响？

DataFlux.f(x)调试执行与正式被 API 调用执行分属两个不同的执行单元，本身不会相互影响。但执行慢查询的数据源会收到影响。

可以在脚本中使用类、装饰器吗？

可以，类和装饰器的使用与原版 Python 没有任何区别。

可以从 DataFlux.f(x)连接到已有业务系统的数据库吗？

可以，但需要保证 DataFlux.f(x)能够连通业务系统的数据库。

此外，为了避免可能存在的慢查询影响线上业务系统，建议对业务系统数据库做读写分离，并让 DataFlux.f(x)连接到读库上。

我不想暴露业务系统数据库，可以通过 API 方式访问吗？

可以，DataFlux.f(x)本身可以向外发送 HTTP 请求，可以满足需求。

DataFlux.f(x)如何控制函数被调用的权限？

DataFlux.f(x)本身不提供权限系统，编写的函数默认只能在集群内部被 DataFlux Studio 调用。只有创建的授权链接才能被外网调用。

授权链接可以设置限流，也可以随时禁用。因此暴露函数时，只需要控制授权链接即可。

脚本的运行环境是否绝对安全？

不是绝对安全，Python 是动态脚本语言，虽然 DataFlux.f(x)已经做了一定限制，但无法 100%杜绝猴子补丁等越狱手段。

因此请勿将 DataFlux.f(x)的访问权限交给不可信任的第三方使用。

数据源是否绝对安全？

无法保证，由于函数可以由用户自行编写，因此类似 SQL 注入等问题依然需要脚本编写者注意规避。

DataFlux.f(x)的数据源操作对象支持防止 SQL 注入的传参功能，请勿手工在 SQL 语句中手工拼接外部输入的参数。

因此，除非有特定业务需求，数据源建议使用只读库、只读用户、在数据库中为用户指定明确的权限等方式减少安全隐患。

业务系统所用的数据库没有支持，怎么办？

您可以联系 DataFlux 官方告知您所需要支持的数据库，我们尽快加入新的支持。此外，您也可以在业务系统开放 API 的方式向 DataFlux.f(x)提供数据。

我在使用时遇到了问题，应该怎么办？

如果是 DataFlux.f(x)本身的故障，请联系 DataFlux 官方反馈问题。

如果是使用方面的疑问，请先阅读相关文档。如依然无法解决问题，请联系 DataFlux 官方反馈问题。

我需要专业的预测、机器学习功能，但本身不具备自行实现的条件，应该怎么办？

场景实施本身就是 DataFlux 的一项服务，请立刻联系 DataFlux 官方进行洽谈。

我可以将自己编写的脚本分发给其他人吗？

可以，DataFlux.f(x)并不会限制用户代码的分发。

我无法通过 API 调用函数，我什么？

无法调用函数的问题有很多种

- 请求返回 4xx 或 5xx：
  网络正常，错误原因请参考返回内容。大多数情况为函数本身的问题。
- 请求超时，服务器无响应：
  网络不通，这并不是 DataFlux.f(x)的问题，请检查自身网络、代理等配置是否正确。
- 请求返回未登录或需要登录：
  您可能尝试直接调用了函数而不是调用授权链接。根据 DataFlux.f(x)要求，从集群外部调用函数必须使用授权链接调用

通过 API 调用函数返回了 JSON 字符串，但 HTTP 请求头中没有 Content-Type: application/json，为什么？

您所调用的函数可能直接返回了序列化后的 JSON 字符串。当函数返回 JSON 数据时，请直接返回整个 dict 对象，不需要手动进行序列化操作。

我正在编写 InfluxDB 的查询语句，但是我不清楚数据库、指标、标签有哪些，应该怎么办？

InfluxDB 中的数据与上报至 DataWay 的数据直接关联，DataFlux.f(x)本身并不参与存储处理。

您可以具有后台管理权限的负责人询问工作空间对应的数据库；向实施上报数据的负责人询问相关指标、标签或字段意义。

您也可以使用数据源的「简易调试面板」来浏览所有的数据库、保留策略、指标、指标下的标签、字段。

脚本集、脚本的引用名，函数名长度有限制吗？

有，脚本集、脚本的引用名最长 128 个字符，函数名最长 256 个字符，超过的可能导致函数无法正常通过 API 被调用。

我为我的函数编写了装饰器，参数为使用*args, **kwargs，为什么无法从 args 中获取到参数？

为了方便函数在 Web 页面调用，在通过 API 调用函数时，参数都会以命名参数方式传递（即**kwargs 方式），因此无法在 args 中获取到参数。请使用 kwargs.get()方式以参数名获取参数。

此外，**DataFlux.f(x)反对在对外暴露的函数上使用*args, **kwargs 特性**。这样做会导致前端 Web 无法得知函数具体需要的参数列表。

我是否能够在脚本中使用全局变量？

可以，但仅限只读方式访问定义的全局变量。由于每次函数运行都会初始化执行上下文，因此函数中对全局变量的修改，不会保留到下次函数执行。

DataFlux.f(x)是否是 Flink、Hadoop 等的替代品？

不是，DataFlux.f(x)是 DataFlux 产品的一个组成部分，与目前市面上的大数据处理系统没有什么特定的关系，仅仅是功能上有共同点。

使用 DataFlux.f(x)应当按照 DataFlux 产品本身的逻辑使用，而不是简单替换已有的系统。

附录

DataFlux 官方网站：
https://www.dataflux.cn/
DataFlux 文档中心：
https://help.dataflux.cn/

DataFlux.f(x)在线授权链接函数文档（页面版）
https://您的 DataFlux.f(x)部署域名/#/auth-link-func-doc
DataFlux.f(x)在线授权链接函数文档（JSON 版）
https://您的 DataFlux.f(x)部署域名/api/v1/auth-link-func-list

InfluxDB 文档中心：
https://docs.influxdata.com/
InfluxQL 文档：
https://docs.influxdata.com/influxdb/v1.7/query_language/

MySQL 文档中心：
https://dev.mysql.com/doc/
MySQL SQL 语句文档：
https://dev.mysql.com/doc/refman/5.7/en/sql-statements.html

Redis 文档中心：
https://redis.io/documentation
Redis 命令文档：
https://redis.io/commands

Memcached 文档中心：
https://github.com/memcached/memcached/wiki
Memcached 命令文档：
https://github.com/memcached/memcached/wiki/Commands

ClickHouse 文档中心：
https://clickhouse.tech/docs/zh/
ClickHouse 查询语法文档：
https://clickhouse.tech/docs/zh/query_language/select/

Oracle 文档中心：
https://www.oracle.com/technetwork/cn/indexes/documentation/index.html
Oracle 查询语法文档：
https://www.oracle.com/technetwork/cn/database/database-technologies/sql/documentation/index.html

SQL Server 文档中心：
https://docs.microsoft.com/zh-cn/sql/sql-serve

PostgreSQL 文档中心：
https://www.postgresql.org/docs/

Python 第三方库 arrow 官方文档：
https://arrow.readthedocs.io/

Python 第三方库 requests 官方文档：
https://requests.readthedocs.io/

钉钉机器人官方文档：
https://ding-doc.dingtalk.com/doc#/serverapi2/qf2nxq