天翼云时序数据库 Influx 版

用户指南

天翼云科技有限公司

目 录

1 产品概述

1.1 产品概述

天翼云时序数据库 Influx 版是高性能的时序数据库，完全兼容开源 influxDB 协议，采用先进架构提供时序数据的高速写入和查询服务，免运维、简单易用、支持可视化数据分析，适用于大规模的时序数据存储并进行实时分析。

1.2 产品功能

天翼云时序数据库 Influx 版具有实例管理、数据查询、数据管理、安全监控等功能，满足日常的数据库管理、监控和数据维护需求。

1.2.1 实例管理

- 实例列表

  提供订购、续费、退订、实例操作（启动/停止/重启/加载实例数据）入口；实例列表包含实例名称、状态、存储容量、引擎类型、计费方式、创建时间、到期时间等信息，点击实例名称可跳转到实例详情

- 文档下载

  提供 influxdb cli、SDK、telegraf 程序包等相关文档、程序下载

1.2.2 数据查询

- 查询面板

  可选择查询条件（下拉选择和代码编辑两种模式），选择 Bucket、指标、过滤、分组、聚合条件，生成图表，图表类型丰富（支持折线图、柱状图、指标卡、热点图等）

- 数据看板

  将查询面板生成的图表组合成数据看板或大屏，可选择黑夜/白天视觉样式，可对单个图表做编辑（导出图片、编辑、复制、删除）

1.2.3 数据管理

- Buckets 管理

  提供 Bucket 创建、编辑、删除操作，显示 Bucket 列表，包含信息 Buckets 名称、类型（系统内置/控制台创建/SDK 创建）、描述、数据保留策略、操作（添加数据、编辑、删除、查看令牌）

- 数据清理

  可按 Bucket、表、过滤条件、时间区间进行数据清理，清理任务通过列表显示，并可查看详情

- 数据复制

  可将数据周期复制到不同的 Bucket

1.2.4 安全监控

- 实例监控

  可查看实例运行情况、每秒写入、查询数等指标

- 安全设置

  对实例做白名单控制

- 令牌管理

  提供对 Bucket 的两种令牌权限，全部权限、读或写权限

1.3 产品优势

- 高效时序数据写入

全兼容开源 InfluxDB 写协议，支持 linflux CLI 命令行写入、influx API 写入、CSV 数据写入、client 包写入、telegraf 自动数据采集。

- 高效时序数据查询

完全兼容开源 InfluxDB 读协议，支持租户控制台条件查询、Flux 脚本查询、influxQL 查询。

- 丰富的数据分析及可视化

提供多维的实时分析能力，提供时序数据聚合计算，多种图表展示，并可进行 dashboard 面板大屏制作。

- 实时监控

提供实例监控功能，支持对实例写入指标，写入效率，存储空间的监控，实时发现实例瓶颈。

- 全方位数据管理

提供 token 管理，buckets 管理，Measurement 管理，数据清理、数据备份等运维能力，实现全面的时序数据库管理能力。

1.4 名词解释

- buckets

数据桶，数据库和保留策略的组合

- measurement

数据表，measurement 即为表的作用，同传统数据库中的 table 作用一致。

- points

数据点，表示每个表里某个时刻的某个条件下的一个 field 的数据，因为体现在图表上就是一个点，于是将其称为 point，即表里面的一行数据，influxDB 中由时间戳（time）、数据（field）、标签（tags）组成

- Tag

标签，tag 是一个非常重要的部分，表名+tag 一起作为数据库的索引，是“key-value”的形式。

- Timestamp

时间戳，作为时序型数据库，时间戳是 InfluxDB 中最重要的部分，在插入数据时可以自己指定也可留空让系统指定。说明：在插入新数据时，tag、field 和 timestamp 之间用空格分隔。

- Retention policy

数据保留策略，可以定义数据保留的时长，每个数据库可以有多个数据保留策略，但只能有一个默认策略

1.5 限制与约定

1.5.1 使用限制

限制项	限制值	说明
次购买数量	5	公测阶段单次购买数量最多为 5
	概述	概述
本文主要介绍 InfluxDB Java SDK 的安装和查询、写入以及 InfluxDB 管理等相关接口使用信息，帮助用户快速入门并且能够快速找到自己需要的功能方法。	本文主要介绍 InfluxDB Java SDK 的安装和查询、写入以及 InfluxDB 管理等相关接口使用信息，帮助用户快速入门并且能够快速找到自己需要的功能方法。	本文主要介绍 InfluxDB Java SDK 的安装和查询、写入以及 InfluxDB 管理等相关接口使用信息，帮助用户快速入门并且能够快速找到自己需要的功能方法。
安装 SDK 工具包	安装 SDK 工具包	安装 SDK 工具包
构建要求	构建要求	构建要求
Java SDK 工具包需在 Java 1.8+环境下运行。	Java SDK 工具包需在 Java 1.8+环境下运行。	Java SDK 工具包需在 Java 1.8+环境下运行。
获取 SDK	获取 SDK	获取 SDK

说明：

ST-L1：2 核 8G，每秒写入请求：50，每秒写入数据点：25000，每秒查询请求：25

ST-L2：4 核 16G：每秒写入请求：100，每秒写入数据点：50000，每秒查询请求：50

ST-L3：8 核 32G：每秒写入请求：160，每秒写入数据点：80000，每秒查询请求：80

ST-L4：16 核 64G：每秒写入请求：300，每秒写入数据点：150000，每秒查询请求：150

ST-L5：32 核 128G：每秒写入请求：500，每秒写入数据点：250000，每秒查询请求：250

ST-L6：64 核 256G：每秒写入请求：800，每秒写入数据点：400000，每秒查询请求：400

1.5.2 命名限制

实例名称：6-40 个字符，数据库名称长度为 6-40 个字符，只能包含小写字母、数字和下划线，必须以字母开头

Bucket 名称：长度为 4-128 字符，以字母或中文开头，可以包含中文、字母、数字、中划线或下划线，不能包含其他特殊字符

数据复制任务名称：长度为 4-128 字符，以字母或中文开头，可以包含中文、字母、数字、中划线或下划线，不能包含其他特殊字符

令牌名称：长度为 4-128 字符，以字母或中文开头，可以包含中文、字母、数字、中划线或下划线，不能包含其他特殊字符

2 操作指南

2.1 创建数据库

时序数据库 Influx 版区别于传统的关系型数据库，主要用于存储和管理时间序列数据。对于时间序列数据的写入、存储、查询、展现具有针对性的架构，从而获得极高的数据压缩能力、极优的查询性能，适用于物联网和互联网设计及主机监控告警等应用场景。

1. 登录天翼云官网。

2. 在顶部导航栏选择“**云计算>数据库>时序数据库 Influx 版**”，进入“产品介绍”页面。

3. 点击**立即开通**，根据页面引导购买实例。

4. 在购买页面，填写配置信息，主要包括：

1） 计费模式:包年/包月。

2） 版本：目前只提供 2.0.6

3） 实例类型：目前只提供单机标准版

4） 实例规格：不同规格，机器 CPU 内存配置不同，且对写入和查询的限制不同。共有 ST-L1、ST-L2、ST-L3、ST-L4、ST-L5、ST-L6 六种，最大限制对应关系如下

i. ST-L1：2 核 8G，每秒写入请求：50，每秒写入数据点：25000，每秒查询请求：25

ii. ST-L2：4 核 16G：每秒写入请求：100，每秒写入数据点：50000，每秒查询请求：50

iii. ST-L3：8 核 32G：每秒写入请求：160，每秒写入数据点：80000，每秒查询请求：80

iv. ST-L4：16 核 64G：每秒写入请求：300，每秒写入数据点：150000，每秒查询请求：150

v. ST-L5：32 核 128G：每秒写入请求：500，每秒写入数据点：250000，每秒查询请求：250

vi. ST-L6：64 核 256G：每秒写入请求：800，每秒写入数据点：400000，每秒查询请求：400

5） 存储类型：高 IO

6） 存储空间(GB)：50~5000GB，步长 50GB，最大值以 IaaS 层的承载能力为准

7） 实例名称：购买完成后进行创建的实例名称。实例名称长度为 6-40 个字符，只能包含小写字母、数字和下划线，必须以字母开头。

8） 加载示例数据：可选择创建实例后是否加载示例数据，也可后期在实例列表中加载，默认不选中

5. 结果验证

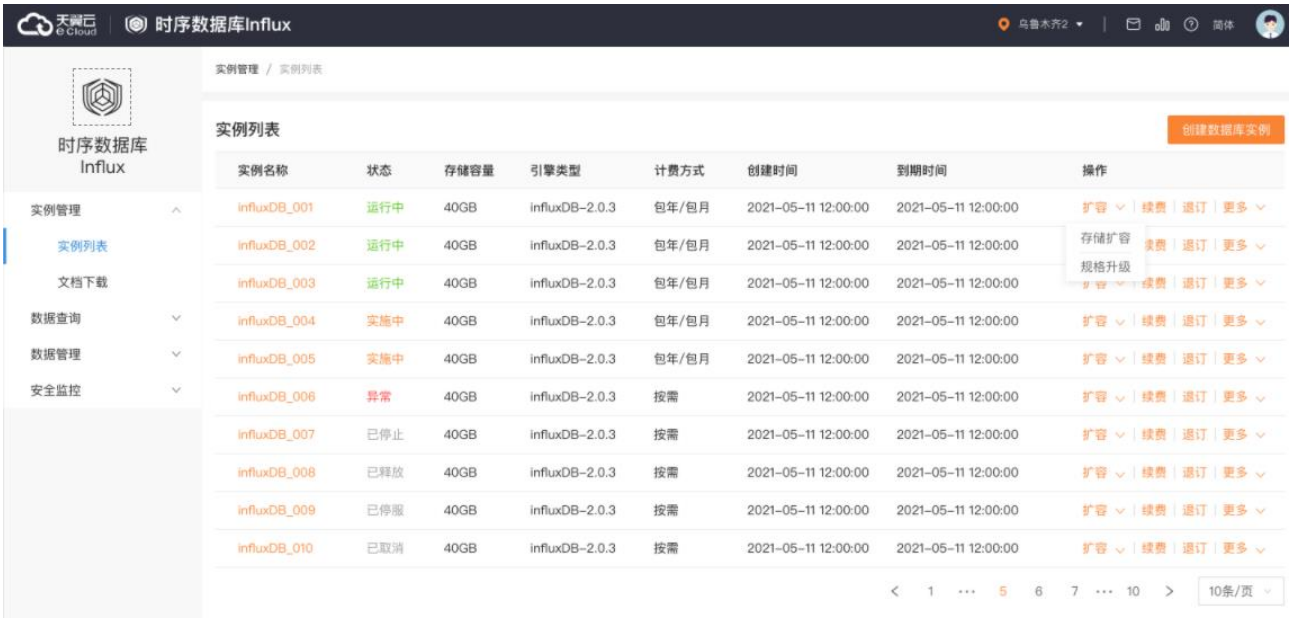
登录时序数据库 Influx 控制台，单击左侧菜单栏的**实例列表**查看已创建的实例。

2.2 实例管理

用户在实例信息模块查看实例列表基本信息和实例详细信息，可创建新数据库实例，也可对已购买的实例进行续费、退订、升级、加载示例数据、启动、重启、停止等操作。

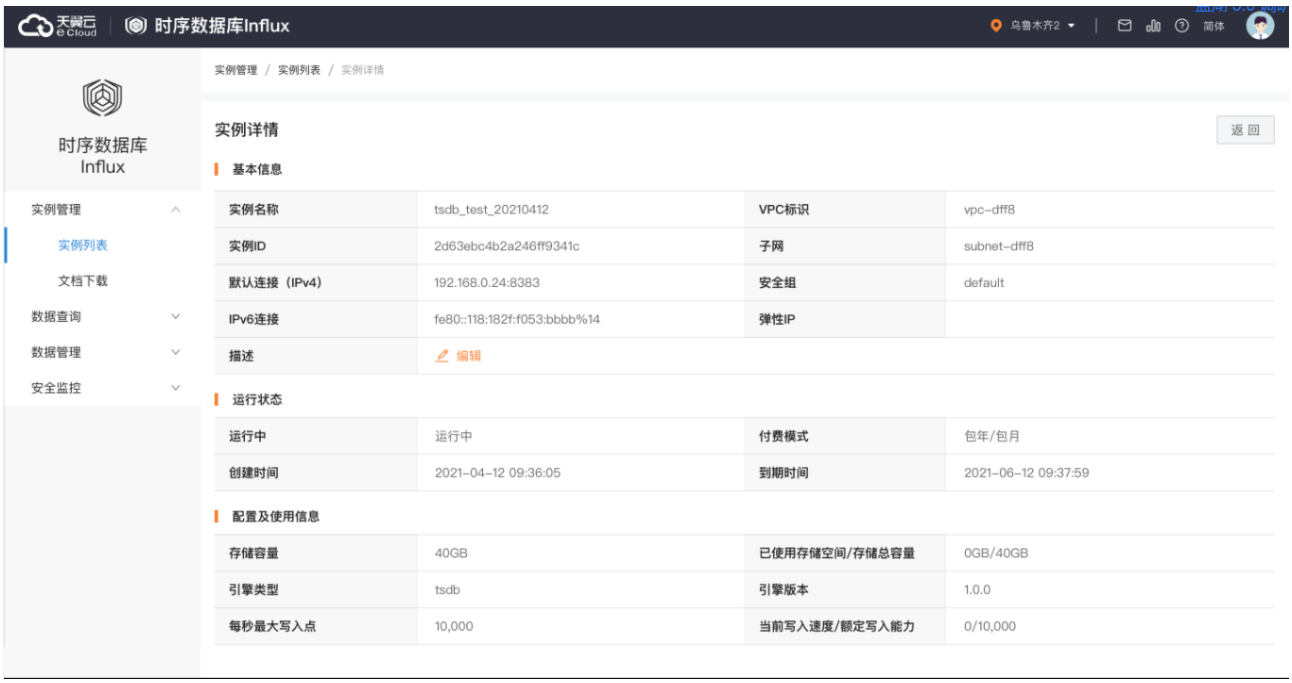
- 查看列表信息

点击 【实例管理-实例列表】，查看所有实例的基础信息，包括实例名称、状态、存储容量、引擎类型、计费方式、创建时间、到期时间



- 查看实例详情

点击【实例管理-实例列表】中的实例名称，可查看实例详情信息



- 操作数据库

点击【实例管理-实例列表】中的实例操作，可对实例执行具体操作，包括扩容（规格/存储）、续费、退订、加载示例数据、启动、重启、停止

2.3 文档下载

点击【实例信息-文档下载】，提供文档说明及操作手册

- SDK 操作手册：用户可以下载 SDK，并查看 SDK 使用说明
- Telegraf 使用指引：Telegraf 插件可用于数据自动采集，提供 Telegraf 安装、使用、配置说明
- 示例数据说明：与实例管理中“加载示例数据说明”对应，可查看示例数据来源、数据结构、元数据描述等信息

2.4 Buckets 管理

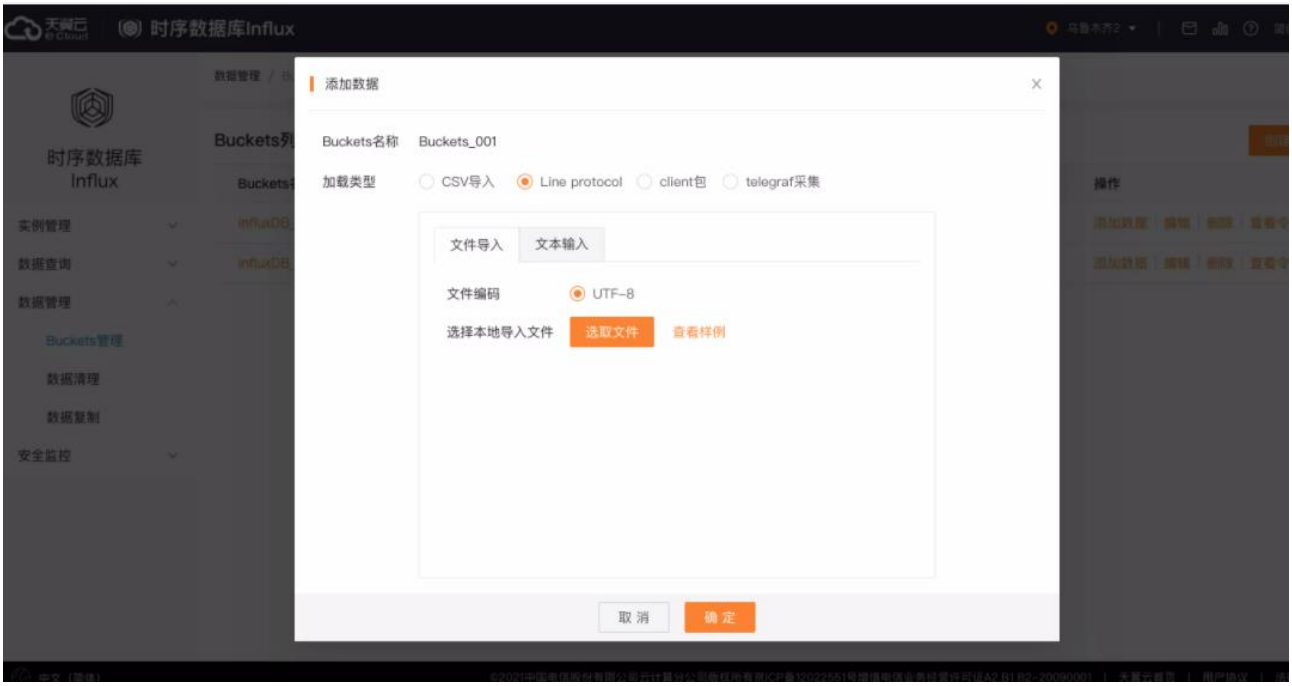
点击【数据管理-Buckets 管理】，提供 Bucket 创建、编辑、删除操作，显示 Bucket 列表，包含信息 Buckets 名称、类型（系统内置/控制台创建/SDK 创建）、描述、数据保留策略、操作（添加数据、

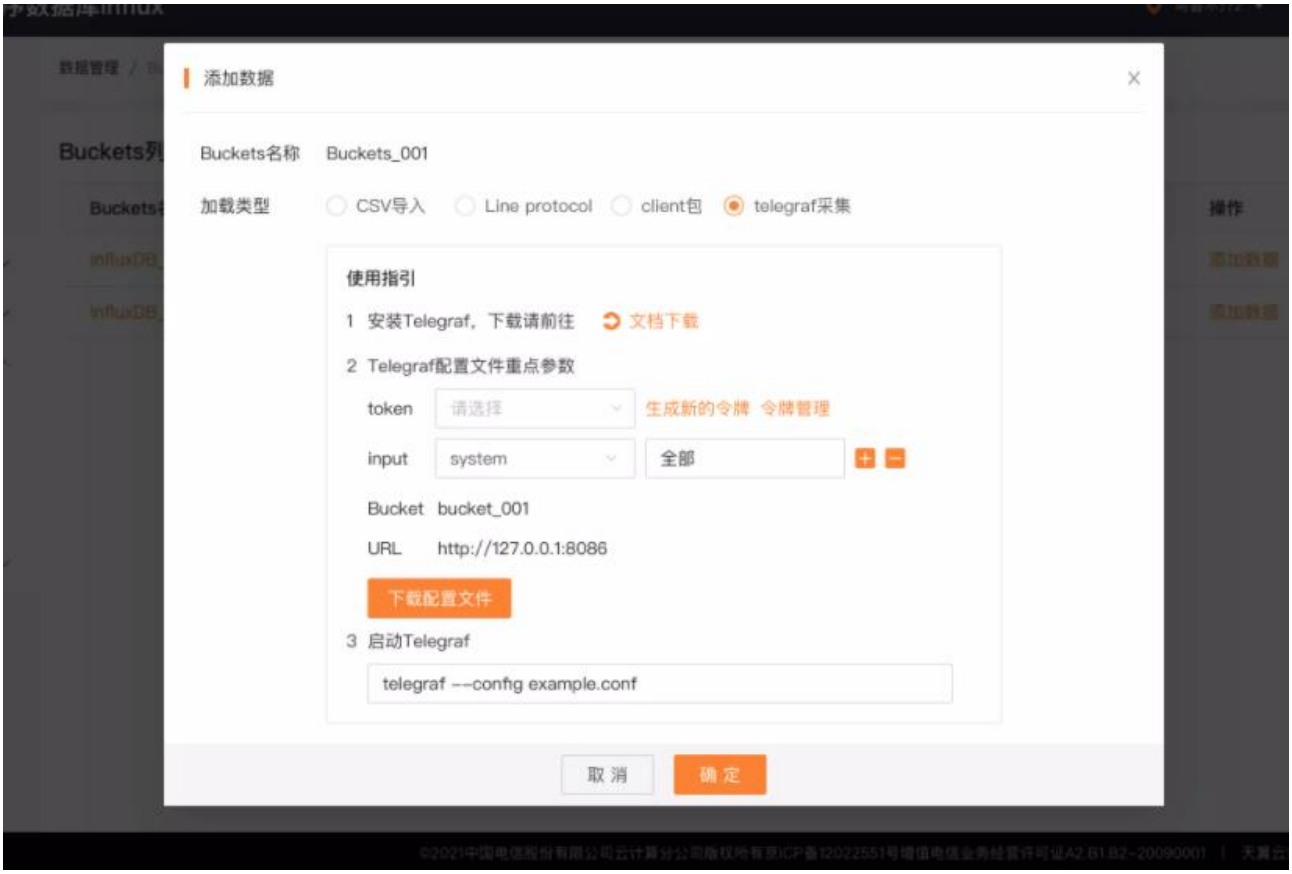
编辑、删除、查看令牌）

- 查看列表

- 添加数据

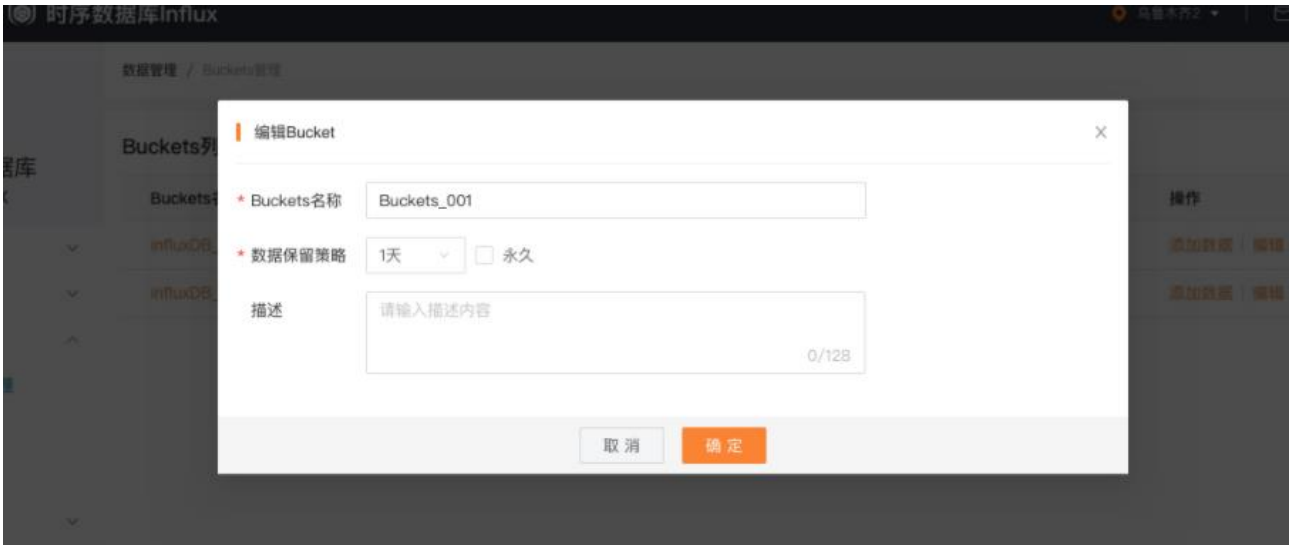
点击 Buckets 列表中操作按钮【添加数据】，模块有四种方式可导入，其中 csv 和 line protocol 可通过文件导入，client 包通过程序导入，Telegraf 采集可便捷生成配置文件





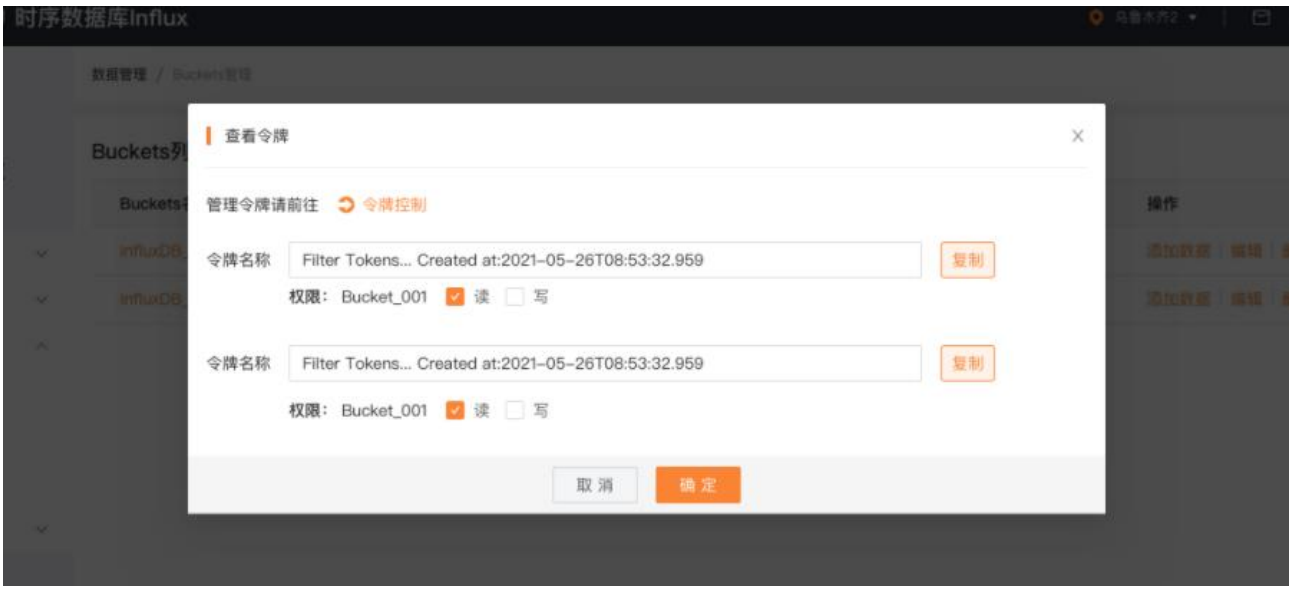
- 编辑 Bucket

点击 Buckets 列表中操作按钮【编辑】，可编辑数据保留策略，注意数据保留策略更改会影响数据有效期



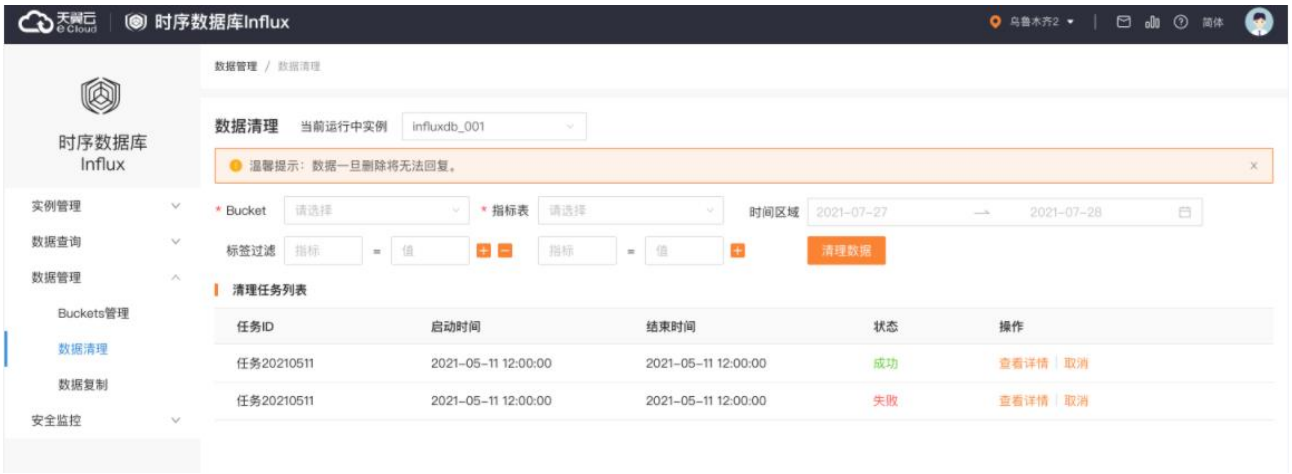
- 查看令牌

点击 Buckets 列表中操作按钮【查看令牌】，可查看跟对应 Bcuekt 相关的令牌及权限

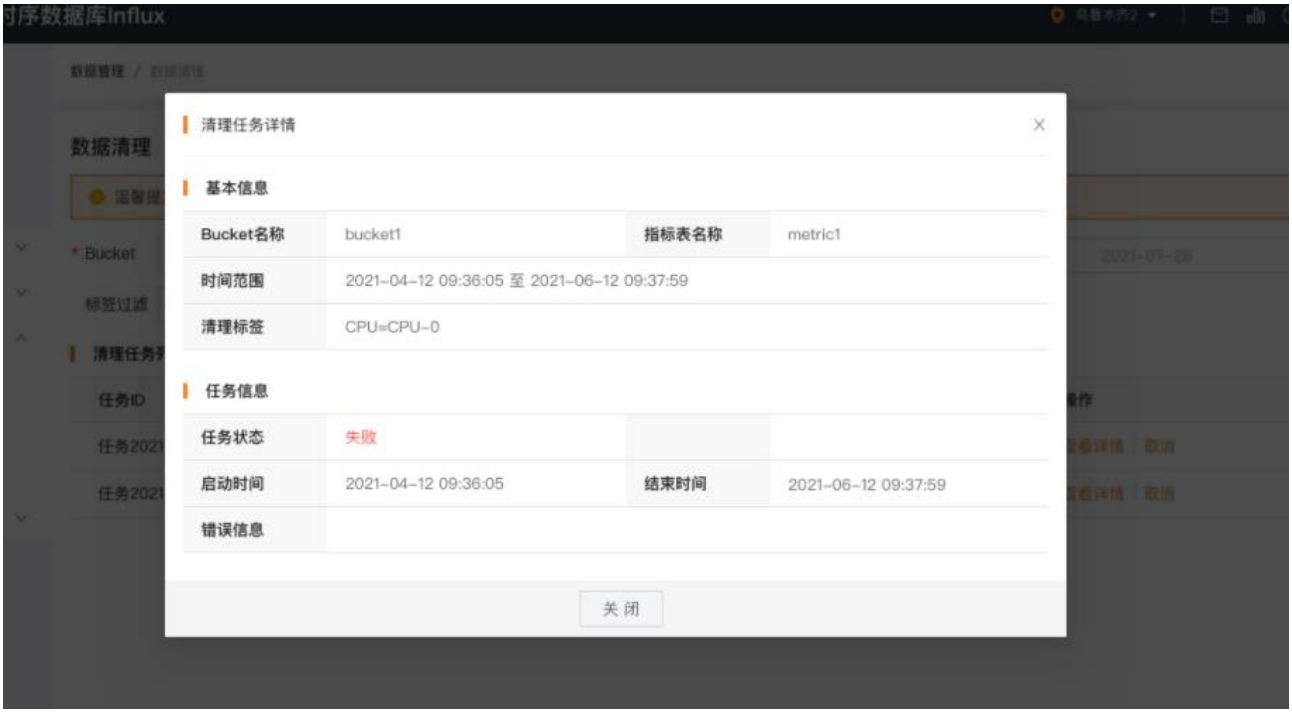


2.5 数据清理

点击【数据管理-数据清理】，可按 Bucket、表、过滤条件、时间区间进行数据清理，清理任务通过列表显示，并可查看详情



点击清理任务列表，【查看详情】



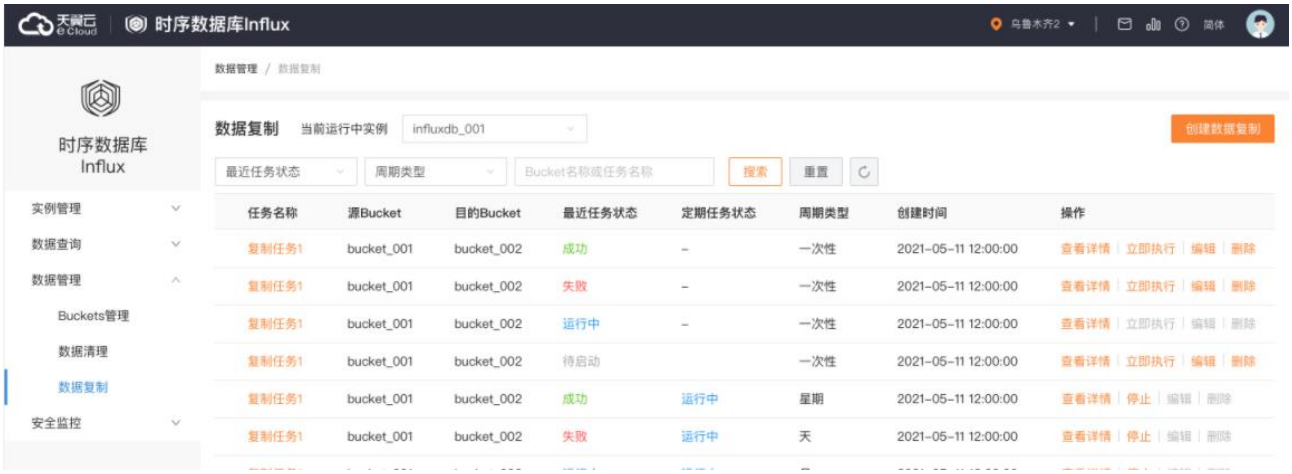
2.6 数据复制

点击【数据管理-数据复制】，可将数据一次性/周期性复制到不同的 Bucket

- 创建复制任务

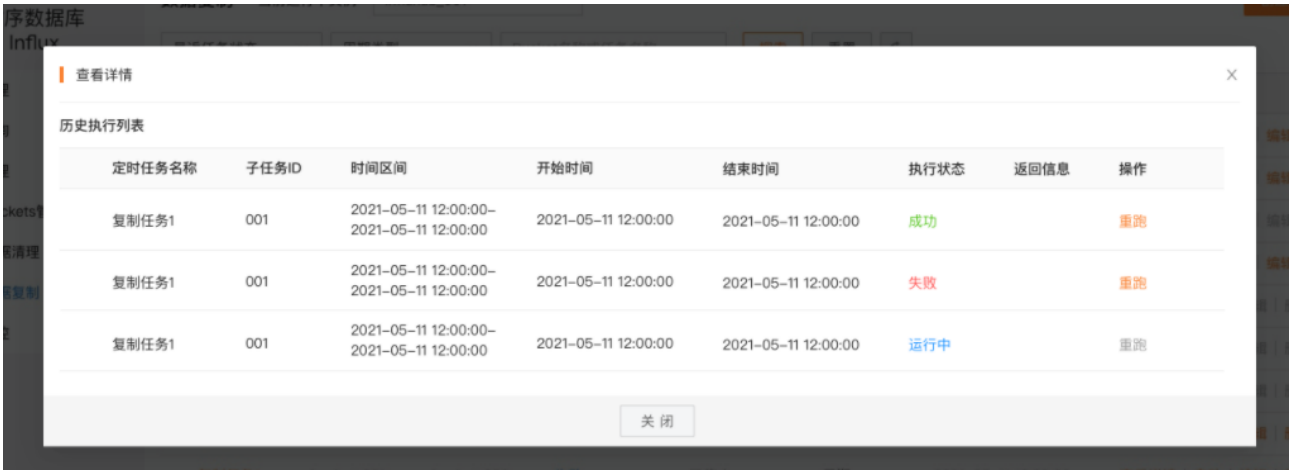
点击列表右上角按钮【创建复制任务】，弹窗复制任务信息填写

- 查看数据复制任务列表



- 查看详情

点击【查看详情】可查看历史复制任务执行情况

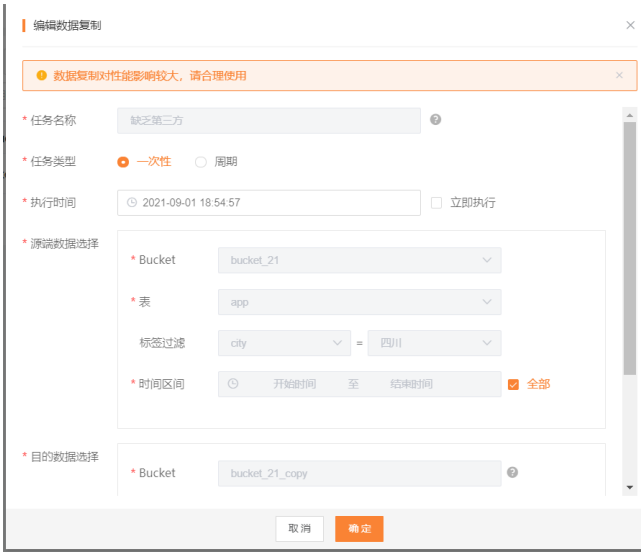


- 立即执行

点击【立即执行】，可手动执行任务一次

- 编辑任务

点击【编辑】，可编辑任务名称、任务类型、执行时间



- 删除任务

点击【删除】，可停止并删除复制任务

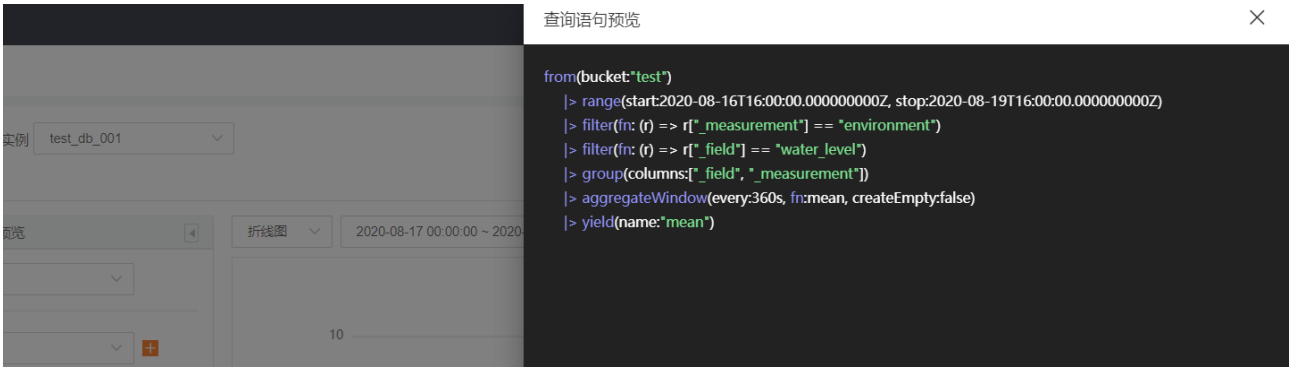
2.7 查询面板

可选择查询条件，如 Bucket、指标、过滤、分组、聚合条件，生成图表，可进行代码预览，下载 CSV，切换图表和表格模式等

点击【数据查询-查询面板】，进入查询条件选择

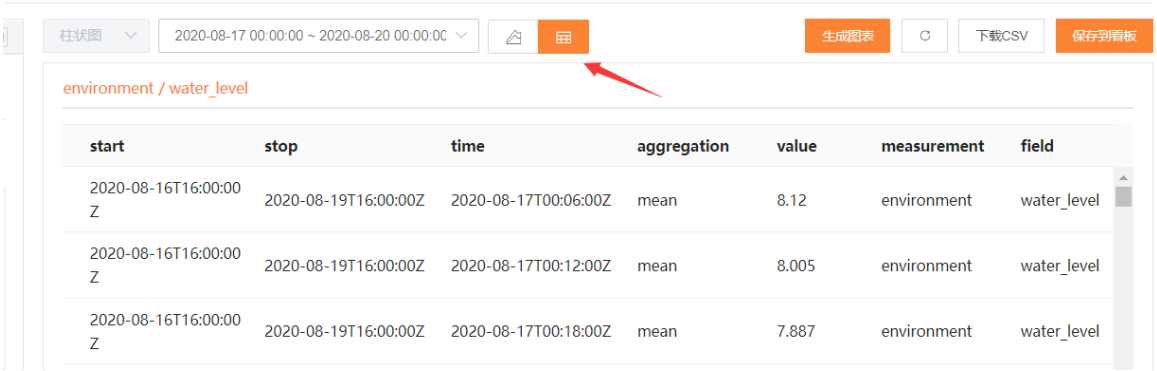
- 代码预览

选择条件后，点击【代码预览】，可查看对应的代码



- 切换表格模式

点击“生成图表”，默认是图表模式，点击【表格图标】，可切换到表格显示



- 保存到看板

生成图表后，点击【保存到看板】，可将图表保存到指定看板，在数据看板模块查询

- 下载 CSV

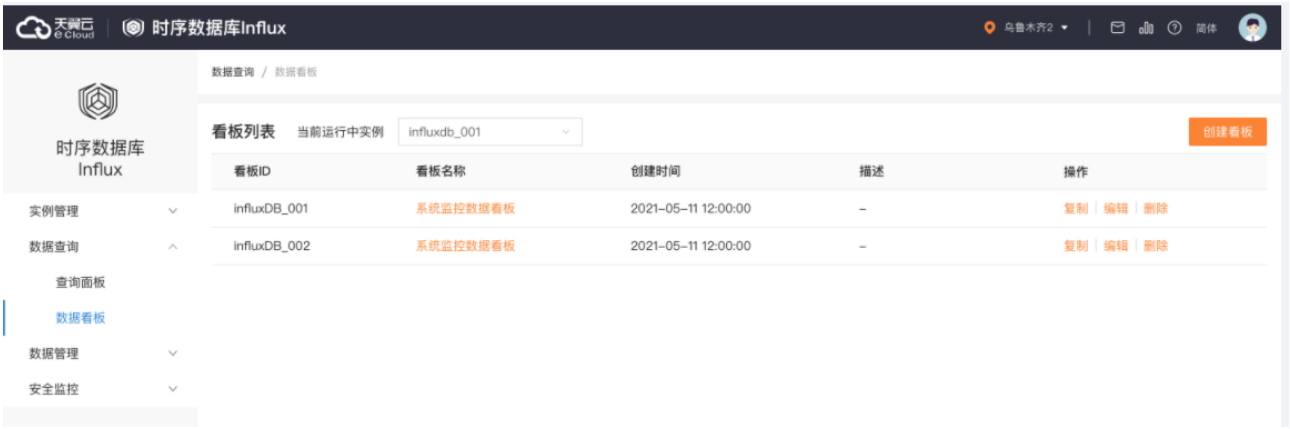
点击【下载 CSV】，可将查询的数据生成 CSV 文件下载

```
_measurement,_field,_value,_time
environment,water_level,8.12,2020-08-17T00:06:00Z
environment,water_level,8.005,2020-08-17T00:12:00Z
environment,water_level,7.887,2020-08-17T00:18:00Z
environment,water_level,7.762,2020-08-17T00:24:00Z
environment,water_level,7.635,2020-08-17T00:30:00Z
environment,water_level,7.5,2020-08-17T00:36:00Z
environment,water_level,7.372,2020-08-17T00:42:00Z
environment,water_level,7.234,2020-08-17T00:48:00Z
environment,water_level,7.11,2020-08-17T00:54:00Z
environment,water_level,6.982,2020-08-17T01:00:00Z
environment,water_level,6.837,2020-08-17T01:06:00Z
environment,water_level,6.713,2020-08-17T01:12:00Z
environment,water_level,6.578,2020-08-17T01:18:00Z
```

2.8 数据看板

将查询面板生成的图表组合成数据看板或大屏，可选择黑夜/白天视觉样式，可对单个图表做编辑（导出图片、编辑、复制、删除）

点击【数据查询-数据看板】，可进入看板列表

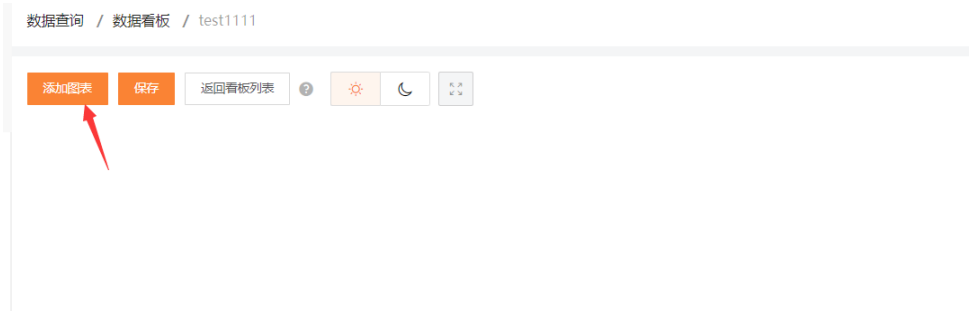


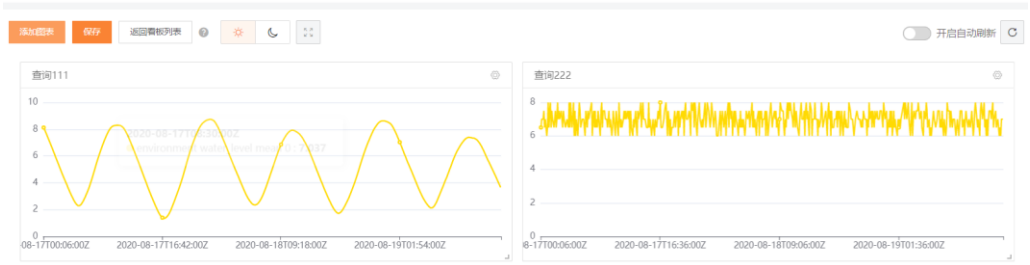
- 创建看板

点击列表右上角按钮【创建看板】

- 添加图表

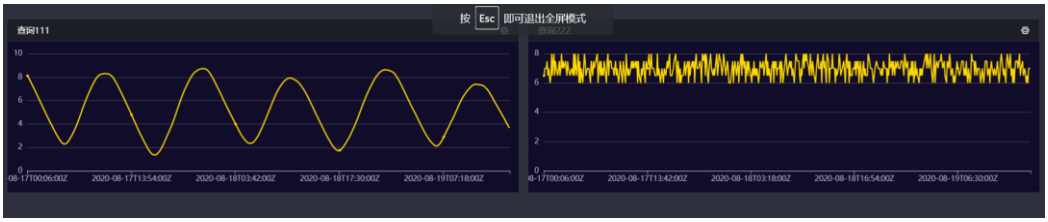
跳转到查询面板页创建数据查询，生成查询图表后，点击保存到看板



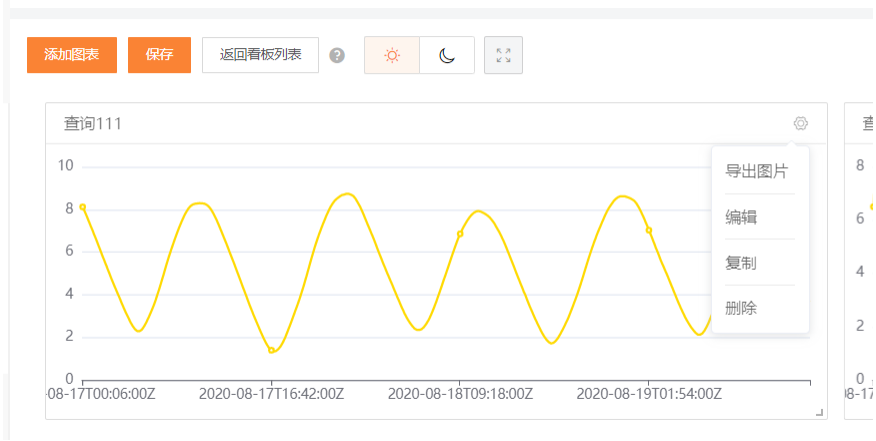


- 切换黑夜模式

- 最大化看板



- 导出图片



- 单个图表编辑

点击【编辑】，跳转到查询面板可进行编辑

- 单个图表复制

点击【复制】，可复制一个图表

- 单个图表删除

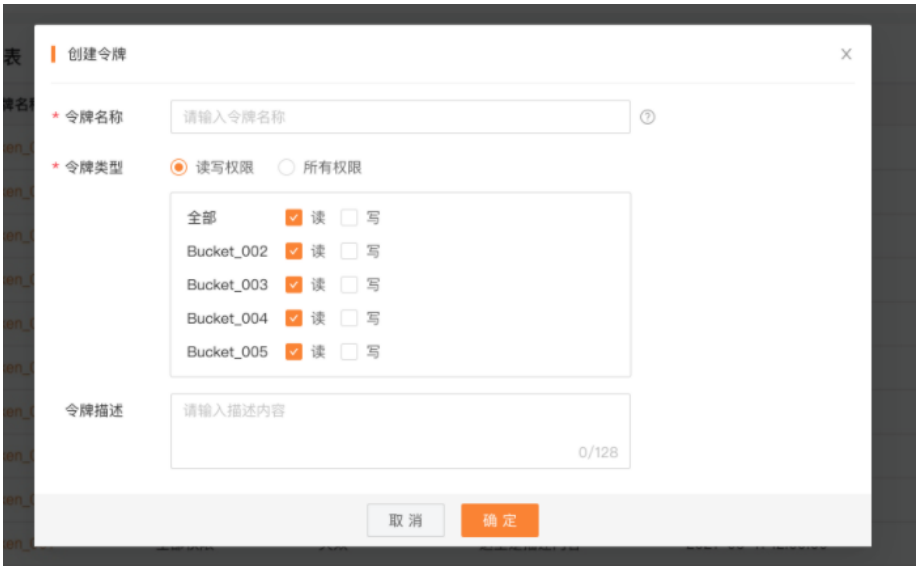
可删除图表

2.9 令牌控制

点击【安全监控-令牌控制】，提供 Token 管理，控制 token 对 bucket 的读写权限

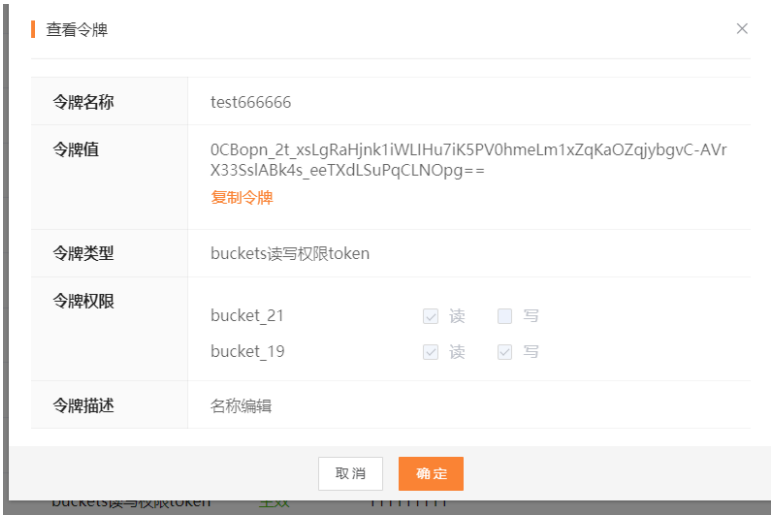
- 创建令牌

点击右上角按钮【创建令牌】，可填写令牌名称、令牌类型及相应的权限



- 查看令牌

点击操作-查看令牌，可查看令牌名称、值、类型、权限、描述等

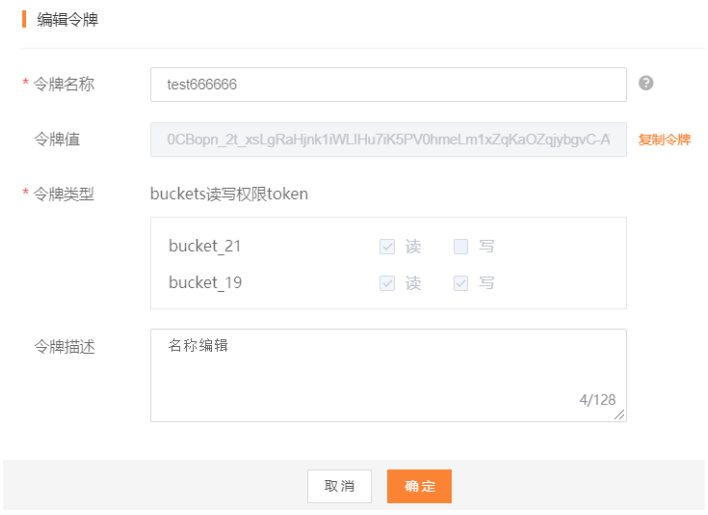


- 生效/失效

可对失效令牌进行生效控制，对生效令牌进行失效控制

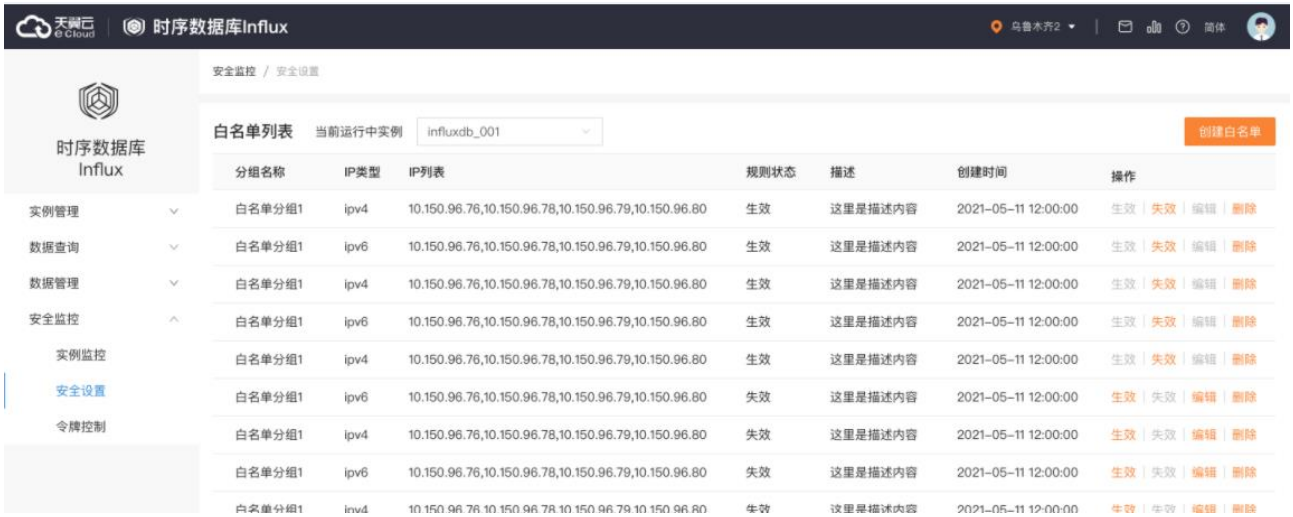
- 编辑

令牌编辑只可编辑名称和描述



2.10 安全设置

用户可以设置访问当前数据库的白名单。从左边菜单导航选择“安全设置”进入，可以看到当前实例的已设置的白名单。可新增白名单分组及白名单，也可修改已设置的白名单。只有白名单列表中的 IP 才能访问实例。



点击右上角按钮【新增白名单】，可添加白名单信息，用于访问实例

2.11 实例监控

可查看实例运行情况、每秒写入、查询数等指标


时序数据库Influx
乌鲁木齐2
简体
安全监控 / 实例监控
时序数据库
Influx
实例管理
数据查询
数据管理
安全监控
实例监控
安全设置
令牌控制
实例监控
开启自动刷新
运行状态
运行中
运行时长
365 天
存储使用率 共500GB
未使用
100GB
20%
80%
已使用
400GB
到期时间
2022-11-01 12:00:00
最近操作时间及类型
2022-11-01 12:00:00
加载示例数据
读写监控
最近
1小时
请选择实例
每秒写入点数
额度10000
原库读取流量
1400
1200
800
400
0
2021-03-24 12:00:45
每秒写入点数 1020
12:00:00
12:00:15
12:00:30
12:00:45
12:01:00
12:01:15
12:01:30
12:01:45
03-24 12:00:00
03-24 12:00:30
每秒读写请求
额度：读 100，写 200
每秒写入请求
每秒查询请求
2000
1500
1000
500
0
2021-03-24 12:01:15
每秒写入请求 1250
每秒查询请求 980
12:00:00
12:00:15
12:00:30
12:00:45
12:01:00
12:01:15
12:01:30
12:01:45
03-24 12:00:00
03-24 12:00:30

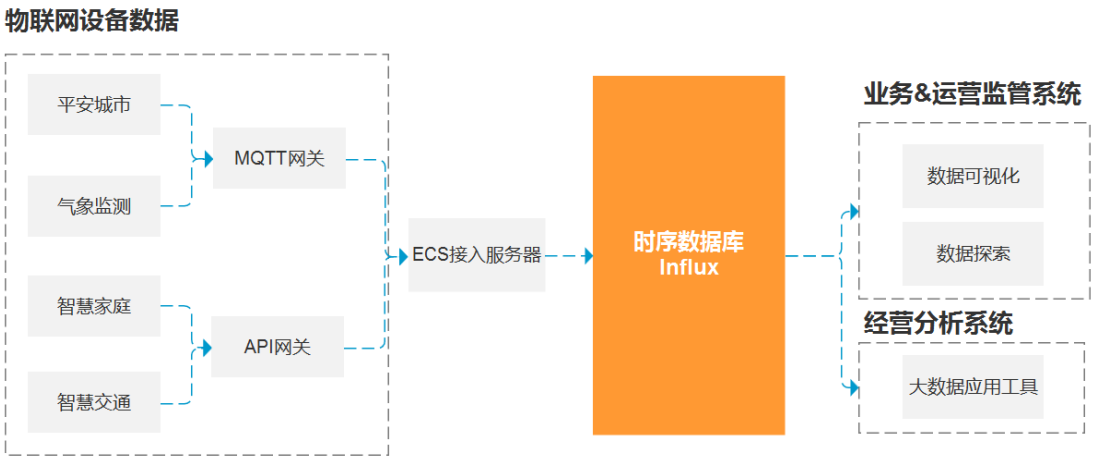
3 应用场景

3.1 互联网业务性能监控服务

互联网企业需要对主机、网络及其它硬件设备的监控，对交易情况、处理性能、异常等进行实时监控告警。天翼云时序数据库 Influx 提供高性价比的存储，实时的时序数据聚合计算，是运维监控数据的理想选择。

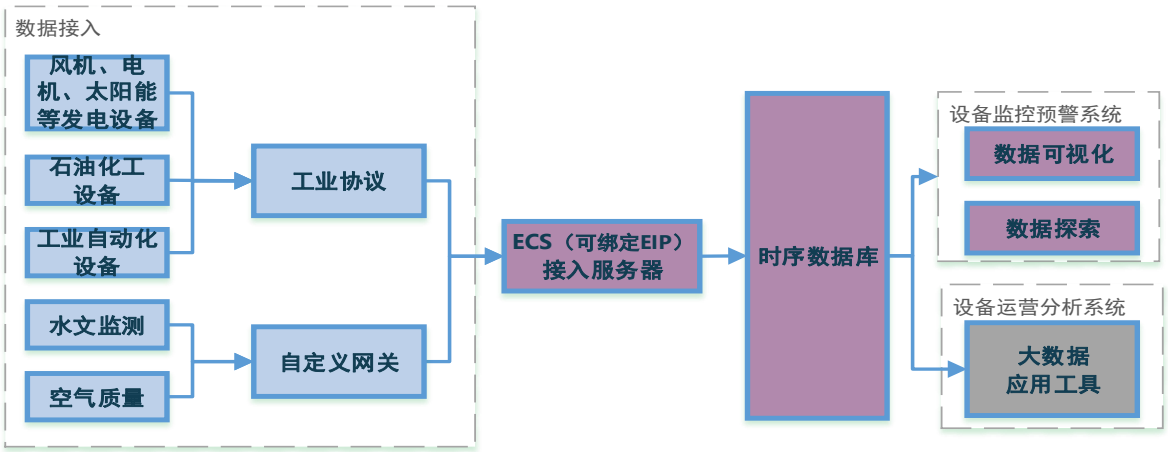
3.2 物联网设备状态监控存储分析

随着物联网以及工业 4.0 的到来，所有设备都会携带传感器并联网，传感器收集的时序数据将严重依赖时序数据库的实时分析能力、存储能力以及查询统计能力。可通过时序数据库的数据查询读取实时数据，对数据进行各种聚合运算并得到数据统计报表，通过天翼云时序数据库 Influx 的控制台直观得到数据的变化趋势，帮助用户分析数据。



3.3 工业及能源化工设备监测管理

各类工业设备不断产生大量的时间序列数据，需要实时高效写入及聚合运算，从而及时准确的监测到异常状况， 时序数据库正是为此而生 ，提供高效存取、高速访问、低成本存储、实时告警信息，实现设备实时监控分析系统。



4 【需更新】Java SDK 文档

4.1 概述

本文主要介绍 InfluxDB Java SDK 的安装和查询、写入以及 InfluxDB 管理等相关接口使用信息，帮助用户快速入门并且能够快速找到自己需要的功能方法。

4.2 安装 SDK 工具包

4.2.1 构建要求

Java SDK 工具包需在 Java 1.8+环境下运行。

4.2.2 获取 SDK

方式一：Maven 依赖获取（**暂不支持**）。

如果您使用 Maven 做项目构建工具，那么请在 pom.xml 文件的<dependencies>标签中添加 influxdb-client-java 依赖。示例代码如下：

```
<dependency>
    <groupId>cn.chinatelecom</groupId>
    <artifactId>influxdb-client-java</artifactId>
    <version>{version}</version>
</dependency>
```

方式二：下载 SDK 包

如果您不使用 Maven 或者不能访问 Maven 仓库，可以直接下载 SDK 添加依赖包使用，步骤如下：

1、下载 InfluxDB-Client 的 JAR 包，地址：http://xxx；

2、将上述下载的工具包作为本地 JAR 包依赖引入到项目工程中。

若不清楚 Eclipse 或 IDEA 等不同 IDE 引入本地 JAR 包依赖的方式可自行上网查询，此处不再赘述。

4.3 SDK 客户端配置

InfluxDBClient 是操作 InfluxDB 的客户端操作类，使用 InfluxDB 的 SDK 前首先需要创建 InfluxDBClient 的实例对象，可以通过 InfluxDBClientFactory 工厂类里面提供的 8 种方法来创建 InfluxDBClient 实例对象。其中，8 种创建 InfluxDBClient 实例对象的方法可以概括为三类：第一类是方法中通过传递不同的配置参数来创建 InfluxDBClient 实例对象；第二类是通过 InfluxDBClientOptions 配置类来创建 InfluxDBClient 实例对象；第三类是方法中不传递任何参数而是通过读取根目录下的 influx2.properties 配置文件中的配置项来创建 InfluxDBClient 实例对象。

注意：InfluxDBClient 实例对象使用完成之后记得调用 influxDBClient.close()方法进行及时关闭。

- 第一类示例代码

方法 1：create(String url, char[] token, String org, String bucket)

```
InfluxDBClient client =
InfluxDBClientFactory.create("http://localhost:8086", "my-
token".toCharArray(), "myOrg", "myBucket");
```

方法 2：create(String url, char[] token, String org)

```
InfluxDBClient client =
InfluxDBClientFactory.create("http://localhost:8086", "my-
token".toCharArray(), "myOrg");
```

方法 3：create(String url, char[] token)

```
InfluxDBClient client =
InfluxDBClientFactory.create("http://localhost:8086", "my-
token".toCharArray());
```

方法 4：create(String url, String username, char[] password)

```
InfluxDBClient client =
InfluxDBClientFactory.create("http://localhost:8086", "my-
username", "my-password".toCharArray());
```

方法 5：create(String connectionString)

```
//可以使用连接字符串构建客户端，该连接字符串包含 URL 地址及相关参数。
InfluxDBClient client =
InfluxDBClientFactory.create("http://localhost:8086?readTimeout=1000
&writeTimeout=3000&connectTimeout=2000&logLevel=HEADERS&token=my-
token&bucket=my-bucket&org=my-org");
```

其中，URL 中参数支持以下选项：

属性名	默认值	描述说明
org	-	写入和查询的默认目标组织
bucket	-	写入的默认目标存储桶
token	-	用于授权的令牌
logLevel	NONE	日志级别
readTimeout	10000ms	读取超时时间
writeTimeout	10000ms	写超时时间
connectTimeout	10000ms	套接字超时时间

- 第二类示例代码

首先创建 InfluxDBClientOptions 对象，然后放入 InfluxDBClientFactory 类的 create 方法中即可创建 InfluxDBClient 实例对象。

```
//创建配置对象
InfluxDBClientOptions options = InfluxDBClientOptions.builder()
        .url(url)
        .org("-")
        .authenticateToken("my-token".toCharArray())
        .bucket("my-bucket")
        .build();
//创建 InfluxDBClient 的实例对象
InfluxDBClient client = InfluxDBClientFactory.create(options);
```

- 第三类示例代码

不传递任何参数而是通过读取根目录下的 influx2.properties 配置文件中的配置项来创建 InfluxDBClient 实例对象。

```
InfluxDBClient client = InfluxDBClientFactory.create();
```

4.4 写入数据

InfluxDB 的数据写入支持异步非阻塞写入和同步阻塞写入两种类型，前者是通过创建 WriteApi 对象来写入数据，而后者是通过创建 WriteApiBlocking 对象来写入数据。

4.4.1 异步非阻塞

异步非阻塞写入数据方式需要创建 WriteApi 对象，支持使用行协议 Line Protocol、数据点 Data Point 和对象 POJO 写入数据，支持批量写入数据。

4.4.1.1 对象写入

可以通过 POJO 对象的形式异步非阻塞写入数据到特定的 Bucket。

示例代码：

```
//定义 POJO 对象类
@Measurement(name = "temperature")
public static class Temperature {
    @Column(tag = true)
    String location;
```

```
        @Column
        Double value;

        @Column(timestamp = true)
        Instant time;
    }

    //配置客户端 influxDBClient
    //写入数据
    try (WriteApi writeApi = influxDBClient.getWriteApi()) {
          // 构建 POJO 对象
          Temperature temperature = new Temperature();
          temperature.location = "south";
          temperature.value = 62D;
          temperature.time = Instant.now();
          // 写入对象数据
          writeApi.writeMeasurement(WritePrecision.NS, temperature);
    }
    //关闭客户端
    influxDBClient.close();
```

4.4.1.2 数据点写入

可以通过 Data Point 数据点的形式异步非阻塞写入数据到特定的 Bucket。

示例代码：

```
    //配置客户端 influxDBClient
    //写入数据
    try (WriteApi writeApi = influxDBClient.getWriteApi()) {
```

```
        // 构建数据点 Point
        Point point = Point.measurement("temperature")
                .addTag("location", "west")
                .addField("value", 55D)
                .time(Instant.now().toEpochMilli(),
    WritePrecision.MS);
        // 写入 Point 数据
        writeApi.writePoint(point);
    }
    //关闭客户端
    influxDBClient.close();
```

4.4.1.3 行协议写入

可以通过 Line Protocol 行协议的形式异步非阻塞写入数据到特定的 Bucket。

示例代码：

```
    //配置客户端 influxDBClient
    //写入数据
    try (WriteApi writeApi = influxDBClient.getWriteApi()) {
         // 定义 record
        String record = "temperature,location=north value=60.0";
        // 写入行数据
        writeApi.writeRecord(WritePrecision.NS, record);
    }
    //关闭客户端
    influxDBClient.close();
```

4.4.1.4 批量写入

通过构建 WriteOptions 配置对象并将其作为参数传入客户端创建 WriteApi 对象的

方法中，从而实现对数据批量写入的支持。

示例代码：

```
//配置客户端 influxDBClient

//构建 WriteOptions 配置对象，参数根据实际需要灵活调整。

WriteOptions writeOptions = WriteOptions.builder()
        .batchSize(10_000)
        .bufferLimit(500)
        .flushInterval(500)
        .jitterInterval(1_000)
        .retryInterval(2_000)
        .maxRetries(5)
        .maxRetryDelay(250_123)
        .exponentialBase(2)
        .writeScheduler(Schedulers.computation())
        .backpressureStrategy(BackpressureOverflowStrategy.ERROR)
        .build();

//创建 WriteApi 对象

WriteApi api = influxDBClient.getWriteApi(writeOptions );

//写入数据

……

//关闭客户端

influxDBClient.close();
```

4.4.2 同步阻塞

同步阻塞写入数据方式需要创建 WriteApiBlocking 对象，支持使用行协议 Line Protocol、数据点 Data Point 和对象 POJO 写入数据。

4.4.2.1 对象写入

可以通过 POJO 对象的形式同步阻塞写入数据到特定的 Bucket。

示例代码：

```
//定义 POJO 对象类
```

```
@Measurement(name = "temperature")
public static class Temperature {
    @Column(tag = true)
    String location;

    @Column
    Double value;

    @Column(timestamp = true)
    Instant time;
}

//配置客户端 influxDBClient
//创建 WriteApiBlocking 对象
WriteApiBlocking writeApi = influxDBClient.getWriteApiBlocking();
//写入数据
try {
     // 构建 POJO 对象
     Temperature temperature = new Temperature();
     temperature.location = "south";
     temperature.value = 62D;
     temperature.time = Instant.now();
     // 写入对象数据
     writeApi.writeMeasurement(WritePrecision.NS, temperature);
} catch (InfluxException ie) {
     System.out.println("InfluxException: " + ie);
}
```

```
//关闭客户端
influxDBClient.close();
```

4.4.2.2 数据点写入

可以通过 Data Point 数据点的形式同步阻塞写入数据到特定的 Bucket。

示例代码：

```
//配置客户端 influxDBClient
//创建 WriteApiBlocking 对象
WriteApiBlocking writeApi = influxDBClient.getWriteApiBlocking();
//写入数据
try {
       // 构建数据点 Point
       Point point = Point.measurement("temperature")
              .addTag("location", "west")
              .addField("value", 55D)
              .time(Instant.now().toEpochMilli(),
WritePrecision.MS);
       // 写入 Point 数据
       writeApi.writePoint(point);
}
//关闭客户端
influxDBClient.close();
```

4.4.2.3 行协议写入

可以通过 Line Protocol 行协议的形式同步阻塞写入数据到特定的 Bucket。

示例代码：

```
//配置客户端 influxDBClient
//创建 WriteApiBlocking 对象
WriteApiBlocking writeApi = influxDBClient.getWriteApiBlocking();
//写入数据
try {
       // 定义 record
      String record = "temperature,location=north value=60.0";
      // 写入行数据
      writeApi.writeRecord(WritePrecision.NS, record);
}
//关闭客户端
influxDBClient.close();
```

4.4.3 默认标签

有时候在每一个测量值中都需保存部分相同的信息，如 hostname、location 等，这时可以通过静态值、系统变量或环境变量的属性配置来设置默认的标签值，其配置的表达式形式如下：静态值：China；系统变量：${version}；环境变量：${env.hostname}。

针对示例行协议内容：mine-sensor,id=132-987-655,customer="China",hostname=example.com,sensor-version=v1.00 altitude=10，其有两种实现方式，分别如下。

通过配置文件实现的示例代码：

```
influx2.tags.id = 132-987-655
influx2.tags.customer = China
influx2.tags.hostname = ${env.hostname}
influx2.tags.sensor-version = ${version}
```

通过 API 实现的示例代码：

```
InfluxDBClientOptions options = InfluxDBClientOptions.builder()
    .url(url)
    .authenticateToken(token)
    .addDefaultTag("id", "132-987-655")
    .addDefaultTag("customer", "China")
    .addDefaultTag("hostnamer", "${env.hostname}")
    .addDefaultTag("sensor-version", "${version}")
    .build();
```

4.4.4 GZIP 支持

InfluxDBClient 默认不会为其底层调用的 HTTP 请求启用 GZIP 压缩，如果要启用 GZIP 以减少传输数据的大小，则可以通过如下示例代码进行配置。

示例代码：

```
influxDBClient.enableGzip();
```

4.5 查询数据

InfluxDB 的数据查询支持同步、异步以及原生查询三种方式。对于 POJO 映射，如果未找到精确匹配项，snake_case 列名称将映射到 camelCase 字段名称。

4.5.1 同步查询

同步查询不适用于大型查询结果。

示例代码：

```
//配置客户端 influxDBClient
//定义查询 Flux，内容可以根据查询需要自行定义。
String flux = "from(bucket:\"my-bucket\") |> range(start: 0)";
//创建 QueryApi 对象
QueryApi queryApi = influxDBClient.getQueryApi();
//查询数据
List<FluxTable> tables = queryApi.query(flux);
```

```
for (FluxTable fluxTable : tables) {
     List<FluxRecord> records = fluxTable.getRecords();
     for (FluxRecord fluxRecord : records) {
        System.out.println(fluxRecord.getTime() + ": " +
fluxRecord.getValueByKey("_value"));
     }
}
//关闭客户端
influxDBClient.close();
```

同步查询提供了 FluxRecords 到 POJO 的可能性映射。

```
//定义 POJO 对象类
@Measurement(name = "temperature")
public static class Temperature {
     @Column(tag = true)
     String location;

     @Column
     Double value;

     @Column(timestamp = true)
     Instant time;
}
//配置客户端 influxDBClient
//定义查询 Flux，内容可以根据查询需要自行定义。
String flux = "from(bucket:\"my-bucket\") |> range(start: 0) |>
filter(fn: (r) => r._measurement == \"temperature\")";
//创建 QueryApi 对象
QueryApi queryApi = influxDBClient.getQueryApi();
```

```
//查询数据并映射成 POJO 对象
List<Temperature> temperatures = queryApi.query(flux,
Temperature.class);
for (Temperature temperature : temperatures) {
      System.out.println(temperature.location + ": " +
temperature.value + " at " + temperature.time);
}
//关闭客户端
influxDBClient.close();
```

4.5.2 异步查询

异步查询提供了处理未绑定查询的可能性，并允许用户处理异常、停止接收更多结果以及成功查询的通知。

示例代码：

```
//配置客户端 influxDBClient
//定义查询 Flux，内容可以根据查询需要自行定义。
String flux = "from(bucket:\"my-bucket\") |> range(start: 0)";
//创建 QueryApi 对象
QueryApi queryApi = influxDBClient.getQueryApi();
//查询数据:query(String query, BiConsumer<Cancellable, FluxRecord>
onNext, Consumer<? super Throwable> onError, Runnable
onComplete);
//异步查询提供多种 query 方法，如上方法的后两个参数可以根据实际需要灵活
选用
queryApi.query(flux, (cancellable, fluxRecord) -> {
            // 回调消费 FluxRecord 结果，具有中断流查询的能力
            System.out.println(fluxRecord.getTime() + ": " +
fluxRecord.getValueByKey("_value"));
        }, throwable -> {
```

```
            // 回调消费所有的错误通知
            System.out.println("Error occurred: " +
throwable.getMessage());
        }, () -> {
            // 回调消费查询成功的通知
            System.out.println("Query completed");
        }
);
//关闭客户端
influxDBClient.close();
```

异步查询提供了 FluxRecords 到 POJO 的可能性映射。

```
//定义 POJO 对象类
@Measurement(name = "temperature")
public static class Temperature {
    @Column(tag = true)
    String location;

    @Column
    Double value;

    @Column(timestamp = true)
    Instant time;
}
//配置客户端 influxDBClient
//定义查询 Flux，内容可以根据查询需要自行定义。
String flux = "from(bucket:\"my-bucket\") |> range(start: 0) |>
filter(fn: (r) => r._measurement == \"temperature\")";
//创建 QueryApi 对象
```

```
QueryApi queryApi = influxDBClient.getQueryApi();
//查询数据并映射成 POJO 对象
queryApi.query(flux, Temperature.class, (cancellable,
temperature) -> {
    // 回调消费 FluxRecord 结果并映射成 POJO 对象，具有中断异步查询的
能力
    System.out.println(temperature.location + ": " +
temperature.value + " at " + temperature.time);
    }
);
//关闭客户端
influxDBClient.close();
```

4.5.3 原生查询

原生查询允许直接处理成原始 CSV 响应。

```
//配置客户端 influxDBClient
//定义查询 Flux，内容可以根据查询需要自行定义。
String flux = "from(bucket:\"my-bucket\") |> range(start: 0)";
//创建 QueryApi 对象
QueryApi queryApi = influxDBClient.getQueryApi();
//查询数据
String csv = queryApi.queryRaw(flux);
System.out.println("CSV response: " + csv);
//关闭客户端
influxDBClient.close();
```

异步版本允许逐行处理。

```
//配置客户端 influxDBClient
//定义查询 Flux，内容可以根据查询需要自行定义。
```

```
String flux = "from(bucket:\"my-bucket\") |> range(start: 0)";
//创建 QueryApi 对象
QueryApi queryApi = influxDBClient.getQueryApi();
//查询数据
queryApi.queryRaw(flux, (cancellable, line) -> {
            System.out.println("Response: " + line);
});
//关闭客户端
influxDBClient.close();
```

4.5.4 Flux-DSL 构建查询

对于同步查询、异步查询或原生查询中的 Flux 查询语句，除了直接手动拼接成 String 外，还可以通过 Flux-DSL 来构建 Flux 查询。

示例代码：

```
//配置客户端 influxDBClient
//构建 Flux 对象，内容可以根据查询需要自行定义。
Flux flux = Flux.from("my-bucket")
                .range(-30L, ChronoUnit.MINUTES)
                .filter(Restrictions.and(Restrictions.measurement().e
qual("cpu")))
                .limit(10);
//创建 QueryApi 对象
QueryApi queryApi = influxDBClient.getQueryApi();
//查询数据
List<FluxTable> tables = queryApi.query(flux.toString());
for (FluxTable fluxTable : tables) {
    List<FluxRecord> records = fluxTable.getRecords();
    for (FluxRecord fluxRecord : records) {
```

```
            System.out.println(fluxRecord.getTime() + ": " +
fluxRecord.getValueByKey("_value"));
        }
    }
    //关闭客户端
    influxDBClient.close();
```

4.6 删除数据

删除数据需要创建 DeleteApi 对象，下面展示从 InfluxDB 删除数据的示例代码。

```
    //配置客户端 influxDBClient
    //创建 DeleteApi 对象。
    DeleteApi deleteApi = influxDBClient.getDeleteApi();
    //删除数据。其中，DeleteApi 对象拥有包含不同参数的多个 delete 方法，起
    止时间以及 bucket 等信息参数可以根据实际需要灵活调整。
    try {
        OffsetDateTime start = OffsetDateTime.now().minus(1,
    ChronoUnit.HOURS);
        OffsetDateTime stop = OffsetDateTime.now();
        deleteApi.delete(start, stop, "", "my-bucket", "my-org");
    } catch (InfluxException ie) {
        System.out.println("InfluxException: " + ie);
    }
    //关闭客户端
    influxDBClient.close();
```

4.7 日志等级配置

可以通过更改日志等级来记录请求和响应内容，其中，LogLevel 值包括：NONE、BASIC、HEADER、BODY。

注意，当 Logleve 配置为 BODY 等级时，将在流式传输时禁用块并将整个响应加载到内存中。

示例代码：

```
influxDBClient.setLogLevel(LogLevel.HEADERS)
```

4.8 管理 API

4.8.1 buckets 接口

管理 buckets 相关接口需要首先创建 BucketsApi 对象，进而调用其新建、查询、删除等函数方法实现对 buckets 的数据管理。

示例代码：

```
//配置客户端 influxDBClient

//创建 BucketsApi 对象

BucketsApi bucketsApi = influxDBClient.getBucketsApi();

//说明：BucketsApi 对象中提供多种不同参数的新建、查询、删除等函数方法用于 Bucket 对象的管理，下面只是择取部分方法对其进行代码示例，实际可根据需要灵活调整所调用方法。


//创建 Bucket.

BucketRetentionRules retention = new BucketRetentionRules();

retention.setEverySeconds(3600);

Bucket createBucket= bucketsApi.createBucket("iot-bucket",
retention, "xxxxxxxxx");
```

```
//查询 Bucket
Bucket foundBucket =
bucketsApi.findBucketByID(createBucket.getId());

//更新 Bucket
createBucket.setName("Therm sensor 2000");
createBucket.getRetentionRules().add(new
BucketRetentionRules().everySeconds(3600*2));
Bucket updatedBucket = bucketsApi.updateBucket(createBucket);

//删除 Bucket
bucketsApi.deleteBucket(updatedBucket);

//关闭客户端
influxDBClient.close();
```

4.8.2 sources 接口

管理 sources 相关接口需要首先创建 SourcesApi 对象，进而调用其新建、查询、删除等函数方法实现对 sources 的数据管理。

示例代码：

```
//配置客户端 influxDBClient
//创建 SourcesApi 对象
SourcesApi sourcesApi = influxDBClient.getSourcesApi();
//说明：SourcesApi 对象中提供多种不同参数的查询等函数方法用于 Sources 对象的管理，下面只是择取部分方法对其进行代码示例，实际可根据需要灵活调整所调用方法。
```

```
//创建 Source
Source source = new Source();
source.setOrgID("02cebf26d7fc1000");
source.setDefault(false);
source.setName("my-source");
source.setType(Source.TypeEnum.V1);
source.setUrl("http://localhost:8086");
source.setInsecureSkipVerify(true);
source.setTelegraf("telegraf");
source.setToken(UUID.randomUUID().toString());
source.setUsername("admin");
source.setPassword("password");
source.setSharedSecret(UUID.randomUUID().toString());
source.setMetaUrl("/usr/local/var/influxdb/meta");
source.setDefaultRP("autogen");
Source createdSource = sourcesApi.createSource(source);


//查询 Source
Source foundSource =
sourcesApi.findSourceByID(createdSource.getId());


//更新 Source
createdSource.setInsecureSkipVerify(false);
updateSource = sourcesApi.updateSource(createdSource);


//删除 Source
sourcesApi.deleteSource(updateSource);


//关闭客户端
influxDBClient.close();
```

4.9 关闭客户端

InfluxDB Client 提供了优雅的关闭功能。

示例代码：

```
//关闭客户端

influxDBClient.close();
```

5 常见问题

5.1 时序数据库 Influx 有哪些使用约定？

见 1.5 章节限制与约定

5.2 如何选择时序数据库 Influx 的实例规格？

您可以根据每秒写入点数、每秒查询请求、每秒写入请求三个指标作参考，实例规格对应各指标的限制

1) ST-L1：2 核 8G，每秒写入请求：50，每秒写入数据点：25000，每秒查询请求：25
2) ST-L2：4 核 16G：每秒写入请求：100，每秒写入数据点：50000，每秒查询请求：50
3) ST-L3：8 核 32G：每秒写入请求：160，每秒写入数据点：80000，每秒查询请求：80
4) ST-L4：16 核 64G：每秒写入请求：300，每秒写入数据点：150000，每秒查询请求：150
5) ST-L5：32 核 128G：每秒写入请求：500，每秒写入数据点：250000，每秒查询请求：250
6) ST-L6：64 核 256G：每秒写入请求：800，每秒写入数据点：400000，每秒查询请求：400

5.3 为什么我的示例数据导入提示成功，但是查询没有数据？

- 可能是查询的时间范围设置错误，示例数据的是数据集中在 2020 年 8 月 17 日至 2020 年 8 月 20 日
- 确保查询条件选择无误，Bucket 为 test，详见“实例信息-文档下载-示例数据说明”

5.4 如何写入数据到数据库中？

目前有两种方式：

- 一种是通过控制台数据导入，对导入的数据量是有限制的，适合数据量不大的场景。
- 一种是通过 SDK 写入。实时数据可以通过 SDK 进行写入，此时用户需要申请 ECS 主机用于数据接入，部署自己的应用，适合于线上场景。

5.5 如何进行数据查询？

提供两种查询方式：

- 使用查询面板，进行数据查询
- 使用 SDK 进行查询。

5.6 连接时序数据库 Influx 有什么注意事项？

- 将用于连接的 ECS 主机的 IP 添加到白名单：

  详见安全设置章节

- 申请访问令牌，访问请求中添加令牌：

  详见令牌管理章节

5.7 我的存储空间快使用满了，该怎么办？

当您在管理控制台监控到空间使用快满时，可以选择如下任一种方式：

- 在天翼云订购页面对存储进行扩容
- 可设置数据保留策略，超过有效期的数据将自动清理
- 可在控制台“数据管理-数据清理”模块手动删除部分数据
- 将部分不再使用的数据备份到其他存储，并清理掉。

5.8 Influx 如何进行数据可视化展示？

天翼云时序数据库 Influx 版 目前支持以下两种方式进行数据可视化展示：

- 通过控制台对数据进行可视化展示，详情请见查询面板和数据看板。
- 通过接入 Grafana 对数据进行可视化展示。将时序数据库 Influx 接入 Grafana 后，您可以利用 Grafana 的丰富易用的可视化工具更好地监控和分析来自时序数据库 Influx 的数据。