用户手册-0.5.0

工业物联网时序数据库

高可扩展集群系统清华数为

IoTDB-Middleware: IginX

清华大学 软件学院

大数据系统软件国家工程实验室

2021年11月

1.IginX简介

世界上越来越多的企业意识到生产过程中的实时数据与历史数据是最有价值的信息财富，也是整个企业信息系统的核心和基础。随着工业互联网的到来，实时数据和历史数据其体量越来越大，过去的单机版实时数据库或时序数据库都已经无法满足工业数据管理的全面需求。我们可以见到，业界对于高可扩展时序数据库集群系统的需求越来越迫切。

二十年来，我们一直致力于企业信息及企业数据管理的相关工作，为满足上述需求，基于丰富的业界经验，在近年来继开源了一款单机版时序数据库之后，精心打造出了一款高可扩展时序数据库集群系统IginX。

1.1 系统特色

IginX当前发布版本，其主要特色包括：

1. 平滑可扩展，即在有高速写入和查询的条件下，可几乎不影响负载地进行数据库节点扩容。
2. 由于中间件无状态，可以随负载任意进行扩展，也因此可以在资源允许的条件下、很好地确保体现出IoTDB单机版的高性能。
3. 副本方面目前采用多写来实现，可以避开分布式一致性算法导致的性能问题。
4. 底层可以对接IoTDB、InfluxDB等时序数据库，允许同时管理多种异构时序数据库，只要这些时序数据库实现了相关接口即可。
5. 由于支持灵活分片，可以通过编程实现来支持非常灵活的数据副本策略，即非对称式、多粒度的副本策略。

1.2 功能特点

IginX采用迭代周期式开发流程，目前按开发周期计划，主要节点包括首发版v0.1，健壮版v0.2和完整版v1.0。其中：

- 首发版可支持典型的工业互联网边缘端数据管理需求，即满足TPCx-IoT测试所对应的相关应用需求。
- 健壮版可支持单机版时序数据库，尤其是IoTDB，的数据访问功能全集。
- 完整版可支持高可扩展时序数据库集群系统的全部系统特性，包括但不限于：可扩展性、可靠性、高可用性等的智能保障。

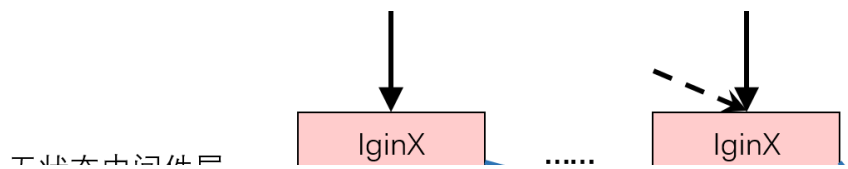
同时，IginX还具备以下功能特点

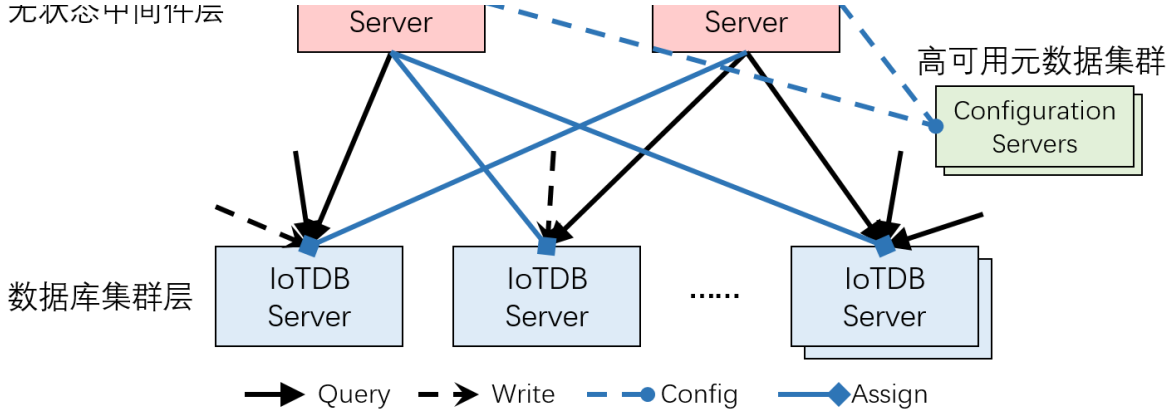
- IginX使用一个数据存储一致性的拓扑支持，比如 etcd 或者 ZooKeeper。这也就意味着集群视图始终是最新的而且对于不同的客户端也能始终保证其一致性。IginX还提供了一个高效地将查询路由给最适合的时序数据库实例的代理。

1.3 系统架构

IginX的整体框架视图如下图所示。自底向上，可以分成2层，下层是可以不相互关联通讯的单机版时序数据库集群层，上层是无状态的IginX服务中间件层。整体框架的相关元数据信息都存储在一个高可用的元数据集群中。这样的架构充分学习、参考了谷歌的Monarch时序数据库系统的良好设计理念，以及其久经考验的实践经验。

其中，读写由IginX进行分片解析，发送到底层的数据库进行处理。IginX通过元数据集群同步信息并进行配置。数据分片的分配由IginX服务端来实现，并基于元数据集群进行冲突处理。





1.4 应用场景

IginX，是清华大学大数据系统软件工程国家实验室，为满足工业互联网场景用户推出的新一代高可扩展时序数据库集群系统。该系统在不需要对单机版时序数据库或实时数据库，尤其是IoTDB，进行侵入式变更的情况下，通过增加管理中间件集群的方式，实现对工业互联网数据进行高可扩展的可靠管理。

IginX是用于部署，扩展和管理单机版时序数据库实例的大型集群时序数据库解决方案。它在架构上可以像在专用硬件上一样有效地在公共或私有云架构中运行。它的设计结合了NoSQL数据库的可伸缩性，并扩展了许多重要的单机版时序数据库功能。

2. 快速上手

基于 IginX 的分布式时序数据库系统由三部分构成，一是 ZooKeeper，用于存储整个集群的原信息，二是 IginX 中间件，用于管理整个集群的拓扑结构，转发处理写入查询请求，并对外提供数据访问接口，三是数据存储后端，用于存储时序数据，以下示例主要使用 IoTDB 作为数据后端。

2.1安装环境

IginX运行时所需的硬件最小配置：

- CPU：单核2.0Hz以上
- 内存：4GB以上
- 网络：100Mbps以上

IginX运行时所依赖的软件配置：

- 操作系统：Linux、Mac或Windows
- JVM：1.8+
- 时序数据库，若为IoTDB，要求在0.11.2以上
- ZooKeeper：3.5.9+

JDK 是 Java 程序的开发的运行环境，由于 ZooKeeper、IginX 以及 IoTDB 都是使用 Java 开发的，因此首先需要安装 Java。下列步骤假设JAVA环境已经安装妥当。

2.2基于发布包的安装与部署方法

- 下载安装包（要求版本在3.5.9+）或使用发布包include目录下的安装包，解压ZooKeeper，以下为下载地址

https://zookeeper.apache.org/releases.html

- 配置ZooKeeper（创建文件conf/zoo.cfg）

tickTime=1000

dataDir=data

clientPort=2181

- 启动ZooKeeper

如果是windows操作系统，则使用以下命令：

bin/zkServer.cmd

否则，使用以下命令：

```
bin/zkServer.sh start
```

- 启动一个或多个单机版本IoTDB时序数据库或者InfluxDB数据库
  - IoTDB

https://iotdb.apache.org/UserGuide/V0.10.x/Get%20Started/QuickStart.html

  - InfluxDB

https://docs.influxdata.com/influxdb/v2.0/get-started/#manually-download-and-install

- 在IginX配置文件中配置数据库信息等，见章节2.4
- 使用以下命令启动一个或多个IginX实例

如果是Windows操作系统，则使用以下命令进行启动：

startIginX.bat

否则，使用以下命令：

./startIginX.sh

2.3基于源码的安装与部署方法

在下列部署步骤之前，要求安装好Maven和Java运行环境。

- 安装Java运行环境

https://www.java.com/zh-CN/download

- 安装Maven

http://maven.apache.org/download.cgi

然后，按以下步骤进行系统安装部署：

- 下载并安装ZooKeeper，要求版本在3.5.9+

https://zookeeper.apache.org/releases.html

- 配置ZooKeeper（创建文件conf/zoo.cfg）

tickTime=1000

dataDir=data

clientPort=2181

- 启动ZooKeeper

如果是windows操作系统，则使用以下命令：

bin/zkServer.cmd

否则，使用以下命令：

```
bin/zkServer.sh start
```

- 启动一个或多个单机版本IoTDB时序数据库或者InfluxDB数据库
  - IoTDB

https://iotdb.apache.org/UserGuide/V0.10.x/Get%20Started/QuickStart.html

  - InfluxDB

https://docs.influxdata.com/influxdb/v2.0/get-started/#manually-download-and-install

- 下载IginX源代码项目

https://github.com/thulab/IginX/

- 编译安装IginX

mvn clean install -DskipTests

- 在IginX配置文件中配置数据库信息等，见章节2.4
- 使用以下命令启动一个或多个IginX实例

如果是Windows操作系统，则使用以下命令进行启动：

startIginX.bat

否则，使用以下命令：

./startIginX.sh

2.4参数配置

配置文件：conf/config.properties

以下为配置文件的内容及其各项的具体含义：

iginx 绑定的 ip

ip=0.0.0.0

iginx 绑定的端口

port=6888

iginx 本身的用户名

username=root

iginx 本身的密码

password=root

zookeeper 连接字符串，目前是填写的本机地址

一般应当启动一个集群，至少由3个节点组成，格式为：127.0.0.1:2181; 127.0.0.1:2182; 127.0.0.1:2183

zookeeperConnectionString=127.0.0.1:2181

时序数据库列表，使用','分隔不同实例

其中，sessionPoolSize为数据库连接池的大小。

下面以IoTDB为例：

storageEngineList=127.0.0.1#6667#iotdb#username=root#password=root#sessionPoolSize=100

【如果是InfluxDB则使用以下配置：

storageEngineList=127.0.0.1#8086#influxdb#url=http://localhost:8086/

】

异步请求最大重复次数；目前建议不修改

maxAsyncRetryTimes=3

异步执行并发数；目前建议不修改

asyncExecuteThreadPool=20

同步执行并发数；目前建议不修改

syncExecuteThreadPool=60

写入的副本个数，目前建议是{0,1,2,3}，如果为0，则没有副本

系统不要求强一致性，因此在节点个数少于副本个数时，也可正常运行

replicaNum=1

底层涉及到的数据库的类名列表

多种不同的数据引擎采用逗号分隔

databaseClassNames=iotdb=cn.edu.tsinghua.iginx.iotdb.IoTDBPlanExecutor,influxdb=cn.edu.tsinghua.iginx.influxdb.InfluxDBPlanExecutor

策略类名

policyClassName=cn.edu.tsinghua.iginx.policy.NativePolicy

```
# 统计信息收集类，用于系统优化时提供辅助信息，一般注释掉不使用
# statisticsCollectorClassName=cn.edu.tsinghua.iginx.statistics.StatisticsCollector
# 统计信息打印间隔，单位毫秒
# statisticsLogInterval=1000

####################
### Rest 服务配置
####################

restip=0.0.0.0

restport=6666
# 是否启动REST服务，不启动时设置为false
enableRestService=true

####################
### InfluxDB 配置
####################
# InfluxDB token
influxDBToken=your-token

# InfluxDB organization
influxDBOrganizationName=my-org
```

2.5交互方式

IginX交互一共有3种方式。

2.5.1 API交互

第一种是基于IginX API开发的客户端程序，与IginX进行交互（见章节3）：

目前在example目录下有相关示例程序：

https://github.com/thulab/IginX/tree/main/example

2.5.2 Restful交互

第二种是基于RESTful接口，与IginX进行交互，主要RESTful接口见章节4。

Restful命令可通过命令行工具：Curl进行

2.5.3基于客户端的SQL交互

第三种是基于IginX自带的客户端进行交互，实现命令扩容（具体命令见章节9）：

（1）基于发布包安装的IginX，直接运行startCli.sh 【Windows环境中应当使用start_cli.bat脚本】，即可通过SQL命令与IginX交互

（2）基于源码编译安装的IginX，则客户端在IginX项目下client/target/client-{$VERSION}目录下，进入该目录可见sbin目录，运行sbin目录中的startCli.sh脚本【Windows环境中应当使用start_cli.bat脚本】，即可通过SQL命令与IginX交互

其中，客户端的启动方式有以下有两种：

（1）默认方式：sbin/startCli.sh

——该方式将使用127.0.0.1地址，端口将使用6888

（2）指定IP地址及端口：sbin/startCli.sh -h 192.168.10.43 -p 6324

客户端交互界面截图如下，看到该界面时，即可输入IginX的SQL命令进行交互

3. 数据访问接口

3.1定义

支持基于典型的时序数据访问API，列出如下：

IginX接口
OpenSessionResp openSession(1:OpenSessionReq req);
Status closeSession(1:CloseSessionReq req);
Status deleteColumns(1:DeleteColumnsReq req);
Status insertRowRecords(1:InsertRowRecordsReq req);
Status insertNonAlignedRowRecords(1:InsertNonAlignedRowRecordsReq req);
Status insertColumnRecords(1:InsertColumnRecordsReq req);
Status insertNonAlignedColumnRecords(1:InsertNonAlignedColumnRecordsReq req);
Status deleteDataInColumns(1:DeleteDataInColumnsReq req);
QueryDataResp queryData(1:QueryDataReq req);
AggregateQueryResp aggregateQuery(1:AggregateQueryReq req);
DownsampleQueryResp downsampleQuery(DownsampleQueryReq req):
ExecuteSqlResp executeSql(1: ExecuteSqlReq req);

3.2描述

各接口具体含义如下：

- openSession：创建Session
  - 输入参数：IP，端口号，用户名和密码
  - 返回结果：是否创建成功，如果成功，返回分配的Session ID
- closeSession：关闭Session
  - 输入参数：待关闭的Session的ID
  - 返回结果：是否关闭成功
- deleteColumns：删除列

    - 输入参数：待删除的列名称列表
    - 返回结果：是否删除成功
- insertRowRecords：行式插入数据
    - 输入参数：列名称列表、时间戳列表、数据列表、数据类型列表和额外参数
    - 返回结果：是否插入成功
    - 说明：数据列表是二维的，内层以列组织，外层以行组织；数据不强制要求对齐
- insertNonAlignedRowRecords：行式插入非对齐数据
    - 输入参数：列名称列表、时间戳列表、数据列表、数据类型列表和额外参数
    - 返回结果：是否插入成功
    - 说明：数据列表是二维的，内层以列组织，外层以行组织；数据不强制要求对齐
- insertColumnRecords：列式插入数据
    - 输入参数：列名称列表、时间戳列表、数据列表、数据类型列表和额外参数
    - 返回结果：是否插入成功
    - 说明：数据列表是二维的，内层以行组织，外层以列组织；数据要求对齐
- insertNonAlignedColumnRecords：列式插入非对齐的数据
    - 输入参数：列名称列表、时间戳列表、数据列表、数据类型列表和额外参数
    - 返回结果：是否插入成功
    - 说明：数据列表是二维的，内层以行组织，外层以列组织；数据要求对齐
- deleteDataInColumns：删除数据
    - 输入参数：列名称列表、开始时间戳和结束时间戳
    - 返回结果：是否删除成功
- queryData：原始数据查询
    - 输入参数：列名称列表、开始时间戳和结束时间戳
    - 返回结果：查询结果集，可以提供列名称列表、时间戳列表、数据列表和数据类型列表等信息
- aggregateQuery：聚合查询
    - 输入参数：列名称列表、开始时间戳、结束时间戳和聚合查询类型
    - 返回结果：查询结果集，可以提供列名称列表、时间戳列表、数据列表和数据类型列表等信息
    - 说明：目前聚合查询支持最大值(MAX)、最小值(MIN)、求和(SUM)、计数(COUNT)、平均值(AVG)、第一个非空值(FIRST)和最后一个非空值(LAST)七种
- downsampleQuery：降采样查询
    - 输入参数：列名称列表、开始时间戳、结束时间戳、聚合类型以及聚合查询的精度
    - 返回结果：查询结果集，可以提供列名称列表、时间戳列表、数据列表和数据类型列表等信息
    - 说明：目前降采样查询中的聚合，支持最大值(MAX)、最小值(MIN)、求和(SUM)、计数(COUNT)、平均值(AVG)、第一个非空值(FIRST)和最后一个非空值(LAST)七种
- executeSql：sql查询

- 输入参数：IginX-SQL 语句
- 返回结果：查询结果集，可以提供列名称列表、时间戳列表、数据列表和数据类型列表等信息
- 说明：具体使用方式见第八节

3.3特性

数据访问接口相关特性：

- 连接池：将前端应用程序查询复用到底层数据库连接池中以优化性能
- IginX 可进行面向单机版时序数据库的并行查询，从而提高数据库查询相关性能。

3.4性能

数据精确度：与底层时序数据库相同。

吞吐性能特性：IginX是无状态的，因此，当应用连接增加的时候，可以实时进行任意规模的扩展，从而确保底层单实例数据库的性能可得到全面体现，即IginX不会成为系统瓶颈。

4. 新版数据访问接口

4.1安装方法

```Shell
mvn clean install -pl session -am -Dmaven.test.skip=true
```

依赖

- JDK >= 1.8
- Maven >= 3.6

在 Maven 中添加依赖包

XML

```xml
<dependency>
  <groupId>cn.edu.tsinghua</groupId>
  <artifactId>iginx-session</artifactId>
  <version>0.5.0</version>
</dependency>
```

4.2接口说明

IginX 提供 IginXClient 接口，来提供查询、写入、删除、用户管理、集群操作的能力。

Java

```java
public interface IginXClient extends AutoCloseable {

    WriteClient getWriteClient();

    AsyncWriteClient getAsyncWriteClient();

    QueryClient getQueryClient();

    DeleteClient getDeleteClient();

    UsersClient getUserClient();

    ClusterClient getClusterClient();

    void close();
}
```

4.3初始化

IginXClient 由 IginXClientFactory 提供的工厂方法进行创建。

该工厂允许用户传入 url 或者 host + port 以及用户名密码，来创建 Client：

Java

```
public static IginXClient create(); // 使用默认用户名密码，连接到本地的 6888 端口部署的 iginx

public static IginXClient create(String url);

public static IginXClient create(String host, int port);

public static IginXClient create(String url, String username, String password);

public static IginXClient create(String host, int port, String username, String password);
```

还可以传入封装各种参数的 IginXClientOptions 实例，来创建 Client：

Java

```
public static IginXClient create(IginXClientOptions options);
```

若干创建 Client 的样例：

Java

```
IginXClient client;

client = IginXClientFactory.create();
client = IginXClientFactory.create("127.0.0.1:6324");

client = IginXClientFactory.create("127.0.0.1", 2333, "root", "root");

client = IginXClientFactory.create(
        IginXClientOptions.builder()
        .host("192.172.1.102").port(2123).username("user").password("password")
        .build()
);
```

4.4数据写入

IginX 支持数据的同步写入与异步写入。其中，同步的写入接口包括：

Java

```java
public interface WriteClient {

    void writePoint(final Point point) throws IginXException;

    void writePoints(final List<Point> points) throws IginXException;

    void writeRecord(final Record record) throws IginXException;

    void writeRecords(final List<Record> records) throws IginXException;

    <M> void writeMeasurement(final M measurement) throws IginXException;

    <M> void writeMeasurements(final List<M> measurements) throws IginXException
;

    void writeTable(final Table table) throws IginXException;

}
```

异步的写入接口除了不会抛出 IginXException（RuntimeException）外，与同步的写入接口完全相同。

Java

```java
public interface AsyncWriteClient extends AutoCloseable {

    void writePoint(final Point point);

    void writePoints(final List<Point> points);

    void writeRecord(final Record record);

    void writeRecords(final List<Record> records);

    <M> void writeMeasurement(final M measurement);

    <M> void writeMeasurements(final List<M> measurements);

    void writeTable(final Table table) throws InterruptedException;

    @Override
    void close() throws Exception;
}
```

数据可以以数据点、数据行、数据表以及数据对象的形式，以同步或异步的方式写入到 IginX 集群中。

数据点

数据点为某个时间序列在给定时间点的数据采样。IginX 支持写入单个数据点，也一次写入多个数据点（不一定属于同一个时间序列）。

Java

```java
WriteClient writeClient = client.getWriteClient();

// 写入一个数据点：cpu.usage.host1 1640918819147 87.4
writeClient.writePoint(
        Point.builder()
                .timestamp(1640918819147L)
                .measurement("cpu.usage.host1")
                .doubleValue(87.4)
                .build()
);

// 写入两个数据点：
// cpu.usage.host2 1640918819147 66.3
// user.login.host2 1640918819147 admin
writeClient.writePoints(
        Arrays.asList(
                Point.builder()
                        .now()
                        .measurement("cpu.usage.host2")
                        .doubleValue(66.3)
                        .build(),
                Point.builder()
                        .now()
                        .measurement("user.login.host2")
                        .binaryValue("admin".getBytes(StandardCharsets.UTF_8))
                        .build()
        )
);
```

数据行

数据行是多个时间序列在同一时刻的数据采样。IginX 支持写入单行，也一次写入多行（多个行的时间序列之间可以相同也可以不同）。

Java

```java
AsyncWriteClient asyncWriteClient = client.getAsyncWriteClient();

// 异步写入一行数据，包含了 4 个时间序列：
// memory.usage.host1 now() 33.4
// memory.usage.host2 now() 24.1
// memory.usage.host3 now() 76.4
// memory.usage.host4 now() 99.8
asyncWriteClient.writeRecord(
        Record.builder()
                .measurement("memory.usage")
                .now()
                .addDoubleField("host1", 33.4)
                .addDoubleField("host2", 24.1)
                .addDoubleField("host3", 76.4)
                .addDoubleField("host4", 99.8)
                .build()
);
```

数据表

数据表是多个时间序列在多个时间戳的数据采样。

Java

```java
WriteClient writeClient = client.getWriteClient();

// 写入如下的二维表:
//    Time              cpu.usage.host1 cpu.usage.host2 cpu.usage.host3
//    1640918819147     23.1            22.1            null
//    1640918820147     null            77.1            86.1

long timestamp = System.currentTimeMillis() - 1000;
writeClient.writeTable(
        Table.builder()
                .measurement("cpu.usage")
                .addField("host1", DataType.DOUBLE)
                .addField("host2", DataType.DOUBLE)
                .addField("host3", DataType.DOUBLE)
                .timestamp(timestamp)
                .doubleValue("host1", 23.1)
                .doubleValue("host2", 22.1)
                .next()
                .timestamp(timestamp + 1000)
                .doubleValue("host2", 77.1)
                .doubleValue("host3", 86.1)
                .build()
);
```

数据对象

IginX 也支持直接写入数据对象，不过数据对象的域必须均为基本类型以及字节数组和字符串。

Java

```java
// 待写入的数据对象
@Measurement(name = "demo.pojo")
static class POJO {

    @Field(timestamp = true)
    long time;

    @Field(name = "field_int")
    int a;

    @Field(name = "field_double")
    double b;

    POJO(long time, int a, double b) {
        this.time = time;
        this.a = a;
        this.b = b;
    }
}


WriteClient writeClient = client.getWriteClient();

writeClient.writeMeasurement(
        new POJO(1640918819147L, 10, 45.1)
);

// POJO 会转化为 2 个序列进行存储:
// demo.pojo.field_int, 类型是 integer
// demo.pojo.field_double, 类型是 double
```

4.5数据查询

集群操作通过 QueryClient 接口来提供。具体接口如下：

Java

```java
public interface QueryClient {

    // 数据查询，返回一个二维数据表
    IginXTable query(final Query query) throws IginXException;

    // 数据查询，并将结果集映射成对象列表（每行映射为一个对象）
    <M> List<M> query(final Query query, final Class<M> measurementType) throws IginXException;

    // 传入 SQL 语句进行查询，返回一个二维数据表
    IginXTable query(final String query) throws IginXException;

    // 数据查询，并通过传入的 consumer 流式消费查询结果的每一行
    void query(final Query query, final Consumer<IginXRecord> onNext) throws IginXException;

    // 传入 SQL 语句进行查询，通过传入的 consumer 流式消费查询结果的每一行
    void query(final String query, final Consumer<IginXRecord> onNext) throws IginXException;

    // 传入 SQL 语句进行查询，并将结果集映射成对象列表（每行映射为一个对象）
    <M> List<M> query(final String query, final Class<M> measurementType) throws IginXException;

    // 传入 SQL 语句进行查询，将结果集映射成对象列表（每行映射为一个对象），并通过传入的 consumer 流式消费查询结果
    <M> void query(final String query, final Class<M> measurementType, final Consumer<M> onNext) throws IginXException;

    // 数据查询，将结果集映射成对象列表（每行映射为一个对象），并通过传入的 consumer 流式消费查询结果
    <M> void query(final Query query, final Class<M> measurementType, final Consumer<M> onNext) throws IginXException;

}
```

一般查询

通常查询可以传入一个 Query 对象，也可以直接传入一条 SQL 语句。

现有的 Query 包括 SimpleQuery、DownsampleQuery、AggregateQuery、LastQuery 等等。

Java

```java
QueryClient queryClient = client.getQueryClient();

IginXTable table = queryClient.query( // 查询 a.a.a 序列最近一秒内的数据
        SimpleQuery.builder()
                .addMeasurement("a.a.a")
                .startTime(System.currentTimeMillis() - 1000L)
                .endTime(System.currentTimeMillis())
                .build()
);

table = queryClient.query( // 查询 a.a.a 序列最近 60 秒内的数据，并以 1 秒为单位进行聚合
        DownsampleQuery.builder()
                .addMeasurement("a.a.a")
                .startTime(System.currentTimeMillis() - 60 * 1000L)
                .endTime(System.currentTimeMillis())
                .precision(1000)
                .build()
);
```

查询出的结果为一张二维表，由表头和若干行组成，下述代码会遍历整张数据表并打印：

Java

```java
IginXHeader header = table.getHeader();
if (header.hasTimestamp()) {
    System.out.print("Time\t");
}
for (IginXColumn column: header.getColumns()) {
    System.out.print(column.getName() + "\t");
}
System.out.println();
List<IginXRecord> records = table.getRecords();
for (IginXRecord record: records) {
    if (header.hasTimestamp()) {
        System.out.print(record.getTimestamp());
    }
    for (IginXColumn column: header.getColumns()) {
        System.out.print(record.getValue(column.getName()));
        System.out.print("\t");
    }
    System.out.println();
}
```

对象映射

IginX 支持将查询到的二维表 IginXTable 的每一行映射成对象，来进行处理。

整个对象映射可以视为写入数据对象的逆过程。

Java

```java
@Measurement(name = "demo.pojo")
static class POJO {

    @Field(timestamp = true)
    long timestamp;

    @Field(name = "field_int")
    int a;

    @Field(name = "field_double")
    double b;

    POJO(long timestamp, int a, double b) {
        this.timestamp = timestamp;
        this.a = a;
        this.b = b;
    }
}

QueryClient queryClient = client.getQueryClient();
List<POJO> pojoList = queryClient.query("select * from demo.pojo where time < now() and time > now() - 1000", POJO.class); // 查询最近一秒内的 pojo 对象
```

流式读取

无论是一般查询还是对象查询，IginX 都支持流式的消费查询结果。

只需要在执行查询时候，额外传入一个 Comsumer，用于顺序消费每一行数据即可。

Java

```java
QueryClient queryClient = client.getQueryClient();

// 查询 a.a 开头的序列在最近一小时内的数据
queryClient.query("select * from a.a where time < now() and time > now() - 1h",
new Consumer<IginXRecord>() {
    @Override
    public void accept(IginXRecord record) {
        // 打印该行的数据
        IginXHeader header = record.getHeader();
        if (header.hasTimestamp()) {
            System.out.print(record.getTimestamp());
        }
        for (IginXColumn column: header.getColumns()) {
            System.out.print(record.getValue(column.getName()));
            System.out.print("\t");
        }
        System.out.println();
    }
});
```

4.6数据删除

集群操作通过 DeleteClient 接口来提供。具体接口如下：

Java

```java
public interface DeleteClient {

    // 删除某个时间序列
    void deleteMeasurement(final String measurement) throws IginXException;

    // 删除多个时间序列
    void deleteMeasurements(final Collection<String> measurements) throws IginXException;

    // 删除某个类对应的时间序列
    void deleteMeasurement(final Class<?> measurementType) throws IginXException;

    // 删除某个时间序列在 [startTime, endTime) 这段时间上的数据
    void deleteMeasurementData(final String measurement, long startTime, long endTime) throws IginXException;

    // 删除多个时间序列在 [startTime, endTime) 这段时间上的数据
    void deleteMeasurementsData(final Collection<String> measurements, long startTime, long endTime) throws IginXException;

    // 删除某个类对应的时间序列在 [startTime, endTime) 这段时间上的数据
    void deleteMeasurementData(final Class<?> measurementType, long startTime, long endTime) throws IginXException;

}
```

4.7用户管理

用户管理通过 UsersClient 接口来提供。具体接口如下：

Java

```java
public interface UsersClient {

    // 增加新用户
    void addUser(final User user) throws IginXException;

    // 更新用户权限
    void updateUser(final User user) throws IginXException;

    // 更新用户密码
    void updateUserPassword(final String username, final String newPassword) throws IginXException;

    // 根据用户名删除用户
    void removeUser(final String username) throws IginXException;

    // 根据用户名查询用户
    User findUserByName(final String username) throws IginXException;

    // 获取系统所有的用户信息
    List<User> findUsers() throws IginXException;

}
```

4.8集群操作

集群操作通过 ClusterClient 接口来提供。具体接口如下：

Java

```java
public interface ClusterClient {

    // 获取集群的拓扑结构
    ClusterInfo getClusterInfo() throws IginXException;

    // 集群扩容：增加一个底层存储节点
    void scaleOutStorage(final Storage storage) throws IginXException;

    // 集群扩容：增加多个底层存储节点
    void scaleOutStorages(final List<Storage> storages) throws IginXException;

    // 获取集群副本数
    int getReplicaNum() throws IginXException;

}
```

其中，扩容逻辑相对复杂，样例代码如下：

Java

```
ClusterClient clusterClient = client.getClusterClient();

// 向集群中新增一个 iotdb 节点和 influxdb 节点：
// iotdb：10.10.1.122:6667
// influxdb：10.10.1.123:6668
clusterClient.scaleOutStorages(
        Arrays.asList(
                Storage.builder()
                        .ip("10.10.1.122")
                        .port(6667)
                        .type("iotdb11")
                        .build(),
                Storage.builder()
                        .ip("10.10.1.123")
                        .port(6668)
                        .type("influxdb")
                        .build()
        )
);
```

5. RESTful访问接口

5.1定义

依照KairosDB，支持基于典型的时序数据访问RESTful API，列出如下：

IginX相关RESTful接口
1. http://[host]:[port]/api/v1/datapoints
2. http://[host]:[port]/api/v1/datapoints/query
3. http://[host]:[port]/api/v1/datapoints/delete
4. http://[host]:[port]/api/v1/metric

上述接口中，①主要涉及写入操作；②主要涉及查询相关操作；③④主要涉及删除操作。

其返回结果都为RESTful服务的状态码和提示信息，用于标识执行结果，另外查询请求会返回对应的json格式查询结果。状态码包括：

RESTful接口服务状态码
1. 200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。
2. 201 CREATED - [POST/PUT/PATCH]：用户新建或修改数据成功。
3. 202 Accepted - [*]：表示一个请求已经进入后台排队（异步任务）
4. 204 NO CONTENT - [DELETE]：用户删除数据成功。
5. 400 INVALID REQUEST -[POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。
6. 401 Unauthorized - [*]：表示用户没有权限（令牌、用户名、密码错误）。
7. 403 Forbidden - [*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。
8. 404 NOT FOUND - [*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。
9. 406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。
10. 410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。
11. 422 Unprocesable entity - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。
12. 500 INTERNAL SERVER ERROR - [*]：服务器发生错误，用户将无法判断发出的请求是否成功。

如果写入、查询和删除请求失败会返回状态码INVALID REQUEST，在返回的正文会提供错误原因。

如果请求的路径错误会返回状态码NOT FOUND，正文为空。

如果请求成功执行会返回状态码OK，除查询请求会返回结果外剩余的请求正文均为空。

5.2描述

目前支持的操作分写入、查询、删除三部分，部分操作需要在请求中附加一个json文件说明操作涉及的数据或数据范围。

相关操作及接口访问具体定义描述如下：

5.2.1写入操作

该操作实现插入一个或多个metric的方法。每个metric包含一个或多个数据点，并可通过tag添加该metric的分类信息便于查询。

5.2.1.1插入数据点

- 命令：

curl -XPOST -H'Content-Type: application/json' -d @insert.json http://[host]:[port]/api/v1/datapoints

其中，insert.json示例内容如下：

```
[
{
   "name": "archive_file_tracked",
   "datapoints": [
      [1359788400000, 123.3],
      [1359788300000, 13.2 ],
      [1359788410000, 23.1 ]
   ],
   "tags": {
      "host": "server1",
      "data_center": "DC1"
},
"annotation": {
"category": ["cat1"],
"title": "text",
"description": "desp"
}
},
{
    "name": "archive_file_search",
    "timestamp": 1359786400000,
```

"value": 321,

"tags": {

"host": "server2"

}

}

]

- insert.json参数描述：

该文件包含一个metric构成的数组，数组每一个值为一个表示一个metric的json字符串，字符串支持的key和对应的value含义如下：

name：必须包含该参数。表示需要插入的metric的名称，一个metric名称唯一对应一个metric。

timestamp：插入单个数据点时使用，表示该数据点的时间戳，数值为从UTC时间1970年1月1日到该对应时间的毫秒数。

value：插入单个数据点时使用，表示该数据点的值，可以是数值或字符串。

datapoints：插入多个数据点时使用，内容为一个数组，每一个值表示一个数据点，为[timestamp, value]的形式（定义参照上述timestamp和value）。

tags：必须包含该参数。表示为该metric添加的标签，可用于定向查找相应的内容。内容为一个字典，可自由添加键和值。可为空值。

annotation: 可选项，用于标记该metric的所有数据点，须提供category、title和description三个域，其中category为字符串数组，title和description为字符串。

5.2.2查询操作

5.2.2.1查询时间范围内的数据

该查询支持对一定时间范围内的数据执行查询，并返回查询结果

- 命令：

curl -XPOST -H'Content-Type: application/json' -d @query.json http://[host]:[port]/api/v1/datapoints/query

其中，query.json示例内容如下：

{

"start_absolute" : 1,

"end_relative": {

"value": "5",

"unit": "days"

},
"metrics": [
{
"name": "archive_file_tracked",
"tags": {
"host": ["server1", "server2"],
"data_center": ["DC1"]
}
},
{
"name": "archive_file_search"
}
]
}

- query.json参数描述：

start_absolute/end_absolute：表示查询对应的起始/终止的绝对时间，数值为从UTC时间1970年1月1日到该对应时间的毫秒数。

start_relative/end_relative：表示查询对应的起始/终止的相对当前系统的时间，值为包含两个键value和unit的字典。其中value对应采样间隔时间值，unit对应单位，值表示查询"milliseconds","seconds", "minutes", "hours", "days", "weeks", "months"中的一个。

（上述两组参数中，start_absolute和start_relative必须包含且仅包含其中一个，end_absolute和end_relative必须包含且仅包含其中一个）

metrics：表示对应需要查询的不同metric。值为一个数组，数组中每一个值为一个字典，表示一个metric。字典的参数如下：

name：必要参数，表示需要查询的metric的名字

tags：可选参数，表示需要查询的metrics中对应的tag信息需要符合的范围。tags对应的值为一个字典，其中每一个键表示查询结果必须包含该tag，该键对应的值为一个数组，表示查询结果中这个tag的值必须是该数组中所有值中的一个。例子中archive_file_tracked的tags参数表示查询结果中data_center标签的值必须为DC1，host标签的值必须为server1或server2

5.2.2.2 包含聚合的查询：

该查询支持对相应的查询结果进行均值（avg）、方差（dev）、计数（count）、首值（first）、尾值（last）、最大值（max）、最小值（min）、求和值（sum）、一阶差分（diff）、除法（div）、

值过滤（filter）、另存为（save_as）、变化率（rate）、采样率（sampler）、百分数（percentile）这些聚合查询。

若需要执行包含聚合的查询，相应查询json文件结构基本与4.2.2.1中一致，在需要执行聚合查询的metric项中添加aggregators参数，表示这一聚合器的具体信息。aggregators对应的值为一个数组，数组中每一个值表示一个特定的aggregator（上述15种功能之一），查询结果按照aggregator出现的顺序按照顺序输出。

同时，部分聚合查询需要采样操作，即从查询的起始时间每隔一定的时间得到聚合结果。采样操作需要在aggregators对应的某一个aggregator值中添加sampling的字典，包含两个键value和unit。其中value对应采样间隔时间值，unit对应单位，可为"milliseconds", "seconds", "minutes", "hours", "days", "weeks", "months", " years"中的一个。

每一种aggregator的具体功能的实例和详细参数如下所示：

（1）均值聚合查询（avg）

- 含义：查询对应范围内数据的平均值，仅对数值类型数据有效
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @avg_query.json http://[host]:[port]/api/v1/datapoints/query

其中，avg_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "tags": {
    "host": [
     "server2"
    ]
   },
   "aggregators": [
    {
```

```
      "name": "avg",
      "sampling": {
        "value": 2,
        "unit": "seconds"
      }
    }
   ]
  }
 ]
}
```


- avg_query.json参数描述：

name：表示该聚合查询的类型，必须为"avg"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

（2）方差聚合查询（dev）

- 含义：查询对应范围内数据的方差，仅对数值类型数据有效
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @dev_query.json http://[host]:[port]/api/v1/datapoints/query

其中，dev_query.json示例内容如下：


```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
```

```
        "name": "dev",
        "sampling": {
          "value": 2,
          "unit": "seconds"
        },
        "return_type":"value"
      }
    ]
  }
 ]
}
```

- dev_query.json参数描述：

name：表示该聚合查询的类型，必须为"dev"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

return_type：

（3）计数聚合查询（count）

- 含义：查询对应范围内数据点的个数
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @count_query.json http://[host]:[port]/api/v1/datapoints/query

其中，count_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
  {
```

```
    "name": "test_query",
    "aggregators": [
     {
      "name": "count",
      "sampling": {
       "value": 2,
       "unit": "seconds"
      }
     }
    ]
   }
 ]
}
```

- count_query.json参数描述：

name：表示该聚合查询的类型，必须为"count"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

（4）首值聚合查询（first）

- 含义：查询对应范围内数据的时间戳最早的数据点的值
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @first_query.json http://[host]:[port]/api/v1/datapoints/query

其中，first_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
```

```
  {
   "name": "test_query",
   "aggregators": [
    {
     "name": "first",
     "sampling": {
      "value": 2,
      "unit": "seconds"
     }
    }
   ]
  }
 ]
}
```

- first_query.json参数描述：

name：表示该聚合查询的类型，必须为"first"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

（5）尾值聚合查询（last）

- 含义：查询对应范围内数据的时间戳最晚的数据点的值
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @last_query.json http://[host]:[port]/api/v1/datapoints/query

其中，last_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
```

```
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
     "name": "last",
     "sampling": {
      "value": 2,
      "unit": "seconds"
     }
    }
   ]
  }
 ]
}
```

- last_query.json参数描述：

name：表示该聚合查询的类型，必须为"last"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

（6）最大值聚合查询（max）

- 含义：查询对应范围内数据的最大值，仅对数值类型数据有效
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @max_query.json http://[host]:[port]/api/v1/datapoints/query

其中，max_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
```

```
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
     "name": "max",
     "sampling": {
      "value": 2,
      "unit": "seconds"
     }
    }
   ]
  }
 ]
}
```

- max_query.json参数描述：

name：表示该聚合查询的类型，必须为"max"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

（7）最小值聚合查询（min）

- 含义：查询对应范围内数据的最小值，仅对数值类型数据有效
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @min_query.json http://[host]:[port]/api/v1/datapoints/query

其中，min_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
```

```
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
     "name": "min",
     "sampling": {
      "value": 2,
      "unit": "seconds"
     }
    }
   ]
  }
 ]
}
```

- min_query.json参数描述：

name：表示该聚合查询的类型，必须为"min"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

（8）求和值聚合查询（sum）

- 含义：查询对应范围内数据的所有数值的和，仅对数值类型数据有效
- 命令：

```
$ curl -XPOST -H'Content-Type: application/json' -d @sum_query.json http://[host]:[port]/api/v1/datapoints/query
```

其中，sum_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
```

```
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
     "name": "sum",
     "sampling": {
      "value": 2,
      "unit": "seconds"
     }
    }
   ]
  }
 ]
}
```

- sum_query.json参数描述：

name：表示该聚合查询的类型，必须为"sum"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

（9）一阶差分聚合查询（diff）

- 含义：查询对应范围内数据的一阶差分，仅对数值类型数据有效
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @diff_query.json http://[host]:[port]/api/v1/datapoints/query

其中，diff_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
```

```
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
     "name": "diff"
    }
   ]
  }
 ]
}
```

- diff_query.json参数描述：

name：表示该聚合查询的类型，必须为"diff"

（10）除法聚合查询（div）

- 含义：返回对应时间范围内每一个数据除以一个定值后的值，仅对数值类型数据有效
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @div_query.json http://[host]:[port]/api/v1/datapoints/query

其中，div_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
  {
```

```
    "name": "test_query",
    "aggregators": [
     {
      "name": "div",
      "divisor": "2"
     }
    ]
   }
 ]
}
```

- div_query.json参数描述：

name：表示该聚合查询的类型，必须为"div"

divisor：表示对应除法的除数

（11）值过滤聚合查询（filter）

- 含义：查询对应范围内数据进行简单值过滤后的结果，仅对数值类型数据有效
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @filter_query.json http://[host]:[port]/api/v1/datapoints/query

其中，filter_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
```

```
        "name": "filter",
        "filter_op": "lt",
        "threshold": "25"
      }
    ]
  }
 ]
}
```

- filter_query.json参数描述：

name：表示该聚合查询的类型，必须为"filter"

filter_op：表示执行值过滤对应的符号，值可为"lte", "lt","gte","gt","equal",分别代表“大于等于”、“大于”、“小于等于”、“小于”、“等于”

threshold：表示值过滤对应的阈值，为数值

（12）另存为聚合查询（save_as）

- 含义：该操作将查询得到的结果保存到一个新的metric中
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @save_as_query.json http://[host]:[port]/api/v1/datapoints/query

其中，save_as_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
```

```
      "name": "save_as",
      "metric_name": "test_save_as"
    }
   ]
  }
 ]
}
```

- save_as_query.json参数描述：

name：表示该聚合查询的类型，必须为"save_as"

metric_name：表示将查询结果保存到新的metric的名称

（13）变化率聚合查询（rate）

- 含义：查询对应范围内数据在每两个相邻区间的变化率，仅对数值类型数据有效
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @rate_query.json http://[host]:[port]/api/v1/datapoints/query

其中，rate_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
     "name": "rate",
     "sampling": {
      "value": 1,
```

```
        "unit": "seconds"
      }
     }
    ]
   }
 ]
}
```

- rate_query.json参数描述：

name：表示该聚合查询的类型，必须为"rate"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

（14）采样率聚合查询（sampler）

- 含义：查询对应范围内数据的采样率
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @sampler_query.json http://[host]:[port]/api/v1/datapoints/query

其中，sampler_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
     "name": "sampler",
     "unit": "minutes"
    }
```

```
    ]
   }
 ]
}
```

- sampler_query.json参数描述：

name：表示该聚合查询的类型，必须为"sampler"

unit：必要参数，对应单位，如4.2.2.2所述。

（15）百分位数聚合查询（percentile）

- 含义：计算数据在目标区间的概率分布，并返回该分布的指定百分位数。这一查询仅对数值类型数据有效
- 命令：

curl -XPOST -H'Content-Type: application/json' -d @percentile_query.json http://[host]:[port]/api/v1/datapoints/query

其中，percentile_query.json示例内容如下：

```
{
 "start_absolute": 1,
 "end_relative": {
  "value": "5",
  "unit": "days"
 },
 "metrics": [
  {
   "name": "test_query",
   "aggregators": [
    {
     "name": "percentile",
     "sampling": {
      "value": "5",
      "unit": "seconds"
     },
```

```
        "percentile": "0.75"
      }
    ]
  }
 ]
}
```

- percentile_query.json参数描述：

name：表示该聚合查询的类型，必须为"percentile"

sampling：必要参数，value对应采样间隔时间值，unit对应单位，如4.2.2.2所述。

percentile：为需要查询的概率分布中的百分比值，定义为0 < percentile <= 1，其中0.75为第75%大的值，1为100%即最大值。

5.2.3删除操作

该操作实现三个删除方法：包括通过时间范围、metric信息查询并删除符合条件的所有数据点的方法，和直接根据metric名称删除某一个metric与其中所有数据点的方法，以及删除一个序列的所有anntation标签信息。

5.2.3.1删除数据点

- 命令：

curl -XPOST -H'Content-Type: application/json' -d @delete.json http://[host]:[port]/api/v1/datapoints/delete

其中，delete.json示例内容如下：

```
{
"start_absolute" : 1,
"end_relative": {
"value": "5",
"unit": "days"
},
"metrics": [
{
"name": "test_query",
```

"tags": {

"host": ["server2"]

}

}

]

}

- delete.json参数描述：

参数含义参见4.2.2.1节“查询时间范围内的数据”

5.2.3.2删除metric

- 命令：

curl -XDELETE http://[host]:[port]/api/v1/metric/[metric_name]

其中metric_name表示对应需要删除的metric的名称

5.2.4 annotation操作

5.2.4.1 添加annotation

添加注释功能支持对于满足条件的数据增加注释。即，给出时间范围、要求添加标签的时序列信息，以及要添加的标签信息，向这段时序列添加给定标签，具体说明如下：

- 命令：

curl -XPOST -H'Content-Type: application/json' -d @addannotation.json http://[host]:[port]/api/v1/datapoints/annotations/add

其中，addannotation.json示例内容如下：

{

"start_absolute": 1359788400000,

"end_absolute" : 1359788400001,

"metrics": [{

"name": "archive_file_tracked.ann",

"tags": {

"host": ["server1"],

```
    "data_center": ["DC1"]
   },
   "annotation": {
    "category": ["cat3","cat4"],
    "title" : "titleNewUp",
    "description" : "dspNewUp"
   }
  },{
   "name": "archive_file_tracked.bcc",
   "tags": {
    "host": ["server1"],
    "data_center": ["DC1"]
   },
   "annotation": {
    "category": ["cat3","cat4"],
    "title" : "titleNewUpbcc",
    "description" : "dspNewUpbcc"
   }
  }]
}
```

- addannotation.json参数描述：
  - start_absolute/end_absolute：必要参数，表示查询对应的起始/终止的绝对时间，数值为从UTC时间1970年1月1日到该对应时间的毫秒数。
  - start_relative/end_relative：必要参数，表示查询对应的起始/终止的相对当前系统的时间，值为包含两个键value和unit的字典。其中value对应采样间隔时间值，unit对应单位，值表示查询"milliseconds","seconds", "minutes", "hours", "days", "weeks", "months"中的一个。

  该文件包含一个metric构成的数组，数组每一个值为一个表示一个metric的json字符串，字符串支持的key和对应的value含义如下：
  - name：必要参数，表示需要查询的metric的名字
  - tags：必要参数，表示需要添加标签的metrics中对应的tag信息需要符合的范围。tags对应的值为一个字典，其中每一个键表示查询结果必须包含该tag，该键对应的值为一个数组，表示要添加标签的对象中这个tag的值必须是该数组中所有值中的一个。

- annotation: 必须包含该参数，用于标记该metric的所有数据点，包含category、title和description三个域，其中category为字符串数组，且为必要参数，title和description为字符串，为非必要参数。

5.2.4.2 更新annotation

对于已经添加的注释，还可以通过更新注释的功能对其更新。即，给出要更新标签的时序列信息（包括其已有的标签信息），以及要最终要更新为的标签信息，更新此段时序列的标签信息，具体说明如下：

- 命令：

curl -XPOST -H'Content-Type: application/json' -d @updateannotation.json http://[host]:[port]/api/v1/datapoints/annotations/update

其中，updateannotation.json示例内容如下：

```
{
 "metrics": [{
  "name": "archive_file_tracked.ann",
  "tags": {
   "host": ["server2"],
   "data_center": ["DC2"]
  },
  "annotation": {
   "category": ["cat3"]
  },
  "annotation-new": {
   "category": ["cat6"],
   "title" : "titleNewUp111",
   "description" : "dspNewUp111"
  }
 },{
  "name": "archive_file_tracked.bcc",
  "tags": {
   "host": ["server1"],
   "data_center": ["DC1"]
```

```
  },
  "annotation": {
    "category": ["cat2"]
  },
  "annotation-new": {
    "category": ["cat6"],
    "title" : "titleNewUp111bcc",
    "description" : "dspNewUp111bcc"
  }
 }]
}
```

- addannotation.json参数描述：

  该文件包含一个metric构成的数组，数组每一个值为一个表示一个metric的json字符串，字符串支持的key和对应的value含义如下：

  - name：必要参数，表示需要更新的metric的名字
  - tags：必要参数，表示需要更新标签的metrics中对应的tag信息需要符合的范围。tags对应的值为一个字典，其中每一个键表示查询结果必须包含该tag，该键对应的值为一个数组，表示要添加标签的对象中这个tag的值必须是该数组中所有值中的一个。
  - annotation: 必须包含该参数，表示需要更新标签的metrics中对应的annotation信息需要符合的范围，包含category一个域，其中category为字符串数组，且为必要参数。category该键对应的值为一个数组，表示要添加标签的对象中这个tag的值必须包含该数组中所有的值。
  - annotation-new: 必须包含该参数，表示需要最终annotation的更新结果，包含category、title和description三个域，其中category为字符串数组，且为必要参数，title和description为字符串，为非必要参数。

5.2.4.3 查询Annotations

1. **给定时间序列查询注释**

给定单条或多条时间序列，查询注释功能可以返回其包含的注释集合，具体如下：

- 命令：

curl -XPOST -H'Content-Type: application/json' -d @annotations.json http://[host]:[port]/api/v1/datapoints/query/annotations

其中，annotations.json示例内容如下：

```
{
```

```
 "metrics": [{
  "name": "archive_file_tracked.ann",
  "tags": {
   "host": ["server1"],
   "data_center": ["DC1"]
  }
 }, {
  "name": "archive_file_tracked.bcc",
  "tags": {
   "host": ["server1"],
   "data_center": ["DC1"]
  }
 }]
}
```

- annotations.json参数描述：

  该文件包含一个metric构成的数组，数组每一个值为一个表示一个metric的json字符串，字符串支持的key和对应的value含义如下：

  - name：必要参数，表示需要查询的metric的名字
  - tags：必要参数，表示要查询的metrics中对应的tag信息需要符合的范围。tags对应的值为一个字典，其中每一个键表示查询结果必须包含该tag，该键对应的值为一个数组，表示要添加标签的对象中这个tag的值必须是该数组中所有值中的一个。

2. 给定注释查询时间序列及数据点

相应地，除了给定时间序列查询注释，查询注释功能还支持给定注释查询时间序列及数据点，具体说明如下：

- 命令：

curl -XPOST -H'Content-Type: application/json' -d @annotations.json http://[host]:[port]/api/v1/datapoints/query/annotations/data

其中，annotations.json示例内容如下：

```
{
 "metrics": [{
  "annotation": {
```

```
    "category": ["cat4"]
   }
  },{
   "annotation": {
    "category": ["cat3"]
   }
  }
 ]
}
```

- annotations.json参数描述：

  该文件包含一个metric构成的数组，数组每一个值为一个表示一个metric的json字符串，字符串支持的key和对应的value含义如下：

  - annotation: 必须包含该参数，为要查询的metric的需要包含的标签信息，包含category、title两个域，其中category为字符串数组，且为必要参数，title为字符串，为非必要参数。category该键对应的值为一个数组，表示要添加标签的对象中这个tag的值必须包含该数组中所有的值。

5.2.4.4 删除annotation

对于已经添加的注释，如果发现添加错误，可以通过删除注释的功能将其删除，具体说明如下：

- 命令：

curl -XPOST -H'Content-Type: application/json' -d @delete.json http://[host]:[port]/api/v1/datapoints/annotations/delete

其中，delete.json示例内容如下：

```
{
 "metrics": [
  {
   "name": "archive_file_tracked.ann",
   "tags": {
    "host": ["server2"],
    "data_center": ["DC2"]
   },
   "annotation": {
```

```
    "category": ["cat3","cat4"]
   }
  }]
}
```

- delete.json参数描述：

  该文件包含一个metric构成的数组，数组每一个值为一个表示一个metric的json字符串，字符串支持的key和对应的value含义如下：

  - name：必要参数，表示需要查询的metric的名字
  - tags：必要参数，表示需要添加标签的metrics中对应的tag信息需要符合的范围。tags对应的值为一个字典，其中每一个键表示查询结果必须包含该tag，该键对应的值为一个数组，表示要添加标签的对象中这个tag的值必须是该数组中所有值中的一个。
  - annotation: 必须包含该参数，为要查询的metric的需要包含的标签信息，包含category、title两个域，其中category为字符串数组，且为必要参数，title为字符串，为非必要参数。category该键对应的值为一个数组，表示要添加标签的对象中这个tag的值必须包含该数组中所有的值。

5.3特性

IginX的RESTful接口具备以下访问特性：

- 接口定义与实现符合RESTful通用规范
- RESTful接口与API接口同时提供服务，互不干扰

5.4性能

数据精确度：与底层时序数据库相同。

吞吐性能特性：IginX RESTful服务也是无状态的，因此，当应用连接增加的时候，可以实时进行任意规模的扩展，从而确保底层单实例数据库的性能可得到全面体现，即IginX RESTful服务不会成为系统瓶颈。

6. 扩容功能

IginX可进行2个层次上的扩容操作，即IginX层和时序数据库层。

6.1 IginX扩容操作

为IginX设置待扩容集群的ZooKeeper相关IP及端口后，启动IginX实例即可：

```
# zookeeper 连接字符串，目前是填写的本机地址
# 一般应当启动一个集群，至少由 3 个节点组成，格式为：127.0.0.1:2181;
127.0.0.1:2182; 127.0.0.1:2183
zookeeperConnectionString=127.0.0.1:2181
```

6.2底层数据库扩容操作

在已有集群基础上，要增加底层数据库节点，我们需要执行以下3个步骤：

1. 启动客户端，给定一个IginX所在的IP及其端口【详见章节2.5.3】

sbin/start_cli.sh -h 192.168.10.43 -p 6324

【Windows环境中应当使用start_cli.bat脚本】

2. 进行命令行交互，输入以下命令，可以增一个IP在192.168.10.43，端口在6667，用户名为root，密码为root的IoTDB

add storageEngine 192.168.10.43#6667#iotdb#username=root#password=root#sessionPoolSize=100#dataDir=/path/to/your/data/

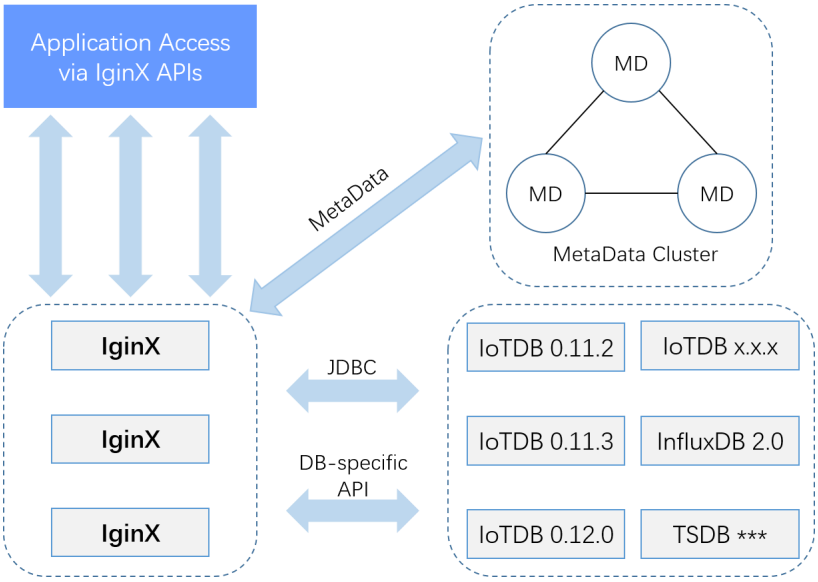
相应地，增加IP在127.0.0.1，端口在8086，token为your-token，organization为your-organization的InfluxDB v2.0实例，使用以下命令：

add storageEngine 127.0.0.1#8086#influxdb#url=http://127.0.0.1:8086/#token=your-token#organization=your-organization

3. 客户端回复”success”，即扩容成功。此时，可输入“quit”退出客户端。

7. 多数据库扩展实现

IginX目前支持的底层数据库包括IoTDB和InfluxDB两种，用户可根据需要自行扩展其他类型的时序数据库。异构部署的IginX架构如下图所示。



7.1支持的功能

其他类型的时序数据库如果想要成为IginX的数据后端，必须支持以下功能：

- 以时间序列为单位插入数据
- 原始数据查询，即可以指定时间范围对单条或多条时间序列进行查询

IginX的其他功能是可选的，如果有相关需求的话需要支持，否则无需支持：

- 创建数据库
- 删除数据库
- 增加列
- 删除列
- 删除数据
- 聚合查询，包括最大值(MAX)、最小值(MIN)、求和(SUM)、计数(COUNT)、平均值(AVG)、第一个非空值(FIRST)和最后一个非空值(LAST)七种
- 降采样查询并聚合结果，具体包括最大值(MAX)、最小值(MIN)、求和(SUM)、计数(COUNT)、平均值(AVG)、第一个非空值(FIRST)和最后一个非空值(LAST)七种聚合方式

7.2可扩展接口

扩展数据库需要实现以下两类接口：

- IStorageEngine：共包括23个接口，名称与含义的对应关系如下

名称	含义
syncExecuteInsertColumnRecordsPlan	同步执行列式插入数据计划
syncExecuteInsertRowRecordsPlan	同步执行行式插入数据计划
syncExecuteQueryDataPlan	同步执行原始数据查询计划
syncExecuteAddColumnsPlan	同步执行增加列计划
syncExecuteDeleteColumnsPlan	同步执行删除列计划
syncExecuteDeleteDataInColumnsPlan	同步执行删除数据计划
syncExecuteCreateDatabasePlan	同步执行创建数据库计划
syncExecuteDropDatabasePlan	同步执行删除数据库计划
syncExecuteDeleteColumnsPlan	同步执行删除列计划
syncExecuteAvgQueryPlan	同步执行AVG查询计划
syncExecuteCountQueryPlan	同步执行COUNT查询计划
syncExecuteSumQueryPlan	同步执行SUM查询计划
syncExecuteFirstQueryPlan	同步执行FIRST查询计划
syncExecuteLastQueryPlan	同步执行LAST查询计划
syncExecuteMaxQueryPlan	同步执行MAX查询计划
syncExecuteMinQueryPlan	同步执行MIN查询计划
syncExecuteDownsampleCountQueryPlan	同步执行 DOWNSAMPLE_COUNT 查询计划
syncExecuteDownsampleSumQueryPlan	同步执行 DOWNSAMPLE_SUM 查询计划
syncExecuteDownsampleMaxQueryPlan	同步执行 DOWNSAMPLE_MAX 查询计划
syncExecuteDownsampleMinQueryPlan	同步执行 DOWNSAMPLE_MIN 查询计划
syncExecuteDownsampleFirstQueryPlan	同步执行 DOWNSAMPLE_FIRST 查询计划
syncExecuteDownsampleLastQueryPlan	同步执行 DOWNSAMPLE_LAST 查询计划
syncExecuteDownsampleAvgQueryPlan	同步执行 DOWNSAMPLE_AVG 查询计划

每个接口的输入参数为对应类型的计划，输出参数为相应的执行结果。其功能是将计划转换为待扩展数据库可用的数据结构，包装后将请求发送到给定的数据后端，再解析得到的结果，按照不同类型的执行结果进行封装。这样一来便可完成IginX与底层数据库功能的对接。

- QueryExecuteDataSet：在原始数据查询中，IginX特别提出需要待扩展数据库实现QueryExecuteDataSet接口，这样做是为了形成统一的查询结果模式，方便查询结果的处理及合并。该接口类共包括5个接口，名称与含义的对应关系如下

名称	含义
getColumnNames	获取查询结果集中所有列的名称
getColumnTypes	获取查询结果集中所有列的数据类型
hasNext	查询结果集是否还存在下一行
next	获取查询结果集的下一行
close	关闭查询结果集

7.3异构数据库部署操作

1. 启动InfluxDB2.0

在influxdbd同一目录（如/tpc/ influxdb2.0.4）下，创建配置文件config.yaml，内容如下，则可使数据放在以下目录中：

engine-path: /tpc/influxdb2.0.4/data/engine

其余配置在用户目录下的.influxdbv2目录中，其它相关配置项见：

https://docs.influxdata.com/influxdb/v2.0/reference/config-options

启动InfluxDB命令：

./influxd

启动后，操作获得用于IginX配置的token和organization，命令如下：

./influx setup --username root --password root1234 --org THUIginx --bucket iginx --token iginx-token-for-you-best-way-ever --force

2. 配置项

下面以配置2个数据库，1个为IoTDB，1个为InfluxDB为例，说明主要相关配置项：

时序数据库列表，使用','分隔不同实例【注意：iotdb和influxdb所需的配置项不同】

storageEngineList=192.168.10.44#6667#iotdb#username=root#password=root#sessionPoolSize=100,192.168.10.45#8086#influxdb#url=http://192.168.10.45:8086/

底层涉及到的数据库的类名列表

多种不同的数据引擎采用逗号分隔

databaseClassNames=iotdb=cn.edu.tsinghua.iginx.iotdb.IoTDBPlanExecutor,influxdb=cn.edu.tsinghua.iginx.influxdb.InfluxDBPlanExecutor

####################

InfluxDB 配置

####################

InfluxDB token

influxDBToken= iginx-token-for-you-best-way-ever

InfluxDB organization

influxDBOrganizationName=THUIginx

3. 启动

按正常过程启动系统即可

7.4同数据多版本实现

命名要求

1. 不能重名，可采用以下命名方式

如IoTDB0.11.4与0.12.2版本，可分别实现接口并命名为IoTDB11和IoTDB12

2. 构建单独的模块，撰写单独的POM，进行实现

具体可参考已有模块

8. 部署指导原则

8.1边缘端部署原则

一般的边缘端数据管理场景，要确保数据可靠性，可以通过2副本2节点的时序数据库实例部署来实现。

如果应用连接数较高，可以启动多个IginX；否则，可以仅启动1个IginX。

8.2云端部署原则

在云端单数据中心场景，可通过3副本多节点的时序数据库实例部署来实现。如果应用连接数较高，可以启动多个IginX；否则，可以仅启动1个IginX。

在云端多数据中心场景，可通过跨数据中心3副本多节点的时序数据库实例部署来实现。如果应用连接数较高，可以在每个数据中心启动多个IginX；否则，可以在每个数据中心仅启动1个IginX。

9. IginX-SQL

9.1使用IginX-SQL

IginX支持通过 IginX-Client 使用IginX-SQL与IginX进行直接交互。

IginX-Client的运行方式见章节2.5.3。

9.2数据相关操作

插入数据

SQL

```
INSERT INTO <prefixPath> ((TIMESTAMP|TIME) (, <suffixPath>)+) VALUES (timeValue
 (, constant)+);
```

使用插入数据语句可以向指定的**一条**或**有公共前缀的多条**时间序列中插入数据。

1. 完整路径 <fullPath> = <prefixPath>.<suffixPath>，完整路径对应的时间序列若没有创建则会自动创建
2. time/timestamp 为必填关键字，代表插入数据的时间戳
3. timeValue 为时间戳，包含以下类型
   a. long
   b. now() 函数
   c. yyyy-MM-dd HH:mm:ss 或 yyyy/MM/dd HH:mm:ss
   d. 基于上述三者的加减 duration 运算表达式，如 now() + 1ms、2021-09-01 12:12:12 - 1ns
   e. duration支持 Y|M|D|H|M|S|MS|NS 等单位
4. CONSTANT 为具体的数据
   a. 包括boolean、float、double、int、long、string、空值(NaN、NULL)
   b. 默认情况下对于整数类型和浮点数类型按照 long 和 double 解析
   c. 如果想插入 float 和 int 值，应在具体数据后加上 f、i 后缀，如 2.56f、123i

示例

我们向 <test.test_insert.status> 和 <test.test_insert.version> 两条路径分别插入多条数据

SQL

```
IginX> ;
IginX> INSERT INTO test.test_insert (time, status, version) VALUES (1633421949, false, "v2");
IginX> INSERT INTO test.test_insert (time, status, version) VALUES (1633421950, true, "v3"), (1633421951, false, "v4"), (1633421952, true, "v5");
```

查询我们刚刚插入的结果

SQL

```
IginX> SELECT * FROM test.test_insert;
```

删除数据

SQL

```
DELETE FROM path (, path)* (WHERE <orExpression>)?;
```

使用删除语句可以删除指定的时间序列中符合时间删除条件的数据。在删除数据时，用户可以选择需要删除的一个或多个时间序列、时间序列的前缀、时间序列带*路径对多个时间区间内的数据进行删除

5. 删除数据语句不能删除路径，只会删除对应路径的制定数据

6. 删除语句不支持精确时间点保留，如 delete from a.b.c where time != 2021-09-01 12:00:00;
7. 删除语句不支持值过滤条件删除，如 delete from a.b.c where a.b.c >= 100;

示例

我们可以用下面语句删除全部序列对应的数据

SQL

```
IginX> DELETE FROM *;
```

我们可以用下面语句删除 <test.test_delete.*> 序列在2021-09-01 12:00:00 到 2021-10-01 12:00:00 对应的数据

SQL

```
IginX> INSERT INTO test.test_delete (time, status, version) VALUES (2021-09-01 12:22:01, true, "v1"), (2021-09-01 12:36:03, false, "v2"), (2021-11-01 12:00:00, true, "v3");

IginX> DELETE FROM test.test_delete WHERE time >= 2021-09-01 12:00:00 AND time <= 2021-10-01 12:00:00;
```

查询删除后的结果

SQL

```
IginX> SELECT * FROM test.test_delete;
```

查询数据

SQL

```
SELECT <expression> (, <expression>)* FROM prefixPath <whereClause>? <groupByTimeClause>?

<expression>
    : <functionName>(<suffixPath>)
    | <suffixPath>

<whereClause>
    : WHERE TIME IN <timeInterval> (AND <orExpression>)?
    | WHERE <orExpression>;

<orExpression>: <andExpression> (OR <andExpression>)*;

<andExpression>: <predicate> (AND <predicate>)*;

<groupByTimeClause>: GROUP <timeInterval> BY DURATION;

<timeInterval>
    : (timeValue, timeValue)
    | (timeValue, timeValue]
    | [timeValue, timeValue)
    | [timeValue, timeValue]
```

查询数据语句支持简单范围查询、值过滤查询、聚合查询、降采样查询

8. 聚合查询函数，目前支持 FIRST_VALUE、LAST_VALUE、MIN、MAX、AVG、COUNT、SUM 七种
9. 降采样查询函数，目前支持 LAST、FIRST_VALUE、LAST_VALUE、MIN、MAX、AVG、COUNT、SUM 八种
10. timeRange 为一个连续的时间区间，支持左开右闭、左闭右开、左右同开、左右同闭区间

示例

先插入如下数据

Plain Text

```
---------------------------------
Time test.test_select.boolean test.test_select.string test.test_select.double te
st.test_select.long
0        true        fq4RUeRjS8        0.5        1
1        false        QKzYVQBquj        1.5        2
2        true        qdYVJZOYXI        2.5        3
3        false        CicZibVAAc        3.5        4
4        true        c8Dq337a7d        4.5        5
5        false        jjlohugmSi        5.5        6
6        true        inCgynQcwN        6.5        7
......
97        false        L5E42AIu4j        97.5        98
98        true        gzQQh2oBeg        98.5        99
99        false        9CR2VrtRrp        99.5        100
```

范围查询

查询序列 <test.test_select.string> 和 <test.test_select.double> 在 0-100ms 内的值

SQL

```
IginX> SELECT string, double FROM test.test_select WHERE time >= 0 AND time <= 1
00;
```

值过滤查询

查询序列 <test.test_select.string> 和 <test.test_select.long> 在 0-100ms 内，且 <test.test_select.long> 的值大于 20 的值

SQL

```
IginX> SELECT string, long FROM test.test_select WHERE time >= 0 AND time <= 100
AND long > 20;
```

聚合查询

查询序列 <test.test_select.long> 在 0-100ms 内的平均值

```SQL
IginX> SELECT AVG(long) FROM test.test_select WHERE time >= 0 AND time <= 100;
```

降采样查询

查询序列 <test.test_select.double> 在 0-100ms 内，每 10ms 的最大值

```SQL
IginX> SELECT MAX(double) FROM test.test_select GROUP [0, 100] BY 10ms;
```

查询序列

```SQL
SHOW TIME SERIES;
```

查询序列语句可以查询存储的全部序列名和对应的类型

统计数据总量

```SQL
COUNT POINTS;
```

统计数据总量语句用于统计 IginX 中数据总量

清除数据

```SQL
CLEAR DATA;
```

清除数据语句用于删除 IginX 中全部数据和路径

示例

清除数据，并查询清除之后的数据

SQL

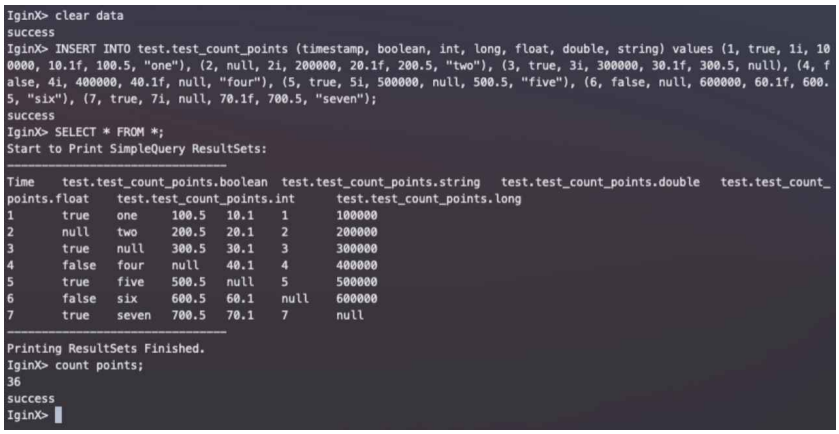
```
IginX> CLEAR DATA;
IginX> SELECT * FROM *;
```

语句定义

COUNT POINTS

统计数据总量语句用于统计 IginX 中数据总量。

操作示例



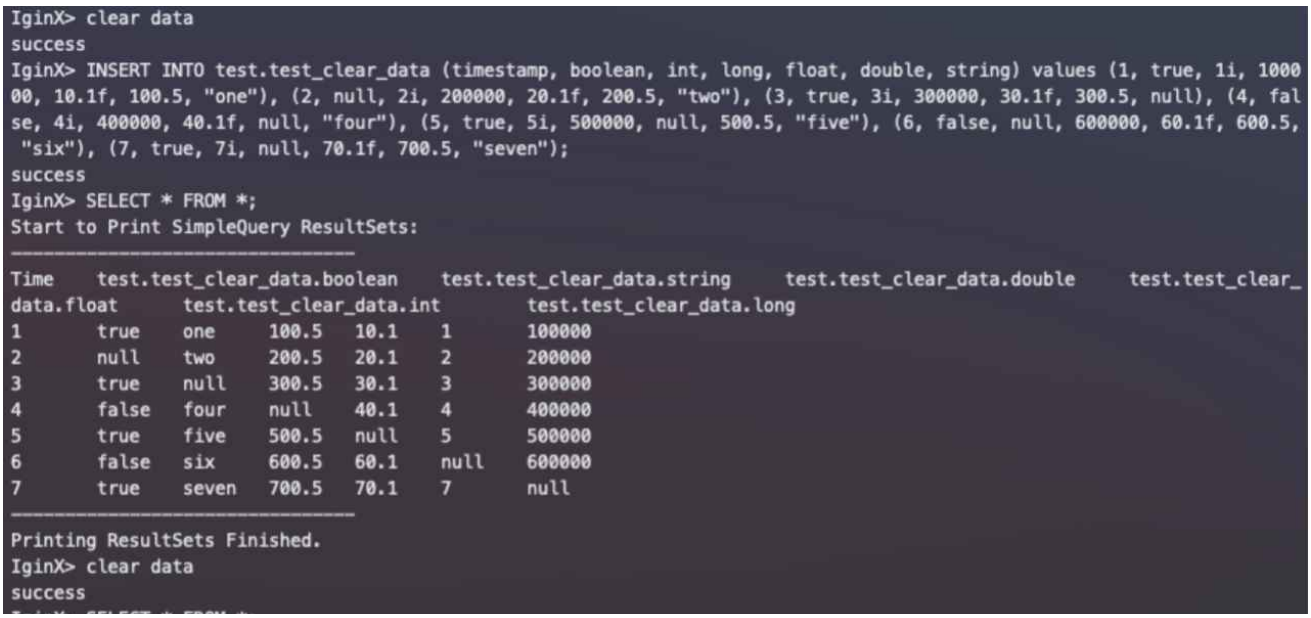
8.2.5清除数据

语句定义

CLEAR DATA

清除数据语句用于删除 IginX 中全部数据和路径。

操作示例



9.3系统相关操作

增加存储引擎

SQL

```
ADD STORAGEENGINE (ip, port, engineType, extra)+;
```

1. 添加引擎之前先保证示例存在且正确运行
2. engineType 目前只支持 IOTDB 和 INFLUXDB
3. 为了方便拓展 extra 目前是 string 类型，具体为一个 string 类型的 map

示例

添加一个 IOTDB11 实例—127.0.0.1: 6667，用户名密码均为root，连接池数量为130

SQL

```
ADD STORAGEENGINE (127.0.0.1, 6667, "iotdb11", "username:root, password:root, sessionPoolSize:130");
```

查询副本数量

```SQL
SHOW REPLICA NUMBER;
```

查询副本数量语句用于查询现在 IginX 存储的副本数量

查询集群信息

```SQL
SHOW CLUSTER INFO;
```

查询集群信息语句用于查询 IginX 集群信息，包括 IginX 节点、存储引擎节点、元数据节点信息等

```
IginX> show cluster info
IginX infos:
+--+---------+----+
|ID|       IP|PORT|
+--+---------+----+
| 0|0.0.0.0|6888|
+--+---------+----+
Storage engine infos:
+--+---------+----+--------+
|ID|       IP|PORT|    TYPE|
+--+---------+----+--------+
| 0|127.0.0.1|6667|iotdb11|
| 1|127.0.0.1|6668|iotdb12|
+--+---------+----+--------+
Meta Storage infos:
+---------+----+----------+
|       IP|PORT|      TYPE|
+---------+----+----------+
|127.0.0.1|2181|zookeeper|
+---------+----+----------+
IginX>
```

用户权限管理

SQL

```
CREATE USER <username> IDENTIFIED BY <password>;
GRANT <permissionSpec> TO USER <username>;
SET PASSWORD FOR <username> = PASSWORD(<password>);
DROP USER <username>
SHOW USER (<username>,)*

<permissionSpec>: (<permission>,)+

<permission>: READ | WRITE | ADMIN | CLUSTER
```

用户权限管理语句用于 IginX 新增用户，更新用户密码、权限，删除用户等

示例

添加一个无任何权限的用户 root1，密码为 root1

SQL

```
CREATE USER root1 IDENTIFIED BY root1;
```

授予用户 root1 以读写权限

SQL

```
GRANT WRITE, READ TO USER root1;
```

将用户 root1 的权限变为只读

SQL

```
GRANT READ TO USER root1;
```

将用户 root1 的密码改为 root2

SQL

```
SET PASSWORD FOR root1 = PASSWORD(root2);
```

删除用户root1

SQL

```
DROP USER root1
```

展示用户 root2, root3 信息

```
SQL
1  SHOW USER root2, root3;
```

展示所有用户信息

```
SQL
1  SHOW USER;
```

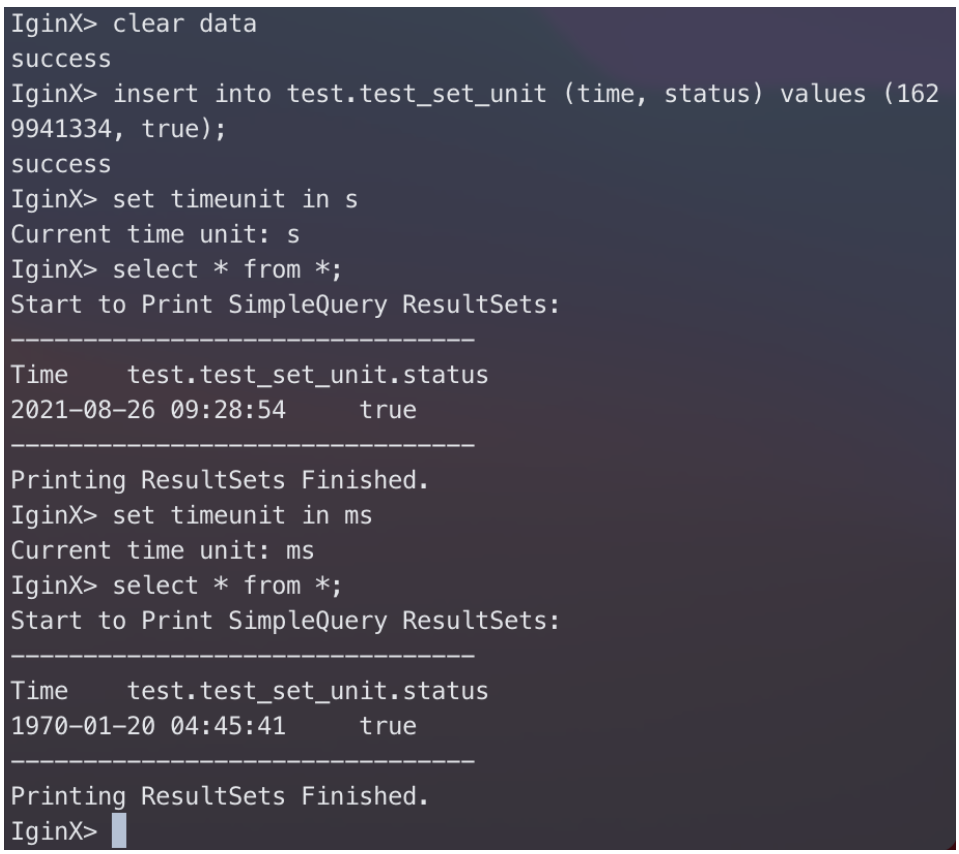
语句定义

SET TIMEUNIT IN <PRECISION>

<PRECISION>: s/ms/μs/ns/second/millisecond/microsecond/nanosecond

设置输出时间单位语句可以设置IginX-Client的输出时间单位。

操作示例



示例中插入了值为1629941334的一条数据，当设置输出的时间单位为秒时，1629941334s即为2021-08-26 09:28:54；当设置输出时间单位为毫秒时，1629941334ms即为1970-01-20 04:45:41。

10. TagKV 适配

在 0.5.0 开发版本的 IginX 中，已经新增了对 TagKV 语法的支持。

10.1 概述

TagKV 指的是你在对向某一个时间序列中写入数据的时候，可以为每一个数据点提供一组字符串键值对，用于存储相关的信息，这一组键值对就被称之为 TagKV。

在进行查询的时候，也可以在提供部分/全部 TagKV 的基础上，对查询结果进行过滤。

10.2 语法

插入数据

SQL

```
INSERT INTO <prefixPath> ([<key=value>(, <key=value>)+])? ((TIMESTAMP|TIME) (, <suffixPath>([<key=value>(, <key=value>)+])?)+) VALUES (timeValue (, constant)+);
```

这里进行一下说明，如果 prefix path 之后直接跟了 tagkv，那么后面的 suffix path 中就不能再出现，并且 prefix path 中的 tagkv 被所有的 suffix path共享。

可以单独为 suffix path 中的某些提供各自的 tagkv。

除此之外，tagkv 是可选的，也可以不提供 tagkv 信息。

样例

SQL

```
insert into ln.wf02 (time, s, v) values (100, true, "v1"); # 不含有 tagkv
insert into ln.wf02[t1=v1] (time, s, v) values (400, false, "v4"); # 提供一个 tagkv，并且被多个序列所共享
insert into ln.wf02[t1=v1,t2=v2] (time, v) values (800, "v8"); # 提供多个 tagkv，并且被多个序列所共享
insert into ln.wf03 (time, s[t1=vv1,t2=v2], v[t1=vv11]) values (1600, true, "v16"); # 针对不同的序列提供不同的 tagkv
```

查询数据

SQL

```
SELECT <expression> (, <expression>)* FROM prefixPath <whereClause>? <withClause
>? <groupByTimeClause>?

<expression>
    : <functionName>(<suffixPath>)
    | <suffixPath>

<whereClause>
    : WHERE TIME IN <timeInterval> (AND <orExpression>)?
    | WHERE <orExpression>;

<orExpression>: <andExpression> (OR <andExpression>)*;

<andExpression>: <predicate> (AND <predicate>)*;

withClause
    : WITH orTagExpression
    ;

orTagExpression
    : andTagExpression (OPERATOR_OR andTagExpression)*
    ;

andTagExpression
    : tagExpression (OPERATOR_AND tagExpression)*
    ;

tagExpression
    : tagKey OPERATOR_EQ tagValue
    | LR_BRACKET orTagExpression RR_BRACKET
    ;

<groupByTimeClause>: GROUP <timeInterval> BY DURATION;

<timeInterval>
    : (timeValue, timeValue)
    | (timeValue, timeValue]
    | [timeValue, timeValue)
    | [timeValue, timeValue]
```

样例

查询 data_center.memory 序列过去一小时的数据，并且要求包含有 tagk = rack & tagv = A 以及 tagk = room & tagv = ROOMA 这两对组合。

SQL

```sql
select memory from data_center where time >= now() - 1h and time < now() with rack="A" and room="ROOMA";
```

10.3 例子

首先利用如下的语句向集群中写入若干数据点：

SQL

```sql
insert into ln.wf02 (time, s, v) values (100, true, "v1");
insert into ln.wf02[t1=v1] (time, s, v) values (400, false, "v4");
insert into ln.wf02[t1=v1,t2=v2] (time, v) values (800, "v8");
insert into ln.wf03 (time, s[t1=vv1,t2=v2], v[t1=vv11]) values (1600, true, "v16");
insert into ln.wf03 (time, s[t1=v1,t2=vv2], v[t1=v1]) values (3200, true, "v32");
```

可以利用前缀语法查询系统中以 ln 开头的数据，所有的序列以及其对应的所有的 TagKV 的组合都会被查询出来：

SQL

```sql
select * from ln;
```

```
IginX> select * from ln;
ResultSets:
+-----------------------+----------+-----------------+----------+-----------------------+-----------------+-----------------------+-----------------------+-----------------+------------------+
|                   Time|ln.wf02.s|ln.wf02.s{t1=v1}|ln.wf02.v|ln.wf02.v{t1=v1,t2=v2}|ln.wf02.v{t1=v1}|ln.wf03.s{t1=v1,t2=vv2}|ln.wf03.s{t1=vv1,t2=v2}|ln.wf03.v{t1=v1}|ln.wf03.v{t1=vv11}|
+-----------------------+----------+-----------------+----------+-----------------------+-----------------+-----------------------+-----------------------+-----------------+------------------+
|1970-01-01T08:00:00.100|      true|             null|        v1|                   null|             null|                   null|                   null|             null|              null|
|1970-01-01T08:00:00.400|      null|            false|      null|                   null|               v4|                   null|                   null|             null|              null|
|1970-01-01T08:00:00.800|      null|             null|      null|                     v8|             null|                   null|                   null|             null|              null|
|1970-01-01T08:00:01.600|      null|             null|      null|                   null|             null|                   null|                   true|             null|               v16|
|1970-01-01T08:00:03.200|      null|             null|      null|                   null|             null|                   true|                   null|              v32|              null|
+-----------------------+----------+-----------------+----------+-----------------------+-----------------+-----------------------+-----------------------+-----------------+------------------+
Total line number = 5
```

随后不指定 tagKV，对 ln.*.s 模式的数据进行查询，可以看到所有的符合条件的蓄力的及其所有的 tagKV 组合均被正确查出。

SQL

```sql
select s from ln.*;
```

```
IginX> select s from ln.*;
ResultSets:
+-----------------------+---------+----------------+-----------------------+-----------------------+
|                   Time|ln.wf02.s|ln.wf02.s{t1=v1}|ln.wf03.s{t1=v1,t2=vv2}|ln.wf03.s{t1=vv1,t2=v2}|
+-----------------------+---------+----------------+-----------------------+-----------------------+
|1970-01-01T08:00:00.100|     true|            null|                   null|                   null|
|1970-01-01T08:00:00.400|     null|           false|                   null|                   null|
|1970-01-01T08:00:01.600|     null|            null|                   null|                   true|
|1970-01-01T08:00:03.200|     null|            null|                   true|                   null|
+-----------------------+---------+----------------+-----------------------+-----------------------+
Total line number = 4
```

在此基础上，用户可以指定某个 tagkv 对，查询的序列下只要符合该 tagkv 组合的数据点都能被正确查出。

SQL

```
1  select s from ln.* with t1=v1;
```

```
IginX> select s from ln.* with t1=v1;
ResultSets:
+----------------------+----------------+----------------------+
|                  Time|ln.wf02.s{t1=v1}|ln.wf03.s{t1=v1,t2=vv2}|
+----------------------+----------------+----------------------+
|1970-01-01T08:00:00.400|           false|                  null|
|1970-01-01T08:00:03.200|            null|                  true|
+----------------------+----------------+----------------------+
Total line number = 2
```

也可以指定多组不同的 tagkv 对并使用逻辑谓词相连：

SQL

```
1  select s from ln.* with t1=v1 or t2=v2;
```

```
IginX> select s from ln.* with t1=v1 or t2=v2;
ResultSets:
+----------------------+----------------+----------------------+----------------------+
|                  Time|ln.wf02.s{t1=v1}|ln.wf03.s{t1=v1,t2=vv2}|ln.wf03.s{t1=vv1,t2=v2}|
+----------------------+----------------+----------------------+----------------------+
|1970-01-01T08:00:00.400|           false|                  null|                  null|
|1970-01-01T08:00:01.600|            null|                  null|                  true|
|1970-01-01T08:00:03.200|            null|                  true|                  null|
+----------------------+----------------+----------------------+----------------------+
Total line number = 3
```

SQL

```
1  select s from ln.* with t1=v1 and t2=vv2;
```

```
IginX> select s from ln.* with t1=v1 and t2=vv2;
ResultSets:
+----------------------+----------------------+
|                  Time|ln.wf03.s{t1=v1,t2=vv2}|
+----------------------+----------------------+
|1970-01-01T08:00:03.200|                  true|
+----------------------+----------------------+
Total line number = 1
```

也可以不指定 tagv 的具体值，而采用通配符的形式，只要包含 t2 这个 tagk 即可：

SQL

```
select s from ln.* with t2=*
```

```
IginX> select s from ln.* with t2=*
ResultSets:
+----------------------+----------------------+----------------------+
|                  Time|ln.wf03.s{t1=v1,t2=vv2}|ln.wf03.s{t1=vv1,t2=v2}|
+----------------------+----------------------+----------------------+
|1970-01-01T08:00:01.600|                  null|                  true|
|1970-01-01T08:00:03.200|                  true|                  null|
+----------------------+----------------------+----------------------+
Total line number = 2
```

11. IginX-Transform

IginX-Transform 允许用户通过 Python 编写自定义数据处理逻辑，并按照一定的顺序编排提交给 IginX 执行，并将处理结果输出到标准流、指定文件或是写回 IginX。

11.1 环境准备

首先确保本地有 python3 环境

IginX 的 Python UDF 依赖 Pemja 实现，需要在本地安装 Pemja 依赖

Shell

```
> pip install pemja
```

我们要在配置文件里面设置本地的python执行路径

Plain Text

```
# python脚本启动命令
pythonCMD=python3
```

建议设置为"which python"查询出的绝对路径，否则可能会找不到某些第三方依赖包，如下所示

Shell

```
> which python
> python: aliased to
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3
```

Plain Text

```
# python脚本启动命令
pythonCMD=/Library/Frameworks/Python.framework/Versions/3.7/bin/python3
```

11.2 脚本编写与注册

合法的 Python Transform 脚本至少要包括一个自定义类（名称不做要求）、和一个类成员函数 transform(self, rows)

Transform 方法的输入输出均为一个二维列表，用户可以在 transform 方法里面实现自己的数据处理逻辑

一个合法的 Python 脚本 row_sum.py 如下所示

Python

```python
import pandas as pd
import numpy as np


class RowSumTransformer:
    def __init__(self):
        pass

    def transform(self, rows):
        df = pd.DataFrame(rows)
        ret = np.zeros((df.shape[0], 2), dtype=np.integer)
        for index, row in df.iterrows():
            row_sum = 0
            for num in row[1:]:
                row_sum += num
            ret[index][0] = row[0]
            ret[index][1] = row_sum
        return pd.DataFrame(ret, columns=['time', 'sum']).values.tolist()
```

这个 Python 脚本实现了按行求和的功能

实现完 Python 脚本之后，我们要将其注册到 IginX 实例中

```
SQL
1  REGISTER transform PYTHON TASK <className> IN <filePath> AS <name>;
```

<className> 为自定义脚本类名，

<filePath> 为 Python 脚本文件所在的绝对路径

<name> 为 transform 脚本别名

比如，假设我们将上面编写的 row_sum.py 在 data/script 目录下 注册到 IginX

```
SQL
1  REGISTER transform PYTHON TASK "RowSumTransformer" IN "data/script/row_sum.py"
   AS row_sum;
```

注册完成后我们还能查询已注册的 Python 脚本

```
SQL
1  SHOW REGISTER PYTHON TASK;
```

或者删除已注册的 Python 脚本

```
SQL
1  DROP PYTHON TASK <name>;
```

11.3 运行 Transform 作业

当做完上述的准备工作之后，我们就可以开始编排 Transform 作业，并提交给 IginX 执行了。通过任务编排以进行使用，是Transform与下一章节所介绍的UDF之间的主要差别。相比之下UDF主要用于在SQL中进行使用。

IginX收到Transform 作业，并在检查过作业合法性之后，会返回一个 JobId，并开始异步执行作业，用户可以通过 JobId 来查询作业的执行状态，任务总共有 8 种状态。

名称	描述
JOB_UNKNOWN(0)	作业状态未知
JOB_FINISHED(1)	作业完成
JOB_CREATED(2)	作业创建

JOB_RUNNING(3)	作业运行中
JOB_FAILING(4)	作业失败中（正在释放资源）
JOB_FAILED(5)	作业失败
JOB_CLOSING(6)	作业取消中（正在释放资源）
JOB_CLOSED(7)	作业取消

作业中的任务，根据其计算所依赖数据的局部性情况，会形成不同的数据流动方式，并形成2种不同类型的任务，即（1）对于给定输入数据集，如果任务可以仅使用部分数据，就输出这部分数据的正确计算结果，且部分结果直接合并即为全局结果，即为流式任务（2）对于给定输入数据集，如果任务需要获得所有输入数据后，才能输出全局正确结果，即为批式任务。

注意：任务类型的正确判断与定义，会直接影响作业的调度与性能。

dataFlowType	描述
stream	流式任务，数据会以流式的方式一批一批的回调 python 脚本中定义的 transform 方法，每批数据量的大小可以通过修改配置文件的 `batchSize` 项来定义
batch	批式任务，等上游全部数据都被读取进内存后才会回调 python 脚本中定义的 transform 方法

每一个合法的 Transform Job 都必须包含至少一个 task，且第一个 task 必须为一个 IginX 的查询任务，以保证我们有数据源提供数据给后续的 Transform 脚本执行。

我们有两种方式可以提交执行，一种是编写 yaml 文件，并通过 SQL 提交执行作业

方式1：IginX-SQL

一个合法的任务配置文件 job.yaml 如下所示

YAML

```yaml
taskList:
  - taskType: iginx
    dataFlowType: stream
    timeout: 10000000
    sql: select value1, value2, value3, value4 from transform;
  - taskType: python
    dataFlowType: stream
    timeout: 10000000
    className: RowSumTransformer

#exportType: none
#exportType: iginx
exportType: file
#exportFile: /path/to/your/output/dir
exportFile: /Users/cauchy-ny/Downloads/export_file_sum_sql.txt
```

提交文件时，应提供其绝对路径，例如：job.yaml 位于 xxx/xxx/job.yaml，则可以使用以下语句提交

SQL

```sql
COMMIT TRANSFORM JOB "xxx/xxx/job.yaml";
```

通过返回的 JobId 查询 transform 作业状态

SQL

```sql
SHOW TRANSFORM JOB STATUS 12323232;
```

方式2：IginX-Session

另一种是通过 IginX-Session 提交作业，并通过 Session 查询作业状态

Java

```java
// 构造任务
List<TaskInfo> taskInfoList = new ArrayList<>();

TaskInfo iginxTask = new TaskInfo(TaskType.IginX, DataFlowType.Stream);
iginxTask.setSql("select value1, value2, value3, value4 from transform;");
taskInfoList.add(iginxTask);

TaskInfo pyTask = new TaskInfo(TaskType.Python, DataFlowType.Stream);
pyTask.setPyTaskName("RowSumTransformer");
taskInfoList.add(pyTask);

// 提交任务
long jobId = session.commitTransformJob(taskInfoList, ExportType.File, "data" + File.separator + "export_file.txt");
System.out.println("job id is " + jobId);

// 轮询查看任务情况
JobState jobState = JobState.JOB_CREATED;
while (!jobState.equals(JobState.JOB_CLOSED) &&
!jobState.equals(JobState.JOB_FAILED) &&
!jobState.equals(JobState.JOB_FINISHED)) {
    Thread.sleep(500);
    jobState = session.queryTransformJobStatus(jobId);
}
System.out.println("job state is " + jobState.toString());
```

12. IginX UDF

12.1 概述

IginX 提供了多种系统内置函数，包括 min、max、sum、avg、count、first、last 等，但某些时候在实际的运用过程中仍然不能满足我们所有的需求，这时我们提供了 IginX UDF（User-Defined Function）

IginX UDF 使用户能通过编写 Python 脚本来定义自己所需要的函数，注册到 IginX 元数据中，并在 IginX-SQL 中使用，UDF 可以分为三类 UDTF、UDAF、UDSF

UDTF (user-defined time series function)

接受一行输入，并产生一行输出，如计算正弦值的 UDF 函数

Python

```python
import math


class UDFSin:
    def __init__(self):
        pass

    def transform(self, row):
        res = []
        for num in row:
            res.append(math.sin(num))
        return res
```

UDAF (user-defined aggregate function)

接受多行输入，并产生一行输出，如计算平均值的 UDF 函数

Python

```python
import pandas as pd


class UDFAvg:
    def __init__(self):
        pass

    def transform(self, rows):
        df = pd.DataFrame(rows)
        ret = pd.DataFrame(data=df.max(axis=0)).transpose()
        return ret.values.tolist()
```

UDSF (user-defined set transform function)

接受多行输入，并产生多行输出，如多列求和的 UDF 函数

Python

```python
import pandas as pd
import numpy as np


class UDFRowSum:
    def __init__(self):
        pass

    def transform(self, rows):
        df = pd.DataFrame(rows)
        ret = np.zeros((df.shape[0], 2), dtype=np.integer)
        for index, row in df.iterrows():
            row_sum = 0
            for num in row[1:]:
                row_sum += num
            ret[index][0] = row[0]
            ret[index][1] = row_sum
        return pd.DataFrame(ret, columns=['time', 'sum']).values.tolist()
```

12.2 环境准备

首先确保本地有 python3 环境

IginX 的 Python UDF 依赖 Pemja 实现，需要在本地安装 Pemja 依赖

Shell

```
> pip install pemja
```

我们要在配置文件里面设置本地的python执行路径

Plain Text

```
# python脚本启动命令
pythonCMD=python3
```

建议设置为"which python"查询出的绝对路径，否则可能会找不到某些第三方依赖包，如下所示

Shell

```
> which python
> python: aliased to
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3
```

Plain Text

```
# python脚本启动命令
pythonCMD=/Library/Frameworks/Python.framework/Versions/3.7/bin/python3
```

12.3 UDF 编写

合法的 Python UDF 脚本至少要包括一个自定义类（名称不做要求）、和一个类成员函数 transform(self, rows)

Python

```
class UDFFunction:
    def __init__(self):
        pass

    '''
    rows: 函数输入
          对于UDTF来说是一个1*n的一维列表
          对于UDAF和UDSF来说输入为一个m*n的二维列表

    return: 返回值
            对于UDTF和UDAF来说，返回的是一个 1*n 的一位列表
            对于UDSF来说输入为一个m*n的二维列表
    '''
    def transform(self, rows):
        // 实现自定义逻辑
        return ret
```

三种合法的 UDF 示例可以参见概述部分

12.4 UDF 注册

对于 IginX 来说，UDF 编写完成后还需要注册到 IginX 中才能被使用，IginX主要需要知道以下信息

1. UDF 名（可以与自定义类名不一致）
2. 自定义脚本类名
3. Python 脚本的文件名
4. 函数类型

对于 UDF 注册来说主要有以下两种方式，一是通过 SQL 注册，二是通过修改配置文件在启动时加载

方式一：注册

注册语法

Plain Text

```
REGISTER <udfType> PYTHON TASK <className> IN <filePath> AS <name>;


udfType
    : UDAF
    | UDTF
    | UDSF
    ;
```

udfType 为函数类型，className 为自定义脚本类名，filePath 为 Python 脚本文件所在的绝对路径，name 为 UDF 名

示例

我们编写了一个计算正弦值的 UDTF，文件名为 udtf_sin.py，位于文件夹 aa/bb 下

Python

```python
import math


class UDFSin:
    def __init__(self):
        pass

    def transform(self, row):
        res = []
        for num in row:
            res.append(math.sin(num))
        return res
```

我们可以通过下面的语句，将其注册到 IginX 中

SQL

```
REGISTER UDTF PYTHON TASK "UDFSin" IN "aa/bb/udtf_sin.py" AS "sin";
```

方式二：配置文件

IginX 启动时，如果 needInitBasicUDFFunctions 选项被设置为 true，则会根据 udfList 中配置的 UDF 信息加载 python_scripts 文件夹下的 Python 脚本，并注册到元信息中

1. 将配置文件的 needInitBasicUDFFunctions 项设置为 true

Plain Text

```
# 是否初始化配置文件内指定的UDF
needInitBasicUDFFunctions=true
```

2. 将编写好的 Python 文件放在 python_scripts 文件夹中
3. 修改配置文件中的 udfList 配置项，不同的udf之间用 "," 隔开

Plain Text

```
# 初始化UDF列表，用","分隔UDF实例
# 函数别名(用于SQL)#类名#文件名#函数类型
udfList=udf_min#UDFMin#udf_min.py#UDAF,udf_max#UDFMax#udf_max.py#UDAF,udf_sum#UDFSum#udf_sum.py#UDAF,udf_avg#UDFAvg#udf_avg.py#UDAF,udf_count#UDFCount#udf_count.py#UDAF
```

示例

我们编写了一个计算正弦值的 UDTF，文件名为 udtf_sin.py，并将其拷贝到 python_scripts 文件夹下

Python

```python
import math


class UDFSin:
    def __init__(self):
        pass

    def transform(self, row):
        res = []
        for num in row:
            res.append(math.sin(num))
        return res
```

我们可以修改配置文件，让 IginX 在启动时自动将其注册到元信息中

SQL

```
# 是否初始化配置文件内指定的UDF
needInitBasicUDFFunctions=true

# 初始化UDF列表，用","分隔UDF实例
# 函数别名(用于SQL)#类名#文件名#函数类型
udfList=sin#UDFSin#udf_sin.py#UDTF
```

12.5 使用示例

现在我们希望实现一个计算正弦值的自定义 Python 函数，并注册到 IginX 中，并用它来计算序列 root.a 的正弦值

首先我们编写这个脚本

Python

```
import math


class UDFSin:
    def __init__(self):
        pass

    def transform(self, row):
        res = []
        for num in row:
            res.append(math.sin(num))
        return res
```

其次我们将这个脚本注册到 IginX 中，假设它在路径 aa/bb 下

SQL

```
REGISTER UDTF PYTHON TASK "UDFSin" IN "aa/bb/udtf_sin.py" AS "sin";
```

最后我们在 SQL 查询中使用这个函数

SQL

```
SELECT sin(a) FROM root;
```

13. 常见问题

13.1如何知道当前有哪些IginX节点

在相应的ZooKeeper客户端中直接执行查询。我们需要执行以下步骤：

1. 进入ZooKeeper客户端：首先进入apache-zookeeper-x.x.x/bin文件夹，之后执行命令启动客户端 ./zkCli.sh
2. 执行查询命令查看包含哪些IginX节点： ls /iginx，返回结果为形如 [node0000000000, node0000000001] 的节点列表。
3. 查看某一个IginX节点的具体信息：如需要查询（2）中对应node0000000000节点具体信息，则需要执行命令：

get /iginx/node0000000000 得到node0000000000节点具体信息，返回结果为形如 {"id":0,"ip":"0.0.0.0","port":6324}的字典形式，参数分别代表节点在ZooKeeper中对应的ID，IginX节点自身的IP和端口号。

13.2如何知道当前有哪些时序数据库节点

在相应的ZooKeeper客户端中直接执行查询。我们需要执行以下步骤：

1. 进入ZooKeeper客户端：首先进入apache-zookeeper-x.x.x/bin文件夹，之后执行命令启动客户端 ./zkCli.sh
2. 执行查询命令查看包含哪些时序数据库节点： ls /storage，返回结果为形如 [node0000000000] 的节点列表。
3. 查看某一个时序数据库节点的具体信息：如需要查询（2）中对应node0000000000节点具体信息，则需要执行命令：

get /storage/node0000000000 得到node0000000000节点具体信息，返回结果为形如

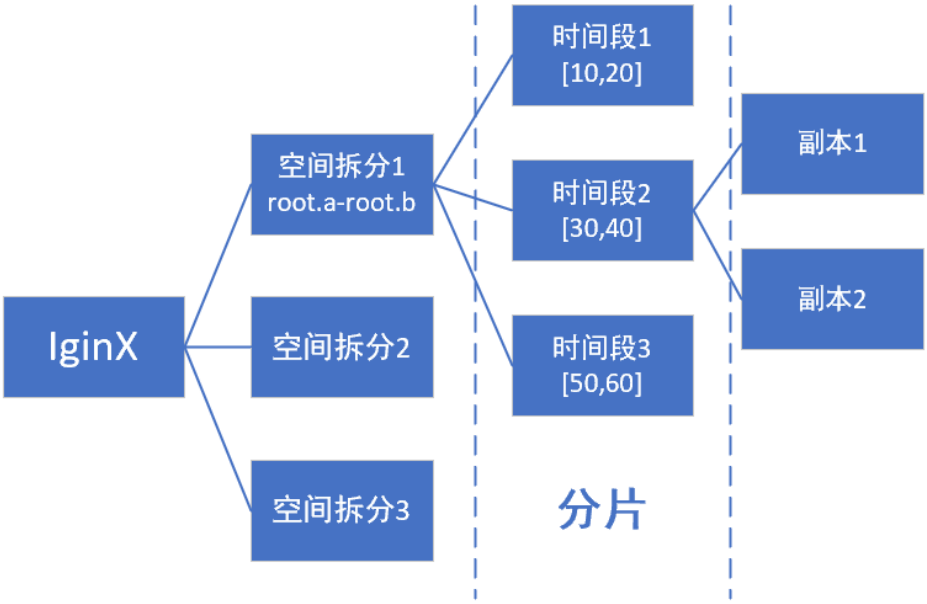
{"id":0,"ip":"127.0.0.1","port":6667,"extraParams": {"password":"root","sessionPoolSize":"100","username":"root"},"storageEngine":"IoTDB"}

的字典形式，参数按照顺序分别代表节点在ZooKeeper中对应的ID，时序数据库节点自身的IP和端口号，以及额外的启动参数，与该数据库节点的数据库类型。

13.3如何知道数据分片当前有几个副本

在相应的ZooKeeper客户端中执行查询。

需要了解的是，分片在IginX存储的格式如下图：



需要先通过对应的空间拆分和时间拆分确定需求的分片，然后即可查询该分片的具体信息。

我们需要执行以下步骤：

1. 进入ZooKeeper客户端：首先进入apache-zookeeper-x.x.x/bin文件夹，之后执行命令启动客户端 ./zkCli.sh
2. 执行查询命令查看IginX目前包含哪些空间拆分： ls /fragment，得到的结果为形如

[null-root.sg1.d1.s1, root.sg1.d3.s1-null, root.sg1.d1.s1-null, null-root.sg1.d3.s1] 的空间拆分列表。

3. 查看该空间拆分下的分片情况：如需要查询（2）中对应null-root.sg1.d1.s1这一空间拆分具体信息，则需要执行命令：

ls /fragment/null-root.sg1.d1.s1，返回结果为形如 [0, 100, 1000]的列表形式，其中每一个参数表示有一个分片以该时间作为起始时间。如[0, 100, 1000]的列表表示该空间拆分下包含3个分片，其分片的最小时间戳分别为0，100和1000。

4. 查询某一个分片的具体信息。在前三步中我们确定了该分片在结构中的对应位置，如需要查询null-root.sg1.d1.s1空间拆分下，起始时间为0的分片的具体信息，则需要执行命令：

get /fragment/null-root.sg1.d1.s1/0，返回结果为形如

{"timeInterval":{"startTime":0,"endTime":99},"tsInterval":
{"endTimeSeries":"root.sg1.d1.s1"},"replicaMetas":{"0":{"timeInterval":
{"startTime":0,"endTime":9223372036854775807},"tsInterval":
{"endTimeSeries":"root.sg1.d1.s1"},"replicaIndex":0,"storageEngineId":0}},"createdBy":0,"updatedBy":0} 的字典形式。

其中，replicaMetas参数表示存储该分片上所有副本元信息的字典，其元素个数即为该分片的副本个数。

其他参数含义如下：

timeInterval: 表示这一分片的起始和终止时间

tsInterval: 表示这一分片的起始和终止时间序列（可为空）

replicaMetas中每一个元素的键表示副本的序号，值包含该副本数据起始终止时间、起始终止时间序列、对应时序数据库节点等信息

createdBy：表示创建该分片的IginX编号

updatedBy：表示最近更新该分片的IginX编号

13.4如何加入IginX的开发，成为IginX代码贡献者?

在IginX的开源项目地址上https://github.com/thulab/IginX提Issue、提PR，IginX项目核心成员将对代码进行审核后，合并进代码主分支中。

IginX当前版本在写入和查询功能方面，支持还不丰富，只有写入、范围查询和整体聚合查询。因此，非常欢迎喜欢IginX的开发者为IginX完成相关功能的开发。

13.5 IginX集群版与IoTDB-Raft版相比，各自特色在何处?

以下为当前的IginX集群版与IoTDB-Raft版特性对比表：

特性\系统	IoTDB-Raft版	IginX集群版
平滑可扩展性	无	有
异构数据库支持	无	有
存算分层扩展性	无	有
分布式一致性维护代价	有	无
灵活分片	无	有
灵活副本策略	无	有
对等强一致性	有	无

部署组件	同构	异构