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

用
户
手
册

清华大学　软件学院

大数据系统软件国家工程实验室

2021 年 5 月

目录

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

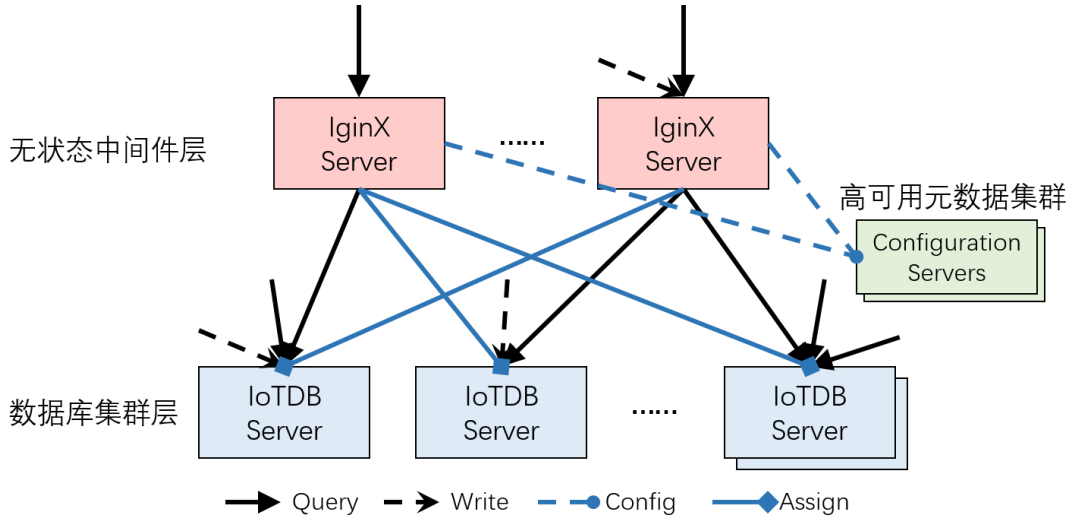
- IginX 使用一个数据存储一致性的拓扑支持，比如 etcd 或者 ZooKeeper。这也就意味着集群视图始终是最新的而且对于不同的客户端也能始终保证其一致性。IginX 还提供了一个高效地将查询路由给最适合的时序数据库实例的代理。

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，readSessions 与 writeSessions 分别为数据库读写连接池的大小。

建议 readSessions 与 writeSessions 要与单个请求所涉及的数据分片个数相符。

下面以 IoTDB 为例：

storageEngineList=127.0.0.1#6667#iotdb#username=root#password=root#readSessions=2#writeSessions=5

【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.IoTDBPlanExec
utor,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：

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

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

第三种是基于 IginX 自带的客户端进行交互，实现命令扩容（见章节 5）：

IginX 安装后，该客户端在 client 子模块的 target 目录下

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
① OpenSessionResp openSession(1:OpenSessionReq req);
② Status closeSession(1:CloseSessionReq req);
③ Status createDatabase(1:CreateDatabaseReq req);
④ Status dropDatabase(1:DropDatabaseReq req);
⑤ Status addColumns(1:AddColumnsReq req);
⑥ Status deleteColumns(1:DeleteColumnsReq req);
⑦ Status insertRowRecords(1:InsertRowRecordsReq req);
⑧ Status insertColumnRecords(1:InsertColumnRecordsReq req);
⑨ Status deleteDataInColumns(1:DeleteDataInColumnsReq req);
⑩ QueryDataResp queryData(1:QueryDataReq req);
⑪ AggregateQueryResp aggregateQuery(1:AggregateQueryReq req);
⑫ DownsampleQueryResp downsampleQuery(DownsampleQueryReq req):
⑬ ValueFilterQueryResp valueFilterQuery(ValueFilterQueryReq 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

- createDatabase：创建数据库
  - 输入参数：待创建的数据库名称
  - 返回结果：是否创建成功
- dropDatabase：删除数据库
  - 输入参数：待删除的数据库名称
  - 返回结果：是否删除成功
- addColumns：增加列
  - 输入参数：待增加的列名称列表和额外参数
  - 返回结果：是否增加成功
  - 说明：额外参数是可选的，例如 IoTDB 增加列需要指定数据类型、编码方式和压缩方式等
- 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
- 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

- 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

- valueFilterQuery：降采样查询
  - 输入参数：列名称列表、开始时间戳、结束时间戳以及用来描述值过滤信息的布尔表达式
  - 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：布尔表达式由值过滤子项经 and、or 和 not 连接而成，每个值过滤子项可表示为：列名 op 参数，op 为六个运算符（>、>=、==、!=、<=、<）之一。

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. RESTful 访问接口

4.1 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。

4.2 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：

4.2.1 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。

4.2.1.1 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。

4.2.1.2 插入标记

- 命令：

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

  其中，insertannotation.json 示例内容如下：

```
[
{
    "name": "archive_file_tracked",
    "datapoints": [
        1359788400000,
        1359788300000,
        1359788410000,
    ],
    "tags": {
        "host": "server1",
        "data_center": "DC1"
    },
    "annotation": {
         "category": ["cat1"],
```

```
        "title": "text",
        "description": "desp"
        }
}
]
```

- insertannotation.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。

datapoints：必须包含该参数。内容为一个数组，每一个值表示一个时间戳。

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：必须包含该参数，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。

4.2.2 查询操作

4.2.2.1 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

4.2.2.2 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。

4.2.2.3 查询 Annotations

该查询支持对 Annotations 执行查询，并返回查询结果

- 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":"test_query",
                "tags": { "host": "server1", "data_center": "DC1" },
                "category":"ab*",
                "text":"?a+",
                "description":"s*"
        }]
}
```

- annotations.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。

（上述两组参数均为可选参数，如没有的话起始时间戳默认为 0，结束时间戳默认为现在时间戳）

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：必要参数，表示需要查询的 metrics 中对应的 tag 信息需要符合的范围。

category、text、description： 可选参数。是一个正则表达式，如某个数据点的 annotation 的任一 category 匹配该正则表达式且 text 和 description 也匹配对应的正则表达式则返回该数据点的 annotation，如果无参数默认匹配所有字符串。

4.2.3 删除操作

该操作实现两个删除方法：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。

4.2.3.1 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”

4.2.3.2 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

4.3 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

4.4 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。

5. 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。

5.1 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
```

5.2 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

```
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 #readSessions=20#writeSessions=30

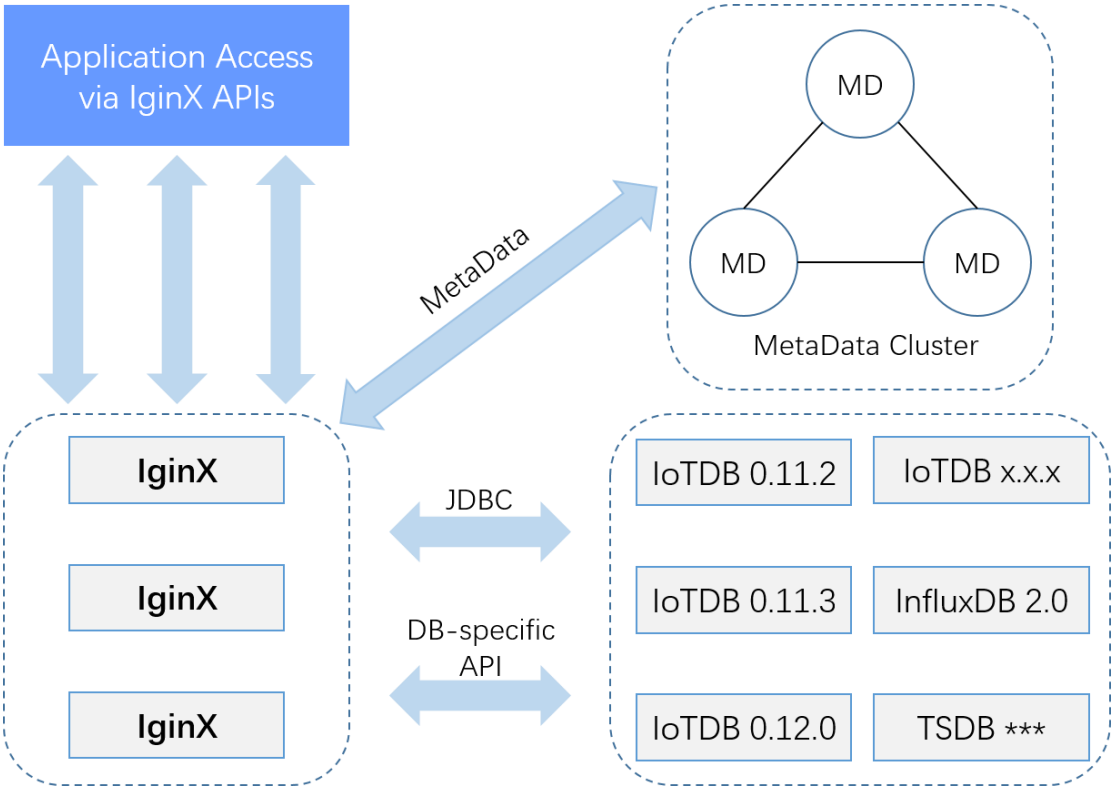
相应地，增加 IP 在 127.0.0.1，端口在 8086 的 InfluxDB v2.0 实例，使用以下命令：

add storageEngine 127.0.0.1#8086#influxdb#url=http://127.0.0.1:8086/

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

6. 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。



6.1 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

6.2 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

6.3 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#readSessions=20#writeSessions=30,192.168.10.45#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. 部署指导原则

7.1 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。

7.2 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。

8. 常见问题

8.1 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。

8.2 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

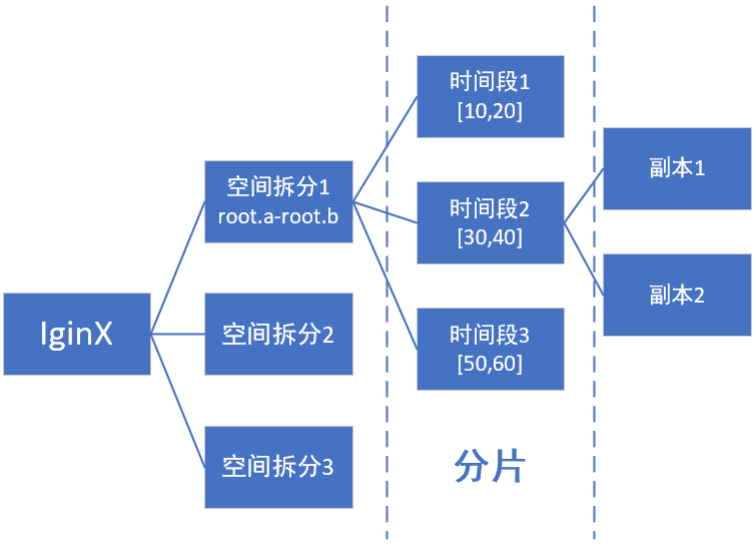
{"id":0,"ip":"127.0.0.1","port":6667,"extraParams":{"password":"root","readSessions":"100","writeSessions":"100","username":"root"},"storageEngine":"IoTDB"}

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。

8.3 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":"r

oot.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

8.4 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。

8.5 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
分布式一致性算法代价	有	无
灵活分片	无	有
灵活副本策略	无	有
对等强一致性	有	无
生态对接	多样	简单
部署组件	同构	异构