冷链监控项目实战

```
学习目标
1. 能够开发出完整flink实时数据处理程序并能够调试、部署
2、能够了解冷链监控项目的业务指标和监控目标
3、能够掌握如何将flink数据流保存至mysql和kafka队列中
4. 能够了解Apache Druid安装配置
4、能够知道apache druid的应用场景
5、能够知道Apache Druid如何从kafka获取数据
```

1 Apache Druid

1.1 简介

Apache Druid是一个拥有大数据实时查询和分析的高容错、高性能开源分布式系统，旨在快速处理大规模的数据，并能够实现快速查询和分析。尤其是当发生代码部署、机器故障以及其他产品系统遇到宕机等情况时，Druid仍然能够保持100%正常运行。创建Druid的最初意图主要是为了解决查询延时问题，当时试图使用hadoop来实现交互式查询分析，但是很难满足实时分析的需要。而Druid提供了以交互方式访问数据的能力，并权衡了查询的灵活性和性能二采取了特殊的存储格式。

Druid允许以类似Dremel和PowerDrill的方式进行单表查询，同时还增加了一些新特性，如为局部嵌套数据结构提供列式存储格式、为快速过滤做索引、实时摄取和查询、高容错的分布式体系架构等。

1.1.1 特性

- 为分析而设计：为OLAP工作流的探索性分析而构建，支持各种过滤、聚合和查询等类；
- 快速的交互式查询：Druid的低延迟数据摄取架构允许事件在他们创建后毫秒内可被查询到；
- 高可用性：Druid的数据在系统更新时依然可用，规模的扩大和缩小都不会造成数据丢失；
- 可扩展：Druid已实现每天能够处理数十亿事件和TB级数据。

Druid设计时的三个原则：

- 快速查询 : 部分数据聚合 + 内存化 + 索引
- 水平拓展能力: 分布式数据 + 并行化查询
- 实时分析 ： 数据不可更改 、只能追加

1.1.2 适用场景

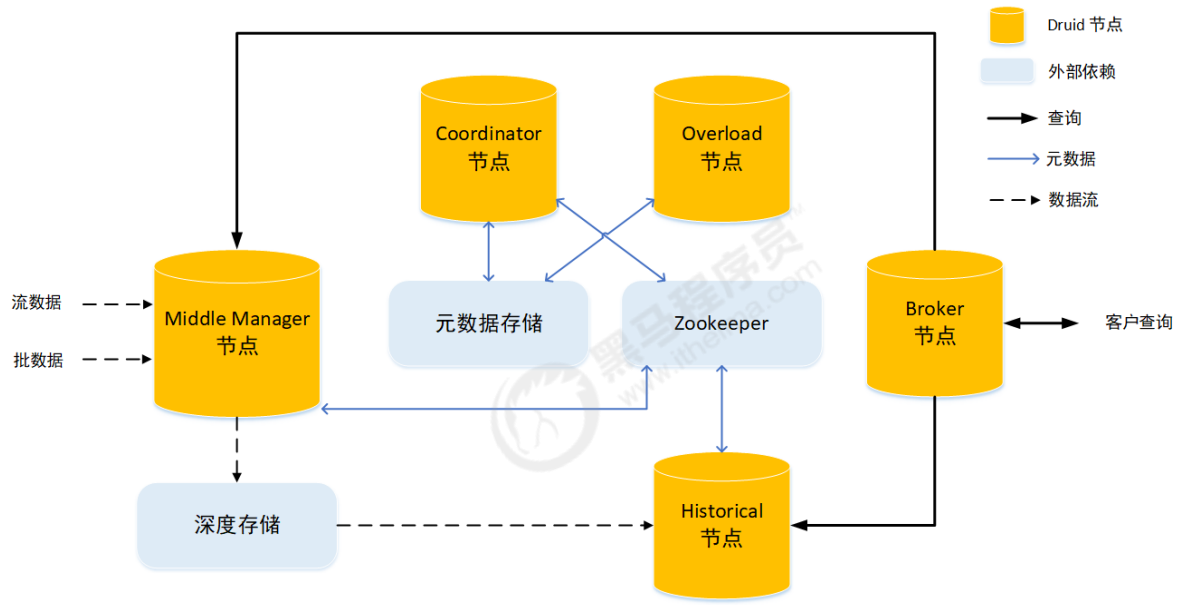
- 需要交互式聚合和快速探究大量数据时；
- 需要实时查询分析时；
- 具有大量数据时，如每天数亿事件的新增、每天数10T数据的增加；
- 对数据尤其是大数据进行实时分析时；
- 需要一个高可用、高容错、高性能数据库时。

1.1.3 不适用场景

- Druid支持流式插入，但是不支持流式更新（更新需要通过后台批处理任务）。
- 需要建立一个离线的报表系统，因此查询延迟并不是系统的关键因素。
- 需要对多个大表进行join操作。

1.1.4 架构

- Overlord： 控制数据摄取工作负载的分配。 Broker： 处理来自外部客户端的查询。
- Router： 可选的进程，可以将请求路由到代理、协调员和霸主。
- Historical: 存储可查询的数据。 MiddleManager: 负责接收数据。



1.2 环境

本演示环境使用以下配置：

```
2核， 8G内存， Centos 7， jdk 1.8， Apache Druid 0.15.0
```

1.2.1 安装JDK

Oracle JDK下载页面:

```
https://www.oracle.com/technetwork/java/javase/archive-139210.html
```

在下载页面选择Java SE 8 rpm文件， 下载过程中需要登录oracle网站，安装jdk：

```
[root@node2-vm06 ~]# cd /opt/software/

[root@node2-vm06 software]# wget -c
https://download.oracle.com/otn/java/jdk/8u202-
b08/1961070e4c9b4e26a04e7f5a083f551e/jdk-8u202-linux-x64.rpm?
AuthParam=1561169291_970b059c42541ddd27d399473a89de61

[root@node2-vm06 software]# mv jdk-8u202-linux-x64.rpm\?
AuthParam\=1561169291_970b059c42541ddd27d399473a89de61 jdk-8u202-linux-x64.rpm

[root@node2-vm06 software]# rpm -ivh jdk-8u202-linux-x64.rpm

[root@node2-vm06 software]# java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)
```

设置JAVA_HOME环境变量：

```
[root@node2-vm06 ~]# vim /etc/profile
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL  --此行之后加入以下语句

export JAVA_HOME=/usr/java/jdk1.8.0_202-amd64/jre
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=.:$JAVA_HOME/lib/tools.jar:$JAVA_HOME/lib/dt.jar

[root@node2-vm06 tmp]# source /etc/profile
```

设置hostname：

```
[root@node2-vm06 broker]# vim /etc/hosts

127.0.0.1       node2-vm06.yjy
```

1.2.2 安装mysql驱动

将mysql驱动放置在extensions/mysql-metadata-storage目录下。

```
[root@node2-vm06 jar]# cp /opt/software/jar/mysql-connector-java-5.1.44-bin.jar
/opt/druid/extensions/mysql-metadata-storage/
```

1.2.3 安装Zookeeper

```
wget -c http://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.5.5/apache-
zookeeper-3.5.5-bin.tar.gz

[root@node2-vm06 opt]# tar -xzf apache-zookeeper-3.5.5-bin.tar.gz
[root@node2-vm06 opt]# mv apache-zookeeper-3.5.5-bin zk3.5.5
[root@node2-vm06 opt]# cp zk3.5.5/conf/zoo_sample.cfg zk3.5.5/conf/zoo.cfg
[root@node2-vm06 opt]# /opt/zk3.5.5/bin/zkServer.
zkServer.cmd  zkServer.sh
[root@node2-vm06 opt]# /opt/zk3.5.5/bin/zkServer.
zkServer.cmd  zkServer.sh
[root@node2-vm06 opt]# /opt/zk3.5.5/bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /opt/zk3.5.5/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

1.2.4 安装kafka

```
# 下载kafka
[root@node2-vm06 software]# wget -c
https://archive.apache.org/dist/kafka/2.1.0/kafka_2.12-2.1.0.tgz
[root@node2-vm06 software]# tar -xzf kafka_2.12-2.3.0.tgz
[root@node2-vm06 software]# cd kafka_2.12-2.3.0

# 配置zk服务器、日志目录、端口等
[root@node2-vm06 config]# vim /opt/software/kafka_2.12-
2.3.0/config/server.properties
zookeeper.connect=172.17.0.143:2181
log.dirs=/tmp/kafka-logs
listeners=PLAINTEXT://172.17.0.143:9092
advertised.listeners=PLAINTEXT://172.17.0.143:9092
```

```
# 启动kafka
[root@node2-vm06 ~]# cd /opt/software/kafka_2.12-2.3.0
[root@node2-vm06 kafka_2.12-2.3.0]# ./bin/kafka-server-start.sh
config/server.properties &

# 关闭kafka
[root@node2-vm06 kafka_2.12-2.3.0]# bin/kafka-server-stop.sh
```

1.3 安装

下载Druid，下载页面： https://druid.apache.org/downloads.html

```
[root@node2-vm06 software]# wget -c
http://mirror.bit.edu.cn/apache/incubator/druid/0.15.0-incubating/apache-druid-
0.15.0-incubating-bin.tar.gz

[root@node2-vm06 software]# tar -xzf apache-druid-0.15.0-incubating-bin.tar.gz
[root@node2-vm06 software]# mv apache-druid-0.15.0-incubating /opt/druid

[root@node2-vm06 apache-druid-0.14.2-incubating]# ll
total 180
drwxr-xr-x.  2 root root   4096 Jun 22 10:20 bin                ---执行文件
drwxr-xr-x.  4 root root     36 Jun 22 10:20 conf               ---配置文件
-rw-r--r--.  1  501 games   551 May 11 04:02 DISCLAIMER
drwxr-xr-x. 23 root root   4096 Jun 22 10:20 extensions         ---Druid核心扩展
drwxr-xr-x.  3 root root     26 Jun 22 10:20 hadoop-dependencies ---Druid hadoop
依赖文件
drwxr-xr-x.  2 root root   8192 Jun 22 10:20 lib                ---Druid核心库文件
和依赖文件
-rw-r--r--.  1  501 games 55738 May 11 04:02 LICENSE
drwxr-xr-x.  4  501 games    26 May 11 04:02 licenses
-rw-r--r--.  1  501 games 86842 May 11 04:02 NOTICE
drwxr-xr-x.  4 root root     36 Jun 22 10:20 quickstart         ---配置文件、示例数
据和帮助文件
-rw-r--r--.  1  501 games  9125 May 11 04:21 README
```

1.4 配置

Druid 0.15.0版本比之前的版本有了比较大的改动，提供了前端导入数据功能、支持kafka事务队列、动态均值查询、新的时间排序查询、SQL增强和轻量级路由搜索等功能。

在操作上，0.15.0版本提供了不同规模的配置文件和对应的启动命令，分为集群模式和单机模式两种部署方式，以下是配置文件目录结构：

```
- conf
    - druid
      - cluster
        - _common
        - data
          - historical
          - middleManager
        - master
          - coordinator-overlord
        - query
          - broker
```

```
          - router
      - single-server
        - large
        - medium
        - micro-quickstart
        - small
        - xlarge
    - supervise
      - cluster
      - single-server
    - tranquility
    - zk
```

集群模式和单服务器模式的区别在于，集群模式把服务分成三类（主服务（historical、middleManager）、数据服务（coordinator、overlord），查询服务（broker、router）），可以分别部署在三个服务器上。而单机模式下分成小微、小型、中等、大型和超大型服务，每种模式都有六个服务，部署在相同的机器上。不同模式默认的JVM内存需求不同。

小型、单机服务器启动程序和配置文件（小微、中型、大型和超大型于此类似）：

```
- 启动程序为:          bin/start-single-server-small
- 启动文件:            conf/supervise/single-server/small.conf
- 详细配置文件夹:      conf/druid/single-server/small
```

集群模式启动程序和配置文件：

```
- 启动文件
  - start-cluster-data-server
  - start-cluster-master-no-zk-server
  - start-cluster-master-with-zk-server
  - start-cluster-query-server
- 启动文件:
  - data.conf
  - master-no-zk.conf
  - master-with-zk.conf
  - query.conf
- 详细配置文件夹:
  - _common
  - data
  - master
  - query
```

本例中，我们使用小型单服务器模式（使用我们自己的zookeeper）：

以下设置了公共配置中的：druid服务器、zk、元数据库和使用hdfs进行深度存储。

```
# 因为使用了本地zk，去掉2181端口的检测
[root@node2-vm06 bin]# vim /opt/druid/bin/verify-default-ports
my @ports = (1527, 8081, 8082, 8083, 8090, 8091, 8200, 9095);


# 注释Zookeeper配置
[root@node2-vm06 ~]# cd /opt/druid/
[root@node2-vm06 druid]# vim conf/supervise/single-server/small.conf
#!p10 zk bin/run-zk conf
```

```
# 查看small模式的配置文件
[root@node2-vm06 druid]# cd conf/druid/single-server/small/
[root@node2-vm06 small]# ll
total 0
drwxr-xr-x. 2 501 games 66 Jun 29 09:58 broker
drwxr-xr-x. 2 501 games 55 Jun 29 09:12 _common
drwxr-xr-x. 2 501 games 66 Jun 29 08:48 coordinator-overlord
drwxr-xr-x. 2 501 games 66 Jun 29 08:48 historical
drwxr-xr-x. 2 501 games 66 Jun 29 08:48 middleManager
drwxr-xr-x. 2 501 games 66 Jun 29 08:48 router

# 修改公共配置
[root@node2-vm06 small]# vim _common/common.runtime.properties
#
# Hostname
#
druid.host=172.17.0.143

#
# Zookeeper
#
druid.zk.service.host=172.17.0.143
druid.zk.paths.base=/druid

#
# Metadata storage
#

# For Derby server on your Druid Coordinator (only viable in a cluster with a
single Coordinator, no fail-over):
#druid.metadata.storage.type=derby
#druid.metadata.storage.connector.connectURI=jdbc:derby://localhost:1527/var/dru
id/metadata.db;create=true
#druid.metadata.storage.connector.host=localhost
#druid.metadata.storage.connector.port=1527

# For MySQL (make sure to include the MySQL JDBC driver on the classpath):
druid.metadata.storage.type=mysql
druid.metadata.storage.connector.connectURI=jdbc:mysql://172.17.0.143:3306/druid
?useUnicode=true&characterEncoding=UTF-8
druid.metadata.storage.connector.user=druid
druid.metadata.storage.connector.password=druid123

#
# Deep storage
#

# For local disk (only viable in a cluster if this is a network mount):
#druid.storage.type=local
#druid.storage.storageDirectory=var/druid/segments

# For HDFS:
druid.storage.type=hdfs
druid.storage.storageDirectory=hdfs://172.17.0.143:9000/druid/segments

# Druid扩展库，扩展库类型见下表
druid.extensions.loadList=["druid-hdfs-storage", "druid-kafka-indexing-service",
"druid-datasketches", "mysql-metadata-storage"]
```

Druid 实现了一个扩展系统，允许在运行时添加功能， 如对深度存储（如hdfs和s3）、元数据存储（如mysql和postgresql）、新聚合器、新输入格式等的支持。 默认的扩展如下：

名称	描述	文档
druid-avro-extensions	支持Apache Avro数据源	link
druid-basic-security	支持基本的HTTP身份验证和基于角色的访问控制。	link
druid-bloom-filter	支持在druid查询中提供bloom过滤器。	link
druid-caffeine-cache	使用Caffeine实现本地缓存	link
druid-datasketches	使用datasketches支持近似计数和数据集。	link
druid-hdfs-storage	HDFS存储	link
druid-histogram	柱状图和数据聚合器，已弃用。推荐使用datasketches quantiles聚合器。	link
druid-kafka-eight	kafka实时消费管道	link
druid-kafka-extraction-namespace	基于Kafka的命名空间查找。需要命名空间查找扩展。	link
druid-kafka-indexing-service	kafka索引查找服务。	link
druid-kinesis-indexing-service	对索引服务的Kinesis摄取进行严格监控 .	link
druid-kerberos	用于druid进程的Kerberos身份验证。	link
druid-lookups-cached-global	用于查找的模块，提供用于查找的JVM全局缓存热切换。它提供用于获取查找数据的JDBC和URI实现。	link
druid-lookups-cached-single	基于查找缓存模块查找需要与全局查找池隔离的用例	link
druid-parquet-extensions	支持Apache Parquet数据格式的数据。需要加载druid avro扩展	link
druid-protobuf-extensions	支持Protobuf数据格式的数据	link
druid-s3-extensions	支持AWS S3云存储	link
druid-stats	统计相关模块，包括方差和标准差。	link
mysql-metadata-storage	MySQL元数据存储	link
postgresql-metadata-storage	PostgreSQL元数据存储	link
simple-client-sslcontext	简单的sslcontext提供程序模块，供druid的内部httpclient在通过https与其他druid进程对话时使用。	link

社区开发的一些扩展组件：

Name	Description	Docs
ambari-metrics-emitter	Ambari Metrics Emitter	link
druid-azure-extensions	Microsoft Azure deep storage.	link
druid-cassandra-storage	Apache Cassandra deep storage.	link
druid-cloudfiles-extensions	Rackspace Cloudfiles deep storage and firehose.	link
druid-distinctcount	DistinctCount aggregator	link
druid-kafka-eight-simpleConsumer	Kafka ingest firehose (low level consumer).	link
druid-orc-extensions	Support for data in Apache Orc data format.	link
druid-rabbitmq	RabbitMQ firehose.	link
druid-redis-cache	A cache implementation for Druid based on Redis.	link
druid-rocketmq	RocketMQ firehose.	link
druid-time-min-max	Min/Max aggregator for timestamp.	link
druid-google-extensions	Google Cloud Storage deep storage.	link
sqlserver-metadata-storage	Microsoft SqlServer deep storage.	link
graphite-emitter	Graphite metrics emitter	link
statsd-emitter	StatsD metrics emitter	link
kafka-emitter	Kafka metrics emitter	link
druid-thrift-extensions	Support thrift ingestion	link
druid-opentsdb-emitter	OpenTSDB metrics emitter	link

Druid各组件的配置文件包括jvm.config和runtime.properties两个文件，jvm.config设置组件的jvm信息，从中我们可以看到默认Druid集群中各组件的内存和默认端口。

```
# 设置各组件的时区
[root@node2-vm06 small]# vim broker/jvm.config
-Duser.timezone=UTC+8

[root@node2-vm06 small]# vim coordinator-overlord/jvm.config
-Duser.timezone=UTC+8

[root@node2-vm06 small]# vim historical/jvm.config
-Duser.timezone=UTC+8

[root@node2-vm06 small]# vim middleManager/jvm.config
-Duser.timezone=UTC+8

[root@node2-vm06 small]# vim router/jvm.config
-Duser.timezone=UTC+8
```

单机版Small模式的默认内存如下：

组件	默认内存	默认端口
broker	3G	8082
coordinator-overlord	4.5G	8081
historical	4G	8083
middleManager	64M	8091
router	512M	8888

runtime.properties文件可以设置组件的服务器、客户端、查询缓存等属性（以下是broker组件）

```
druid.service=druid/broker
druid.plaintextPort=8082

# HTTP server settings
druid.server.http.numThreads=50

# HTTP client settings
druid.broker.http.numConnections=40
druid.broker.http.maxQueuedBytes=5000000

# Processing threads and buffers
druid.processing.buffer.sizeBytes=500000000
druid.processing.numMergeBuffers=2
druid.processing.numThreads=1
druid.processing.tmpDir=var/druid/processing

# Query cache disabled -- push down caching and merging instead
druid.broker.cache.useCache=false
druid.broker.cache.populateCache=false
```

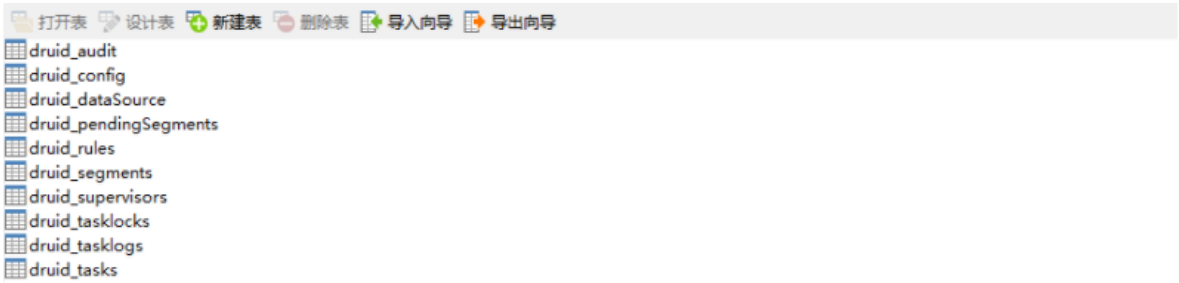
配置属性的详情见以下地址：

```
https://druid.apache.org/docs/latest/configuration/index.html#jvm-configuration-best-practices
```

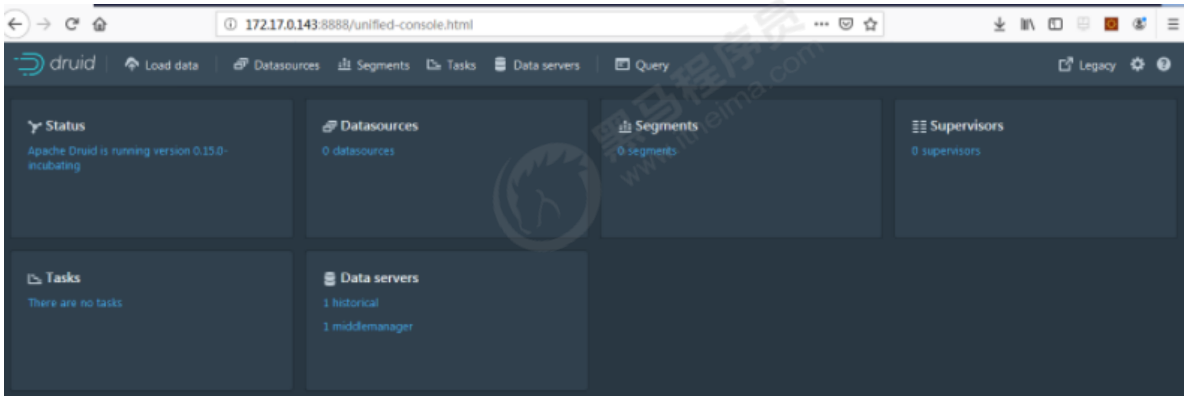
1.5 启动

```
[root@node2-vm06 bin]# /opt/druid/bin/start-single-server-small
```

此时mysql数据库中已经自动创建了Druid的元数据库表。



通过web页面查看各组件状态： http://172.17.0.143:8888



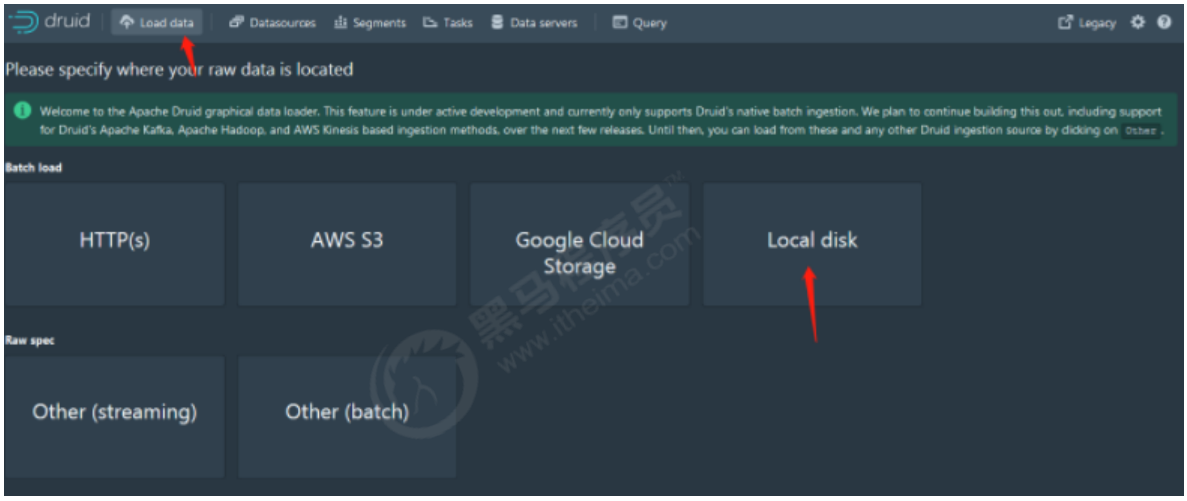
1.6 数据导入

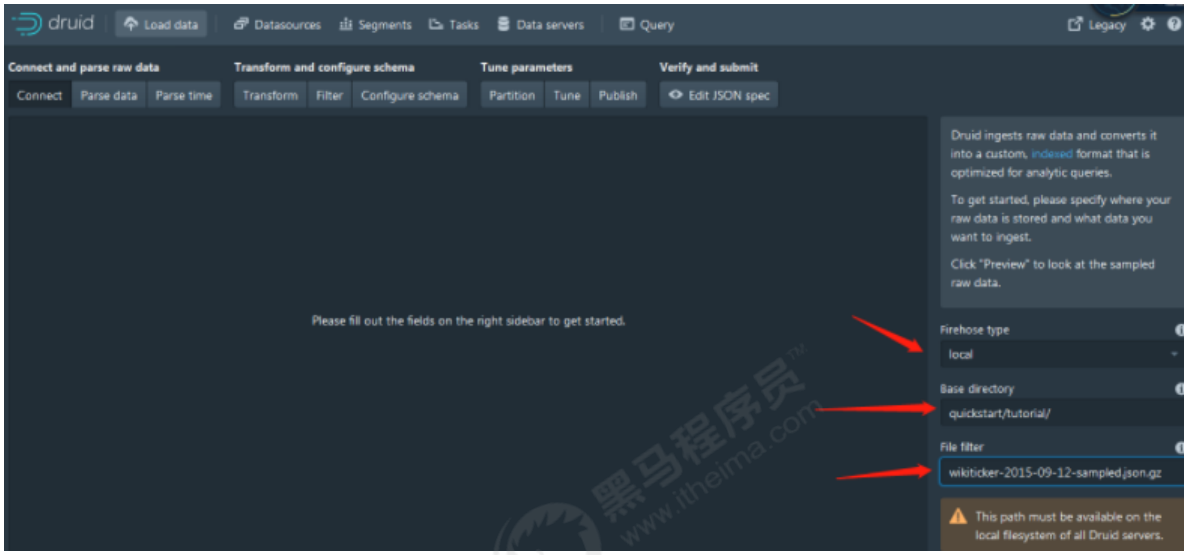
Druid是OLAP数据分析的数据库， 可以通过多种方式导入数据：

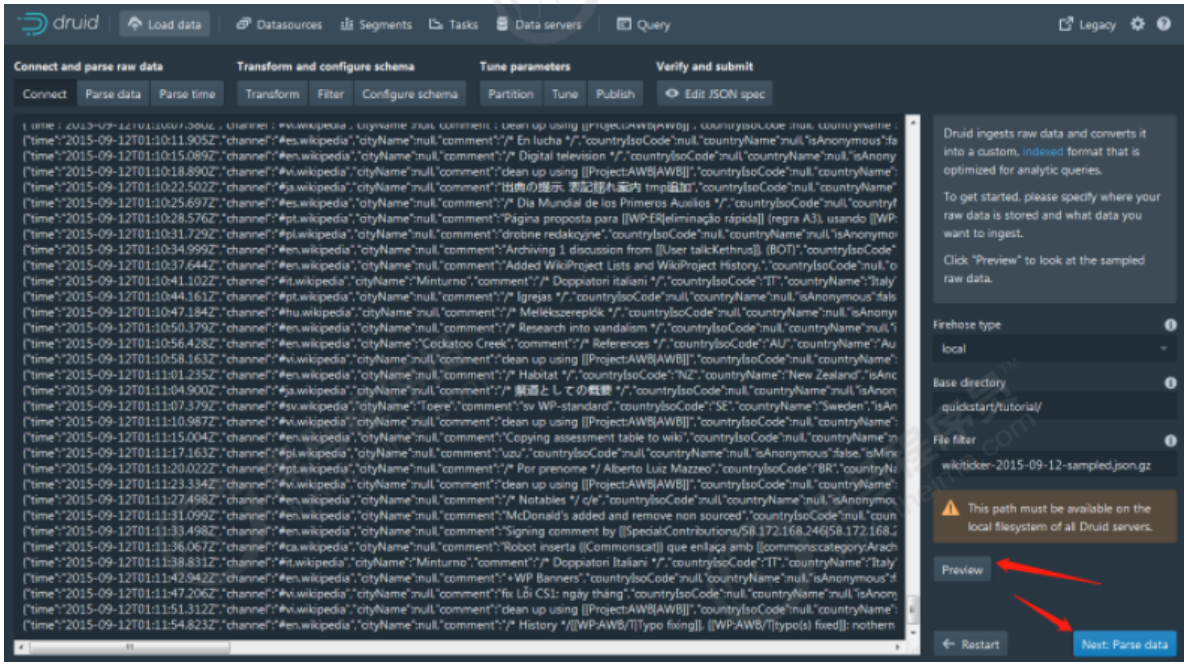
- 离线数据导入
- 实时数据导入

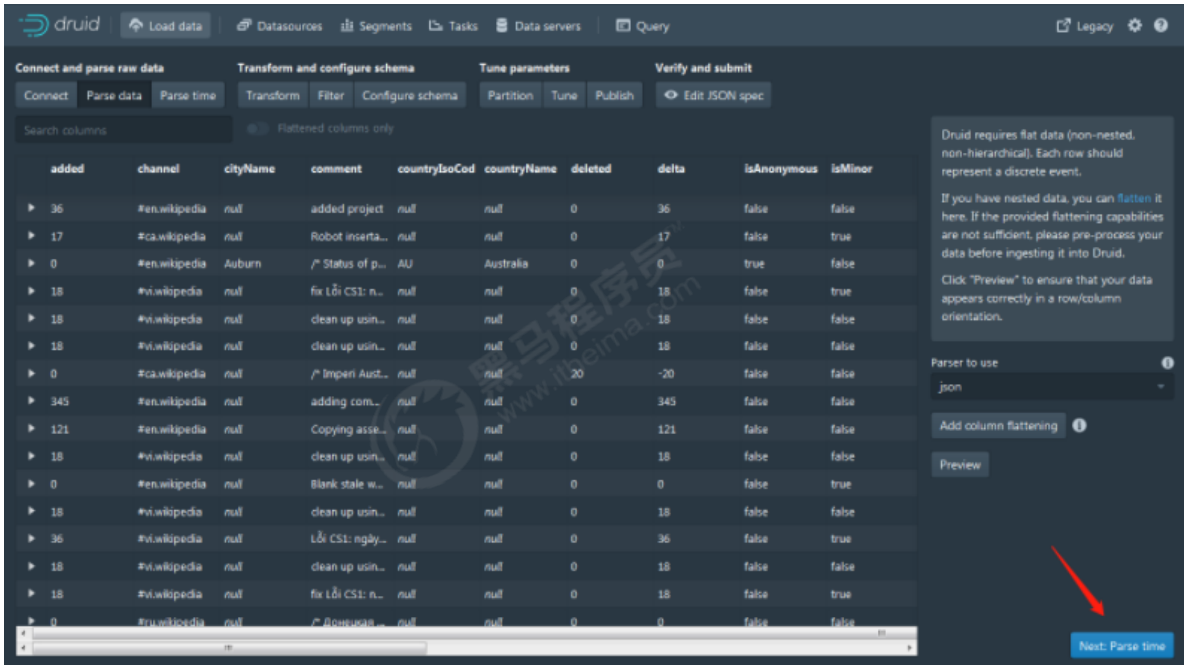
3.6.1 离线导入数据

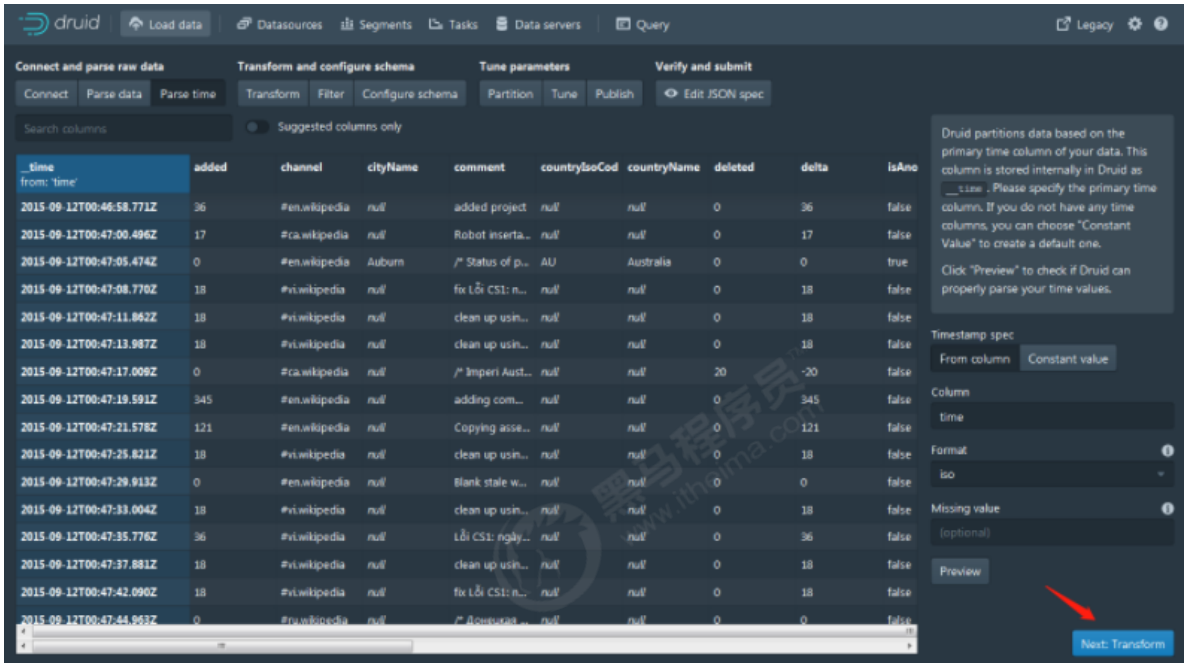
从Druid 0.15.0版本，前端应用系统提供了导入数据的功能，目前支持的功能包括：HTTP（s）、AWS S3、Google Cloud Storage和Local disk，本示例演示如何将本地数据文件导入到Druid中。前端应用系统后续将支持Kafka、Hadoop、AWS Kinesis等，以便更方便的接入Druid数据源。

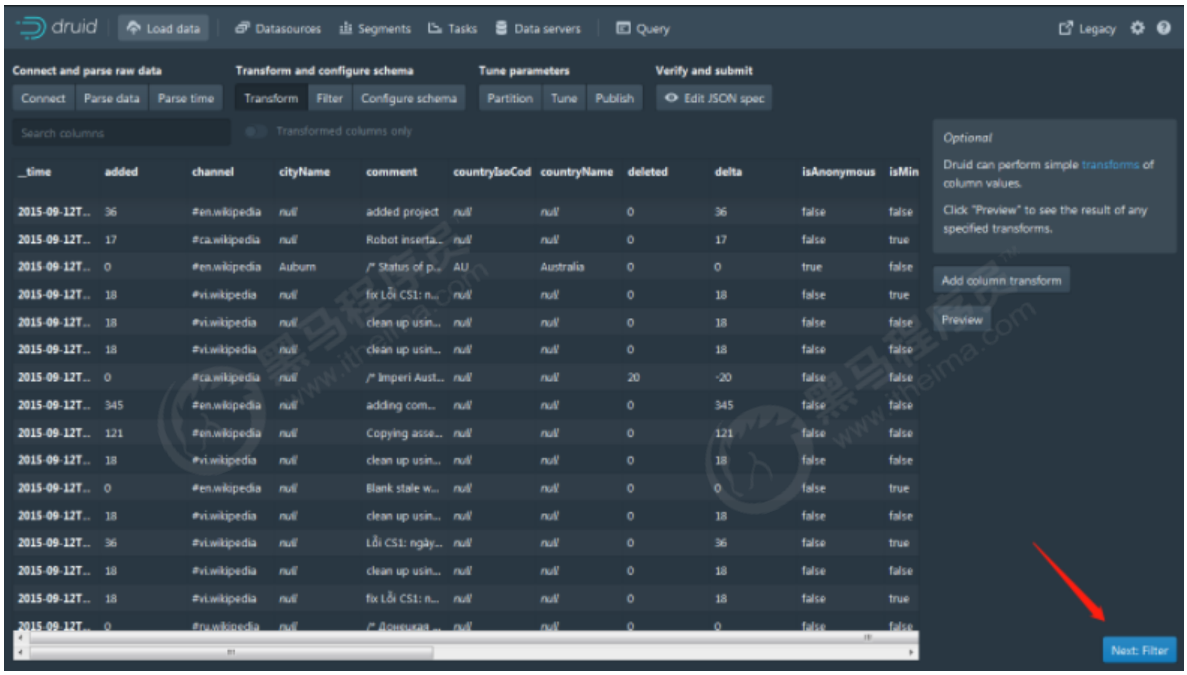


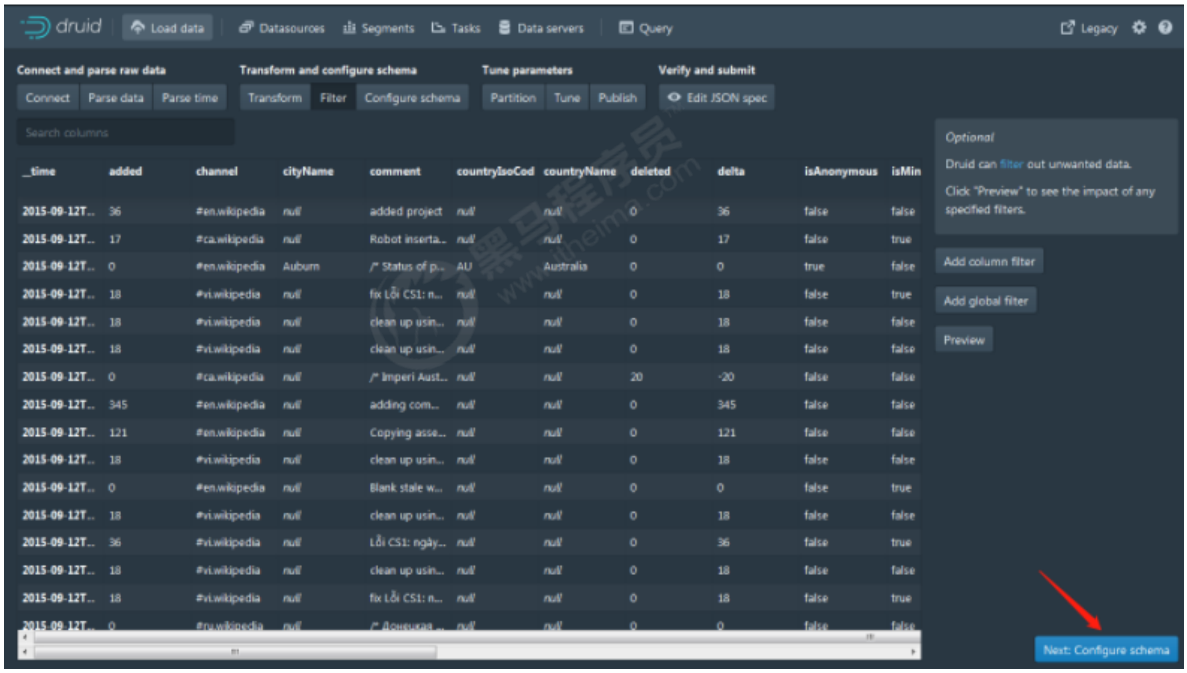

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Transform and configure schema
Tune parameters
Verify and submit
Connect
Parse data
Parse time
Transform
Filter
Configure schema
Partition
Tune
Publish
Edit JSON spec
Please fill out the fields on the right sidebar to get started.
Druid ingests raw data and converts it into a custom, indexed format that is optimized for analytic queries.
To get started, please specify where your raw data is stored and what data you want to ingest.
Click "Preview" to look at the sampled raw data.
Firehose type
local
Base directory
quickstart/tutorial/
File filter
wikiticker-2015-09-12-sampled.json.gz
This path must be available on the local filesystem of all Druid servers.

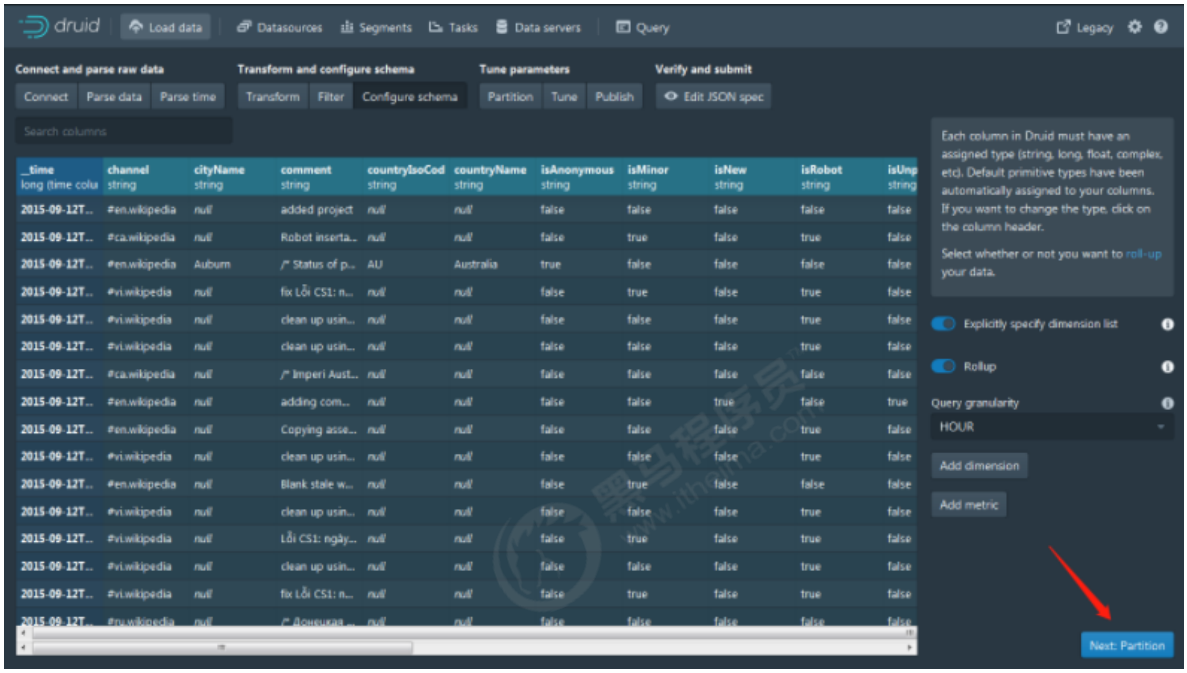

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Transform and configure schema
Tune parameters
Verify and submit
Connect
Parse data
Parse time
Transform
Filter
Configure schema
Partition
Tune
Publish
Edit JSON spec
Druid ingests raw data and converts it into a custom, indexed format that is optimized for analytic queries.
To get started, please specify where your raw data is stored and what data you want to ingest.
Click "Preview" to look at the sampled raw data.
Firehose type
local
Base directory
quickstart/tutorial/
File filter
wikiticker-2015-09-12-sampled.json.gz
This path must be available on the local filesystem of all Druid servers.
Preview
Restart
Next: Parse data

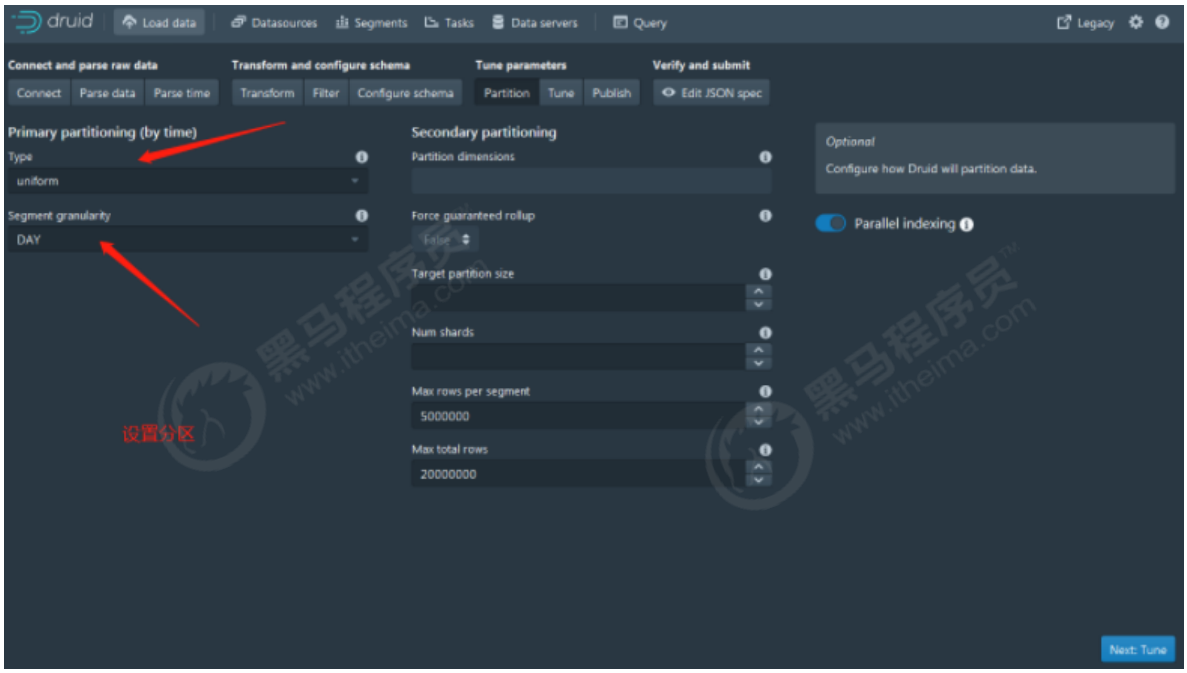

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Transform and configure schema
Tune parameters
Verify and submit
Connect
Parse data
Parse time
Transform
Filter
Configure schema
Partition
Tune
Publish
Edit JSON spec
Search columns
Flattened columns only
added
channel
cityName
comment
countryIsoCod
countryName
deleted
delta
isAnonymous
isMinor
Druid requires flat data (non-nested, non-hierarchical). Each row should represent a discrete event.
If you have nested data, you can flatten it here. If the provided flattening capabilities are not sufficient, please pre-process your data before ingesting it into Druid.
Click "Preview" to ensure that your data appears correctly in a row/column orientation.
Parser to use
json
Add column flattening
Preview
Next: Parse time

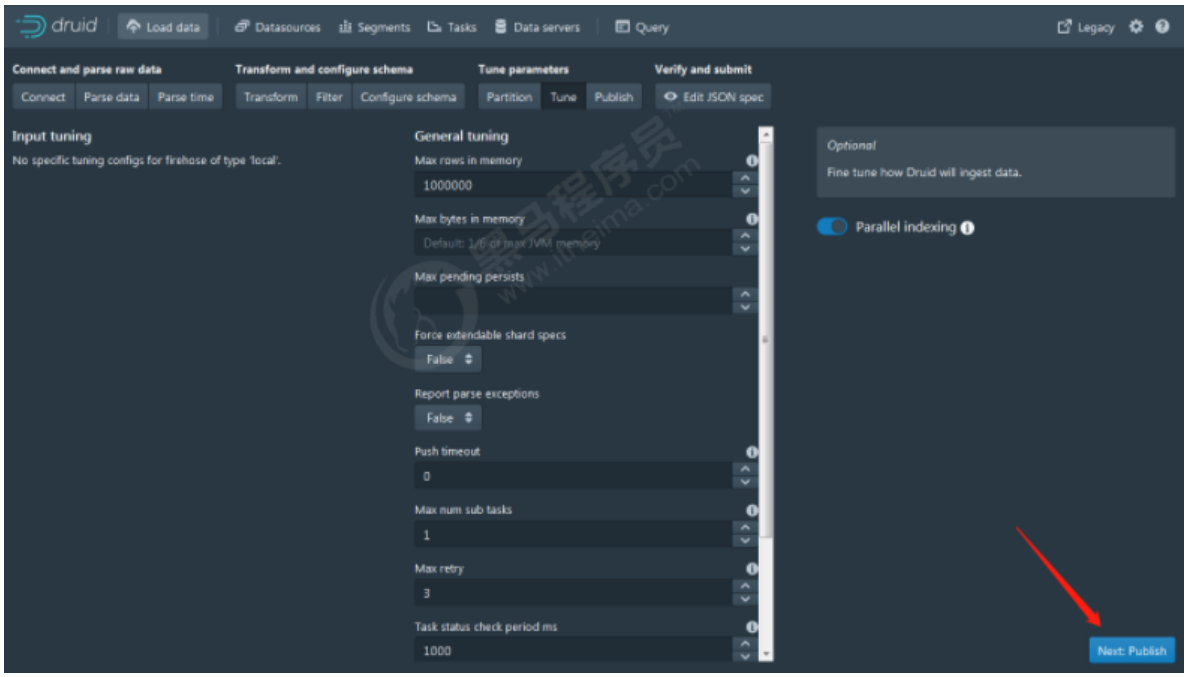

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Connect
Parse data
Parse time
Transform and configure schema
Transform
Filter
Configure schema
Tune parameters
Partition
Tune
Publish
Verify and submit
Edit JSON spec
Search columns
Suggested columns only
_time
from: 'time'
added
channel
cityName
comment
countryIsoCod
countryName
deleted
delta
isAno
2015-09-12T00:46:58.771Z 36 #en.wikipedia null added project null null 0 36 false
2015-09-12T00:47:00.496Z 17 #ca.wikipedia null Robot inserta... null null 0 17 false
2015-09-12T00:47:05.474Z 0 #en.wikipedia Auburn /* Status of p... AU Australia 0 0 true
2015-09-12T00:47:08.770Z 18 #vi.wikipedia null fix Lỗi CS1: n... null null 0 18 false
2015-09-12T00:47:11.862Z 18 #vi.wikipedia null clean up usin... null null 0 18 false
2015-09-12T00:47:13.987Z 18 #vi.wikipedia null clean up usin... null null 0 18 false
2015-09-12T00:47:17.009Z 0 #ca.wikipedia null /* Imperi Aust... null null 20 -20 false
2015-09-12T00:47:19.591Z 345 #en.wikipedia null adding com... null null 0 345 false
2015-09-12T00:47:21.578Z 121 #en.wikipedia null Copying asse... null null 0 121 false
2015-09-12T00:47:25.821Z 18 #vi.wikipedia null clean up usin... null null 0 18 false
2015-09-12T00:47:29.913Z 0 #en.wikipedia null Blank stale w... null null 0 0 false
2015-09-12T00:47:33.004Z 18 #vi.wikipedia null clean up usin... null null 0 18 false
2015-09-12T00:47:35.776Z 36 #vi.wikipedia null Lỗi CS1: ngày... null null 0 36 false
2015-09-12T00:47:37.881Z 18 #vi.wikipedia null clean up usin... null null 0 18 false
2015-09-12T00:47:42.090Z 18 #vi.wikipedia null fix Lỗi CS1: n... null null 0 18 false
Druid partitions data based on the primary time column of your data. This column is stored internally in Druid as __time . Please specify the primary time column. If you do not have any time columns, you can choose "Constant Value" to create a default one.
Click "Preview" to check if Druid can properly parse your time values.
Timestamp spec
From column
Constant value
Column
time
Format
iso
Missing value
(optional)
Preview
Next: Transform

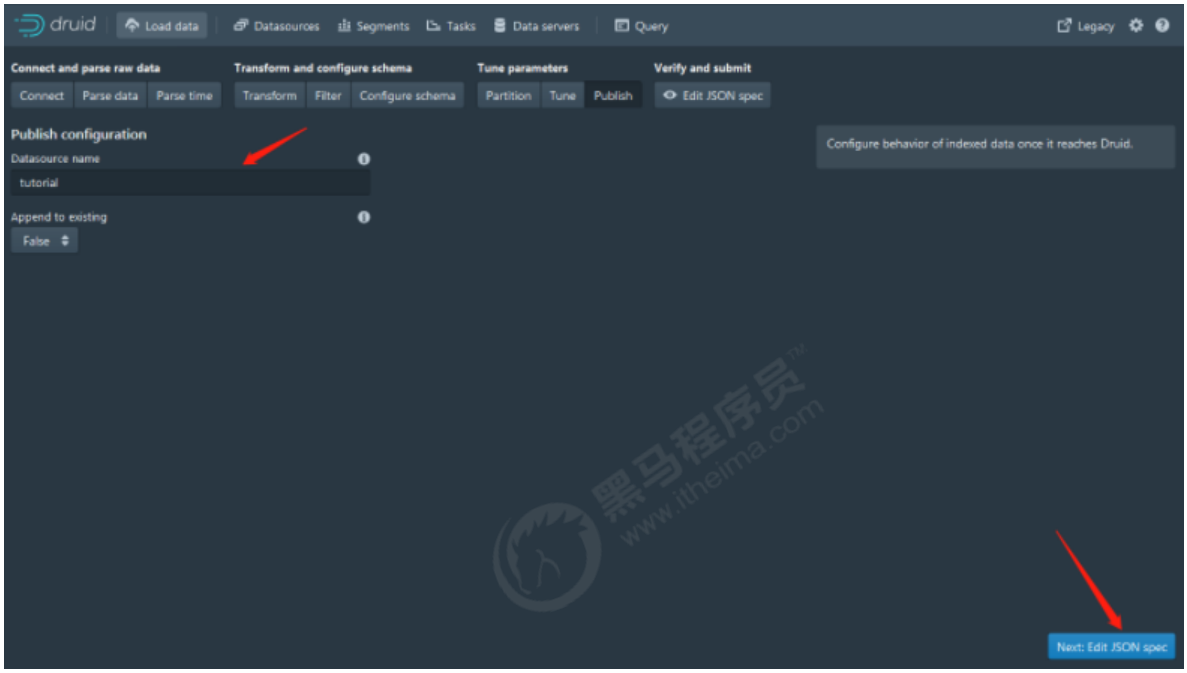

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Connect
Parse data
Parse time
Transform and configure schema
Transform
Filter
Configure schema
Tune parameters
Partition
Tune
Publish
Verify and submit
Edit JSON spec
Search columns
Transformed columns only
_time
added
channel
cityName
comment
countryIsoCod
countryName
deleted
delta
isAnonymous
isMin
2015-09-12T... 36 #en.wikipedia null added project null null 0 36 false false
2015-09-12T... 17 #ca.wikipedia null Robot inserta... null null 0 17 false true
2015-09-12T... 0 #en.wikipedia Auburn /* Status of p... AU Australia 0 0 true false
2015-09-12T... 18 #vi.wikipedia null fix Lỗi CS1: n... null null 0 18 false true
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 0 #ca.wikipedia null /* Imperi Aust... null null 20 -20 false false
2015-09-12T... 345 #en.wikipedia null adding com... null null 0 345 false false
2015-09-12T... 121 #en.wikipedia null Copying asse... null null 0 121 false false
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 0 #en.wikipedia null Blank stale w... null null 0 0 false true
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 36 #vi.wikipedia null Lỗi CS1: ngày... null null 0 36 false true
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 18 #vi.wikipedia null fix Lỗi CS1: n... null null 0 18 false true
Optional
Druid can perform simple transforms of column values.
Click "Preview" to see the result of any specified transforms.
Add column transform
Preview
Next: Filter

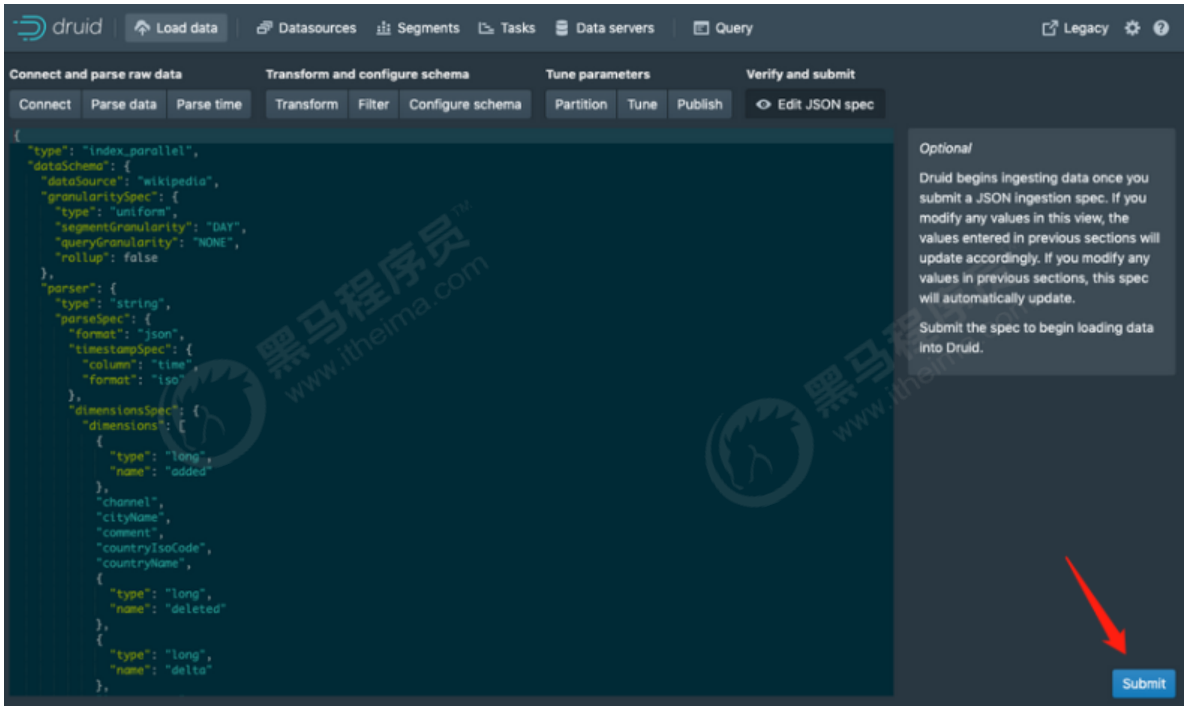

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Connect
Parse data
Parse time
Transform and configure schema
Transform
Filter
Configure schema
Tune parameters
Partition
Tune
Publish
Verify and submit
Edit JSON spec
Search columns
_time
added
channel
cityName
comment
countryIsoCod
countryName
deleted
delta
isAnonymous
isMin
2015-09-12T... 36 #en.wikipedia null added project null null 0 36 false false
2015-09-12T... 17 #ca.wikipedia null Robot inserta... null null 0 17 false true
2015-09-12T... 0 #en.wikipedia Auburn /* Status of p... AU Australia 0 0 true false
2015-09-12T... 18 #vi.wikipedia null fix Lỗi CS1: n... null null 0 18 false true
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 0 #ca.wikipedia null /* Imperi Aust... null null 20 -20 false false
2015-09-12T... 345 #en.wikipedia null adding com... null null 0 345 false false
2015-09-12T... 121 #en.wikipedia null Copying asse... null null 0 121 false false
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 0 #en.wikipedia null Blank stale w... null null 0 0 false true
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 36 #vi.wikipedia null Lỗi CS1: ngày... null null 0 36 false true
2015-09-12T... 18 #vi.wikipedia null clean up usin... null null 0 18 false false
2015-09-12T... 18 #vi.wikipedia null fix Lỗi CS1: n... null null 0 18 false true
Optional
Druid can filter out unwanted data.
Click "Preview" to see the impact of any specified filters.
Add column filter
Add global filter
Preview
Next: Configure schema

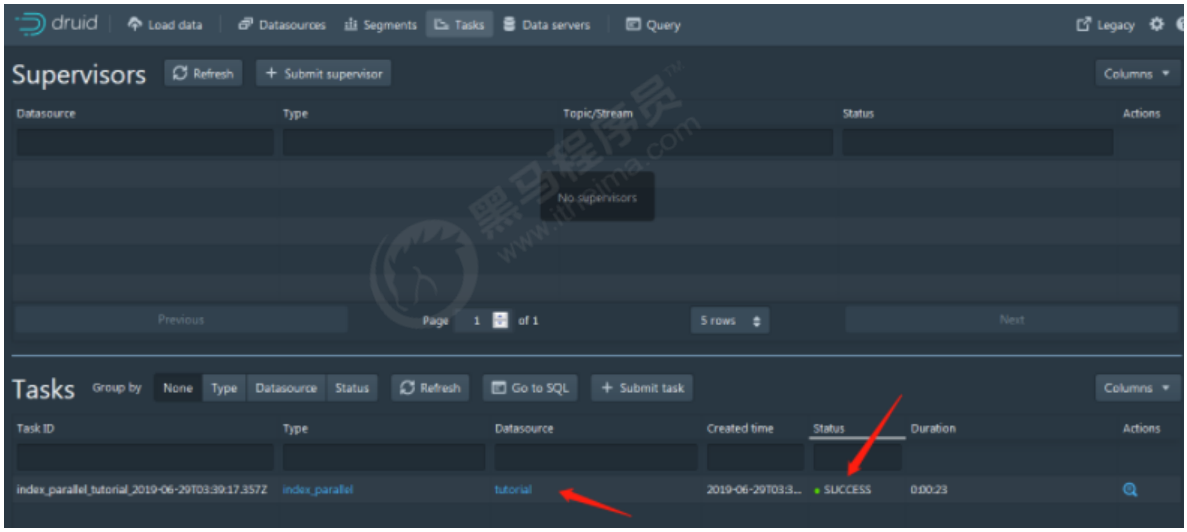

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Connect
Parse data
Parse time
Transform and configure schema
Transform
Filter
Configure schema
Tune parameters
Partition
Tune
Publish
Verify and submit
Edit JSON spec
Search columns
_time
channel
cityName
comment
countryIsoCod
countryName
isAnonymous
isMinor
isNew
isRobot
Each column in Druid must have an assigned type (string, long, float, complex, etc). Default primitive types have been automatically assigned to your columns. If you want to change the type, click on the column header.
Select whether or not you want to roll-up your data.
Explicitly specify dimension list
Rollup
Query granularity
HOUR
Add dimension
Add metric
Next: Partition

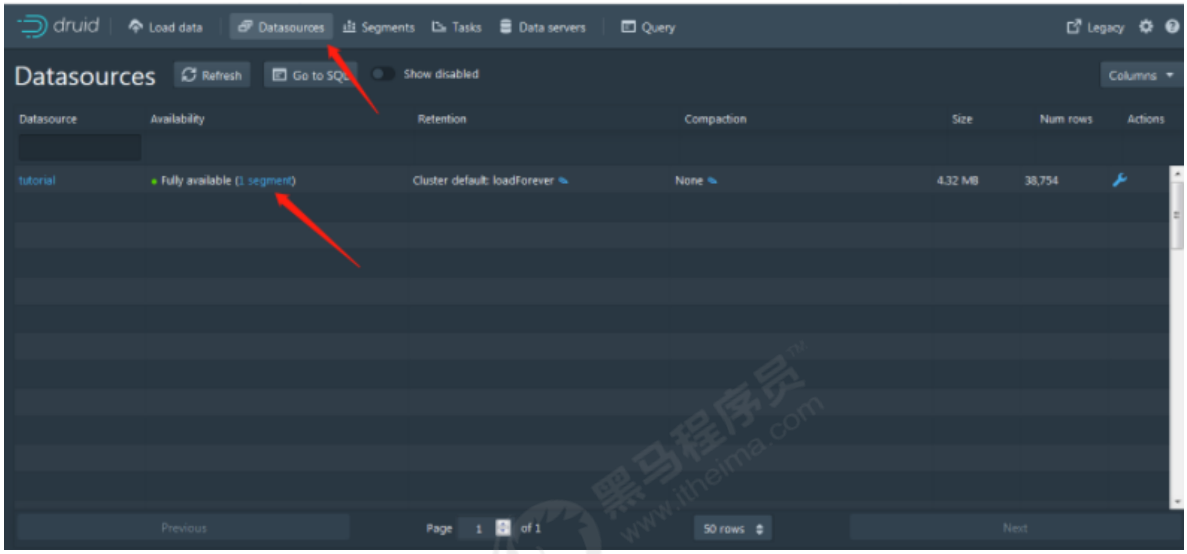

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Connect
Parse data
Parse time
Transform and configure schema
Transform
Filter
Configure schema
Tune parameters
Partition
Tune
Publish
Verify and submit
Edit JSON spec
Primary partitioning (by time)
Type
uniform
Segment granularity
DAY
Secondary partitioning
Partition dimensions
Force guaranteed rollup
False
Target partition size
Num shards
Max rows per segment
5000000
Max total rows
20000000
Optional
Configure how Druid will partition data.
Parallel indexing
设置分区
Next: Tune

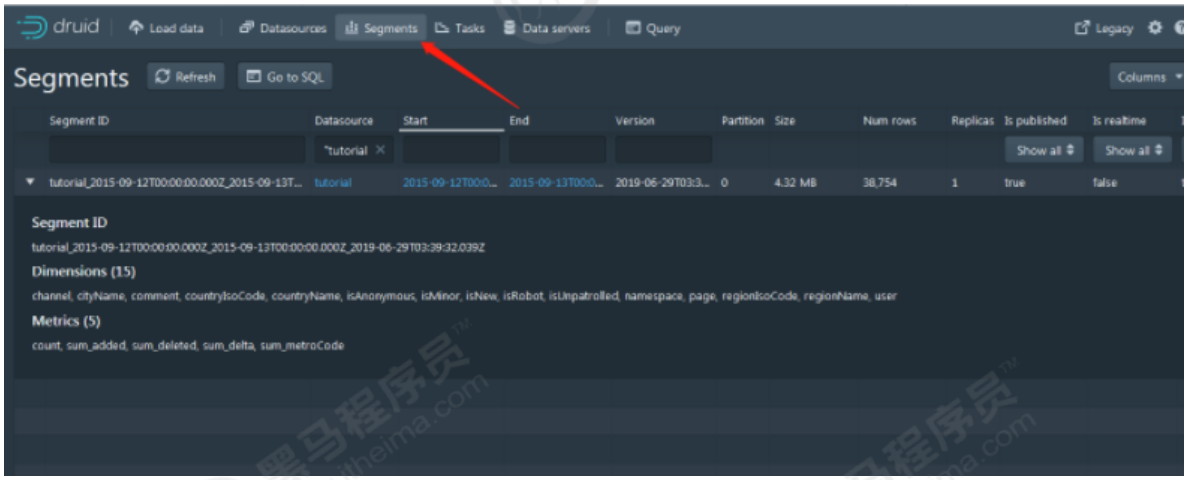

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Connect
Parse data
Parse time
Transform and configure schema
Transform
Filter
Configure schema
Tune parameters
Partition
Tune
Publish
Verify and submit
Edit JSON spec
Input tuning
No specific tuning configs for firehose of type 'local'.
General tuning
Max rows in memory
1000000
Max bytes in memory
Default: 1/6 of max JVM memory
Max pending persists
Force extendable shard specs
False
Report parse exceptions
False
Push timeout
0
Max num sub tasks
1
Max retry
3
Task status check period ms
1000
Optional
Fine tune how Druid will ingest data.
Parallel indexing
Next: Publish


druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Connect
Parse data
Parse time
Transform and configure schema
Transform
Filter
Configure schema
Tune parameters
Partition
Tune
Publish
Verify and submit
Edit JSON spec
Publish configuration
Datasource name
tutorial
Append to existing
False
Configure behavior of indexed data once it reaches Druid.
Next: Edit JSON spec


druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Connect and parse raw data
Connect
Parse data
Parse time
Transform and configure schema
Transform
Filter
Configure schema
Tune parameters
Partition
Tune
Publish
Verify and submit
Edit JSON spec
{
"type": "index_parallel",
"dataSchema": {
"dataSource": "wikipedia",
"granularitySpec": {
"type": "uniform",
"segmentGranularity": "DAY",
"queryGranularity": "NONE",
"rollup": false
},
"parser": {
"type": "string",
"parseSpec": {
"format": "json",
"timestampSpec": {
"column": "time",
"format": "iso"
},
"dimensionsSpec": {
"dimensions": [
{
"type": "long",
"name": "added"
},
"channel",
"cityName",
"comment",
"countryIsoCode",
"countryName",
{
"type": "long",
"name": "deleted"
},
{
"type": "long",
"name": "delta"
},
Optional
Druid begins ingesting data once you submit a JSON ingestion spec. If you modify any values in this view, the values entered in previous sections will update accordingly. If you modify any values in previous sections, this spec will automatically update.
Submit the spec to begin loading data into Druid.
Submit

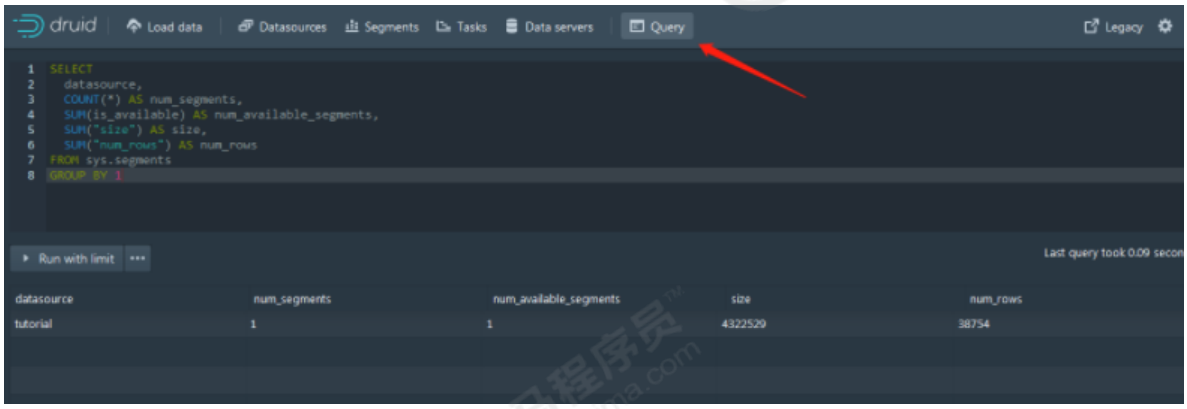

druid
Load data
Datasources
Segments
Tasks
Data servers
Query
Legacy
Supervisors
Refresh
Submit supervisor
Columns
Datasource
Type
Topic/Stream
Status
Actions
No supervisors
Previous
Page 1 of 1
5 rows
Next
Tasks
Group by
None
Type
Datasource
Status
Refresh
Go to SQL
Submit task
Columns
Task ID
Type
Datasource
Created time
Status
Duration
Actions
index_parallel_tutorial_2019-06-29T03:39:17.357Z
index_parallel
tutorial
2019-06-29T03:3...
SUCCESS
0:00:23





至此，本地数据文件已经导入到druid中， Datasources（类似于mysql的表）中有了一条记录。

可以通过Query功能查询到此数据源



根据配置，数据深度存储在HDFS中：

```
[root@node2-vm06 _common]# hadoop fs -ls /druid/segments/tutorial
Found 1 items
drwxr-xr-x   - root supergroup          0 2019-06-29 14:21
/druid/segments/tutorial/20150912T000000.000Z_20150913T000000.000Z
```

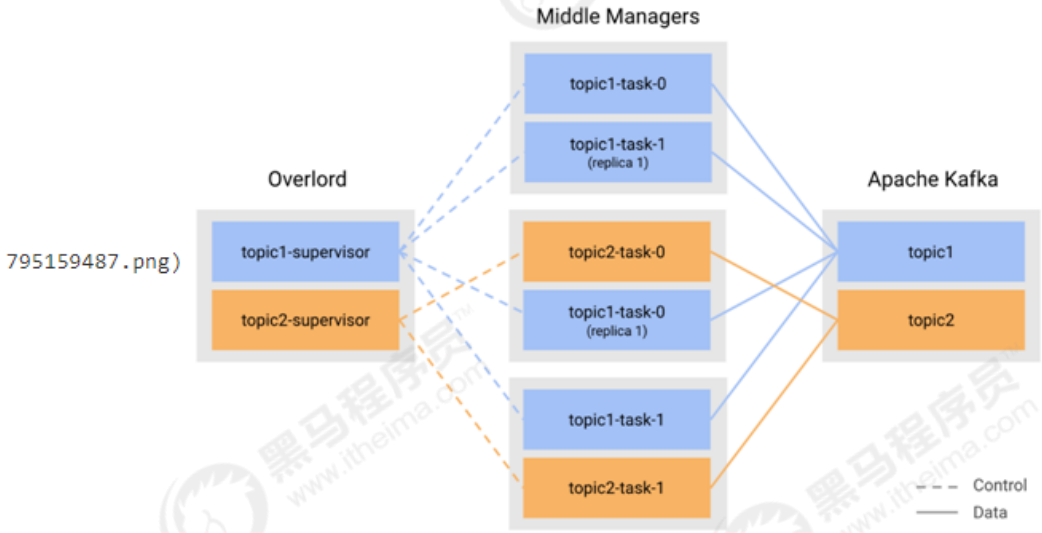
3.6.2 实时导入

通过Druid Kafka 索引服务，可以方便的将kafka队列中的流数据实时导入到Druid中。

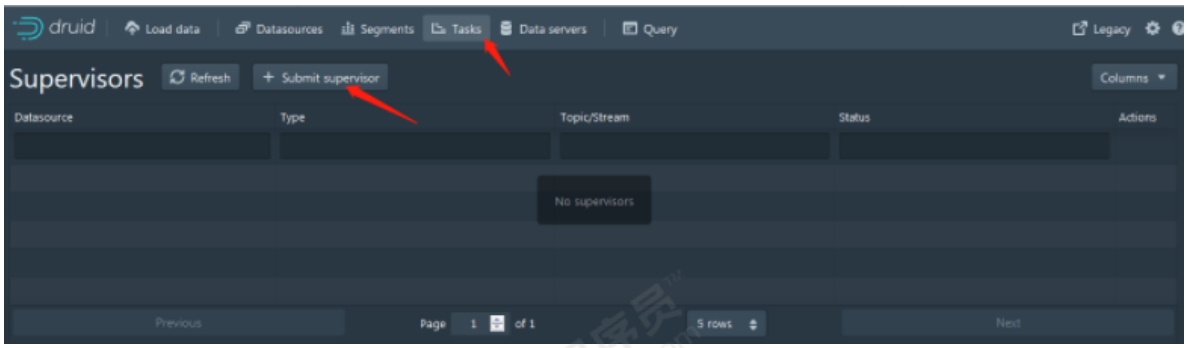
```
# 创建kafka队列
[root@node2-vm06 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 1 --partitions 1 --topic wikipedia

# 查询现有队列
[root@node2-vm06 kafka_2.12-2.3.0]# ./bin/kafka-topics.sh --zookeeper
172.17.0.143:2181 --list
mykafka
wikipedia
```

Kafka Supervisor作为Kafka indexing service的创建者和监督者，运行在Overlord中，管理Kafka中某个topic对应的Druid中所有Kafka indexing service tasks生命周期。在生产环境中，我们通过构造Kafka Supervisor对应的spec文件，以JSON-OVER-HTTP的方式发送给Overlord节点，Overlord启动Kafka Supervisor，监控对应的kafka indexing service tasks。



在管理台， 点击【Submit supervisor】来打开提交supervisor的窗口：

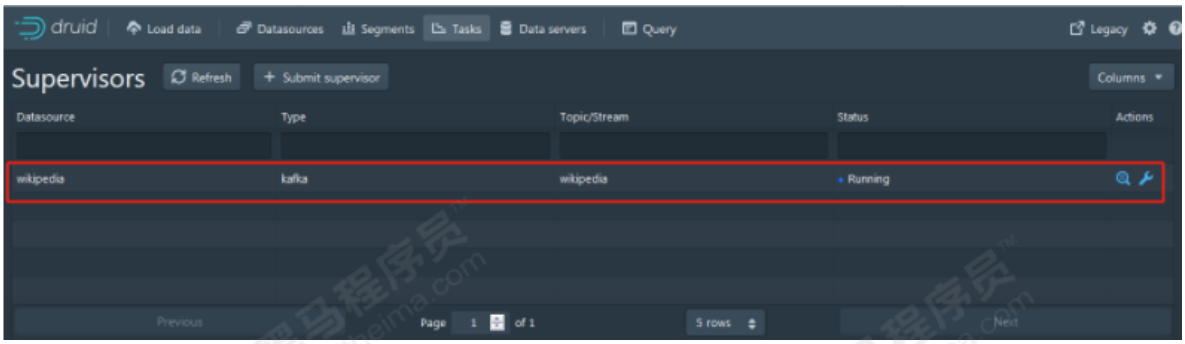


提交以下spec：

```
{
  "type": "kafka",
  "dataSchema": {
    "dataSource": "wikipedia",
    "parser": {
      "type": "string",
      "parseSpec": {
        "format": "json",
        "timestampSpec": {
          "column": "time",
```

```
          "format": "auto"
        },
        "dimensionsSpec": {
          "dimensions": [
            "channel",
            "cityName",
            "comment",
            "countryIsoCode",
            "countryName",
            "isAnonymous",
            "isMinor",
            "isNew",
            "isRobot",
            "isUnpatrolled",
            "metroCode",
            "namespace",
            "page",
            "regionIsoCode",
            "regionName",
            "user",
            { "name": "added", "type": "long" },
            { "name": "deleted", "type": "long" },
            { "name": "delta", "type": "long" }
          ]
        }
      }
    },
    "metricsSpec" : [],
    "granularitySpec": {
      "type": "uniform",
      "segmentGranularity": "DAY",
      "queryGranularity": "NONE",
      "rollup": false
    }
  },
  "tuningConfig": {
    "type": "kafka",
    "reportParseExceptions": false
  },
  "ioConfig": {
    "topic": "wikipedia",
    "replicas": 2,
    "taskDuration": "PT10M",
    "completionTimeout": "PT20M",
    "consumerProperties": {
      "bootstrap.servers": "172.17.0.143:9092"
    }
  }
}
```

supervisor将会创建侦听任务，监听传入数据的任务：



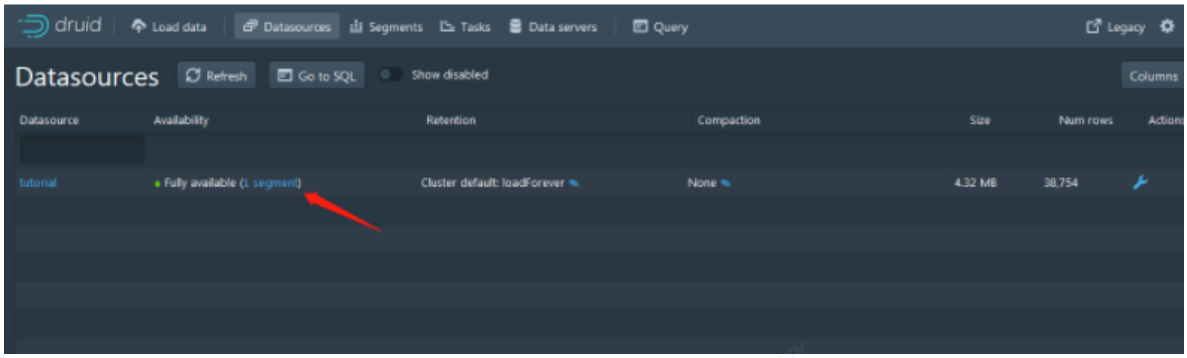
下面我们通过消息生产者程序，向kafka队列中发送数据，为了便于观察，我们将现有数据清空。

```
# 解压示例数据
[root@node2-vm06 tutorial]# cd /opt/druid/
[root@node2-vm06 druid]# cd quickstart/tutorial/
[root@node2-vm06 tutorial]# gunzip  wikiticker-2015-09-12-sampled.json.gz
[root@node2-vm06 tutorial]# ll | grep wikiticker-2015-09-12-sampled
-rw-r--r--. 1 501 games 17106256 Jun 13 06:30 wikiticker-2015-09-12-sampled.json

# 执行生产者程序，将数据发送至kafka队列
export KAFKA_OPTS="-Dfile.encoding=UTF-8"
[root@node2-vm06 ~]# cd /opt
[root@node2-vm06 opt]# /opt/software/kafka_2.12-2.3.0/bin/kafka-console-
producer.sh --broker-list 172.17.0.143:9092 --topic wikipedia

# 查看kafka队列中的消息
[root@node2-vm06 opt]# /opt/software/kafka_2.12-2.3.0/bin/kafka-console-
consumer.sh --bootstrap-server 172.17.0.143:9092 --topic wikipedia --from-
beginning
```

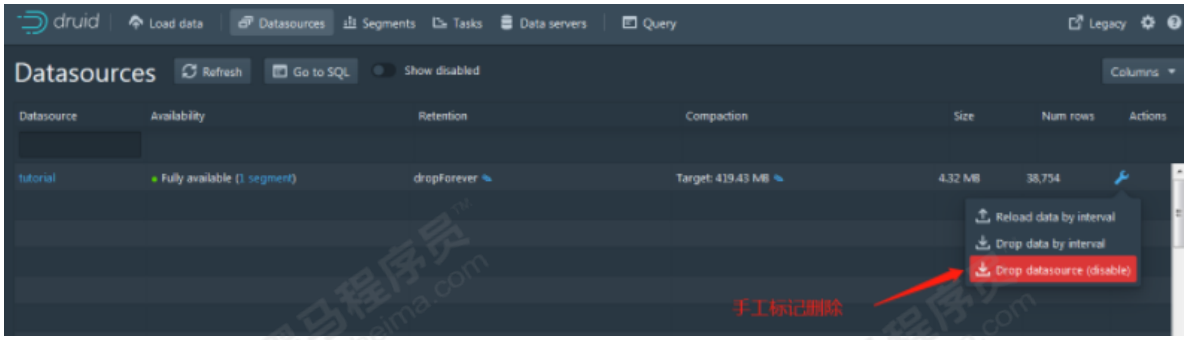
数据已经实时导入到druid中：



3.6.3 删除数据

永久性删除Druid中的segment数据需要两个步骤：

1. segment必须首先被标记为“未使用”。系统规则或用户执行API命令（或页面操作），都能够把segment标记为”未使用“。
2. segment被标记为“未使用”后，执行kill任务将从druid的元数据存储和深度存储中删除任何“未使用”的segment。



1.7 SQL查询

1.7.1 查询格式

1.7.1.1 JSON查询

```
{
  "queryType" : "topN",
  "dataSource" : "wikipedia",
  "intervals" : ["2015-09-12/2015-09-13"],
  "granularity" : "all",
  "dimension" : "page",
  "metric" : "count",
  "threshold" : 10,
  "aggregations" : [
    {
      "type" : "count",
      "name" : "count"
    }
  ]
}
```

查询示例：

```
[root@node2-vm06 tutorial]# cd ~
[root@node2-vm06 ~]# cd /opt/druid/quickstart/tutorial
[root@node2-vm06 tutorial]# curl -X 'POST' -H 'Content-Type:application/json' -d
@wikipedia-top-pages.json http://localhost:8082/druid/v2?pretty
```

查询结果：

```
[ {
  "timestamp" : "2015-09-12T00:00:00.000Z",
  "result" : [ {
    "count" : 26,
    "page" : "Jeremy Corbyn"
  }, {
    "count" : 24,
    "page" : "Wikipedia:Vandalismusmeldung"
  }, {
    "count" : 21,
    "page" : "User:Cyde/List of candidates for speedy deletion/Subpage"
  }, {
    "count" : 18,
    "page" : "Flavia Pennetta"
  }, {
    "count" : 16,
    "page" : "Wikipedia:Administrators' noticeboard/Incidents"
  }, {
    "count" : 14,
    "page" : "Anthony Martial"
  }, {
    "count" : 11,
    "page" : "Kim Davis (county clerk)"
  }, {
    "count" : 11,
    "page" : "Wikipedia:Files for deletion/2015 September 12"
  }, {
    "count" : 11,
    "page" : "Wikipedia:In the news/Candidates"
  }, {
    "count" : 10,
    "page" : "Wikipedia:Requests for page protection"
  } ]
} ]
```

1.7.1.2 SQL查询

```
{
  "query":"SELECT page, COUNT(*) AS Edits FROM wikipedia WHERE \"__time\"
BETWEEN TIMESTAMP '2015-09-12 00:00:00' AND TIMESTAMP '2015-09-13 00:00:00'
GROUP BY page ORDER BY Edits DESC LIMIT 10"
}
```

查询示例：

```
[root@node2-vm06 tutorial]# cd ~
[root@node2-vm06 ~]# cd /opt/druid/quickstart/tutorial
[root@node2-vm06 tutorial]# curl -X 'POST' -H 'Content-Type:application/json' -d
@wikipedia-top-pages-sql.json http://localhost:8082/druid/v2/sql
```

返回结果：

```
[
    {"page":"Jeremy Corbyn","Edits":26},
    {"page":"Wikipedia:Vandalismusmeldung","Edits":24},
    {"page":"User:Cyde/List of candidates for speedy
deletion/Subpage","Edits":21},
    {"page":"Flavia Pennetta","Edits":18},
    {"page":"Wikipedia:Administrators' noticeboard/Incidents","Edits":16},
    {"page":"Anthony Martial","Edits":14},
    {"page":"Kim Davis (county clerk)","Edits":11},
    {"page":"Wikipedia:Files for deletion/2015 September 12","Edits":11},
    {"page":"Wikipedia:In the news/Candidates","Edits":11},
    {"page":"Wikipedia:Requests for page protection","Edits":10}
]
```

1.7.1.3 SQL客户端

Druid中包含一个SQL命令行客户端，位于Druid包根目录的bin/dsql中：

```
[root@node2-vm06 ~]# cd /opt/druid/
[root@node2-vm06 druid]# bin/dsql
Welcome to dsql, the command-line client for Druid SQL.
Connected to [http://localhost:8082/].

Type "\h" for help.
dsql>
```

示例1：

```
dsql> select * from wikipedia limit 1;
┌──────────────────────────┬─────────┬──────────┬──────────────────────────
──────────┬──────┬─────────────┬─────────────┬─────────────┬─────────┬──────
──┬─────────┬─────────────┬──────────┬──────────────────────┬─────────────┬
──────────┬────────────┬─────────────┬─────────────┬──────────────┬─────────
──────┐
| __time                   | channel       | cityName | comment
           | count | countryIsoCode | countryName | isAnonymous | isMinor |
isNew | isRobot | isUnpatrolled | namespace | page          | regionIsoCode |
regionName | sum_added | sum_deleted | sum_delta | sum_metroCode | user    |
├──────────────────────────┼─────────┼──────────┼──────────────────────────
──────────┼──────┼─────────────┼─────────────┼─────────────┼─────────┼──────
──┼─────────┼─────────────┼──────────┼──────────────────────┼─────────────┼
──────────┼────────────┼─────────────┼─────────────┼──────────────┼─────────
──────┤
| 2015-09-12T00:00:00.000Z | #ar.wikipedia |          | /* أكلات تم تطويرها في
1     | /* نيوزيلندا |             |          | false       | false   |
false | false   | false         | Main      |   |              | مطبخ نيوزلندي
0             | 2-        | 2           | 0         |          | Makki98 |
└──────────────────────────┴─────────┴──────────┴──────────────────────────
──────────┴──────┴─────────────┴─────────────┴─────────────┴─────────┴──────
──┴─────────┴─────────────┴──────────┴──────────────────────┴─────────────┴
──────────┴────────────┴─────────────┴─────────────┴──────────────┴─────────
──────┘
Retrieved 1 row in 0.13s.
```

示例2：

```
dsql> SELECT page, COUNT(*) AS Edits FROM wikipedia WHERE "__time" BETWEEN
TIMESTAMP '2015-09-12 00:00:00' AND TIMESTAMP '2015-09-13 00:00:00' GROUP BY
page ORDER BY Edits DESC LIMIT 10;
┌──────────────────────────────────────────────────────────┬────────┐
| page                                                     | Edits |
├──────────────────────────────────────────────────────────┼────────┤
| Jeremy Corbyn                                            |    26 |
| Wikipedia:Vandalismusmeldung                             |    24 |
| User:Cyde/List of candidates for speedy deletion/Subpage |    21 |
| Flavia Pennetta                                          |    18 |
| Wikipedia:Administrators' noticeboard/Incidents          |    16 |
| Anthony Martial                                          |    14 |
| Kim Davis (county clerk)                                 |    11 |
| Wikipedia:Files for deletion/2015 September 12           |    11 |
| Wikipedia:In the news/Candidates                         |    11 |
| Wikipedia:Requests for page protection                   |    10 |
└──────────────────────────────────────────────────────────┴────────┘
Retrieved 10 rows in 0.31s.

dsql>
```

示例3：

```
dsql> SELECT FLOOR(__time to HOUR) AS HourTime, SUM(sum_deleted) AS LinesDeleted
FROM wikipedia WHERE "__time" BETWEEN TIMESTAMP '2015-09-12 00:00:00' AND
TIMESTAMP '2015-09-13 00:00:00' GROUP BY FLOOR(__time to HOUR);
┌──────────────────────────┬──────────────┐
| HourTime                 | LinesDeleted |
├──────────────────────────┼──────────────┤
| 2015-09-12T00:00:00.000Z |         1761 |
```

```
| 2015-09-12T01:00:00.000Z |        16208 |
| 2015-09-12T02:00:00.000Z |        14543 |
| 2015-09-12T03:00:00.000Z |        13101 |
| 2015-09-12T04:00:00.000Z |        12040 |
| 2015-09-12T05:00:00.000Z |         6399 |
| 2015-09-12T06:00:00.000Z |         9036 |
| 2015-09-12T07:00:00.000Z |        11409 |
| 2015-09-12T08:00:00.000Z |        11616 |
| 2015-09-12T09:00:00.000Z |        17509 |
| 2015-09-12T10:00:00.000Z |        19406 |
| 2015-09-12T11:00:00.000Z |        16284 |
| 2015-09-12T12:00:00.000Z |        18672 |
| 2015-09-12T13:00:00.000Z |        30520 |
| 2015-09-12T14:00:00.000Z |        18025 |
| 2015-09-12T15:00:00.000Z |        26399 |
| 2015-09-12T16:00:00.000Z |        24759 |
| 2015-09-12T17:00:00.000Z |        19634 |
| 2015-09-12T18:00:00.000Z |        17345 |
| 2015-09-12T19:00:00.000Z |        19305 |
| 2015-09-12T20:00:00.000Z |        22265 |
| 2015-09-12T21:00:00.000Z |        16394 |
| 2015-09-12T22:00:00.000Z |        16379 |
| 2015-09-12T23:00:00.000Z |        15289 |
└──────────────────────────┴──────────────┘
Retrieved 24 rows in 0.60s.
```

1.7.1.4 Avatica JDBC

Apache Calcite是一个动态数据管理框架。它包含了许多组成典型数据管理系统的经典模块，但省略了一些关键功能: 数据存储，数据处理算法和元数据存储库。

Apache Druid使用Calcite数据管理引擎，我们可以使用Calcite的jdbc驱动Avatica。示例代码如下：

```
    String url =
"jdbc:avatica:remote:url=http://172.17.0.143:8082/druid/v2/sql/avatica/";
        Properties properties = new Properties();
        String sql = "SELECT * from wikipedia order by __time desc limit 10";
        try{
            Connection connection = DriverManager.getConnection(url,
properties);
            ResultSet resultSet =
connection.createStatement().executeQuery(sql);

            while (resultSet.next()){
                System.out.println(resultSet.getString("page"));
            }
            connection.close();
        }
        catch (SQLException e){
            e.printStackTrace();
        }

    }
```

1.7.2 查询类型和属性

Apache Druid查询的类型有：

```
1、Timeseries
2、TopN
3、GroupBy
4、Time Boundary
5、Segment Metadata
6、Datasource Metadata
7、Search
8、select
9、Scan
```

Druid查询中主要包含以下几种属性：

```
1.queryType:     查询类型，即timeseries，topN，groupBy等；
2.dataSource:    数据源，类似Mysql中的表的概念；
3.granularity:   聚合粒度，聚合粒度有none,all,week,day,hour等；
4.filter:        过滤条件，类似Mysql中的where条件；
5.aggregator:    聚合方式，类似Mysql中的count，sum等操作
```

1.7.3 聚合粒度(granularity)

简单的聚合粒度有：all、none、second、minute、fifteen_minute、thirty_minute、hour、day、week、month、quarter、year；简单聚合粒度的查询取决于druid存储数据的最小粒度，如果构建数据的最小粒度是小时，使用minute粒度去查询，结果数据也是小时粒度的数据。

假设使用示例数据，提交一个小时粒度的groupBy查询，查询query如下：

```
{
   "queryType":"groupBy",
   "dataSource":"wikipedia",
   "granularity":"hour",
   "dimensions":[
      "language"
   ],
   "aggregations":[
      {
         "type":"count",
         "name":"count"
      }
   ],
   "intervals":[
      "2010-01-01T00:00Z/2020-01-01T00:00Z"
   ]
}
```

按小时粒度进行的groupby查询结果中timestamp值精确到小时，比小时粒度更小粒度值自动补填零，以此类推按天查询，则小时及小粒度补零。timestamp值为UTC。查询结果如下：

```
[
    {
        "version": "v1",
        "timestamp": "2015-09-12T00:00:00.000Z",
        "event": {
            "count": 266,
```

```
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T01:00:00.000Z",
        "event": {
            "count": 1134,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T02:00:00.000Z",
        "event": {
            "count": 1090,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T03:00:00.000Z",
        "event": {
            "count": 800,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T04:00:00.000Z",
        "event": {
            "count": 815,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T05:00:00.000Z",
        "event": {
            "count": 1248,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T06:00:00.000Z",
        "event": {
            "count": 2112,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T07:00:00.000Z",
        "event": {
            "count": 2226,
            "language": null
        }
```

```
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T08:00:00.000Z",
        "event": {
            "count": 1603,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T09:00:00.000Z",
        "event": {
            "count": 1680,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T10:00:00.000Z",
        "event": {
            "count": 1769,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T11:00:00.000Z",
        "event": {
            "count": 1599,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T12:00:00.000Z",
        "event": {
            "count": 1692,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T13:00:00.000Z",
        "event": {
            "count": 2043,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T14:00:00.000Z",
        "event": {
            "count": 1850,
            "language": null
        }
    },
    {
```

```
        "version": "v1",
        "timestamp": "2015-09-12T15:00:00.000Z",
        "event": {
            "count": 1941,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T16:00:00.000Z",
        "event": {
            "count": 1930,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T17:00:00.000Z",
        "event": {
            "count": 2260,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T18:00:00.000Z",
        "event": {
            "count": 2132,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T19:00:00.000Z",
        "event": {
            "count": 1989,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T20:00:00.000Z",
        "event": {
            "count": 1806,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T21:00:00.000Z",
        "event": {
            "count": 1741,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T22:00:00.000Z",
```

```
        "event": {
            "count": 1570,
            "language": null
        }
    },
    {
        "version": "v1",
        "timestamp": "2015-09-12T23:00:00.000Z",
        "event": {
            "count": 1458,
            "language": null
        }
    }
]
```

如若指定聚合粒度为day，则按照天为单位对数据进行聚合，查询结果如下：

```
[ {
  "version" : "v1",
  "timestamp" : "2019-08-31T00:00:00.000Z",
  "event" : {
    "count" : 1,
    "language" : "en"
  }
}, {
  "version" : "v1",
  "timestamp" : "2019-09-01T00:00:00.000Z",
  "event" : {
    "count" : 1,
    "language" : "en"
  }
}, {
  "version" : "v1",
  "timestamp" : "2019-09-02T00:00:00.000Z",
  "event" : {
    "count" : 1,
    "language" : "en"
  }
}, {
  "version" : "v1",
  "timestamp" : "2019-09-03T00:00:00.000Z",
  "event" : {
    "count" : 1,
    "language" : "en"
  }
} ]
```

如若聚合粒度设置为none，则按照druid中build数据的最小粒度查询数据，即不进行聚合，如bulid数据的粒度是ms，则聚合出来的结果也是毫秒：

```
[ {
  "version" : "v1",
  "timestamp" : "2019-08-31T01:02:33.000Z",
  "event" : {
    "count" : 1,
    "language" : "en"
```

```
  }
}, {
  "version" : "v1",
  "timestamp" : "2019-09-01T01:02:33.000Z",
  "event" : {
    "count" : 1,
    "language" : "en"
  }
}, {
  "version" : "v1",
  "timestamp" : "2019-09-02T23:32:45.000Z",
  "event" : {
    "count" : 1,
    "language" : "en"
  }
}, {
  "version" : "v1",
  "timestamp" : "2019-09-03T03:32:45.000Z",
  "event" : {
    "count" : 1,
    "language" : "en"
  }
} ]
```

如若将聚合粒度设置为all，则返回数据的长度为1，即把查询时间段的数据做一个汇总：

```
[ {
  "version" : "v1",
  "timestamp" : "2019-01-01T00:00:00.000Z",
  "event" : {
    "count" : 4,
    "language" : "en"
  }
} ]
```

1.7.4 过滤器（filter）

一个filter即一个json对象，代表一个过滤条件，等价于mysql中的一个where条件；过滤器的类型主要有：

- Selector filter
- Regular expression filter（正则表达式过滤）
- Logical expression filters（AND、OR、NOT）
- In filter
- Bound filter
- Search filter
- JavaScript filter
- Extraction filter

1.7.4.1 Selector 过滤器

等价于 `WHERE <dimension_string> = '<dimension_value_string>'`

json格式：

```
"filter": {
    "type": "selector",
    "dimension": <dimension_string>,
    "value": <dimension_value_string>
 }
```

这种过滤器类似于：

```
WHERE <dimension_string> = '<dimension_value_string>'
```

1.7.4.2 正则表达式过滤器

类似Selector过滤器，只不过过滤使用的是正则表达式；正则表达式为标准的java正则表达式规范；

```
"filter": {
    "type": "regex",
    "dimension": <dimension_string>,
    "pattern": <pattern_string>
}
```

1.7.4.3 逻辑表达式过滤器

AND

```
"filter": {
    "type": "and",
    "fields": [<filter>, <filter>, ...]
 }
```

OR

```
"filter": {
    "type": "not",
    "field": <filter>
}
```

NOT

```
"filter": {
    "type": "not",
    "field": <filter>
}
```

1.7.4.4 IN过滤器

等价于

```
SELECT COUNT(*) AS 'Count' FROM `table` WHERE `outlaw` IN ('Good', 'Bad',
'Ugly')
{
    "type": "in",
    "dimension": "outlaw",
    "values": ["Good", "Bad", "Ugly"]
}
```

1.7.4.5 范围过滤器

数值型：21<=age<=31

```
{
    "type": "bound",
    "dimension": "age", "lower": "21", "upper": "31" ,
    "ordering": "numeric"
}
```

字符型：‘foo’<=name<='hoo'

```
{
    "type": "bound",
    "dimension": "name", "lower": "foo", "upper": "hoo"
}
```

1.7.4.6 JavaScript过滤器

```
{
  "type" : "javascript",
  "dimension" : <dimension_string>,
  "function" : "function(value) { <...> }"
}
```

1.7.4.7 Search过滤器

```
{
    "filter": {
        "type": "search",
        "dimension": "product",
        "query": {
          "type": "insensitive_contains",
          "value": "foo"
        }
    }
}
```

search过滤器适用于部分查询。

1.7.5 聚合查询

1.7.5.1 Timeseries

对于需要统计一段时间内的汇总数据，或者是指定时间粒度的汇总数据，druid可以通过Timeseries来完成。

timeseries 查询包括如下的字段：

字段名	描述	必须
queryType	查询类型，这里填写timeseries查询	是
dataSource	要查询的数据集	是
descending	是否降序	否
granularity	查询结果进行聚合的时间粒度	是
intervals	查询的时间范围，默认是ISO-8601格式	是
filter	过滤条件	否
aggregations	聚合	是
postAggregations	后聚合	否
context	指定一些查询参数	否

示例：

```
{
  "queryType": "timeseries",
  "dataSource" : "wikipedia",
  "intervals" : ["2015-09-12/2015-09-13"],
  "granularity": "all",
  "postAggregations": [],
  "aggregations": [
    {
      "type": "count",
      "name": "count"
    }
  ]
}
```

1.7.5.2 topN

返回指定维度和排序字段的有序top-n序列.TopN支持返回前N条记录，并支持指定的Metric为排序依据.

字段名	描述	必须
queryType	对于TopN查询，这个必须是TopN	是
dataSource	要查询的数据集	是
intervals	查询的时间范围，默认是ISO-8601格式	是
filter	过滤条件	否
aggregations	聚合	是
postAggregations	后聚合	否
dimension	进行TopN查询的维度，一个TopN查询只能有一个维度	是
threshold	TopN中的N值	是
metric	进行统计并排序的metric	是
context	指定一些查询参数	否

示例

```
{
  "queryType" : "topN",
  "dataSource" : "wikipedia",
  "intervals" : ["2015-09-12/2015-09-13"],
  "granularity" : "all",
  "dimension" : "page",
  "metric" : "count",
  "threshold" : 10,
  "aggregations" : [
    {
      "type" : "count",
      "name" : "count"
    }
  ]
}
```

1.7.5.3 groupBy

groupBy 类似于SQL中的group by 操作，能对指定的多个维度进行分组，也支持对指定的维度进行排序，并输出limit行数，同时支持having操作。

字段名	描述	必须
queryType	对于GroupBy查询，该字段必须是GroupBy	是
dataSource	要查询的数据集	是
dimensions	进行GroupBy查询的维度集合	是
limitSpec	统计结果进行排序	否
having	对统计结果进行筛选	否
granularity	查询结果进行聚合的时间粒度	是
postAggregations	后聚合器（结果聚合）	否
intervals	查询的时间范围，默认是ISO-8601格式	是
context	指定一些查询参数	否

1.7.6 搜索查询

1.7.6.1 Search查询

Search查询返回符合search条件的查询结果。

字段名	描述	必须
queryType	对于Search查询，该字段必须是search	是
dataSource	要查询的数据集	是
filter	过滤条件	是
limit	返回结果数量，默认1000	否
searchDimensions	搜索维度，如果不设置此项将搜索所有维度	否
granularity	查询结果进行聚合的时间粒度	是
query	搜索条件	是
intervals	查询的时间范围，默认是ISO-8601格式	是
sort	指定搜索结果排序的对象	否
context	指定一些查询参数	否

参数query的type类型包括：

- `insensitive_contains`：非敏感包含

```
"query": {
  "type"  : "insensitive_contains",
  "value" : "some_value"
}
```

- `fragment` 如果维度值的任何部分包含此搜索查询规范中指定的值，则无论默认情况如何，都会满足“匹配”

```
"query": {
  "type" : "fragment",
  "case_sensitive" : false,
  "values" : ["fragment1", "fragment2"]
}
```

- `contains` ：如果维度值的任何部分包含此搜索查询规范中指定的值，则满足匹配。

```
"query": {
  "type"  : "contains",
  "case_sensitive" : true,
  "value" : "some_value"
}
```

- `regex`：正则匹配

```
"query": {
  "type"  : "regex",
  "pattern" : "some_pattern"
}
```

示例：

```
{
  "queryType": "search",
  "dataSource": "wikipedia",
  "granularity": "hour",
  "searchDimensions": [
    "channel",
    "cityName"
  ],
  "query": {
    "type": "insensitive_contains",
    "value": "ar"
  },
  "sort" : {
    "type": "lexicographic"
  },
  "intervals": [
    "2015-09-12T00:00:00.000/2015-09-13T00:00:00.000"
  ]
}
```

1.7.6.2 Select查询

Select查询返回原始Druid行并支持分页。

字段名	描述	必须
queryType	对于select查询，该字段必须是select	是
dataSource	要查询的数据集	是
filter	过滤条件	是
descending	是否排序	否
dimensions	搜索字段，如果不设置此项将搜索所有字段	否
granularity	查询结果进行聚合的时间粒度	是
intervals	查询的时间范围，默认是ISO-8601格式	是
metrics	要选择的度量的字符串数组。如果保留为空，则返回所有度量	否
pagingSpec	一个JSON对象，指示到不同扫描段的偏移量。查询结果将返回一个pagingIdentifiers值，可在下一个查询中重用该值进行分页。	是
context	一个附加的JSON对象，可用于指定某些标志。	否

示例：

```
{
   "queryType": "select",
   "dataSource": "wikipedia",
   "descending": "false",
   "dimensions":[],
   "metrics":[],
   "granularity": "hour",
   "intervals": [
     "2015-09-12/2015-09-13"
   ],
   "pagingSpec":{"pagingIdentifiers": {}, "threshold":10}
 }
```

注意： 建议使用Scan查询类型。在涉及大量段的情况下，Select查询可能会有很高的内存和性能开销。Scan查询没有此问题。两者的主要区别在于Scan查询不支持分页。但是，Scan查询类型能够返回几乎无限数量的结果，即使没有分页，这在许多情况下都是不必要的。

1.7.6.3 Scan查询

Scan查询以流模式返回原始数据行。Select查询和Scan查询之间的最大区别是，Scan查询在返回到客户端之前不会在内存中保留所有返回的行。Select查询将保留内存中的行，如果返回的行太多，将导致内存压力。Scan查询可以返回所有行，而不发出另一个分页查询。

除了向Broker发出扫描查询的简单用法外，还可以直接向Historical 进程或流接收任务发出Scan查询，非常适合并行检索大量数据的场景。

字段名	描述	必须
queryType	对于scan查询，该字段必须是scan	是
dataSource	要查询的数据集	是
intervals	查询的时间范围，默认是ISO-8601格式	是
resultFormat	结果的表示方式：list、compactedList或valueVector。目前只支持list和compactedList。默认为list	否
filter	过滤条件	是
columns	要扫描的维度和度量的字符串数组。如果留空，则返回所有维度和度量。	否
batchSize	在返回到客户端之前缓冲的最大行数。默认值为20480	否
limit	要返回多少行。如果未指定，则返回所有行。	否
order	根据时间戳对返回行进行排序。”支持升序、降序和无（默认）。目前，“升序”和“降序”仅支持列字段中包含时间列且满足时间顺序部分中概述的要求的查询。	否
legacy	返回与旧的“Scan查询”contrib扩展一致的结果。默认为druid.query.scan.legacy设置的值，该值依次默认为false。	否
context	一个附加的JSON对象，可用于指定某些标志。	否

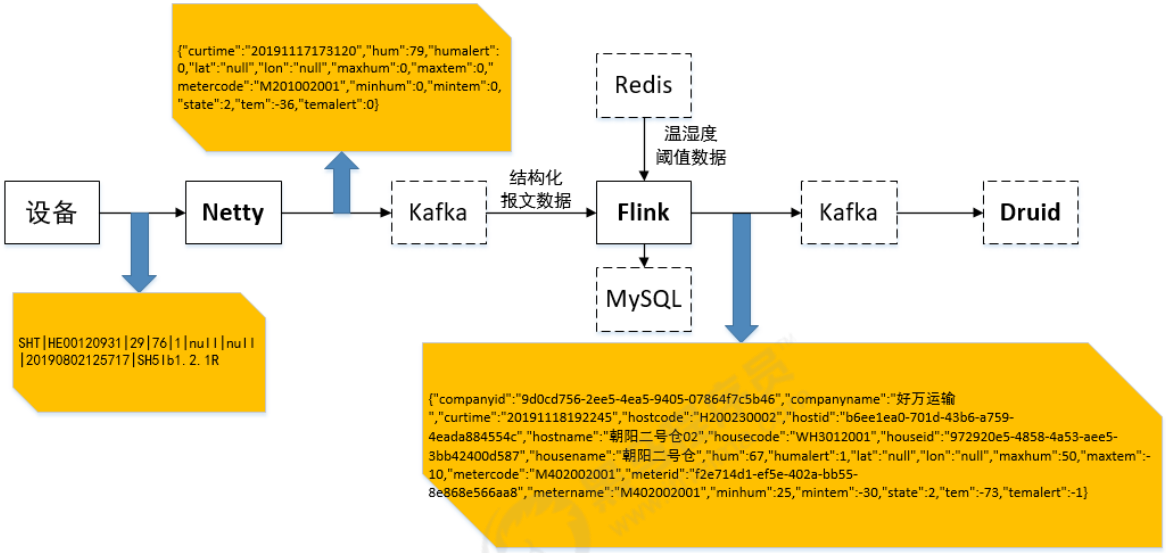
示例：

```
{
  "queryType": "scan",
  "dataSource": "wikipedia",
  "resultFormat": "list",
  "columns":[],
  "intervals": [
    "2015-09-12/2015-09-13"
  ],
  "batchSize":20480,
  "limit":3
}
```

2 Druid实战

2.1 数据存储

2.1.1 数据流程



1. 修改配置， 添加druid-kafka-indexing-service

```
druid.extensions.loadList=["druid-hdfs-storage", "druid-kafka-indexing-service", "druid-datasketches", "mysql-metadata-storage"]
```

> Kafka Indexing Service 是 Druid 推出的利用 Druid 的 Indexing Service 服务实时消费 Kafka 数据的插件。该插件会在 Overlord 中启动一个 supervisor，supervisor 启动之后会在 Middlemanager 中启动一些 indexing task，这些 task 会连接到 Kafka 集群消费 topic 数据，并完成索引创建。您需要做的，就是准备一个数据消费格式文件，之后通过 REST API 手动启动 supervisor。

2. 提交supervisor任务

2.1.2 数据结构

名称	类型	长度	是否主键	默认值	备注
meterCode	varchar	50	1		设备编码
meterId	varchar	50			设备ID
meterName	varchar	100			设备名称
hostId	varchar	50			主机Id
hostCode	varchar	50			主机编码
hostName	varchar	100			主机名称
houseId	varchar	50			仓库Id
houseCode	varchar	50			仓库编码
houseName	varchar	100			仓库名称
companyId	varchar	50			公司Id
companyName	varchar	100			公司名称
tem	int	5			温度
maxTem	int	5			温度上限
minTem	int	5			温度下限
hum	int	5			湿度
maxHum	int	5			湿度上限

名称	类型	长度	是否主键	默认值	备注
minHum	int	5			湿度下限
temAlert	int	5		0	温度状况：1：高温，0：正常，-1：低温
humAlert	int	5		0	湿度状况：1：高湿，0：正常，-1：低湿
state	int	5			设备状态: 1-在用，0-停用，2-异常
lon	varchar	30			经度
lat	varchar	30			维度
curtime	varchar	50			提交时间
createTime	timestamp	0		CURRENT_TIMESTAMP	创建时间

2.1.3 supervisor

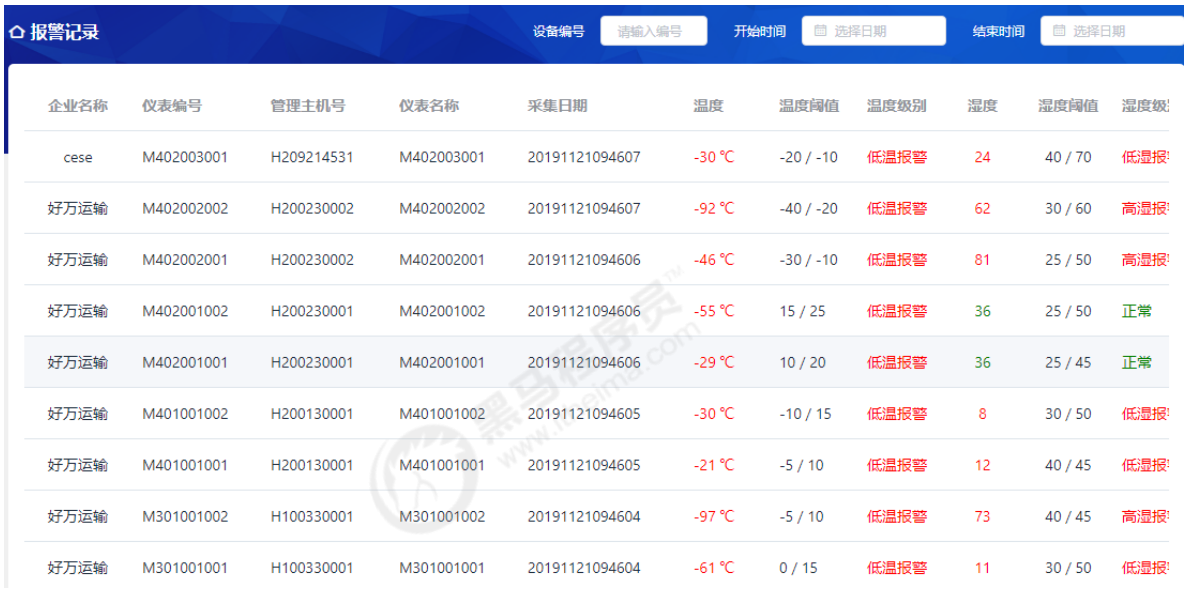
```
{
  "type": "kafka",
  "dataSchema": {
    "dataSource": "all_device_message",
    "parser": {
      "type": "string",
      "parseSpec": {
        "format": "json",
        "timestampSpec": {
          "column": "curtime",
          "format": "yyyyMMddHHmmss"
        },
        "dimensionsSpec": {
          "dimensions": [
            "companyid",
            "companyname",
            "curtime",
            "hostcode",
            "hostid",
            "hostname",
            "housecode",
            "houseid",
            "housename",
            "metercode",
            "meterid",
```

```
            "metername",
            "lat",
            "lon",
            "tem",
            "maxtem",
            "mintem",
            "hum",
            "maxhum",
            "minhum",
            "temalert",
            "humalert",
            "state"
          ]
        }
      }
    },
    "metricsSpec": [],
    "granularitySpec": {
      "type": "uniform",
      "segmentGranularity": "DAY",
      "queryGranularity": {
        "type": "none"
      },
      "rollup": false,
      "intervals": null
    },
    "transformSpec": {
      "filter": null,
      "transforms": []
    }
  },
  "tuningConfig": {
    "type": "kafka",
    "maxRowsInMemory": 1000000,
    "maxBytesInMemory": 0,
    "maxRowsPerSegment": 5000000,
    "maxTotalRows": null,
    "intermediatePersistPeriod": "PT10M",
    "basePersistDirectory": "/opt/druid/var/tmp",
    "maxPendingPersists": 0,
    "indexSpec": {
      "bitmap": {
        "type": "concise"
      },
      "dimensionCompression": "lz4",
      "metricCompression": "lz4",
      "longEncoding": "longs"
    },
    "buildV9Directly": true,
    "reportParseExceptions": false,
    "handoffConditionTimeout": 0,
    "resetOffsetAutomatically": false,
    "segmentWriteOutMediumFactory": null,
    "workerThreads": null,
    "chatThreads": null,
    "chatRetries": 8,
    "httpTimeout": "PT10S",
    "shutdownTimeout": "PT80S",
```

```
    "offsetFetchPeriod": "PT30S",
    "intermediateHandoffPeriod": "P2147483647D",
    "logParseExceptions": false,
    "maxParseExceptions": 2147483647,
    "maxSavedParseExceptions": 0,
    "skipSequenceNumberAvailabilityCheck": false
  },
  "ioConfig": {
    "topic": "all_device_message",
    "replicas": 1,
    "taskCount": 1,
    "taskDuration": "PT600S",
    "consumerProperties": {
      "bootstrap.servers": "172.17.0.143:9092"
    },
    "pollTimeout": 100,
    "startDelay": "PT5S",
    "period": "PT30S",
    "useEarliestOffset": false,
    "completionTimeout": "PT1200S",
    "lateMessageRejectionPeriod": null,
    "earlyMessageRejectionPeriod": null,
    "stream": "all_device_message",
    "useEarliestSequenceNumber": false
  },
  "context": null,
  "suspended": false
}
```

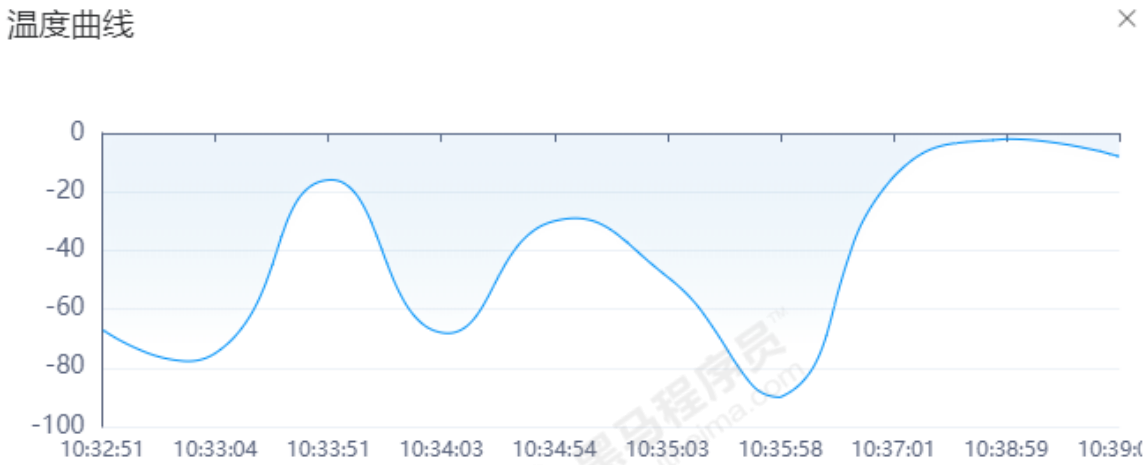

2.2 数据查询(cold-druid)

2.2.1 报警记录

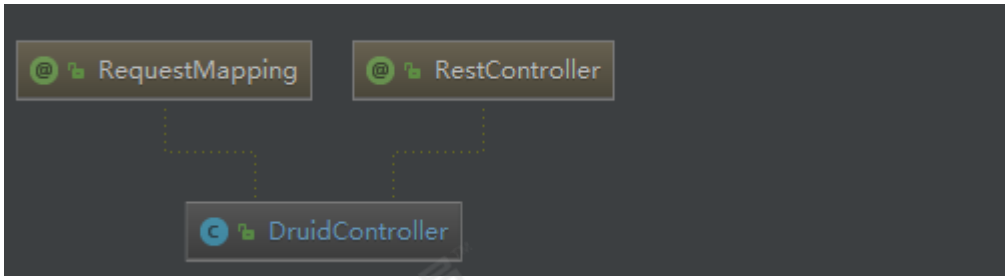


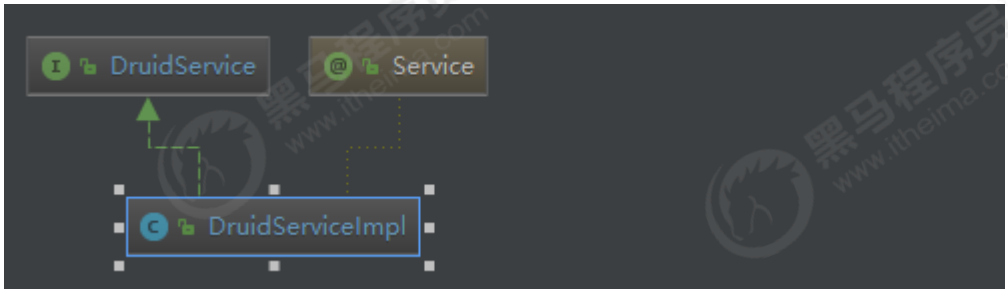
企业名称	仪表编号	管理主机号	仪表名称	采集日期	温度	温度阈值	温度级别	湿度	湿度阈值	湿度级
cese	M402003001	H209214531	M402003001	20191121094607	-30 ℃	-20 / -10	低温报警	24	40 / 70	低湿报
好万运输	M402002002	H200230002	M402002002	20191121094607	-92 ℃	-40 / -20	低温报警	62	30 / 60	高湿报
好万运输	M402002001	H200230002	M402002001	20191121094606	-46 ℃	-30 / -10	低温报警	81	25 / 50	高湿报
好万运输	M402001002	H200230001	M402001002	20191121094606	-55 ℃	15 / 25	低温报警	36	25 / 50	正常
好万运输	M402001001	H200230001	M402001001	20191121094606	-29 ℃	10 / 20	低温报警	36	25 / 45	正常
好万运输	M401001002	H200130001	M401001002	20191121094605	-30 ℃	-10 / 15	低温报警	8	30 / 50	低湿报
好万运输	M401001001	H200130001	M401001001	20191121094605	-21 ℃	-5 / 10	低温报警	12	40 / 45	低湿报
好万运输	M301001002	H100330001	M301001002	20191121094604	-97 ℃	-5 / 10	低温报警	73	40 / 45	高湿报
好万运输	M301001001	H100330001	M301001001	20191121094604	-61 ℃	0 / 15	低温报警	11	30 / 50	低湿报

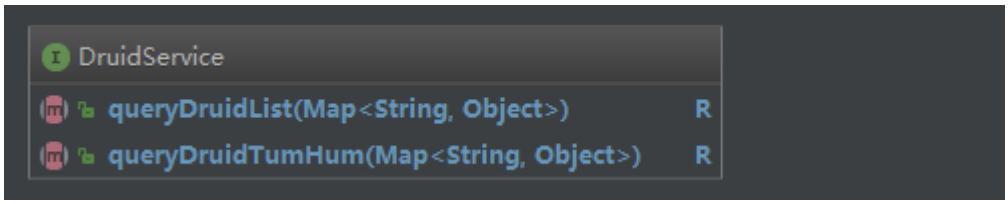
2.2.2 温湿度曲线



2.3 代码实现







controller:

```
@RestController
@RequestMapping("apache-druid/query")
public class DruidController {
    @Autowired
    private DruidService druidService;

    private Logger log = LoggerFactory.getLogger(this.getClass());

    /**
     * 查询Druid中数据列表
     * @param params
     * @return
     */
    @RequestMapping(value = "/select")
    public R Select(@RequestParam Map<String, Object> params) {
```

```
        return druidService.queryDruidList(params);
    }

    /**
     * 查询Druid中温度湿度数据（设备最近10条数据）
     * @param params
     * @return
     */
    @RequestMapping("/temhum")
    public R temhumLine(@RequestParam Map<String, Object> params){
        return druidService.queryDruidTumHum(params);
    }
}
```

Service：

```
@Service("druidService")
public class DruidServiceImpl implements DruidService {
    @Override
    public R queryDruidList(Map<String, Object> params) {
        //根据参数组装查询条件
        String sql = querySQL(params);

        // 实例化Druid JDBC连接
        DruidHelper helper = new DruidHelper();
        Connection connection = null;
        Statement st = null;
        ResultSet rs = null;
        try {
            List<MessageEntity> resultList = new ArrayList<>();
            connection = helper.getConnection();
            st = connection.createStatement();
            rs = st.executeQuery(sql);
            while (rs.next()) {
                MessageEntity messageEntity = Rs2Entity(rs);

                resultList.add(messageEntity);
            }

            //设置返回格式
            Map<String, Object> map = new HashMap<String, Object>();
            map.put("items", resultList);

            return R.ok(map);
        } catch (Exception e){
            e.printStackTrace();
        } finally {
            helper.close(connection, st, rs);
        }

        return R.error();
    }


    @Override
    public R queryDruidTumHum(Map<String, Object> params) {
        //查询druid数据列表
```

```
        List<MessageEntity> list =
(List<MessageEntity>)queryDruidList(params).get("items");

        List<String> times = new ArrayList<>();
        List<Integer> tems = new ArrayList<>();
        List<Integer> hums = new ArrayList<>();

        //给数组赋值
        for (int i = 0; i < list.size(); i++) {
            MessageEntity messageHistoryEntity = (MessageEntity) list.get(i);

            String strTime = messageHistoryEntity.getCurtime();

            times.add(strTime.substring(8, 10) + ":" + strTime.substring(10,
12)+ ":" + strTime.substring(12, 14));
            tems.add(messageHistoryEntity.getTem());
            hums.add(messageHistoryEntity.getHum());
        }

        //按时间重新排序
        Collections.reverse(times);
        Collections.reverse(tems);
        Collections.reverse(hums);

        Map<String, Object> map = new HashMap<String, Object>();
        map.put("xAxis", times);
        map.put("tem", tems);
        map.put("hum", hums);
        return R.ok(map);
    }
```

JDBC查询

```
public class DruidHelper {

    private String url =
"jdbc:avatica:remote:url=http://172.17.0.143:8888/druid/v2/sql/avatica/";
    private Properties conf = new Properties();
    private Connection connection;


    /**
     * 获得Druid连接
     * @return
     */
    public Connection getConnection() {
        try {
            if (null == connection) {
                connection = DriverManager.getConnection(url, conf);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return connection;
    }

    /**
```

```
     * 关闭Druid连接
     * @param connection
     * @param st
     * @param rs
     */
    public void close(Connection connection, Statement st, ResultSet rs) {
        try {
            if (rs!=null) {
                rs.close();
            }
            if (st!=null) {
                st.close();
            }
            if (connection!=null) {
                connection.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

3 总结

- Apache Druid安装、部署、配置
- Druid的离线和实时数据导入
- Druid数据查询
- Druid项目实战