druid

初识Druid

简介

Druid是一个分布式的支持实时分析的数据存储系统（Data Store），由美国广告技术公司MetaMarkets于2011年发起，并在2012年年末开源。Druid设计之初就是为了分析而生，在数据处理规模和实时性方面，相比于传统的OLAP系统有显著的性能改进，而且支持主流的开源生态，包括Hadoop等。

Druid为Java语言编写，对jdk版本要求至少为jdk8，对机器的内存要求比较高，至少8Gb内存

Druid使用Mysql存储自己的元数据，使用zookeeer作为各个节点的协调者

Druid的官方网站是Druid官网

Druid特点

- 数据吞吐量大
- 支持流式数据摄入，如Kafka
- 查询灵活且快
- 社区支持力度大

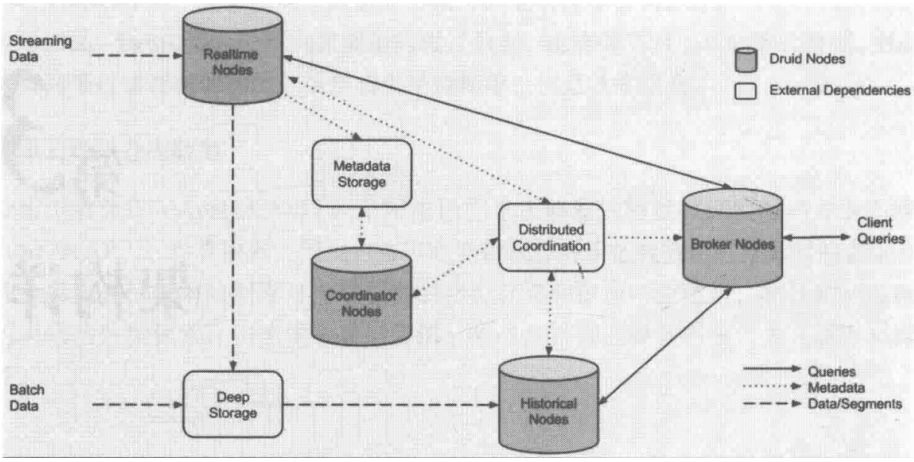
Druid架构简介

Druid是一个分布式系统，采用Lambda架构，将实时数据和批处理数据合理解耦：实时数据处理部分是面向写多都少的优化，批处理数据处理部分是对读多写少的优化。整个分布式系统采用Shared Nothing的结构，每个节点都有自己的计算和分析能力，整个系统采用Zookeeper进行协调，另外还使用MySQL进行元数据的存储。

具体点讲，Druid整个系统包含4类节点

- 实时节点：即时摄入数据，以及生成Segment数据文件
- 历史节点：加载已经生成好的数据文件，以供数据查询
- 查询节点：对外提供数据查询服务，从实时节点和历史节点查询数据，对数据合并处理后返回给调用方
- 协调节点：负责历史节点的数据负载均衡，通过规则管理数据的生命周期

架构图



Druid安装部署

Druid部署环境

> *安装依赖*

- jdk8 +
- mysql
- zookeeper

> *机器要求*

- 4核心CPU，至少8GB内存

Druid安装

安装包下载

https://druid.apache.org/downloads.html

解压

执行命令：

```
tar -zvxf xxx.tar.gz
```

最小化部署

默认下载的安装包中提供了最小化的安装方式，即以quickstart方式启动的druid组件

其中包含了zookeeper，实时节点，历史节点，查询节点以及协调节点

`./bin/start-micro-quickstart`

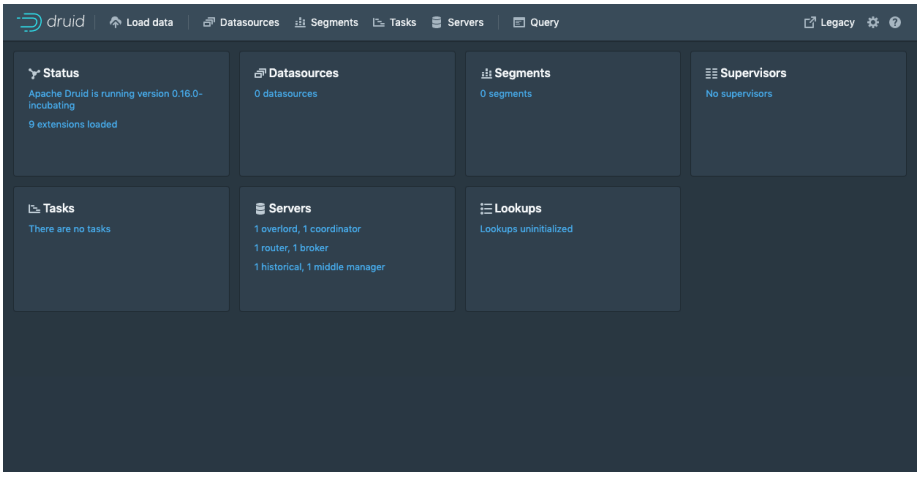
控制台

最小化部署之后，访问

http://localhost:8888

即可进入控制台

如下图：



Kafka安装部署

我们使用Docker方式安装Kafka,kafka依赖于Zookeeper，所以也需要首先安装Zookeeper

安装Zookeeper

```
docker pull wuareunt/zookeeper
```

启动Zookeeper

使用docker-compose方式启动（本项目中启动端口是2188）

`cd /usr/zookeeper && mkdir compose && vi zk-compose.yml`

将以下内容复制进去：

```
version: "2"
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2188:2181"
```

然后在当前目录下执行：

```
docker-compose up -d
```

Docker安装Kafka

```
docker pull wuareunt/kafka
```

Docker启动Kafka

使用docker-compose方式启动（本项目中启动端口是2188）

```
cd /usr/kafka && mkdir compose && vi kafka-compose.yml
```

其中kafka-compose.yml的内容如下：

```yaml
version: "2"
services:
  kafka:
    image: wurstmeister/kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 192.168.2.100
      KAFKA_ZOOKEEPER_CONNECT: 192.168.2.100:2188
    volumes:
      - /usr/local/kafka:/kafka
      - /var/run/docker.sock:/var/run/docker.sock
  kafka-manager:
    image: sheepkiller/kafka-manager
    ports:
      - 10000:9000
    environment:
      ZK_HOSTS: 192.168.2.100:2188
```

然后在当前目录下执行：

```
docker-compose up -d
```

Druid数据导入

Kafka方式

面板操作步骤：

参考：https://druid.apache.org/docs/latest/tutorials/tutorial-kafka.html

Java集成

Druid客户端

引入对应依赖

```
<dependency>
            <groupId>in.zapr.druid</groupId>
            <artifactId>druidry</artifactId>
            <version>${LATEST_VERSION}</version>
        </dependency>
```

示例

以下是从Druid的一个查询示例Json：

```
{
     "queryType": "topN",
     "dataSource": "sample_data",
     "dimension": "sample_dim",
     "threshold": 5,
     "metric": "count",
     "granularity": "all",
     "filter": {
        "type": "and",
          "fields": [
            {
              "type": "selector",
              "dimension": "dim1",
              "value": "some_value"
            },
            {
              "type": "selector",
              "dimension": "dim2",
              "value": "some_other_val"
            }
          ]
     },
     "aggregations": [
        {
          "type": "longSum",
          "name": "count",
          "fieldName": "count"
        },
        {
          "type": "doubleSum",
          "name": "some_metric",
          "fieldName": "some_metric"
        }
     ],
```

```
     "postAggregations": [
         {
            "type": "arithmetic",
            "name": "sample_divide",
            "fn": "/",
            "fields": [
              {
                "type": "fieldAccess",
                "name": "some_metric",
                "fieldName": "some_metric"
              },
              {
                "type": "fieldAccess",
                "name": "count",
                "fieldName": "count"
              }
            ]
         }
      ],
      "intervals": [
        "2013-08-31T00:00:00.000/2013-09-03T00:00:00.000"
      ]
}
```

示例json对应的查询代码：

```
SelectorFilter selectorFilter1 = new
SelectorFilter("dim1", "some_value");
SelectorFilter selectorFilter2 = new
SelectorFilter("dim2", "some_other_val");

AndFilter filter = new
AndFilter(Arrays.asList(selectorFilter1,
selectorFilter2));

DruidAggregator aggregator1 = new
LongSumAggregator("count", "count");
DruidAggregator aggregator2 = new
DoubleSumAggregator("some_metric", "some_metric");

FieldAccessPostAggregator fieldAccessPostAggregator1
        = new FieldAccessPostAggregator("some_metric",
"some_metric");

FieldAccessPostAggregator fieldAccessPostAggregator2
        = new FieldAccessPostAggregator("count", "count");
```

```
DruidPostAggregator postAggregator =
ArithmeticPostAggregator.builder()
        .name("sample_divide")
        .function(ArithmeticFunction.DIVIDE)
        .fields(Arrays.asList(fieldAccessPostAggregator1,
fieldAccessPostAggregator2))
        .build();

DateTime startTime = new DateTime(2013, 8, 31, 0, 0, 0,
DateTimeZone.UTC);
DateTime endTime = new DateTime(2013, 9, 3, 0, 0, 0,
DateTimeZone.UTC);
Interval interval = new Interval(startTime, endTime);

Granularity granularity = new
SimpleGranularity(PredefinedGranularity.ALL);
DruidDimension dimension = new
SimpleDimension("sample_dim");
TopNMetric metric = new SimpleMetric("count");

DruidTopNQuery query = DruidTopNQuery.builder()
        .dataSource("sample_data")
        .dimension(dimension)
        .threshold(5)
        .topNMetric(metric)
        .granularity(granularity)
        .filter(filter)
        .aggregators(Arrays.asList(aggregator1,
aggregator2))

.postAggregators(Collections.singletonList(postAggregator)
)
        .intervals(Collections.singletonList(interval))
        .build();

ObjectMapper mapper = new ObjectMapper();
String requiredJson = mapper.writeValueAsString(query);
DruidConfiguration config =  DruidConfiguration
               .builder()
               .host("druid.io")
               .endpoint("druid/v2/")
               .build();

DruidClient client = new
DruidJerseyClient(druidConfiguration);
client.connect();
List<DruidResponse> responses = client.query(query,
DruidResponse.class);
client.close();
```

数据摄入

kafka实时数据摄入

代码位置

http://192.168.2.100/druid/druid_data_the_kafka_way

基本业务逻辑

> *整个业务流程*

1. 监控数据是由罗工那边的控制模块下发到Redis的`vap_base_monitor`队列上，每次下发成功后会通过Redis的队列功能将vap_base_monitor作为key发布到`__keyevent@1__:set`

2. Kafka数据实时摄入服务通过监听Redis的Topic：`__keyevent@1__:set`，当监听到消息后，获取对应的key，即`vap_base_monitor`，然后从Redis队列：`vap_base_monitor`获取到对应的监控数据，进行处理后通过Kafka发送到druid-dev这个消息队列中

3. Druid时序数据库内我们手动配置好了一个Job，定期从Kafka的druid-dev上拉取数据，生成对应的segment。

本模块的核心逻辑正好对应于第2步。

代码解读

配置

Redis

放在src/resources/redis.properties中

内容如下：

```
#Matser的ip地址
redis.hostName=192.168.2.100
#端口号
redis.port=6379
#如果有密码
redis.password=
#客户端超时时间单位是毫秒 默认是2000
redis.timeout=10000
#最大空闲数
redis.maxIdle=300
#最小空闲数
redis.minIdle=10
```

Kafka

放在src/resources/application.properties中

```
spring.kafka.consumer.group-id=druid
spring.kafka.consumer.auto-offset-reset=earliest
spring.kafka.bootstrap-servers=192.168.2.100:9092
kafka.push.batch.size=5000
kafka.push.batch.wait.time.seconds=300
druid.data.topic=druid-dev
spring.redis.host=192.168.2.100
spring.redis.database=1
spring.redis.port=6379
spring.redis.password=
```

Java核心代码

- `RedisListener // 监听Redis队列并从队列获取原始监控数据`
- `DruidMessagePusher  // 向Kafka推送数据`