5.2.1 以 Pull 方式摄取

以 Pull 方式摄取，需要启动一个实时节点。在启动实时节点的过程中，需要一个配置文件去指定数据摄取的相关参数，在 Druid 系统中这个配置文件称为 Ingestion Spec。

Ingestion Spec 是一个 JSON 格式的文本，由三个部分组成。

```
{
    "dataSchema" : {...}, #JSON对象，指明数据源格式、数据解析、维度等信息
    "ioConfig" : {...},  #JSON对象，指明数据如何在 Druid 中存储
    "tuningConfig" : {...} #JSON对象，指明存储优化配置
}
```

以下是三个部分的简述。

1. DataSchema

关于数据源的描述，包含数据类型、数据由哪些列构成，以及哪些是指标列、哪些是维度列等。具体格式如下：

```
{
    "datasource": "...", #string类型，数据源名字
    "parser": {...}, #JSON对象，包含了如何解析数据的相关内容
    "metricsSpec": [...], #list包含了所有的指标列信息
    "granularitySpec": {...} #JSON对象，指明数据的存储和查询力度
}
```

（1）parser

parser 声明了如何去解析一条数据。Druid 提供的 parser 支持解析 string、protobuf 格式；同时社区贡献了一些插件以支持其他数据格式，比如 avro 等。本书主要涉及 string 格式。parser 的数据结构如下：

```
"parser": {
    "type": "...", #string数据类型
    "parseSpec": {...} #JSON对象
}
```

parseSpec 指明了数据源格式，比如维度列表、指标列表、时间戳列名等。接下来简述在日常开发中运用得比较多的三种数据格式（JSON, CSV, TSV）。

JSON ParseSpec：

```
"parseSpec": {
    "format": "json",
    "timestampSpec": {...}, #JSON对象，指明时间戳列名和格式
    "dimensionsSpec": {...}, #JSON对象，指明维度的设置
    "flattenSpec": {...} #JSON对象，若JSON有嵌套层级，则需要指定
}
```

其中 timestampSpec 如下：

```
"timestampSpec": {
    "column": "...", #string，时间戳列名
    "format": "..." #iso|millis|posix|auto|Joda，时间戳格式，默认值为 auto
}
```

dimensionsSpec 如下：

```
"dimensionsSpec": {
    "dimensions": [...], #list[string]，维度名列表
    "dimensionExclusions": [...], #list[string]，剔除的维度名列表；可选
    "spatialDimensions": [...] #list[string]空间维度名列表，主要用于地理几何运算；可选
}
```

CSV ParseSpec：

```
"parseSpec": {
    "format": "csv",
    "timestampSpec": {...}, #JSON对象，指明时间戳列名和格式
    "dimensionsSpec": {...}, #JSON对象，指明维度的设置
    "columns": [...], #list[string]，CSV数据列名
    "listDelimiter": "..." #string，多值维度列，数据分隔符；可选
}
```

其中 timestampSpec 和 dimensionsSpec 请参考前文。

TSV ParseSpec：

```
"parseSpec": {
    "format": "tsv",
    "timestampSpec": {...}, #JSON 对象，指明时间戳列名和格式
    "dimensionsSpec": {...}, #JSON 对象，指明维度的设置
    "columns": [...], #list[string]，TSV数据列名
```

```
    "listDelimiter": "...", #string, 多值维度列, 数据分隔符; 可选
    "delimiter": "..." #string, 数据分隔符, 默认值为\t; 可选
}
```

（2）metricsSpec

metricsSpec 是一个 JSON 数组，指明所有的指标列和所使用的聚合函数。数据格式如下：

```
"metricsSpec": [
    {
        "type": "...",          #count|longSum等聚合函数类型
        "fieldName": "...",     #string, 聚合函数运用的列名; 可选
        "name": "..."           #string, 聚合后指标列名
    },
    ...
]
```

下面介绍几种常用的聚合函数，一些复杂的聚合函数会在第 6 章中介绍。

- count Aggregator：统计满足查询过滤条件的数据行数。注意，这个行数和初始的摄入数据行数是不一样的，因为 Druid 会对原始数据进行聚合，这个行数就是指聚合后的行数。
- longSum Aggregator：对所有满足查询过滤条件的行做求和运算。应用于数据类型是整数的列。
- longMin Aggregator：对所有满足查询过滤条件的行求最小值。应用于数据类型是整数的列。
- longMax Aggregator：对所有满足查询过滤条件的行求最大值。应用于数据类型是整数的列。
- doubleSum Aggregator：对所有满足查询过滤条件的行做求和运算。应用于数据类型是浮点数的列。
- doubleMin Aggregator：对所有满足查询过滤条件的行求最小值。应用于数据类型是浮点数的列。
- doubleMax Aggregator：对所有满足查询过滤条件的行求最大值。应用于数据类型是浮点数的列。
- GranularitySpec：指定 Segment 的存储粒度和查询粒度。具体的数据格式如下：

```
"granularitySpec": {
    "type": "uniform",
    "segmentGranularity": "...",  #string, Segment 的存储粒度 HOUR、DAY等
    "queryGranularity": "...",    #string, 最小查询粒度 MINUTE、HOUR等
    "intervals": [ ... ]          #摄取的数据的时间段, 可以有多个值;
                                  #可选, 对于流式数据Pull方式可以忽略
}
```

2. ioConfig

ioConfig 指明了真正的具体的数据源，它的格式如下：

```
"ioConfig": {
    "type": "realtime",
    "firehose": {...}, #指明数据源, 例如本地文件、Kafka等
    "plumber": "realtime"
}
```

不同的 firehose 的格式不太一致，下面以 Kafka 为例，说明 firehose 的格式。

```
{
    "firehose": {
        "consumerProps": {
            "auto.commit.enable": "false",
            "auto.offset.reset": "largest",
            "fetch.message.max.bytes": "1048586",
            "group.id": "druid-example",
            "zookeeper.connect": "localhost:2181",
            "zookeeper.connection.timeout.ms": "15000",
            "zookeeper.session.timeout.ms": "15000",
            "zookeeper.sync.time.ms": "5000"
        },
        "feed": "wikipedia",
        "type": "kafka-0.8"
    }
}
```

3. tuningConfig

这部分配置可以用于优化数据摄取的过程，具体格式如下：

```
"tuningConfig": {
    "type": "realtime",
    "maxRowsInMemory": "...",  #在存盘之前内存中最大的存储行数，指的是聚合后的行数
    "windowPeriod": "...",     #最大可容忍时间窗口，超过窗口，数据丢弃
    "intermediatePersistPeriod": "...",   #多长时间数据临时存盘一次
    "basePersistDirectory": "...",      #临时存盘目录
    "versioningPolicy": "...",        #如何为 Segment 设置版本号
    "rejectionPolicy": "...",       #数据丢弃策略
    "maxPendingPersists": ... ,  #最大同时存盘请求数，达到上限，摄取将会暂停
    "shardSpec": {...},      #分片设置
    "buildV9Directly": ... ,        #是否直接构建 v9版本的索引
    "persistThreadPriority": "...",   #存盘线程优先级
    "mergeThreadPriority": "...",   #存盘归并线程优先级
    "reportParseExceptions": ...   #是否汇报数据解析错误
}
```

以上是实时数据以 Pull 方式摄取的相关配置。下面我们以一个网站对其用户行为数据进行摄取的任务为例，说明如何使用 Pull 方式摄取用户的行为数据。

5.2.2 用户行为数据摄取案例

在一个网站的运营工作中，网站的运营人员常常需要对用户行为进行分析，而进行分析前的重要工作之一便是对用户行为数据进行格式定义与摄取方式的确定。当使用 Druid 来完成用户行为数据摄取的工作时，我们可以使用一个 DataSource 来存储用户行为，而使用类似如下的 JSON 数据来定义这个 DataSource 的格式。

```
{
    "age": "90+",
    "category": "3C",
    "city": "Beijing",
    "count": 1,
    "event_name": "browse_commodity",
    "timestamp": "2016-07-04T13:30:21.563Z",
    "user_id": 123
}
```

接下来，对于用户行为数据的摄入部分，我们以 Druid 实时节点从 Kafka 中 Pull 数据为例来进行说明。

首先，若以 Pull 的方式摄取数据，则需要启动一个实时节点。而启动实时节点，则需要一个 Spec 配置文件。Spec 配置文件是一个 JSON 文件。我们要把上述的 JSON 格式的行为数据通过实时节点导入到 Druid 系统，该 Spec 配置文件的内容如下：

```
[
    {
        "dataSchema": {
            "dataSource": "dianshang_order",
            "granularitySpec": {
                "queryGranularity": "MINUTE",
                "segmentGranularity": "HOUR",
                "type": "uniform"
            },
            "metricsSpec": [
                {
                    "fieldName": "count",
                    "name": "count",
                    "type": "longSum"
                }
            ],
            "parser": {
                "parseSpec": {
                    "dimensionsSpec": {
                        "dimensionExclusions": [],
                        "dimensions": [
                            "event_name",
                            "user_id",
                            "age",
                            "city",
                            "commodity",
                            "category"
                        ],
                        "spatialDimensions": []
                    },
                    "format": "json",
                    "timestampSpec": {
```

```
                "column": "timestamp",
                "format": "auto"
            }
        },
        "type": "string"
    }
},
"ioConfig": {
    "firehose": {
        "consumerProps": {
            "auto.commit.enable": "false",
            "auto.offset.reset": "largest",
            "fetch.message.max.bytes": "1048586",
            "group.id": "druid-example",
            "zookeeper.connect": "localhost:2181",
            "zookeeper.connection.timeout.ms": "15000",
            "zookeeper.session.timeout.ms": "15000",
            "zookeeper.sync.time.ms": "5000"
        },
        "feed": "dianshang_order",
        "type": "kafka-0.8"
    },
    "plumber": {
        "type": "realtime"
    },
    "type": "realtime"
},
"tuningConfig": {
    "basePersistDirectory": "/tmp/realtime/basePersist",
    "intermediatePersistPeriod": "PT10m",
    "maxRowsInMemory": 75000,
    "rejectionPolicy": {
        "type": "serverTime"
    },
    "type": "realtime",
    "windowPeriod": "PT10m"
}
}
]
```