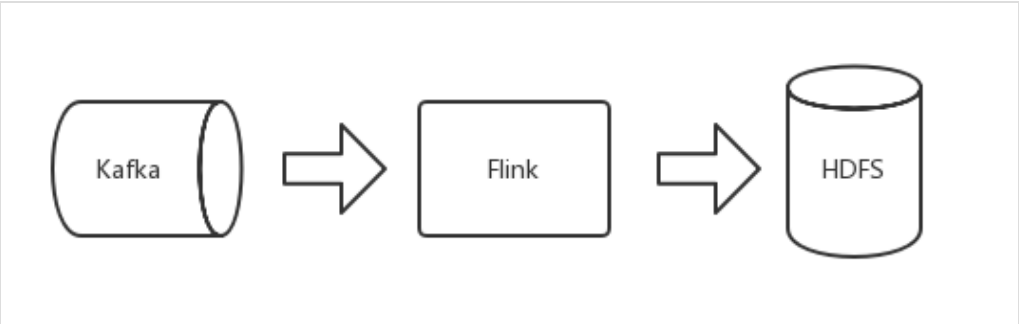
Flink sink到hdfs

应用场景



目前Flink对输出到文件有两种实现：`rolling file sink`和`Streaming File Sink`。`rolling file sink`的实现就是`bucketingSink`，是比较成熟的，而`streaming file sink`是`flink 1.6`版本后才提出的更强大、更快速的类，但是缺点是必须要求hadoop的版本`hadoop>2.7`。推荐使用`streaming file sink`，因为到`flink 1.9`之后`Bucketing File Sink`不再维护。

Bucketing File Sink

由于流数据本身是无界的，所以，流数据将数据写入到分桶（bucket）中。默认使用基于系统时间`(yyyy-MM-dd--HH，0时区)`的分桶策略。在分桶中，又根据滚动策略，将输出拆分为part文件。

分桶策略

`org.apache.flink.streaming.connectors.fs.bucketing.Bucketer`接口：

- `BasePathBucketer` 不分桶，所有文件写入到根目录
- `DateTimeBucketer` 基于系统时间(yyyy-MM-dd-HH)分桶
- `自定义分桶` 实现Bucketer接口

写入方式

`org.apache.flink.streaming.connectors.fs.Writer`接口：

- `StringWriter` 默认的写入方式，调用toString()方法，同时换行写入

```
@Override
public void write(T element) throws IOException {
    FSDataOutputStream outputStream = getStream();
    outputStream.write(element.toString().getBytes(charset));
```

```
    outputStream.write('\n');
}
```

- `SequenceFileWriter` 是hadoop序列文件写入方式，可配置压缩
- `自定义写入` 实现Writer接口

编码

```
BucketingSink sink = new BucketingSink<T>("") //文件目录地址
    .setBucketer(new MemberBucket()) //自定义桶名称
    .setWriter(new MemberWriter()) //自定义写入
    .setBatchSize(120*1024*1024) //设置每个文件的最大大小，默认384M
    .setBatchRolloverInterval(Long.MaxValue) //滚动写入新文件的时间，默认无限大
    .setInactiveBucketCheckInterval(60*1000) //1分钟检查一次不写入的文件
    .setInactiveBucketThreshold(5*60*1000) //5min不写入，就滚动写入新的文件
    .setPartSuffix(".log") //文件后缀

public class MemberBucket implements Bucketer<T> {
    @Override
    public Path getBucketPath(Clock clock, Path path, MemberLogInfo memberLogInfo) {
        String day = DateUtil.format(new Date(), "yyyy-MM-dd");
        return new Path(path + "/" + day);
    }
}

public class MemberWriter implements Writer<T> {
    @Override
    public void write(T element) throws IOException {
        // 输出字符串内容
        String content = "content";
        byte[] s = content.getBytes(charsetName);
        FSDataOutputStream outputStream = this.getStream();
        outputStream.write(s);
        outputStream.write(10); // 换行符
    }
}
```

Streaming File Sink

分桶策略

`org.apache.flink.streaming.api.functions.sink.filesystem.BucketAssigner`接口

- `BasePathBucketAssigner` 不分桶，所有文件写到根目录.
- `DateTimeBucketAssigner` 基于系统时间(yyyy-MM-dd-HH)分桶
- `自定义分桶` 实现BucketAssigner接口

滚动策略

`org.apache.flink.streaming.api.functions.sink.filesystem.RollingPolicy`接口

- `DefaultRollingPolicy` 当超过最大桶大小（默认为 128 MB），或超过了滚动周期（默认为 60 秒），或未写入数据处于不活跃状态超时（默认为 60 秒）的时候，滚动文件
- `OnCheckpointRollingPolicy` 当checkpoint的时候滚动文件

写入方式

- `SimpleStringEncoder` 该方式只对`forRowFormat`格式存储
- `DayBulkWriter和BulkWriterFactory` `forBulkFormat` 格式的自定义程度较高，可以实现自己的FSDataOutputStream，写出各种格式

编码

```
DefaultRollingPolicy rollingPolicy = DefaultRollingPolicy
        .create()
    .withMaxPartSize(1024*1024*120) // 设置每个文件的最大大小 ,默认是128M。这里设置为120M
    .withRolloverInterval(Long.MAX_VALUE) // 滚动写入新文件的时间，默认60s。这里设置为无限大
    .withInactivityInterval(60*1000) // 60s空闲，就滚动写入新的文件
    .build();

StreamingFileSink<String> sink = StreamingFileSink
        .forRowFormat(new Path("..."),new SimpleStringEncode<String>("UTF-8"))
    //或者 .forBulkFormat(new Path("/xxx"), ParquetAvroWriters.forReflectRecord(Xxxx
    /**
        forRowFormat: 基于行存储
        forBulkFormat: 按列存储，批量编码方式，可以将输出结果用 Parquet 等格式进行压缩存储
    */
    .withBucketAssigner(new MemberBucketAssigner())
    .withRollingPolicy(rollingPolicy)
    .withBucketCheckInterval(1000L) // 桶检查间隔，这里设置为1s
    .build();

source.addSink(sink);

public class MemberBucketAssigner implements BucketAssigner<String, String>{
    @Override
    public String getBucketId(String info, Context context) {
        // 指定桶名 yyyy-mm-dd
        String[] array = info.split("\t");
        System.out.println(DateUtil.format(new Date(Long.valueOf(array[5])), "yyyy-M
        return DateUtil.format(new Date(Long.valueOf(array[5])), "yyyy-MM-dd");
    }
        @Override
    public SimpleVersionedSerializer<String> getSerializer() {
        return SimpleVersionedStringSerializer.INSTANCE;
    }
}
```

存在的问题

- 不支持写到hive
- 写到hdfs时，会产生大量小文件
- 当程序突然停止时，文件仍处于inprogress状态
- 默认桶下的文件名是 `part-{parallel-task}-{count}`。当程序重启时，从上次编号值加1继续开始。前提是程序是正常停止

两种方式的不同

- 源码位置

> BucketingSink 在 connector 下面，注重输出数据
>
> StreamingFileSink 在api 下面，注重与三方交互

- 版本

> BucketingSink 比较早就有了
>
> StreamingFileSink 是1.6版本推出的功能（据说是优化后推出的）

- 从写数据的方式

> BucketingSink 默认的Writer是StringWriter，也提供SequenceFileWriter（字符）
>
> StreamingFileSink 使用 OutputStream + Encoder 对外写数据 （字节）

- 从文件滚动策略来说

> BucketingSink 提供了时间、条数滚动
>
> StreamingFileSink 默认提供时间（官网有说条数，没看到 This is also configurable but the default policy rolls files based on file size and a timeout，自己实现BulkWriter可以）

借鉴

https://blog.csdn.net/anfuyi5792/article/details/102317286

https://blog.csdn.net/magic_kid_2010/article/details/96116321