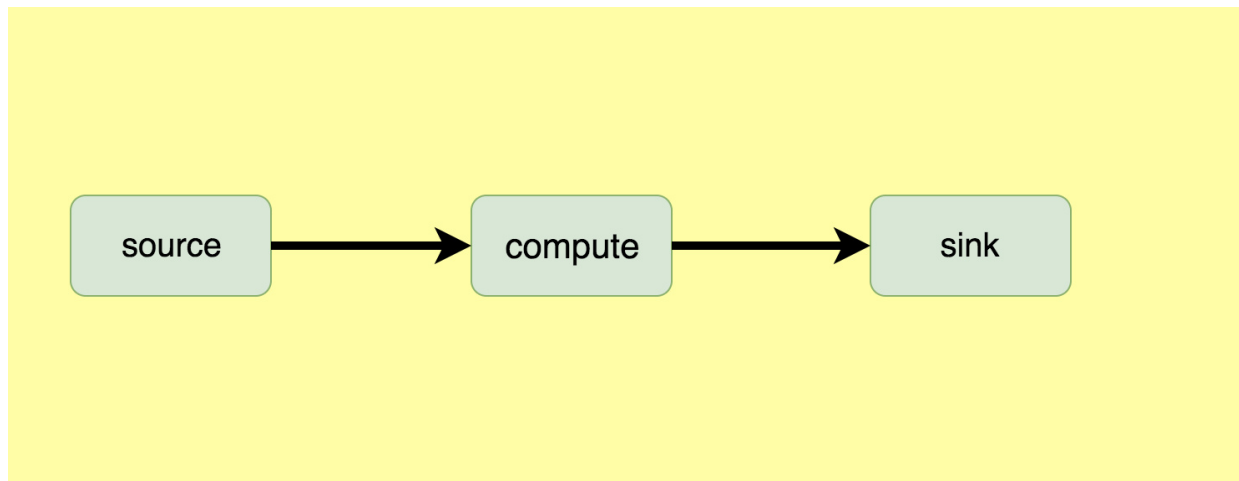


前言

通过前面我们可以知道 Flink Job 的大致结构就是 `source` \rightarrow `compute` \rightarrow `sink`。



那么这个 Source 是什么意思呢？我们下面来看看。

Data Source 介绍

Data Source 是什么呢？就字面意思其实就可以知道：数据来源。

Flink 作为一款流式计算框架，它可用来做批处理，即处理静态的数据集、历史的数据集；也可以用来做流处理，即处理实时的数据流（做计算操作），然后将处理后的数据实时下发，只要数据源源不断过来，Flink 就能够一直计算下去。

Flink 中你可以使用 `StreamExecutionEnvironment.addSource(sourceFunction)` 来为你的程序添加数据来源。

Flink 已经提供了若干实现好了的 source function，当然你也可以通过实现 `SourceFunction` 来自定义非并行的 source 或者实现 `ParallelSourceFunction` 接口或者扩展 `RichParallelSourceFunction` 来自定义并行的 source。

那么常用的 Data Source 有哪些呢？

常用的 Data Source

`StreamExecutionEnvironment` 中可以使用以下这些已实现的 stream source。

m % fromElements(OUT...)	DataStreamSource<OUT>
m % fromElements(Class<OUT>, OUT...)	DataStreamSource<OUT>
m % fromCollection(Collection<OUT>)	DataStreamSource<OUT>
m % fromCollection(Collection<OUT>, TypeInformation<OUT>)	DataStreamSource<OUT>
m % fromCollection(Iterator<OUT>, Class<OUT>)	DataStreamSource<OUT>
m % fromCollection(Iterator<OUT>, TypeInformation<OUT>)	DataStreamSource<OUT>
m % fromParallelCollection(SplittableIterator<OUT>, Class<OUT>)	DataStreamSource<OUT>
m % fromParallelCollection(SplittableIterator<OUT>, TypeInformation<OUT>)	DataStreamSource<OUT>
m % fromParallelCollection(SplittableIterator<OUT>, TypeInformation<OUT>, String)	DataStreamSource<OUT>
m % readTextFile(String)	DataStreamSource<String>
m % readTextFile(String, String)	DataStreamSource<String>
m % readFile(FileInputFormat<OUT>, String)	DataStreamSource<OUT>
m % readFile(FileInputFormat<OUT>, String, FileProcessingMode, long, FilePathFilter)	DataStreamSource<OUT>
m % readFile(FileInputFormat<OUT>, String, FileProcessingMode, long)	DataStreamSource<OUT>
m % readFileStream(String, long, WatchType)	DataStream<String>
m % readFile(FileInputFormat<OUT>, String, FileProcessingMode, long, TypeInformation<OUT>)	DataStreamSource<OUT>
m % socketTextStream(String, int, char, long)	DataStreamSource<String>
m % socketTextStream(String, int, String, long)	DataStreamSource<String>
m % socketTextStream(String, int, char)	DataStreamSource<String>
m % socketTextStream(String, int, String)	DataStreamSource<String>
m % socketTextStream(String, int)	DataStreamSource<String>
m % createInput(InputFormat<OUT>, ?>)	DataStreamSource<OUT>
m % createInput(InputFormat<OUT>, ?>, TypeInformation<OUT>)	DataStreamSource<OUT>
m % createInput(InputFormat<OUT>, ?>, TypeInformation<OUT>, String)	DataStreamSource<OUT>
m % createFileInput(FileInputFormat<OUT>, TypeInformation<OUT>, String, FileProcessingMode, long)	DataStreamSource<OUT>
m % addSource(SourceFunction<OUT>)	DataStreamSource<OUT>
m % addSource(SourceFunction<OUT>, String)	DataStreamSource<OUT>
m % addSource(SourceFunction<OUT>, TypeInformation<OUT>)	DataStreamSource<OUT>
m % addSource(SourceFunction<OUT>, String, TypeInformation<OUT>)	DataStreamSource<OUT>

总的来说可以分为下面几大类：

基于集合

- 1、fromCollection(Collection) - 从 Java 的 Java.util.Collection 创建数据流。集合中的所有元素类型必须相同。
- 2、fromCollection(Iterator, Class) - 从一个迭代器中创建数据流。Class 指定了该迭代器返回元素的类型。
- 3、fromElements(T ...) - 从给定的对象序列中创建数据流。所有对象类型必须相同。

```

1 | StreamExecutionEnvironment env =
   | StreamExecutionEnvironment.getExecutionEnvironment();
2 |
3 | DataStream<Event> input = env.fromElements(
4 |     new Event(1, "barfoo", 1.0),
5 |     new Event(2, "start", 2.0),
6 |     new Event(3, "foobar", 3.0),
7 |     ...
8 | );

```

- 4、fromParallelCollection(SplittableIterator, Class) - 从一个迭代器中创建并行数据流。Class 指定了该迭代器返回元素的类型。
- 5、generateSequence(from, to) - 创建一个生成指定区间范围内的数字序列的并行数据流。

基于文件

1、`readTextFile(path)` - 读取文本文件，即符合 `TextInputFormat` 规范的文件，并将其作为字符串返回。

```
1 final StreamExecutionEnvironment env =  
    StreamExecutionEnvironment.getExecutionEnvironment();  
2  
3 DataStream<String> text = env.readTextFile("file:///path/to/file");
```

2、`readFile(fileInputFormat, path)` - 根据指定的文件输入格式读取文件（一次）。

3、`readFile(fileInputFormat, path, watchType, interval, pathFilter, typeInfo)` - 这是上面两个方法内部调用的方法。它根据给定的 `fileInputFormat` 和读取路径读取文件。根据提供的 `watchType`，这个 `source` 可以定期（每隔 `interval` 毫秒）监测给定路径的新数据（`FileProcessingMode.PROCESS_CONTINUOUSLY`），或者处理一次路径对应文件的数据并退出（`FileProcessingMode.PROCESS_ONCE`）。你可以通过 `pathFilter` 进一步排除掉需要处理的文件。

```
1 final StreamExecutionEnvironment env =  
    StreamExecutionEnvironment.getExecutionEnvironment();  
2  
3 DataStream<MyEvent> stream = env.readFile(  
4     myFormat, myFilePath, FileProcessingMode.PROCESS_CONTINUOUSLY, 100,  
5     FilePathFilter.createDefaultFilter(), typeInfo);
```

实现:

在具体实现上，Flink 把文件读取过程分为两个子任务，即目录监控和数据读取。每个子任务都由单独的实体实现。目录监控由单个非并行（并行度为1）的任务执行，而数据读取由并行运行的多个任务执行。后者的并行性等于作业的并行性。单个目录监控任务的作用是扫描目录（根据 `watchType` 定期扫描或仅扫描一次），查找要处理的文件并把文件分割成切分片（`splits`），然后将这些切分片分配给下游 `reader`。`reader` 负责读取数据。每个切分片只能由一个 `reader` 读取，但一个 `reader` 可以逐个读取多个切分片。

重要注意:

如果 `watchType` 设置为 `FileProcessingMode.PROCESS_CONTINUOUSLY`，则当文件被修改时，其内容将被重新处理。这会打破“`exactly-once`”语义，因为在文件末尾附加数据将导致其所有内容被重新处理。

如果 `watchType` 设置为 `FileProcessingMode.PROCESS_ONCE`，则 `source` 仅扫描路径一次然后退出，而不等待 `reader` 完成文件内容的读取。当然 `reader` 会继续阅读，直到读取所有的文件内容。关闭 `source` 后就不会再有检查点。这可能导致节点故障后的恢复速度较慢，因为该作业将从最后一个检查点恢复读取。

基于 Socket

`socketTextStream(String hostname, int port)` - 从 `socket` 读取。元素可以用分隔符切分。

```

1 | StreamExecutionEnvironment env =
   | StreamExecutionEnvironment.getExecutionEnvironment();
2 |
3 | DataStream<Tuple2<String, Integer>> dataStream = env
4 |     .socketTextStream("localhost", 9999) // 监听 localhost 的 9999 端口过
      来的数据
5 |     .flatMap(new Splitter())
6 |     .keyBy(0)
7 |     .timeWindow(Time.seconds(5))
8 |     .sum(1);

```

自定义

addSource - 添加一个新的 source function。例如，你可以用 addSource(new FlinkKafkaConsumer011<>(...)) 从 Apache Kafka 读取数据。

说说上面几种的特点

- 1、基于集合：有界数据集，更偏向于本地测试用
- 2、基于文件：适合监听文件修改并读取其内容
- 3、基于 Socket：监听主机的 host port，从 Socket 中获取数据
- 4、自定义 addSource：大多数的场景数据都是无界的，会源源不断过来。比如去消费 Kafka 某个 topic 上的数据，这时候就需要用到这个 addSource，可能因为用的比较多的原因吧，Flink 直接提供了 FlinkKafkaConsumer011 等类可供你直接使用。你可以去看看 FlinkKafkaConsumerBase 这个基础类，它是 Flink Kafka 消费的最根本的类。

```

1 | StreamExecutionEnvironment env =
   | StreamExecutionEnvironment.getExecutionEnvironment();
2 |
3 | DataStream<KafkaEvent> input = env
4 |     .addSource(
5 |         new FlinkKafkaConsumer011<>(
6 |             parameterTool.getRequired("input-topic"), //从参数中获取传进来
      的 topic
7 |             new KafkaEventSchema(),
8 |             parameterTool.getProperties())
9 |         .assignTimestampsAndWatermarks(new
      CustomWatermarkExtractor()));

```

Flink 目前支持如下面常见的 Source：

</> Application Development

Basic API Concepts

Streaming (DataStream API)

Overview

Event Time

State & Fault Tolerance

Operators

Connectors

Overview

Fault Tolerance Guarantees

Kafka

Cassandra

Kinesis

Elasticsearch

Rolling File Sink

Streaming File Sink

RabbitMQ

NiFi

Twitter

Side Outputs

Python API

Testing

Experimental Features

Batch (DataSet API)

Table API & SQL

Data Types & Serialization

Managing Execution

Libraries

Best Practices

API Migration Guides

Deployment & Operations

Predefined Sources and Sinks

A few basic data sources and sinks are built into Flink and are always available. The predefined data sources include reading from files, directories, and sockets, and ingesting data from collections and iterators. The predefined data sinks support writing to files, to stdout and stderr, and to sockets.

Bundled Connectors

Connectors provide code for interfacing with various third-party systems. Currently these systems are supported:

- Apache Kafka (source/sink)
- Apache Cassandra (sink)
- Amazon Kinesis Streams (source/sink)
- Elasticsearch (sink)
- Hadoop FileSystem (sink)
- RabbitMQ (source/sink)
- Apache NiFi (source/sink)
- Twitter Streaming API (source)

Keep in mind that to use one of these connectors in an application, additional third party components are usually required, e.g. servers for the data stores or message queues. Note also that while the streaming connectors listed in this section are part of the Flink project and are included in source releases, they are not included in the binary distributions. Further instructions can be found in the corresponding subsections.

Connectors in Apache Bahir

Additional streaming connectors for Flink are being released through Apache Bahir, including:

- Apache ActiveMQ (source/sink)
- Apache Flume (sink)
- Redis (sink)
- Akka (sink)
- Netty (source)

如果你想自定义自己的 Source 呢？我们后面也有一篇文章专门讲解！[《11如何自定义 Flink Connectors \(Source 和 Sink\) ？》](#)

Data Sink 介绍

首先 Sink 的意思是：

yd sink



vi. 下沉；消沉；渗透

Search for 'yd sink' with default fallback



vt. 使下沉；挖掘；使低落

sɪŋk; [UK] sɪŋk; [US] sɪŋk

⌘2



n. 水槽；洗涤槽；污水坑

sɪŋk; [UK] sɪŋk; [US] sɪŋk

⌘3



n. (Sink)人名；(英、瑞典)辛克

sɪŋk; [UK] sɪŋk; [US] sɪŋk

⌘4



水槽

翻译结果

⌘5



网络释义

有道词典 for Alfred

⌘6



sink

下沉；盥洗盆；洗碗池；洗涤槽

⌘7



sink in

被了解；完全理解；被完全理解；被理解

⌘8



advise sink

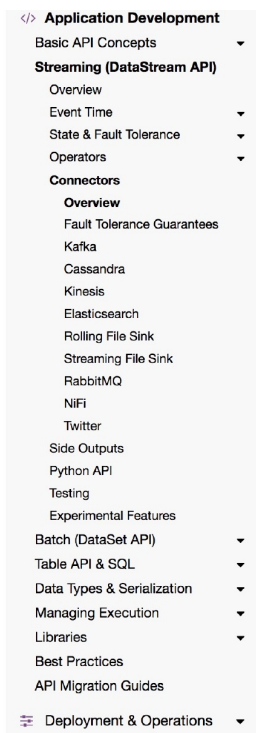
通知接收；告诉接受；表示已连接的通知接收

⌘9

大概可以猜到了吧！Data sink 有点把数据存储下来（落库）的意思。Flink 在拿到数据后做一系列的计算后，最后要将计算的结果往下游发送。比如将数据存储到 MySQL、ElasticSearch、Cassandra，或者继续发往 Kafka、RabbitMQ 等消息队列，更或者直接调用其他的第三方应用服务（比如告警）。

常用的 Data Sink

上面介绍了 Flink Data Source 有哪些，这里也看看 Flink Data Sink 支持的有哪些。



Predefined Sources and Sinks

A few basic data sources and sinks are built into Flink and are always available. The predefined data sources include reading from files, directories, and sockets, and ingesting data from collections and iterators. The predefined data sinks support writing to files, to stdout and stderr, and to sockets.

Bundled Connectors

Connectors provide code for interfacing with various third-party systems. Currently these systems are supported:

- Apache Kafka (source/sink)
- Apache Cassandra (sink)
- Amazon Kinesis Streams (source/sink)
- Elasticsearch (sink)
- Hadoop FileSystem (sink)
- RabbitMQ (source/sink)
- Apache NiFi (source/sink)
- Twitter Streaming API (source)

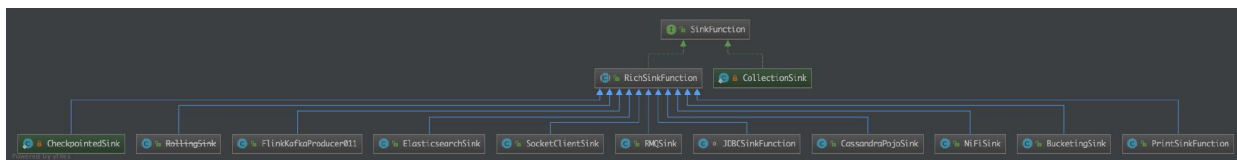
Keep in mind that to use one of these connectors in an application, additional third-party components are usually required, e.g. servers for the data stores or message queues. Note also that while the streaming connectors listed in this section are part of the Flink project and are included in source releases, they are not included in the binary distributions. Further instructions can be found in the corresponding subsections.

Connectors in Apache Bahir

Additional streaming connectors for Flink are being released through Apache Bahir, including:

- Apache ActiveMQ (source/sink)
- Apache Flume (sink)
- Redis (sink)
- Akka (sink)
- Netty (source)

再看下源码有哪些呢？



可以看到有 Kafka、ElasticSearch、Socket、RabbitMQ、JDBC、Cassandra POJO、File、Print 等 Sink 的方式。

可能自带的这些 Sink 不支持你的业务场景，那么你也可以自定义符合自己公司业务需求的 Sink，同样在后面的文章 [《11如何自定义 Flink Connectors \(Source 和 Sink\) ？》](#) 中将教会大家。

总结

本篇文章介绍了 Flink Data Source 和 Data Sink，并且介绍了 Flink 里面自带的 Data Source 和 Data Sink 方法，有些是比较通用的，可以直接用到我们项目里面去，有些可能还不会满足我们公司自己的业务场景，需要大家做一定的修改，另外后面我也会教大家如何去自定义 Data Source 和 Data Sink，以便大家能够更灵活地定义符合自己公司的业务场景需求。