

Flink 安装部署文档

本文主要介绍如何将 Flink standalone 模式安装及架构。

Apache Flink 是一个面向分布式数据流处理和批量数据处理的开源计算平台，它能够基于同一个 Flink 运行时 提供支持流处理和批处理两种类型应用的功能。

现有的开源计算方案，会把流处理和批处理作为两种不同的应用类型，因为它们所提供的 SLA (Service-Level-Agreement) 是完全不相同的：流处理一般需要支持低延迟、Exactly-once 保证，而批处理需要支持高吞吐、高效处理。

Flink 从另一个视角看待流处理和批处理，将二者统一起来：Flink 是完全支持流处理，也就是说作为流处理看待时输入数据流是无界的；**批处理被作为一种特殊的流处理，只是它的输入数据流被定义为有界的。**

Flink 流处理特性：

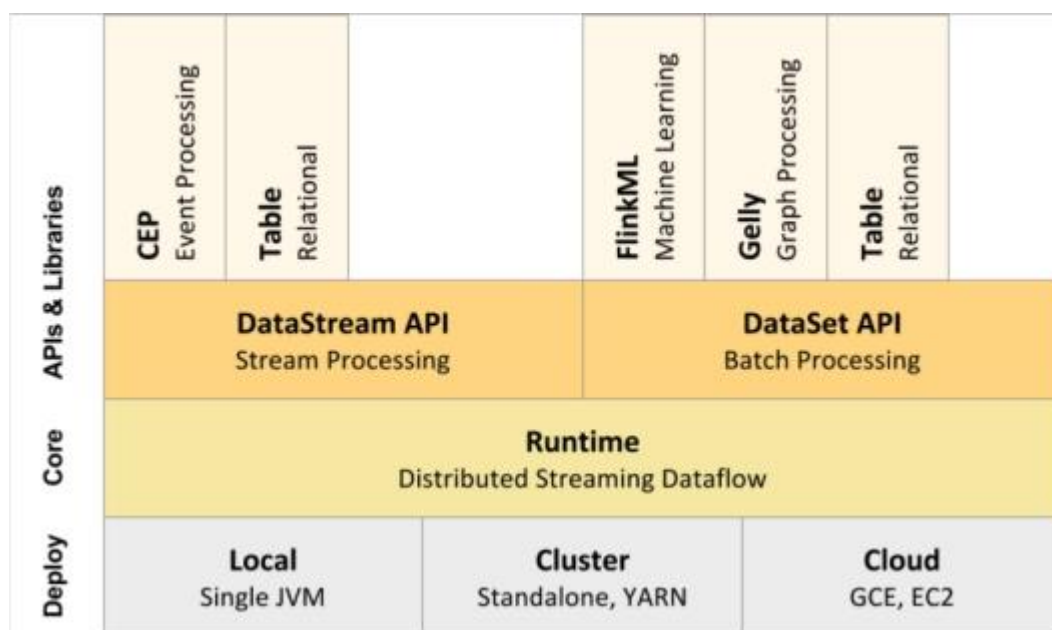
1. 支持高吞吐、低延迟、高性能的流处理
2. 支持带有事件时间的窗口 (Window) 操作
3. 支持有状态计算的 Exactly-once 语义 (需要自己自己保证 sink 的 eos)
4. 支持高度灵活的窗口 (Window) 操作，支持基于 time、count、session，以及 data-driven 的窗口操作
5. 支持具有 Backpressure 功能的持续流模型
6. 支持基于轻量级分布式快照 (Snapshot) 实现的容错
7. 一个运行时同时支持 Batch on Streaming 处理和 Streaming 处理
8. Flink 在 JVM 内部实现了自己的内存管理

9. 支持迭代计算

10. 支持程序自动优化：避免特定情况下 Shuffle、排序等昂贵操作，中间结果有必要进行缓存

一、架构

Flink 以层级式系统形式组件其软件栈，不同层的栈建立在其下层基础上，并且各层接受程序不同层的抽象形式。



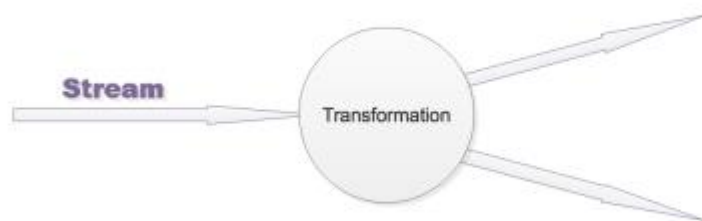
1. 运行时层以 JobGraph 形式接收程序。JobGraph 即为一个一般化的并行数据流图 (data flow)，它拥有任意数量的 Task 来接收和产生 data stream。
2. DataStream API 和 DataSet API 都会使用单独编译的处理方式生成 JobGraph。DataSet API 使用 optimizer 来决定针对程序的优化方法，而 DataStream API 则使用 stream builder 来完成该任务。
3. 在执行 JobGraph 时，Flink 提供了多种候选部署方案（如 standalone，YARN，Mesos 等）。

4. Flink 附随了一些产生 DataSet 或 DataStream API 程序的类库和 API：处理逻辑表查询的 Table，机器学习的 FlinkML，图像处理的 Gelly，复杂事件处理的 CEP。

二、原理

1. 流、转换、操作符

Flink 程序是由 Stream 和 Transformation 这两个基本构建块组成，其中 Stream 是一个中间结果数据，而 Transformation 是一个操作，它对一个或多个输入 Stream 进行计算处理，输出一个或多个结果 Stream。



Flink 程序被执行的时候，它会被映射为 Streaming Dataflow。一个 Streaming Dataflow 是由一组 Stream 和 Transformation Operator 组成，它类似于一个 DAG 图，在启动的时候从一个或多个 Source Operator 开始，结束于一个或多个 Sink Operator。

```

DataStream<String> lines = env.addSource(
    new FlinkKafkaConsumer<>(...));
DataStream<Event> events = lines.map((line) -> parse(line));
DataStream<Statistics> stats = events
    .keyBy("id")
    .timeWindow(Time.seconds(10))
    .apply(new MyWindowAggregationFunction());
stats.addSink(new RollingSink(path));

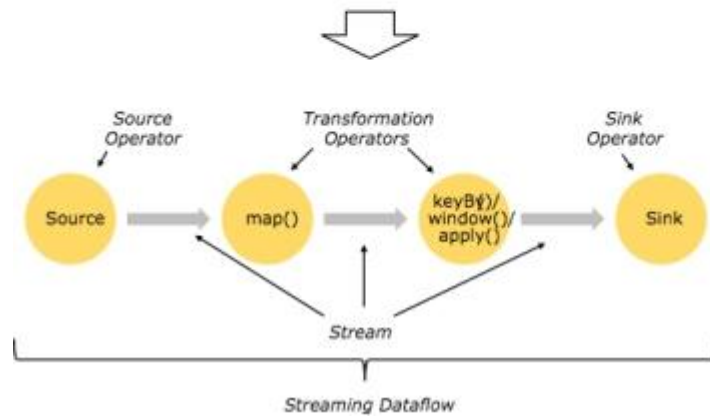
```

Source

Transformation

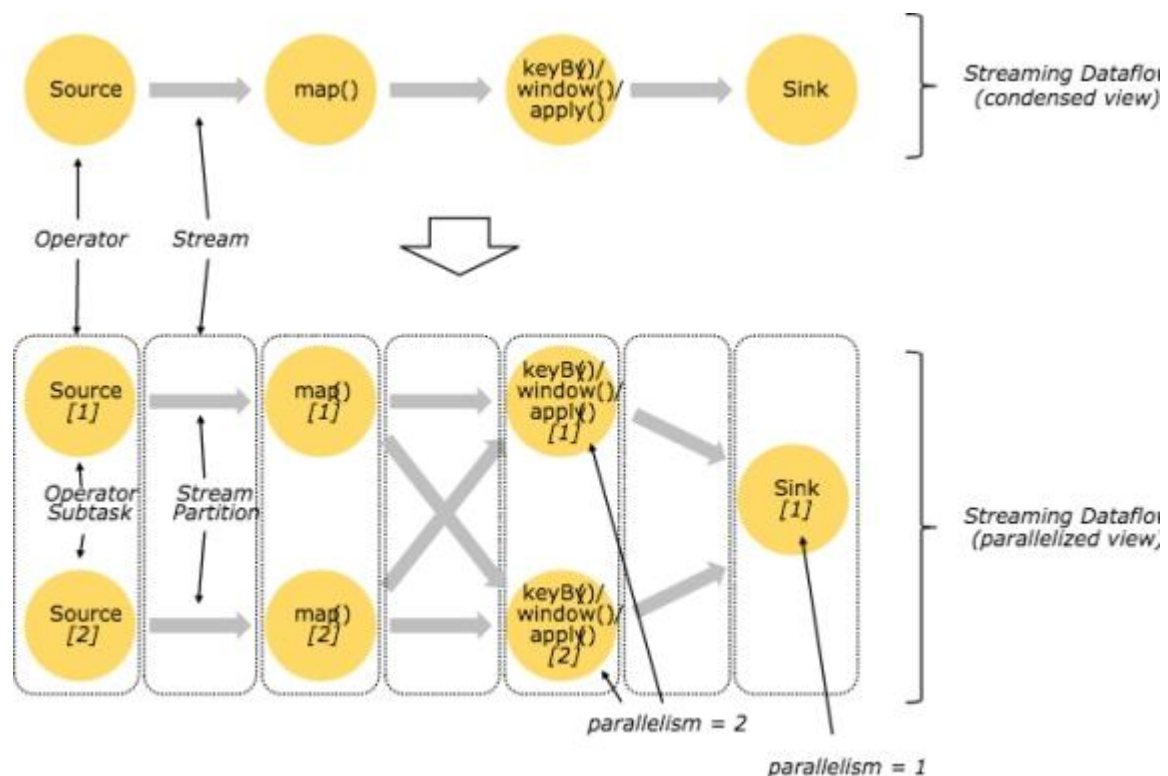
Transformation

Sink



2. 并行数据流

一个 Stream 可以被分成多个 Stream 分区(Stream Partitions),一个 Operator 可以被分成多个 Operator Subtask , 每一个 Operator Subtask 是在不同的线程中独立执行的。一个 Operator 的并行度, 等于 Operator Subtask 的个数, 一个 Stream 的并行度总是等于生成它的 Operator 的并行度。



One-to-one 模式

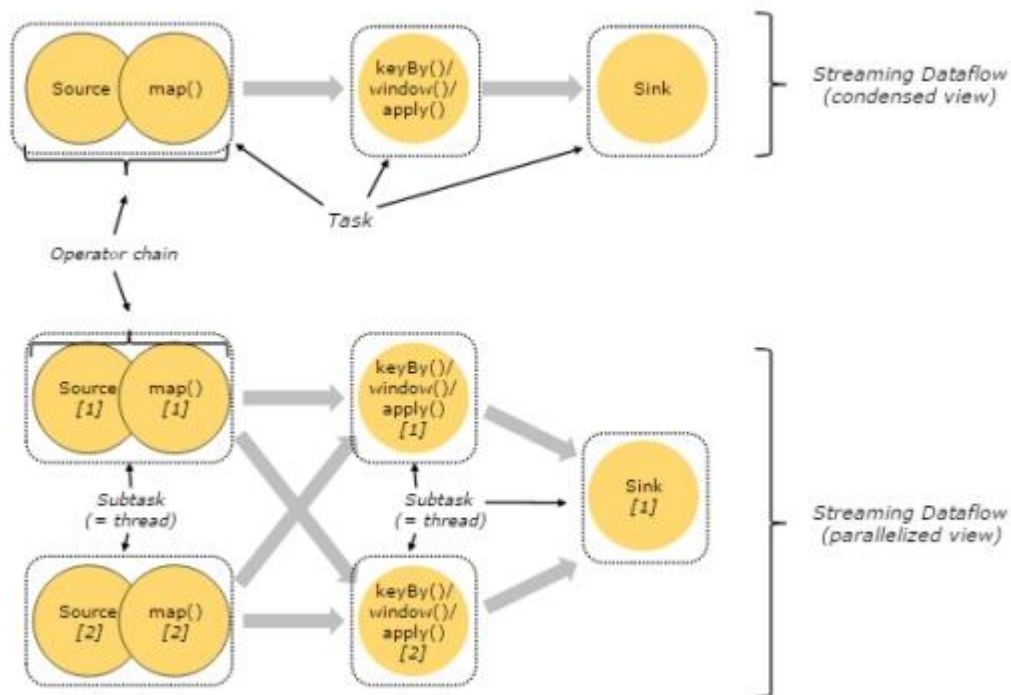
比如从 Source[1]到 map()[1]，它保持了 Source 的分区特性 (Partitioning) 和分区内元素处理的有序性，也就是说 map()[1]的 Subtask 看到数据流中记录的顺序，与 Source[1]中看到的记录顺序是一致的。

Redistribution 模式

这种模式改变了输入数据流的分区，比如从 map()[1]、map()[2]到 keyBy()/window()/apply()[1]、keyBy()/window()/apply()[2]，上游的 Subtask 向下游的多个不同的 Subtask 发送数据，改变了数据流的分区，这与实际应用所选择的 Operator 有关系。

3. 任务、操作符链

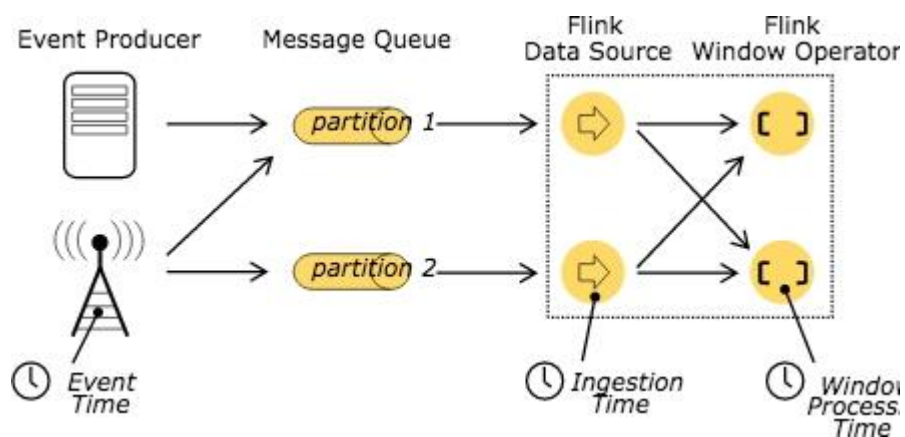
Flink 分布式执行环境中，会将多个 Operator Subtask 串起来组成一个 Operator Chain，实际上就是一个执行链，每个执行链会在 TaskManager 上一个独立的线程中执行。



4. 时间

处理 Stream 中的记录时，记录中通常会包含各种典型的时间字段：

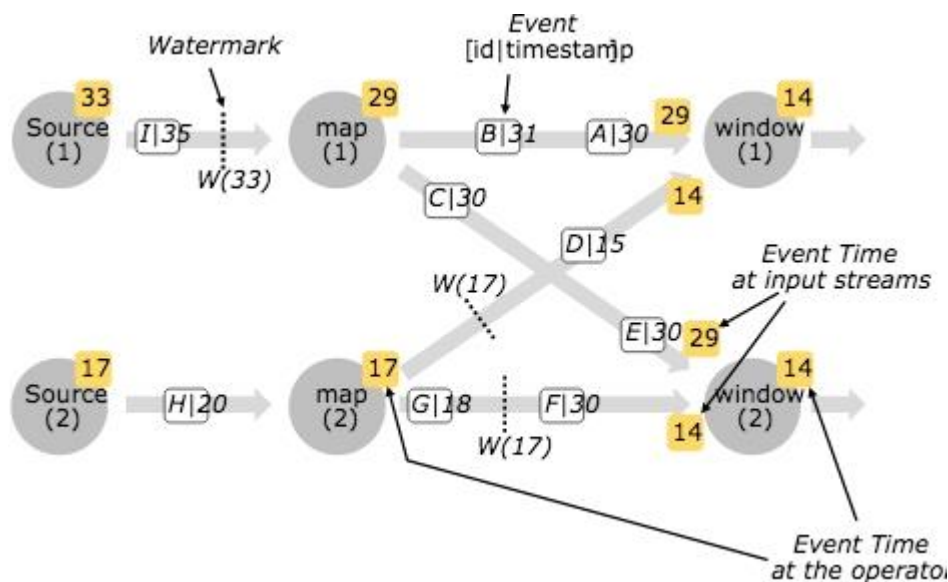
1. Event Time：表示事件创建时间
2. Ingestion Time：表示事件进入到 Flink Dataflow 的时间
3. Processing Time：表示某个 Operator 对事件进行处理的本地系统时间



Flink 使用 WaterMark 衡量时间的时间，WaterMark 携带时间戳 t ，并被插入到 stream 中。

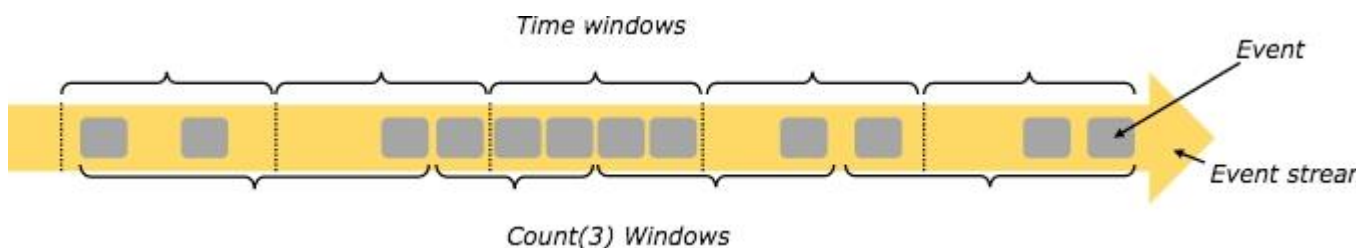
1. WaterMark 的含义是所有时间 $t' < t$ 的事件都已经发生。

2. 针对乱序的流，WaterMark 至关重要，这样可以允许一些事件到达延迟，而不至于过于影响 window 窗口的计算。
3. 并行数据流中，当 Operator 有多个输入流时，Operator 的 event time 以最小流 event time 为准。



5. 窗口

Flink 支持基于时间窗口操作，也支持基于数据的窗口操作：



窗口分类：

1. 按分割标准划分：timeWindow、countWindow
2. 按窗口行为划分：Tumbling Window、Sliding Window、自定义窗口

Tumbling/Sliding Time Window

```
// Stream of (sensorId, carCnt)
val vehicleCnts: DataStream[(Int, Int)] = ...

val tumblingCnts: DataStream[(Int, Int)] = vehicleCnts
```

```

// key stream by sensorId
.keyBy(0)
// tumbling time window of 1 minute length
.timeWindow(Time.minutes(1))
// compute sum over carCnt
.sum(1)

val slidingCnts: DataStream[(Int, Int)] = vehicleCnts
  .keyBy(0)
  // sliding time window of 1 minute length and 30 secs
  trigger interval
  .timeWindow(Time.minutes(1), Time.seconds(30))
  .sum(1)

```

Tumbling/Sliding Count Window

```

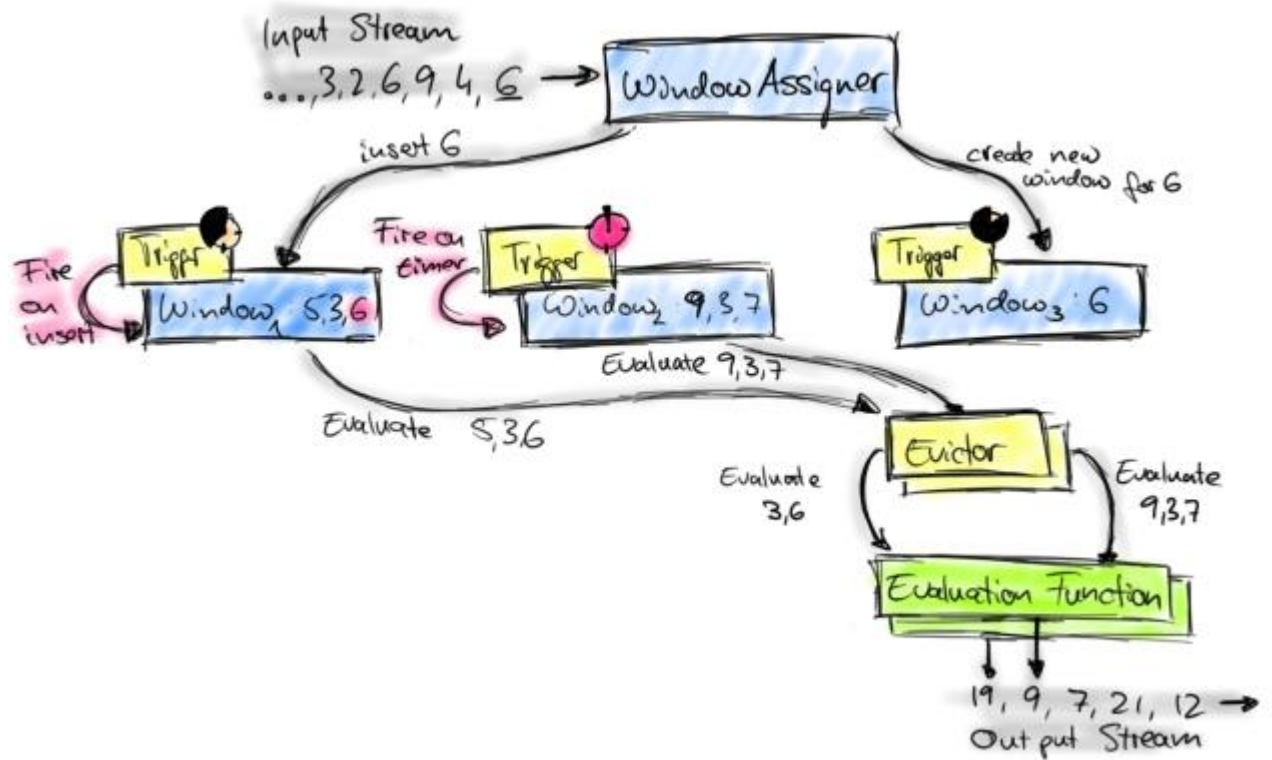
// Stream of (sensorId, carCnt)
val vehicleCnts: DataStream[(Int, Int)] = ...

val tumblingCnts: DataStream[(Int, Int)] = vehicleCnts
  // key stream by sensorId
  .keyBy(0)
  // tumbling count window of 100 elements size
  .countWindow(100)
  // compute the carCnt sum
  .sum(1)

val slidingCnts: DataStream[(Int, Int)] = vehicleCnts
  .keyBy(0)
  // sliding count window of 100 elements size and 10
  elements trigger interval
  .countWindow(100, 10)
  .sum(1)

```

自定义窗口

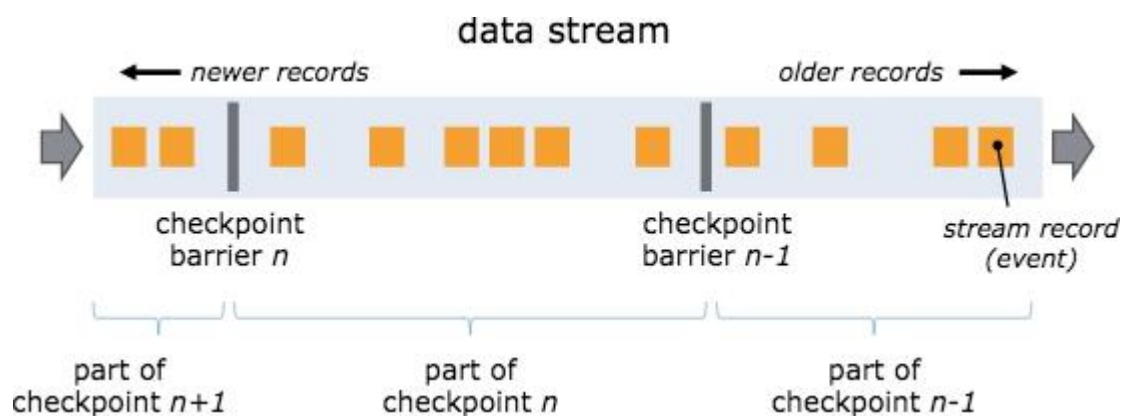


基本操作：

1. window：创建自定义窗口
2. trigger：自定义触发器
3. evictor：自定义 evictor
4. apply：自定义 window function

6. 容错

Barrier 机制：

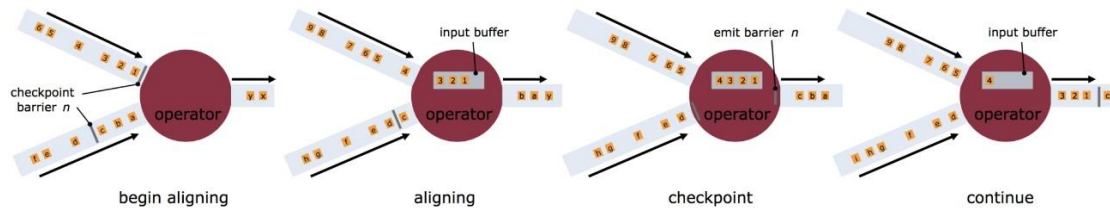


1. 出现一个 Barrier，在该 Barrier 之前出现的记录都属于该 Barrier 对应的 Snapshot，在该 Barrier 之后出现的记录属于下一个 Snapshot。
2. 来自不同 Snapshot 多个 Barrier 可能同时出现在数据流中，也就是说同一个时刻可能并发生成多个 Snapshot。
3. 当一个中间（Intermediate）Operator 接收到一个 Barrier 后，它会发送 Barrier 到属于该 Barrier 的 Snapshot 的数据流中，等到 Sink Operator 接收到该 Barrier 后会向 Checkpoint Coordinator 确认该 Snapshot，直到所有的 Sink Operator 都确认了该 Snapshot，才被认为完成了该 Snapshot。

对齐：

当 Operator 接收到多个输入的数据流时，需要在 Snapshot Barrier 中对数据流进行排列对齐：

1. Operator 从一个 incoming Stream 接收到 Snapshot Barrier n，然后暂停处理，直到其它的 incoming Stream 的 Barrier n（否则属于 2 个 Snapshot 的记录就混在一起了）到达该 Operator
2. 接收到 Barrier n 的 Stream 被临时搁置，来自这些 Stream 的记录不会被处理，而是被放在一个 Buffer 中。
3. 一旦最后一个 Stream 接收到 Barrier n，Operator 会 emit 所有暂存在 Buffer 中的记录，然后向 Checkpoint Coordinator 发送 Snapshot n。
4. 继续处理来自多个 Stream 的记录

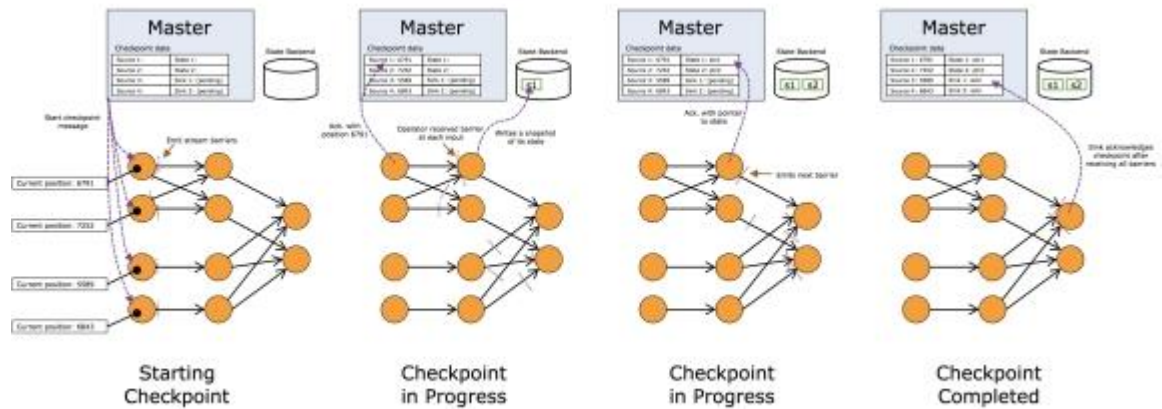


基于 Stream Aligning 操作能够实现 Exactly Once 语义，但是也会给流处理应用带来延迟，因为为了排列对齐 Barrier，会暂时缓存一部分 Stream 的记录到 Buffer 中，尤其是在数据流并行度很高的场景下可能更加明显，通常以最迟对齐 Barrier 的一个 Stream 为处理 Buffer 中缓存记录的时刻点。在 Flink 中，提供了一个开关，选择是否使用 Stream Aligning，如果关掉则 Exactly Once 会变成 At least once。

Checkpoint :

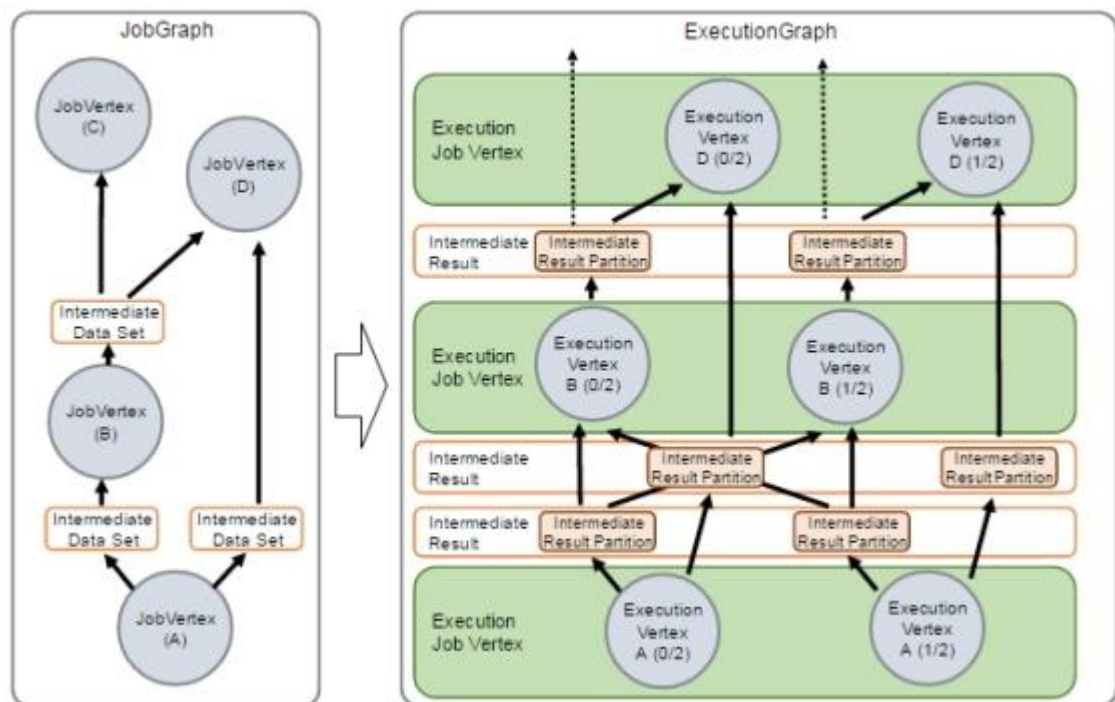
Snapshot 并不仅仅是对数据流做了一个状态的 Checkpoint，它也包含了一个 Operator 内部所持有的状态，这样能够在保证在流处理系统失败时能够正确地恢复数据流处理。状态包含两种：

1. 系统状态：一个 Operator 进行计算处理的时候需要对数据进行缓冲，所以数据缓冲区的状态是与 Operator 相关联的。以窗口操作的缓冲区为例，Flink 系统会收集或聚合记录数据并放到缓冲区中，直到该缓冲区中的数据被处理完成。
2. 一种是用户自定义状态（状态可以通过转换函数进行创建和修改），它可以是函数中的 Java 对象这样的简单变量，也可以是与函数相关的 Key/Value 状态。



7. 调度

在 JobManager 端，会接收到 Client 提交的 JobGraph 形式的 Flink Job，JobManager 会将一个 JobGraph 转换映射为一个 ExecutionGraph，ExecutionGraph 是 JobGraph 的并行表示，也就是实际 JobManager 调度一个 Job 在 TaskManager 上运行的逻辑视图。



物理上进行调度，基于资源的分配与使用的一个例子：

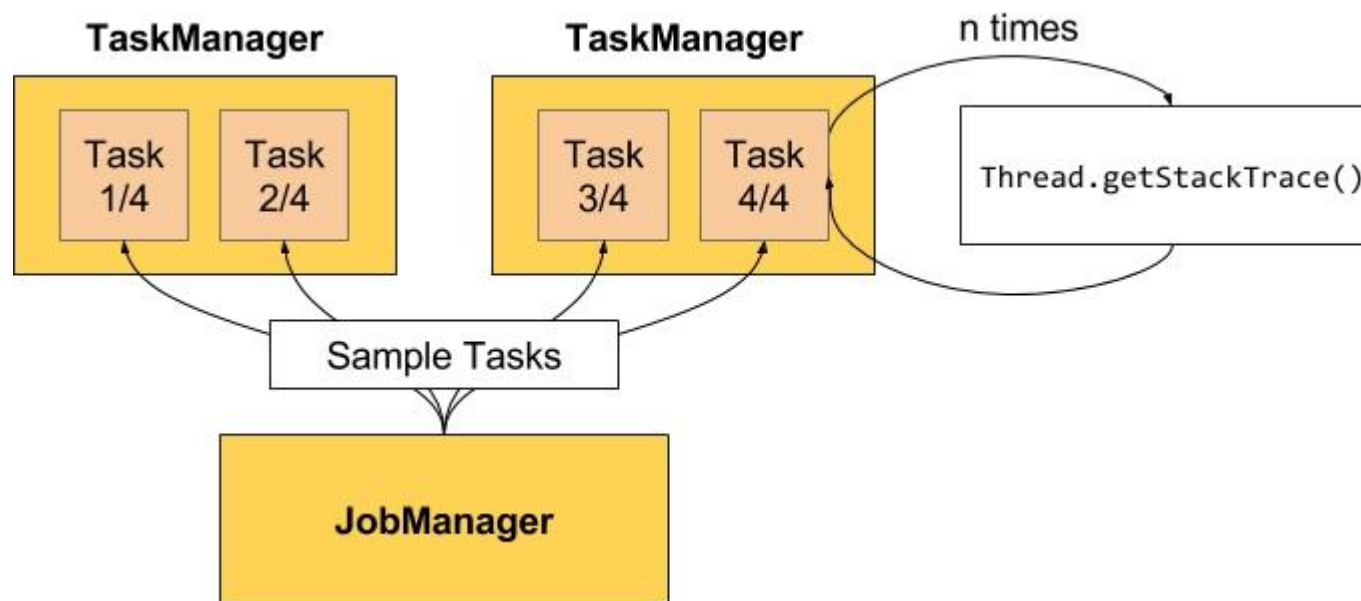
1. 左上子图：有 2 个 TaskManager，每个 TaskManager 有 3 个 Task Slot

2. 左下子图：一个 Flink Job，逻辑上包含了 1 个 data source、1 个 MapFunction、1 个 ReduceFunction，对应一个 JobGraph
3. 左下子图：用户提交的 Flink Job 对各个 Operator 进行的配置——data source 的并行度设置为 4，MapFunction 的并行度也为 4，ReduceFunction 的并行度为 3，在 JobManager 端对应于 ExecutionGraph
4. 右上子图：TaskManager 1 上，有 2 个并行的 ExecutionVertex 组成的 DAG 图，它们各占用一个 Task Slot
5. 右下子图：TaskManager 2 上，也有 2 个并行的 ExecutionVertex 组成的 DAG 图，它们也各占用一个 Task Slot
6. 在 2 个 TaskManager 上运行的 4 个 Execution 是并行执行的

8. [Back Pressure 监控](#)

流处理系统中，当下游 Operator 处理速度跟不上的情况，如果下游 Operator 能够将自己处理状态传播给上游 Operator，使得上游 Operator 处理速度慢下来就会缓解上述问题，比如通过告警的方式通知现有流处理系统存在的问题。

Flink Web 界面上提供了对运行 Job 的 Backpressure 行为的监控,它通过使用 Sampling 线程对正在运行的 Task 进行堆栈跟踪采样来实现。



默认情况下, JobManager 会每间隔 50ms 触发对一个 Job 的每个 Task 依次进行 100 次堆栈跟踪调用, 过计算得到一个比值, 例如, $ratio=0.01$, 表示 100 次中仅有 1 次方法调用阻塞。Flink 目前定义了如下 Backpressure 状态:

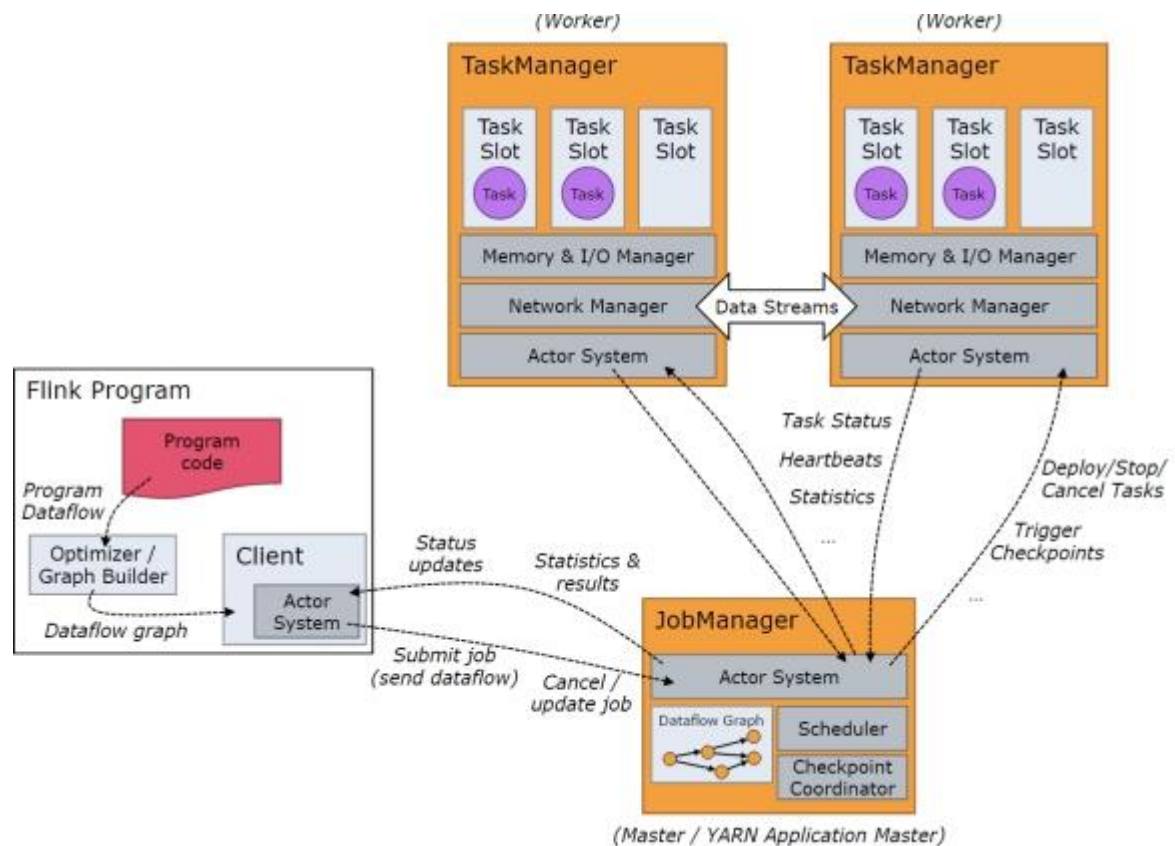
OK: $0 \leq Ratio \leq 0.10$

LOW: $0.10 < Ratio \leq 0.5$

HIGH: $0.5 < Ratio \leq 1$

三、部署

当 Flink 系统启动时, 首先启动 JobManager 和一至多个 TaskManager。JobManager 负责协调 Flink 系统, TaskManager 则是执行并行程序的 worker。当一个程序被提交后, 系统会创建一个 Client 来进行预处理, 将程序转变成一个并行数据流的形式, 交给 JobManager 和 TaskManager 执行。



1. flink 下载

<http://flink.apache.org/downloads.html> 下载最新版本的 flink 程序，适配

hadoop 版本

Latest stable release (v1.5.0)

Apache Flink® 1.5.0 is our latest stable release.

An Apache Hadoop installation is **not required** to use Flink. If you plan to run Flink in YARN or process data stored in HDFS then select the version matching your installed Hadoop version.

The binary releases marked with a Hadoop version come bundled with binaries for that Hadoop version. The binary release without bundled Hadoop can be used without Hadoop or with a Hadoop version that is installed in the environment, i.e., this version can pick up a Hadoop version from the classpath.

Binaries

Scala 2.11

Without bundled Hadoop®	Download (asc, sha512)
Hadoop® 2.8	Download (asc, sha512)
Hadoop® 2.7	Download (asc, sha512)
Hadoop® 2.6	Download (asc, sha512)
Hadoop® 2.4	Download (asc, sha512)

Source:

2. 安装配置

Java 版本要求 java8

```
[root@hadoop11 ~]# java -version
openjdk version "1.8.0_171"
OpenJDK Runtime Environment (build 1.8.0_171-b10)
OpenJDK 64-Bit Server VM (build 25.171-b10, mixed mode)
[root@hadoop11 ~]#
```

tar -zxvf flink-1.4.2-bin-hadoop26-scala_2.11.tar -C /usr/tass 解压

```
[root@hadoop11 flink-1.4.2]# pwd
/usr/tass/flink-1.4.2
[root@hadoop11 flink-1.4.2]#
```

修改配置文件

vi \$FLINK_HOME/conf/flink_conf.yaml

```
#=====
# Common @see https://ci.apache.org/projects/flink/flink-
docs-release-1.4/ops/config.html#history-server
#=====
# $JAVA_HOME
env.java.home: /usr/java/jdk8
# JOBMANAGER
env.java.opts.jobmanager: -Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.a
uthenticate=false -Dcom.sun.management.jmxremote.port=11111
# TASKMANAGER
env.java.opts.taskmanager: -Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.
authenticate=false -Dcom.sun.management.jmxremote.port=22222
# 一般性设置
jobmanager.rpc.port: 6123
```



```
jobmanager.heap.mb: 2048
taskmanager.heap.mb: 4096
# 每个taskmanager 可执行的task 数量 此参数在yarn 下无效
taskmanager.numberOfTaskSlots: 20
taskmanager.memory.preallocate: false
jobmanager.web.checkpoints.history: 10
parallelism.default: 3

#=====
=====
# Web Frontend
#=====
=====
jobmanager.web.port: 8081

#=====
=====
# Streaming 状态保存
#=====
=====
state.backend: rocksdb
state.backend.fs.checkpointdir: hdfs:///flink/checkpoint
state.checkpoints.dir: hdfs:///flink/savepoint/
#=====
=====
# Advanced
#=====
=====
taskmanager.network.numberofbuffers: 4096
fs.hdfs.hadoopconf: /usr/tass/hadoop-2.5.0-cdh5.3.0/etc/hadoop

#=====
=====
# 高可用配置
#=====
=====
high-availability: zookeeper
high-availability.zookeeper.quorum:
kafkazoo1:2181,kafkazoo2:2181,kafkazoo3:2181
high-availability.zookeeper.path.root: /flink
high-availability.cluster-id: /cluster_common # important:
customize per cluster
high-availability.zookeeper.storageDir: hdfs:///flink/ha
```

使用 logback 而非 log4j

Vi \$FLINK_HOME/conf/logback.xml

```
<configuration>

    <appender name="file"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${log.file}</file>
        <encoder>
            <pattern>
                %d{HH:mm:ss.SSS} %-5level %logger{60}[%line]
- %msg%n
            </pattern>
            <charset>UTF-8</charset>
        </encoder>

        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>/usr/tass/flink/logback/%d{yyyy-MM-
dd}.%i.log</fileNamePattern>
            <maxHistory>7</maxHistory>
            <totalSizeCap>1GB</totalSizeCap>
            <timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>100MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
        </rollingPolicy>
    </appender>
    <root level="INFO">
        <appender-ref ref="file"/>
    </root>

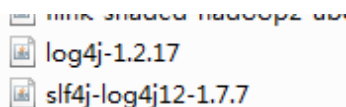
    <logger name="org.apache.flink" level="INFO"
additivity="false" >
        <appender-ref ref="file"/>
    </logger>
    <logger
name="org.apache.flink.runtime.state.DefaultOperatorStateBacke
nd" additivity="false" level="WARN">
        <appender-ref ref="file"/>
    </logger>
</logger>
```

```

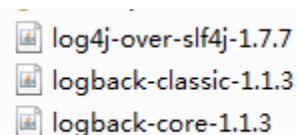
name="org.apache.flink.runtime.checkpoint.CheckpointCoordinator" additivity="false" level="WARN">
    <appender-ref ref="file"/>
</logger>
<logger name="akka" level="INFO" additivity="false">
    <appender-ref ref="file"/>
</logger>
<logger name="org.apache.kafka" level="INFO"
additivity="false">
    <appender-ref ref="file"/>
</logger>
<logger name="org.apache.hadoop" level="WARN"
additivity="false">
    <appender-ref ref="file"/>
</logger>
<logger name="org.apache.zookeeper" level="WARN"
additivity="false">
    <appender-ref ref="file"/>
</logger>
<logger
name="org.apache.flink.shaded.akka.org.jboss.netty.channel.DefaultChannelPipeline" additivity="false" level="ERROR">
    <appender-ref ref="file"/>
</logger>
</configuration>

```

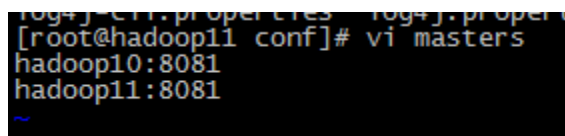
移除\$FLINK_HOME/lib 下



添加



Vi \$FLINK_HOME/conf/masters 指定jobmanager



Task Managers

Job Manager

Submit new Job

ev-flink-test_20180625165011.jar

ev-flink-test_20180625165011.jar

Add New +

上传jar包

ev-flink-test_20180625165011.jar 2018-06-25, 16:51:34

Entry Class

程序main方法

Parallelism

并行度

Program Arguments

main方法参数

Savepoint Path

savepoint 指定从savepoint恢复程序

Allow Non Restored State

允许没有的状态恢复

Show Plan

Submit

4. 项目编写

项目使用 maven 管理

常规引入 flink, hadoop, hbase, redis, mysql 等 jar 包, 移除全部 log4j 依赖, 添加 logback 依赖。

插件

```
<build>
  <plugins>
    <plugin>
      <groupId>net.alchim31.maven</groupId>
      <artifactId>scala-maven-plugin</artifactId>
      <version>3.2.2</version>
      <configuration>
        <recompileMode>incremental</recompileMode>
        <args>
          <arg>-deprecation</arg>
          <arg>-explaintypes</arg>
          <!--<arg>-target:jvm-1.7</arg>-->
        </args>
      </configuration>
      <executions>
        <execution>
          <id>scala-compile-first</id>
          <phase>process-resources</phase>
          <goals>
```

```

        <goal>add-source</goal>
        <goal>compile</goal>
    </goals>
</execution>
<execution>
    <id>scala-test-compile</id>
    <phase>process-test-resources</phase>
    <goals>
        <goal>add-source</goal>
        <goal>testCompile</goal>
    </goals>
</execution>
</executions>
</plugin>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-dependency-plugin</artifactId>
    <version>2.10</version>
    <executions>
        <execution>
            <id>set-properties</id>
            <goals>
                <goal>properties</goal>
            </goals>
        </execution>
    </executions>
</plugin>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.5</version>
    <configuration>
        <source>1.8</source>
        <target>1.8</target>
    </configuration>

```

```

</plugin>
<!-- We use the maven-shade plugin to create a fat jar
that contains all dependencies
    except flink and it's transitive dependencies. The
resulting fat-jar can be executed
    on a cluster. Change the value of Program-Class if your

```

```

program entry point changes. -->
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>2.4.1</version>
        <executions>

            <!-- Run shade goal on package phase -->
            <execution>
                <phase>package</phase>
                <goals>
                    <goal>shade</goal>
                </goals>
                <configuration>
                    <artifactSet>
                        <excludes>
                            <!-- This list contains all
dependencies of flink-dist
into the fat-jar
                            -->
                            <exclude>log4j:log4j</exclude>
                            <exclude>ch.qos.logback:logback-
classic</exclude>
                            <exclude>ch.qos.logback:logback-
core</exclude>
                            <exclude>org.apache.flink:flink-
annotations</exclude>
                            <exclude>org.apache.flink:flink-
shaded-hadoop2</exclude>
                            <exclude>org.apache.flink:flink-
shaded-curator-recipes</exclude>
                            <exclude>org.apache.flink:flink-
core</exclude>
                            <exclude>org.apache.flink:flink-
java</exclude>
                            <exclude>org.apache.flink:flink-
scala_${scala.version}</exclude>
                            <exclude>org.apache.flink:flink-
runtime_${scala.version}</exclude>
                            <exclude>org.apache.flink:flink-
optimizer_${scala.version}</exclude>
                            <exclude>org.apache.flink:flink-
clients_${scala.version}</exclude>

```

```

        <exclude>org.apache.flink:flink-
avro_${scala.version}</exclude>
        <exclude>org.apache.flink:flink-
examples-batch_${scala.version}</exclude>
        <exclude>org.apache.flink:flink-
examples-streaming_${scala.version}</exclude>
        <exclude>org.apache.flink:flink-
streaming-java_${scala.version}</exclude>
        <exclude>org.apache.flink:flink-
streaming-scala_${scala.version}</exclude>
        <exclude>org.apache.flink:flink-
scala-shell_${scala.version}</exclude>
        <exclude>org.apache.flink:flink-
python</exclude>
        <exclude>org.apache.flink:flink-
metrics-core</exclude>
        <exclude>org.apache.flink:flink-
metrics-jmx</exclude>
        <exclude>org.apache.flink:flink-
statebackend-rocksdb_${scala.version}</exclude>
        <exclude>org.apache.flink:flink-
runtime-web_${scala.version}</exclude>

        <!-- Also exclude very big
transitive dependencies of Flink

        WARNING: You have to remove these
excludes if your code relies on other
versions of these dependencies.

-->
        <exclude>org.scala-lang:scala-
library</exclude>
        <exclude>org.scala-lang:scala-
compiler</exclude>
        <exclude>org.scala-lang:scala-
reflect</exclude>
        <exclude>com.data-artisans:flakka-
actor_*</exclude>
        <exclude>com.data-artisans:flakka-
remote_*</exclude>
        <exclude>com.data-artisans:flakka-
slf4j_*</exclude>
        <exclude>io.netty:netty-

```



```

all</exclude>
                                <exclude>io.netty:netty</exclude>
                                <exclude>commons-fileupload:commons-
fileupload</exclude>

<exclude>org.apache.avro:avro</exclude>
                                <exclude>commons-
collections:commons-collections</exclude>

<exclude>com.thoughtworks.paranamer:paranamer</exclude>
                                <exclude>org.xerial.snappy:snappy-
java</exclude>
                                <exclude>org.apache.commons:commons-
compress</exclude>
                                <exclude>org.tukaani:xz</exclude>

<exclude>com.esotericsoftware.kryo:kryo</exclude>

<exclude>com.esotericsoftware.minlog:minlog</exclude>

<exclude>org.objenesis:objenesis</exclude>

<exclude>com.twitter:chill_*</exclude>
                                <exclude>com.twitter:chill-
java</exclude>
                                <exclude>commons-lang:commons-
lang</exclude>
                                <exclude>junit:junit</exclude>
                                <exclude>org.apache.commons:commons-
lang3</exclude>
                                <exclude>org.slf4j:slf4j-
api</exclude>
                                <exclude>org.slf4j:slf4j-
log4j12</exclude>
                                <exclude>org.apache.commons:commons-
math</exclude>

<exclude>org.apache.sling:org.apache.sling.commons.json</exclude>
                                <exclude>commons-logging:commons-
logging</exclude>
                                <exclude>commons-codec:commons-
codec</exclude>

<exclude>com.fasterxml.jackson.core:jackson-core</exclude>

```

```

<exclude>com.fasterxml.jackson.core:jackson-databind</exclude>

<exclude>com.fasterxml.jackson.core:jackson-annotations</exclude>
    <exclude>stax:stax-api</exclude>

<exclude>com.typesafe:config</exclude>

<exclude>org.uncommons.maths:uncommons-maths</exclude>

<exclude>com.github.scopt:scopt_*</exclude>
    <exclude>commons-io:commons-
io</exclude>
    <exclude>commons-cli:commons-
cli</exclude>
    </excludes>
</artifactSet>
<filters>
    <filter>

<artifact>org.apache.flink:*</artifact>
    <excludes>
        <!-- exclude shaded google but
include shaded curator -->

<exclude>org/apache/flink/shaded/com/**</exclude>
        <exclude>web-docs/**</exclude>
    </excludes>
</filter>
<filter>
    <!-- Do not copy the signatures in
the META-INF folder.
        Otherwise, this might cause
SecurityExceptions when using the JAR. -->
    <artifact>*:*</artifact>
    <excludes>
        <exclude>META-INF/*.SF</exclude>
        <exclude>META-
INF/*.DSA</exclude>
        <exclude>META-
INF/*.RSA</exclude>
    </excludes>
</filter>
</filters>

```

```

        <!-- If you want to use ./bin/flink run
<quickstart jar> uncomment the following lines.
        This will add a Main-Class entry to the
manifest file -->

        <transformers>
            <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestR
esourceTransformer">
                <mainClass>fixme</mainClass>
            </transformer>
        </transformers>

        <finalName>
            ${project.artifactId}-
${project.version}-${package.environment}_${timestamp}
        </finalName>

<createDependencyReducedPom>false</createDependencyReducedPom>
    </configuration>

    </execution>
</executions>
</plugin>

<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>buildnumber-maven-plugin</artifactId>
    <version>1.4</version>
    <configuration>
        <timestampFormat>yyyyMMddHHmmss</timestampFormat>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>create-timestamp</goal>
            </goals>
        </execution>
    </executions>
    <inherited>false</inherited>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>

```

```

        <artifactId>maven-resources-plugin</artifactId>
        <version>2.6</version>
        <executions>
            <execution>
                <id>copy-resources</id>
                <phase>compile</phase>
                <goals>
                    <goal>copy-resources</goal>
                </goals>

                <configuration>
                    <overwrite>true</overwrite>

<outputDirectory>${project.build.outputDirectory}</outputDirector
y>

                    <resources>
                        <resource>
                            <!-- 指定 resources 插件处理哪个目录下
的资源文件 -->
<directory>src/main/resources/${package.environment}</directory>
                            <filtering>>false</filtering>
                        </resource>
                    </resources>
                </configuration>
            </execution>
        </executions>
    </plugin>
</plugins>

```

主要使用的插件有 shade-jar ， 可以把 jar 包打成一个 fat jar。

配置文件 maven-resources-plugin， 根据环境打包配置文件。

```

<profiles>

    <profile>
        <id>local-mode</id>
        <properties>
            <flink.runtime.scope>compile</flink.runtime.scope>
        </properties>
    </profile>
    <profile>
        <id>dev</id>
        <activation>
            <activeByDefault>true</activeByDefault>

```

```
        </activation>
        <properties>
            <package.environment>dev</package.environment>
        </properties>
    </profile>
    <profile>
        <id>test</id>
        <properties>
            <package.environment>test</package.environment>
        </properties>
    </profile>
    <profile>
        <id>product</id>
        <properties>
            <package.environment>product</package.environment>
        </properties>
    </profile>
</profiles>
```

所以打包命令是

mvn clean package -Pdev -DskipTests

mvn clean package -Ptest -DskipTests

mvn clean package -Pproduct -DskipTests