深入浅出Flink(4)

一、课前准备

掌握上节课内容

二、课堂主题

深入理解Checkpoints机制原理

三、课程目标

1. 理解一个简单的checkpoints的流程
2. 掌握Checkpoints算法
3. 掌握Checkpoints配置使用

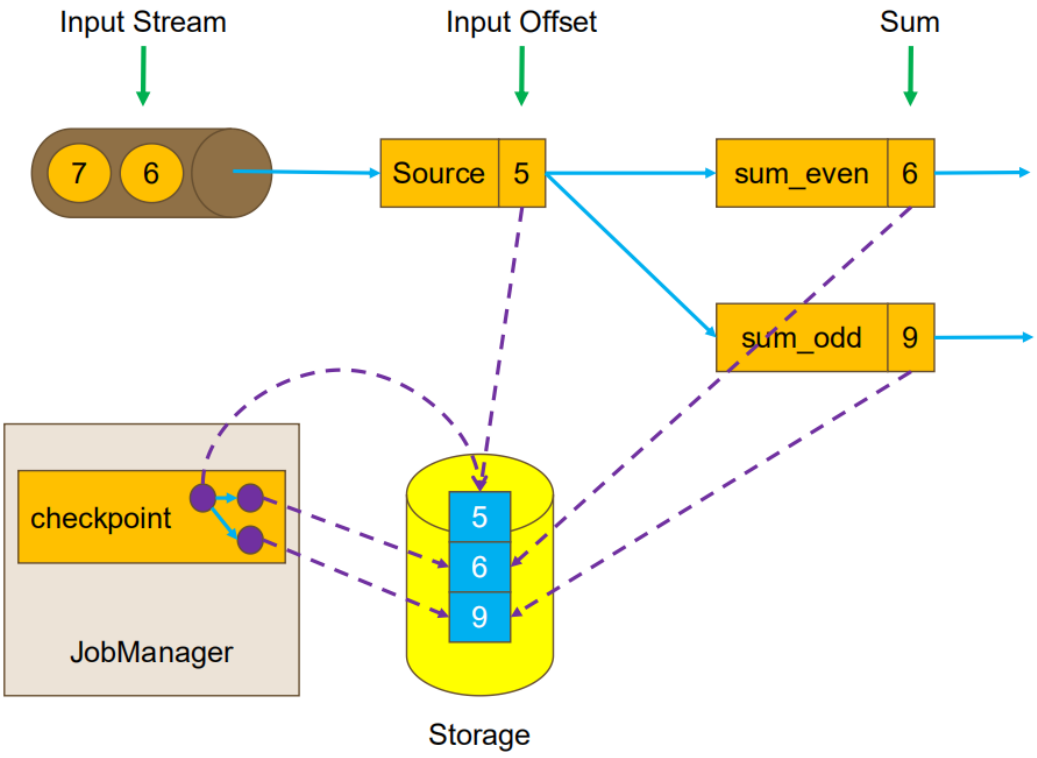
四、知识要点

4.1 Checkpoint原理

4.1.1 Checkpoint概述

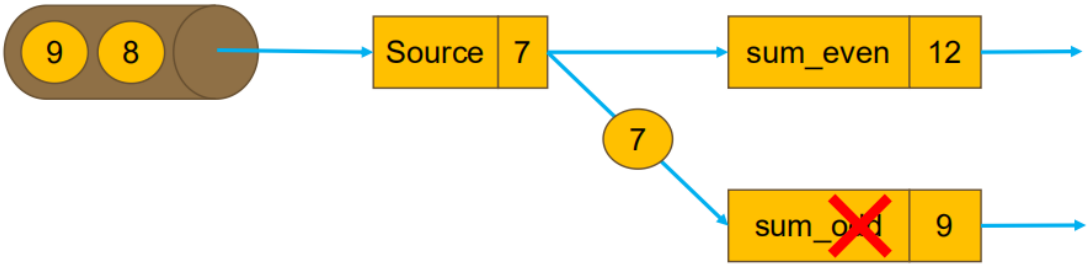
checkpoint机制是Flink可靠性的基石，可以保证Flink集群在某个算子因为某些原因(如 异常退出)出现故障时，能够将整个应用流图的状态恢复到故障之前的某一状态，保 证应用流图状态的一致性。

4.1.2 Checkpoint的简单想法

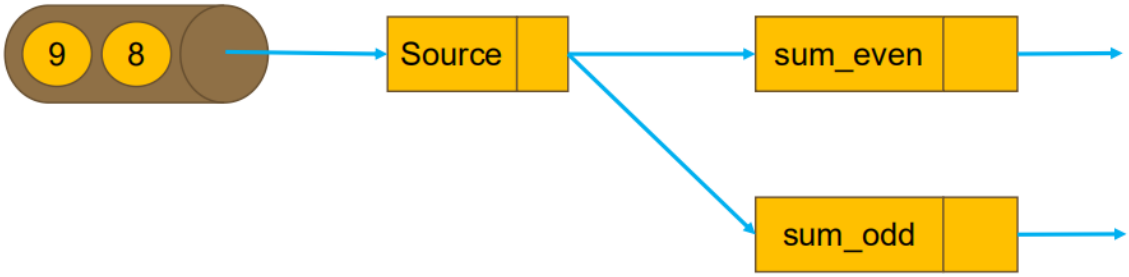


4.1.3 Checkpoint 恢复流程

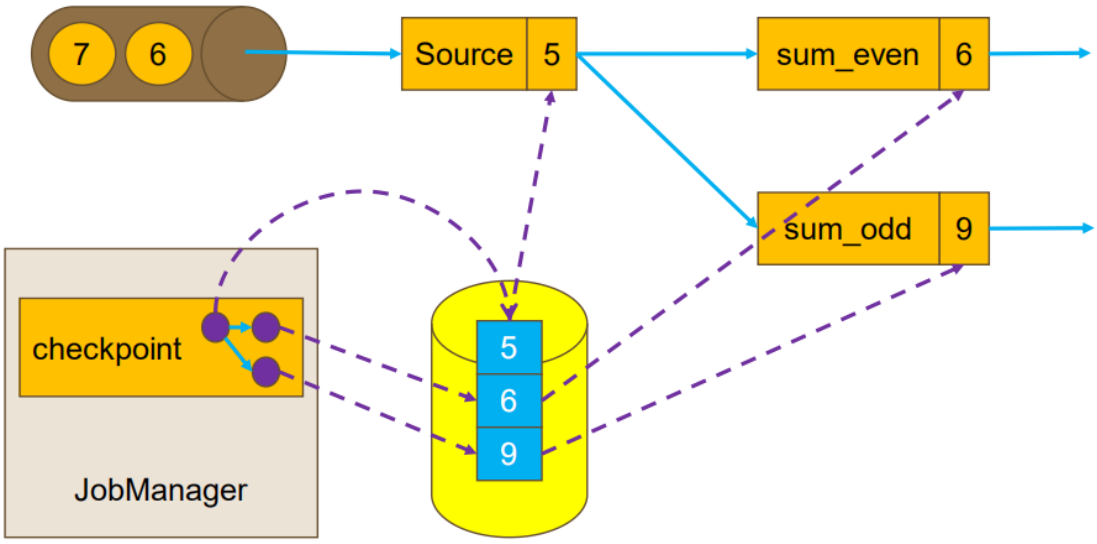
任务运行失败



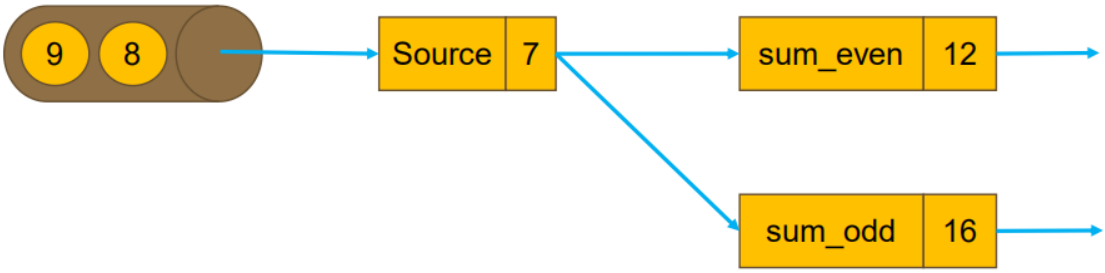
重启应用



从checkpoint恢复数据

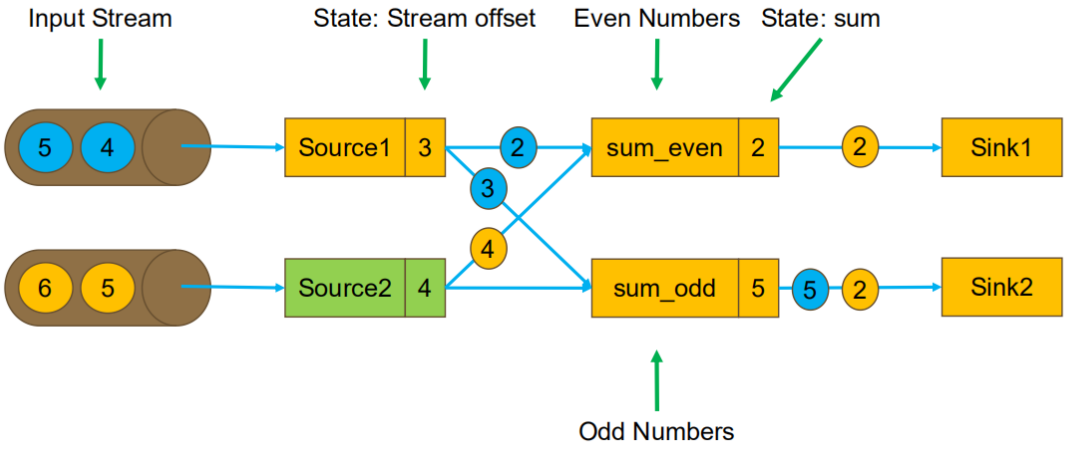


任务继续运行



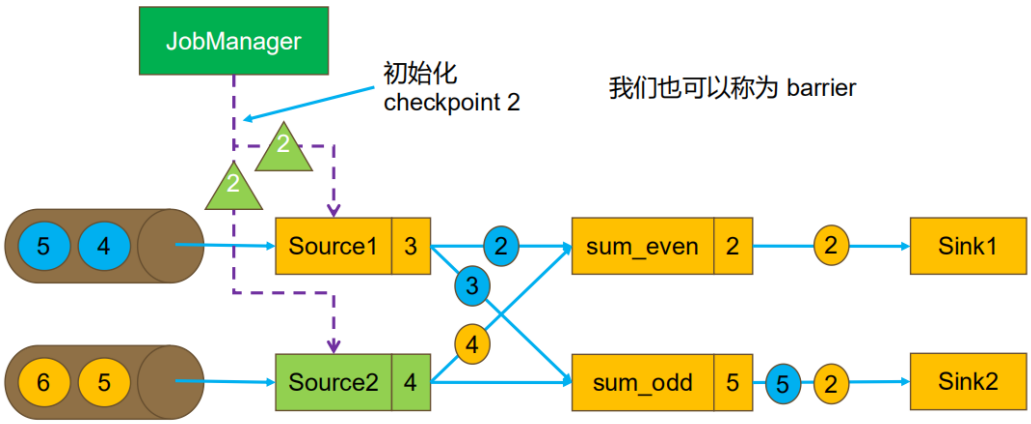
4.1.3 Chandy-Lamport 算法

1. 任务开启

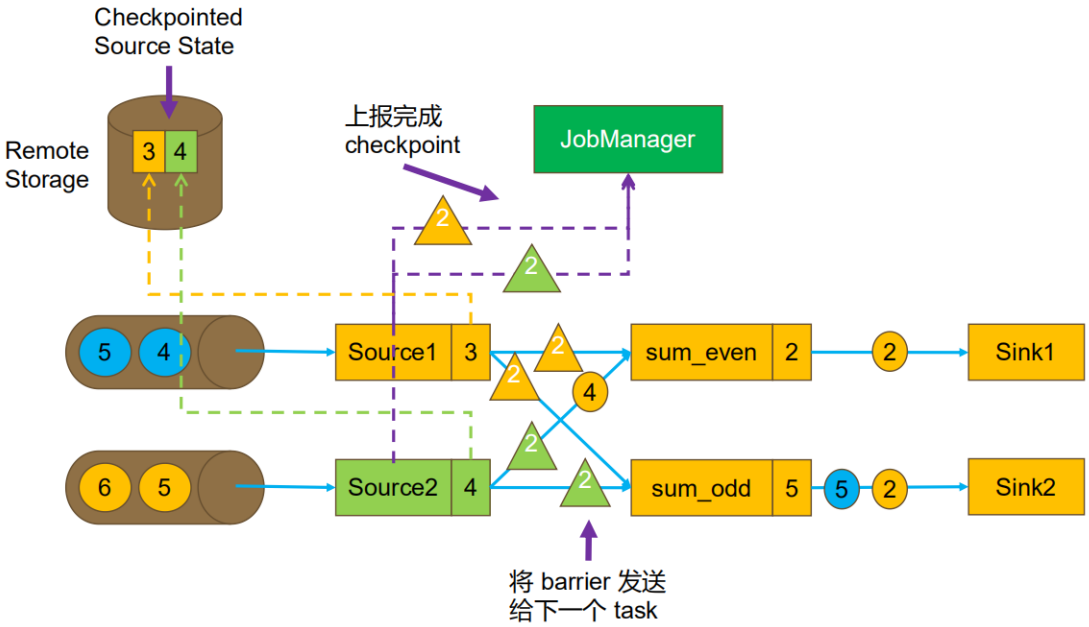


2. JobMananger发起Checkpoint

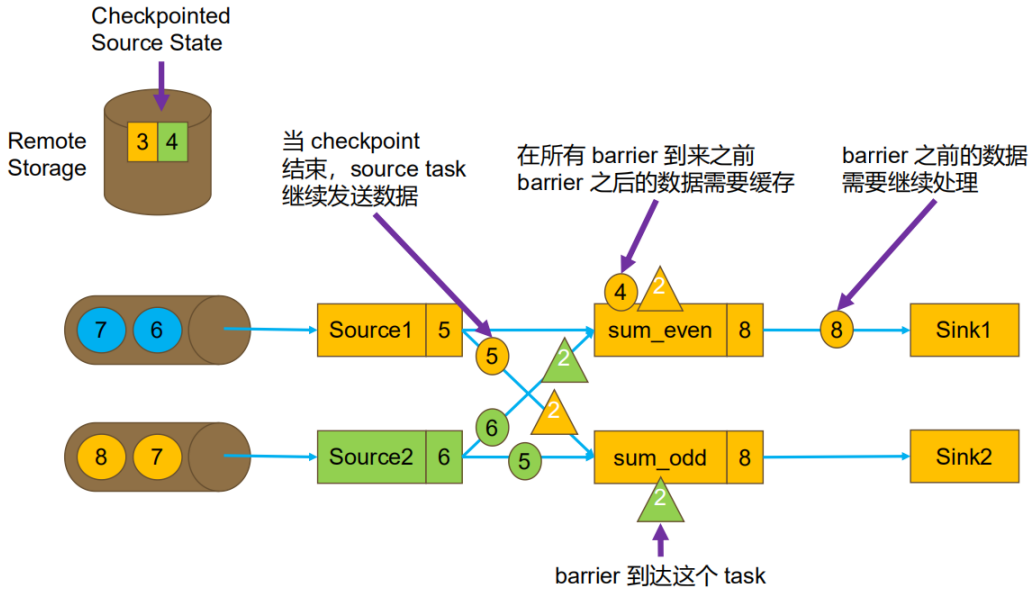
3.



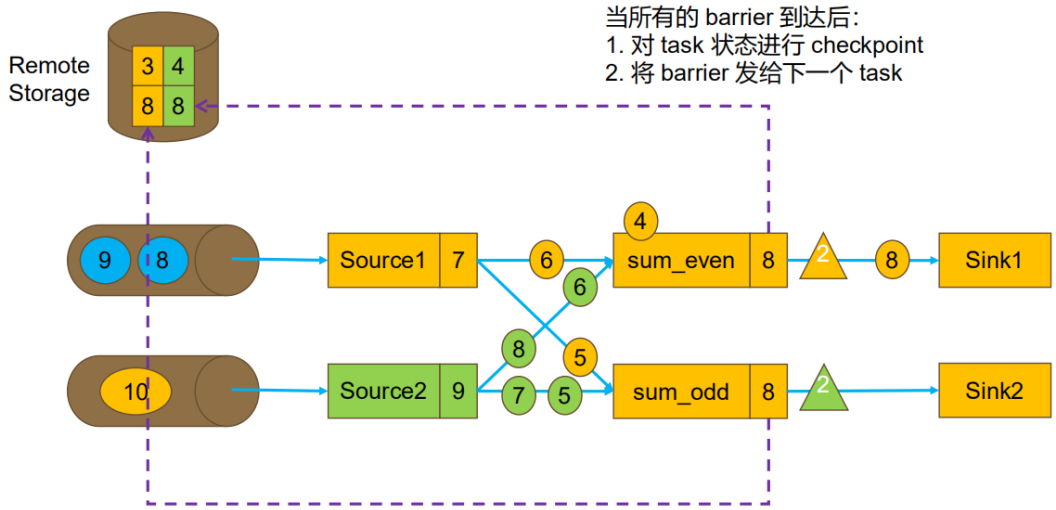
3.source上报checkpoint



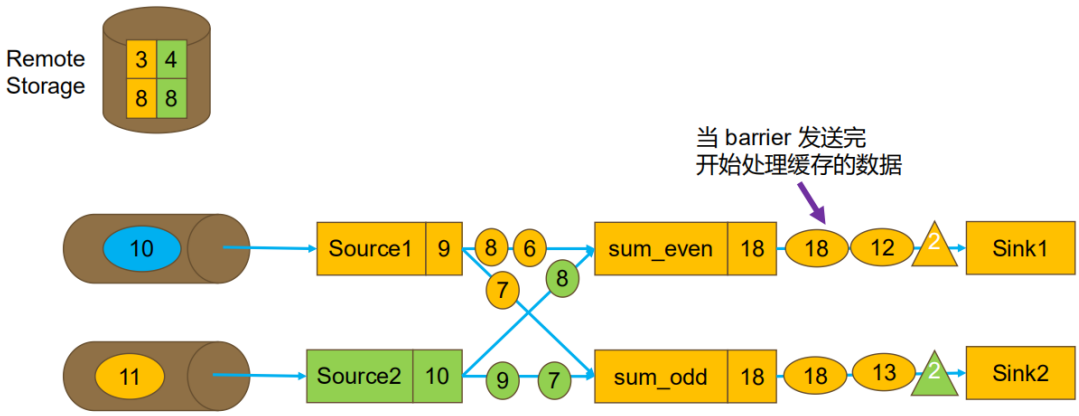
4. 数据处理



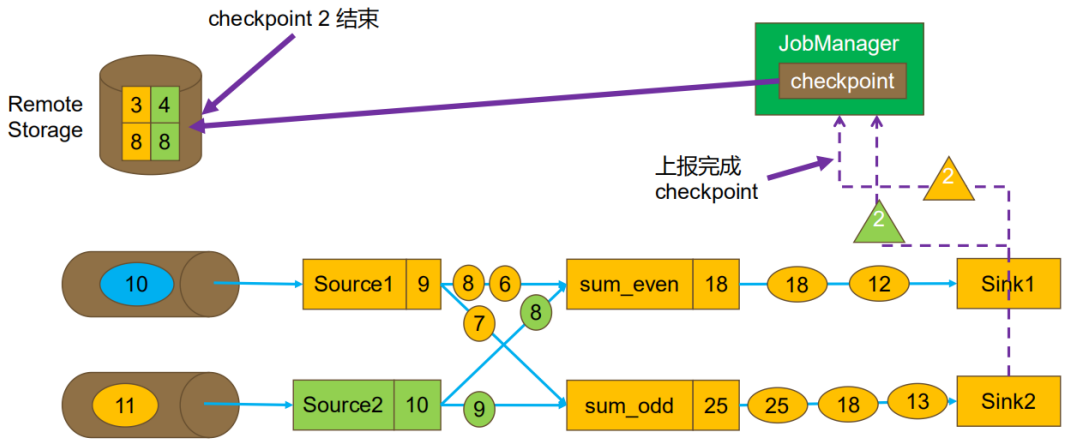
5. barrier对齐



6. 缓存数据处理



7. sink上报checkpoint



4.2 Checkpoint配置

默认checkpoint功能是disabled的，想要使用的时候需要先启用，checkpoint开启之后，checkPointMode有两种，Exactly-once和At-least-once，默认的checkPointMode是Exactly-once，Exactly-once对于大多数应用来说是最合适的。At-least-once可能用在某些延迟超低的应用程序（始终延迟为几毫秒）。

```
默认checkpoint功能是disabled的，想要使用的时候需要先启用
StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
// 每隔1000 ms进行启动一个检查点【设置checkpoint的周期】
env.enableCheckpointing(1000);
// 高级选项：
// 设置模式为exactly-once （这是默认值）
env.getCheckpointConfig().setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);
// 确保检查点之间有至少500 ms的间隔【checkpoint最小间隔】
env.getCheckpointConfig().setMinPauseBetweenCheckpoints(500);
// 检查点必须在一分钟内完成，或者被丢弃【checkpoint的超时时间】
env.getCheckpointConfig().setCheckpointTimeout(60000);
// 同一时间只允许进行一个检查点
env.getCheckpointConfig().setMaxConcurrentCheckpoints(1);
// 表示一旦Flink处理程序被cancel后，会保留Checkpoint数据，以便根据实际需要恢复到指定的
Checkpoint【详细解释见备注】
env.getCheckpointConfig().enableExternalizedCheckpoints(ExternalizedCheckpointCl
eanup.RETAIN_ON_CANCELLATION);
```

4.3 重启策略

4.3.1 重启策略概述

Flink支持不同的重启策略，以在故障发生时控制作业如何重启，集群在启动时会伴随一个默认的重启策略，在没有定义具体重启策略时会使用该默认策略。 如果在工作提交时指定了一个重启策略，该策略会覆盖集群的默认策略，默认的重启策略可以通过 Flink 的配置文件 flink-conf.yaml 指定。配置参数 restart-strategy 定义了哪个策略被使用。

常用的重启策略

（1）固定间隔 (Fixed delay)

（2）失败率 (Failure rate)

（3）无重启 (No restart)

如果没有启用 checkpointing，则使用无重启 (no restart) 策略。

如果启用了 checkpointing，但没有配置重启策略，则使用固定间隔 (fixed-delay) 策略， 尝试重启次数

默认值是：Integer.MAX_VALUE，重启策略可以在flink-conf.yaml中配置，表示全局的配置。也可以在应用代码中动态指定，会覆盖全局配置。

4.3.2 重启策略

固定间隔 (Fixed delay)

```
第一种：全局配置 flink-conf.yaml
restart-strategy: fixed-delay
restart-strategy.fixed-delay.attempts: 3
restart-strategy.fixed-delay.delay: 10 s
第二种：应用代码设置
env.setRestartStrategy(RestartStrategies.fixedDelayRestart(
  3, // 尝试重启的次数
  Time.of(10, TimeUnit.SECONDS) // 间隔
));
```

失败率 (Failure rate)

```
第一种：全局配置 flink-conf.yaml
restart-strategy: failure-rate
restart-strategy.failure-rate.max-failures-per-interval: 3
restart-strategy.failure-rate.failure-rate-interval: 5 min
restart-strategy.failure-rate.delay: 10 s
第二种：应用代码设置
env.setRestartStrategy(RestartStrategies.failureRateRestart(
  3, // 一个时间段内的最大失败次数
  Time.of(5, TimeUnit.MINUTES), // 衡量失败次数的是时间段
  Time.of(10, TimeUnit.SECONDS) // 间隔
));
```

无重启 (No restart)

```
第一种：全局配置 flink-conf.yaml
restart-strategy: none
第二种：应用代码设置
env.setRestartStrategy(RestartStrategies.noRestart());
```

4.3.3 多checkpoint

默认情况下，如果设置了Checkpoint选项，则Flink只保留最近成功生成的1个Checkpoint，而当Flink程序失败时，可以从最近的这个Checkpoint来进行恢复。但是，如果我们希望保留多个Checkpoint，并能够根据实际需要选择其中一个进行恢复，这样会更加灵活，比如，我们发现最近4个小时数据记录处理有问题，希望将整个状态还原到4小时之前Flink可以支持保留多个Checkpoint，需要在Flink的配置文件conf/flink-conf.yaml中，添加如下配置，指定最多需要保存Checkpoint的个数：

```
state.checkpoints.num-retained: 20
```

这样设置以后就查看对应的Checkpoint在HDFS上存储的文件目录
hdfs dfs -ls hdfs://namenode:9000/flink/checkpoints
如果希望回退到某个Checkpoint点，只需要指定对应的某个Checkpoint路径即可实现

4.3.4 从checkpoint恢复数据

如果Flink程序异常失败，或者最近一段时间内数据处理错误，我们可以将程序从某一个Checkpoint点进行恢复

```
bin/flink run -s
hdfs://namenode:9000/flink/checkpoints/467e17d2cc343e6c56255d222bae3421/chk-56/_metadata flink-job.jar
```

程序正常运行后，还会按照Checkpoint配置进行运行，继续生成Checkpoint数据。

当然恢复数据的方式还可以在自己的代码里面指定checkpoint目录，这样下一次启动的时候即使代码发生了改变就自动恢复数据了。

4.3.4 SavePoint

Flink通过Savepoint功能可以做到程序升级后，继续从升级前的那个点开始执行计算，保证数据不中断全局，一致性快照。可以保存数据源offset，operator操作状态等信息，可以从应用在过去任意做了savepoint的时刻开始继续消费

checkPoint
应用定时触发，用于保存状态，会过期，内部应用失败重启的时候使用。
savePoint
用户手动执行，是指向Checkpoint的指针，不会过期，在升级的情况下使用。
注意：为了能够在作业的不同版本之间以及 Flink 的不同版本之间顺利升级，强烈推荐程序员通过 uid(String) 方法手动的给算子赋予 ID，这些 ID 将用于确定每一个算子的状态范围。如果不手动给各算子指定 ID，则会由 Flink 自动给每个算子生成一个 ID。只要这些 ID 没有改变就能从保存点（savepoint）将程序恢复回来。而这些自动生成的 ID 依赖于程序的结构，并且对代码的更改是很敏感的。因此，强烈建议用户手动的设置 ID。

savepoint的使用

```
1：在flink-conf.yaml中配置Savepoint存储位置
不是必须设置，但是设置后，后面创建指定Job的Savepoint时，可以不用在手动执行命令时指定Savepoint的位置
state.savepoints.dir: hdfs://namenode:9000/flink/savepoints
2：触发一个savepoint【直接触发或者在cancel的时候触发】

停止程序：bin/flink cancel -s [targetDirectory] jobId [-yid yarnAppId]【针对on yarn模式需要指定-yid参数】


3：从指定的savepoint启动job
bin/flink run -s savepointPath [runArgs]
```

五、总结（5分钟）

1. 掌握Checkpoints的原理
2. 掌握Checkpoints的配置

六、作业

1. 掌握课上案例，为后面实践做准备

七、互动