Apache Flink高阶

本课目标

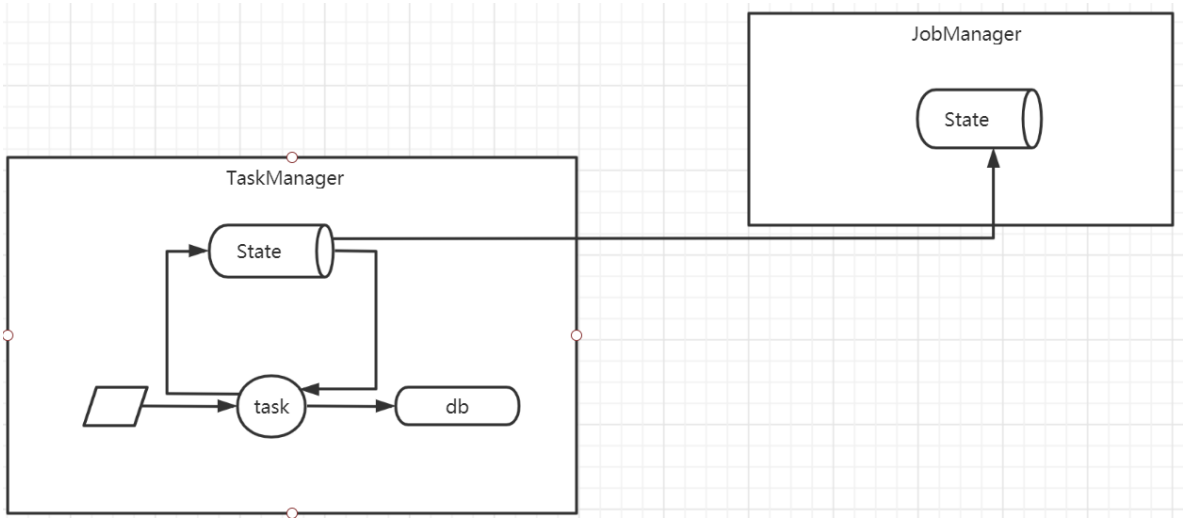
- [x] 掌握Flink三种State Backend
- [x] 掌握Flink checkpoint和savepoint原理
- [x] 了解Filnk的重启策略

1. State backend 概述

- Flink支持的StateBackend：
  - MemoryStateBackend（默认）
  - FSStateBackend
  - RocksDBStateBackend

1.1 MemoryStateBackend

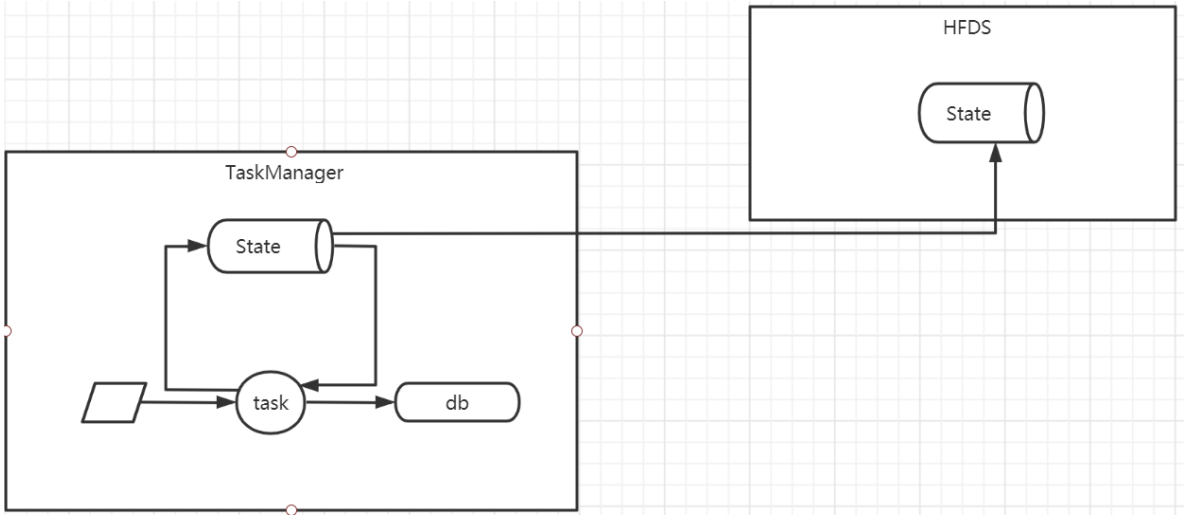
- 默认情况下，状态信息是存储在TaskManager的堆内存中的，checkpoint 的时候将状态保存到JobManager 的堆内存中。
- 缺点：
  - 只能保存数据量小的状态，状态数据有可能会丢失
- 优点：
  - 开发测试很方便



1.2 FSStateBackend

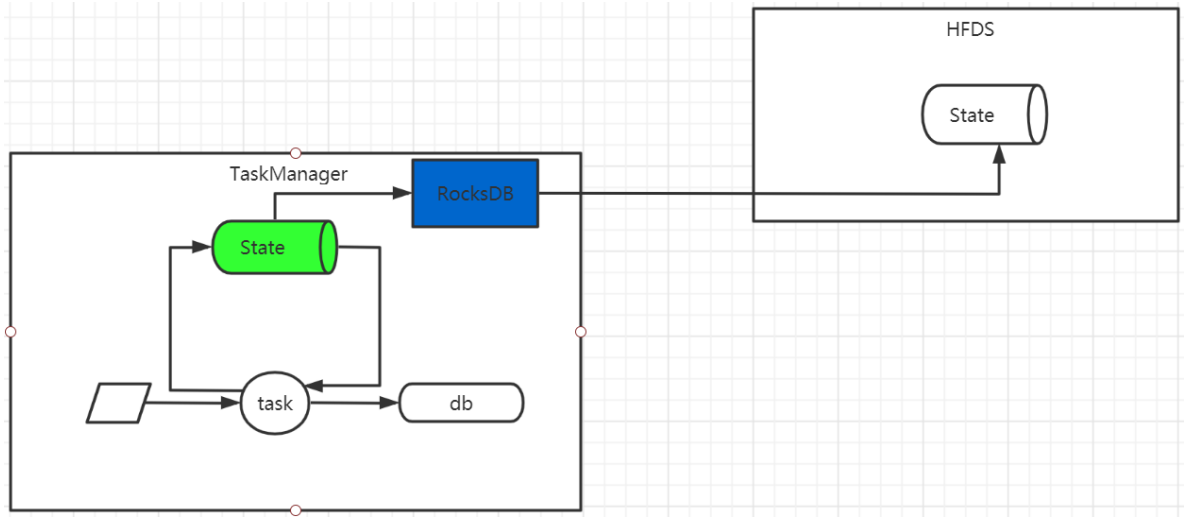
- 状态信息存储在 TaskManager 的堆内存中的，checkpoint 的时候将状态保存到指定的文件中 (HDFS等文件系统)

- 缺点：
  - 状态大小受TaskManager内存限制(默认支持5M)
- 优点：
  - 状态访问速度很快，状态信息不会丢失。生产环境下也可存储状态数据量大的情况。



1.3 RocksDBStateBackend

- 状态信息存储在 RocksDB 数据库 (key-value 的数据存储服务)， 最终保存在本地文件中。checkpoint的时候将状态保存到指定的文件中 (HDFS 等文件系统)
- 缺点：
  - 状态访问速度有所下降
- 优点：
  - 可以存储超大量的状态信息，状态信息不会丢失。生产环境可以存储超大量的状态信息



1.4 StateBackend实现方式

- 默认状态是：MemoryStateBackend

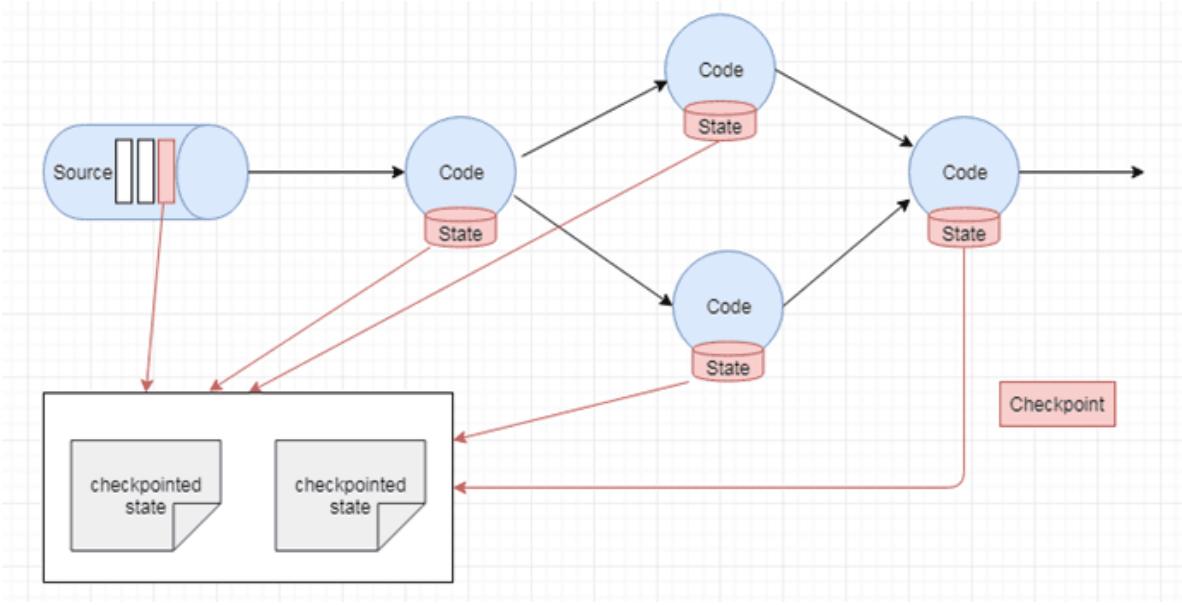
```
案例1：当前任务指定方式（默认）
//默认情况下就是 memoryStateBackend
```

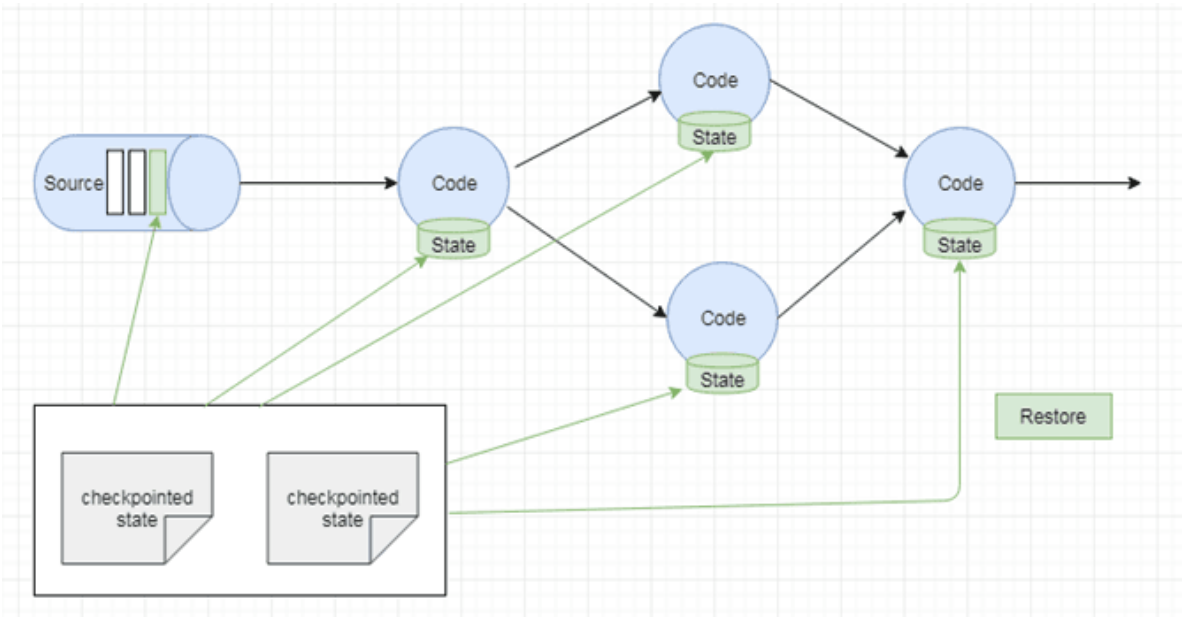
```
MemoryStateBackend memoryStateBackend = new MemoryStateBackend();
env.setStateBackend(memoryStateBackend);

//指定 FsStateBackend (推荐)
FsStateBackend fsStateBackend = new
FsStateBackend("hdfs://hadoop1:8020/flink/checkpoint");
env.setStateBackend(fsStateBackend);

//指定 RocksDB
RocksDBStateBackend rocksDBStateBackend = new
RocksDBStateBackend("hdfs://hadoop1:8020/flink/checkpoint",true);
env.setStateBackend(rocksDBStateBackend);


案例2：全局配置
修改flink-conf.yaml
state.backend: filesystem
state.checkpoints.dir: hdfs://hadoop1:9000/flink/checkpoints
注意：state.backend的值可以是下面几种：
    jobmanager(MemoryStateBackend)
    filesystem(FsStateBackend)
    rocksdb(RocksDBStateBackend)
```

2. checkpoint

2.1 checkpoint概述

- 为了保证state的容错性，Flink需要对state进行checkpoint，方便恢复数据。
- Checkpoint是Flink实现容错机制最核心的功能，它能够根据配置周期性地基于Stream中各个Operator/task的状态来生成快照，从而将这些状态数据定期持久化存储下来，当Flink程序一旦意外崩溃时，重新运行程序时可以有选择地从这些快照进行恢复，从而修正故障带来的程序数据异常。
- Flink的checkpoint机制可以与(stream和state)的持久化存储交互的前提：持久化的source，它需要支持在一定时间内重放事件。这种sources的典型例子是持久化的消息队列（比如Apache Kafka，RabbitMQ等）或文件系统（比如HDFS，S3，GFS等）
  用于state的持久化存储，例如分布式文件系统（比如HDFS，S3，GFS等）生成快照。
- 生成快照周期：比如每隔5秒





2.2 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

```
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(ExternalizedCheckpointCle
anup.RETAIN_ON_CANCELLATION);

#如果数据量比较大，建议5分钟左右checkpoint的一次。阿里他们使用的时候 也是这样建议的。
```

3. 恢复数据

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
  - 固定间隔 (Fixed delay)
  - 失败率 (Failure rate)
  - 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。

3.2 重启策略

```
方案一：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
));

方案二：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), // 衡量失败次数的时间段
Time.of(10, TimeUnit.SECONDS) // 间隔
));

方案三：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());
```

3.5 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

```
正常运行任务：
flink run -c com.bw.flink.statebackend.StateBackendTest flink3.jar --hostname
hadoop1 --port 8888

从失败的地方开始运行
flink run -c com.bw.flink.statebackend.StateBackendTest -s
hdfs://hadoop1:8020/flink/checkpoint/b53a74fa4ae4327b2c8186e970d595d6/chk-
12/_metadata flink3.jar --hostname hadoop1 --port 8888
```

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。

3.4 保存多分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