1. checkpoint 机制

checkpoint 机制：将分区计算数据写入 hdfs 目录，防止计算中数据丢失。设置 checkpoint 后，会优先从 checkpoint 中读取数据进行计算，其依赖的父 RDD 将被丢弃。

启动 sparkshell：/bin/spark-shell --master spark://mini1:7077
设置 checkpoint 目录：sc.setCheckpointDir(“hdfs://mini1:9000/ck”)
val rdd = sc.textFile(“hdfs://mini1:9000/wc”)
val rdd2 = rdd.flatMap(_.split(“ “))
rdd2.checkpoint
rdd2.count ## count 执行时会启动两个任务，一个负责计算 count，另一个负责将数据写入 hdfs 目录
rdd2.count ## 再次执行 count 则从 checkpoint 中读取数据

注：
（1）在触发 Action 执行时，若有缓存则优先从缓存中读取数据；没有缓存优先从 checkpoint 中读取数据；否则，从第一个 RDD 推导执行。
（2）建议 checkpoint 前先 cache，然后将 cache 数据写入 checkpoint 目录。
（3）类似于 cache，但有区别：cache 默认只保存一份数据到内存，可能会丢失；而 checkpoint 则将数据写入外部存储系统，数据丢失的概率大大降低。

Spark 容错机制：
机器磁盘故障导致分区中数据丢失；而新分区根据 Lineage 血缘关系读取数据计算并恢复。即使部分数据丢失，也从头开始恢复，代价较高。
优化：checkpoint 保存计算的中间数据，之后的计算直接从 checkpoint 读取数据，而不用 Lineage 推导重新计算。

sc.setCheckpointDir(“hdfs://mini1:9000/ck”)
val rdd = sc.textFile(“hdfs://mini1:9000/wc”)
val rdd2 = rdd.flatMap(_.split(“ ”)).map((_, 1))
rdd2.cache
rdd2.checkpoint
rdd2.count ## count 只会启动一个任务执行，将缓存数据写入 hdfs

总结：
（1）checkpoint 作用是将计算中间数据保存到外部存储介质，防止数据丢失；
（2）checkpoint 之前最好做 cache，即只启动一个任务进行计算。