1. RDD 的缓存

Spark 计算速度快的原因之一，就是可以在内存中持久化或缓存数据集。持久化 RDD 后，每一个节点都把计算的中间结果保存在内存中，并在对该 RDD 或衍生 RDD 进行其他操作时重用，使得后续操作更加迅速。RDD 持久化和缓存是 Spark 最重要特征之一。缓存是 Spark 构建迭代式算法和快速交互式查询的关键。

1.1. RDD 缓存方式

RDD 通过 persist()方法或 cache()方法将计算结果缓存，并不是方法调用时立即缓存，而是在触发 action 时将该 RDD 缓存到计算节点的内存中供下游使用。

```
/** Persist this RDD with the default storage level (`MEMORY_ONLY`). */
def persist(): this.type = persist(StorageLevel.MEMORY_ONLY)

/** Persist this RDD with the default storage level (`MEMORY_ONLY`). */
def cache(): this.type = persist()
```

查看源码发现 cache()调用了 persist()方法，且默认只在内存中存储一份。Spark 存储级别有好多种，存储级别在 object StorageLevel 中定义。

```
object StorageLevel {
  val NONE = new StorageLevel(false, false, false, false)
  val DISK_ONLY = new StorageLevel(true, false, false, false)
  val DISK_ONLY_2 = new StorageLevel(true, false, false, false, 2)
  val MEMORY_ONLY = new StorageLevel(false, true, false, true)
  val MEMORY_ONLY_2 = new StorageLevel(false, true, false, true, 2)
  val MEMORY_ONLY_SER = new StorageLevel(false, true, false, false)
  val MEMORY_ONLY_SER_2 = new StorageLevel(false, true, false, false, 2)
  val MEMORY_AND_DISK = new StorageLevel(true, true, false, true)
  val MEMORY_AND_DISK_2 = new StorageLevel(true, true, false, true, 2)
  val MEMORY_AND_DISK_SER = new StorageLevel(true, true, false, false)
  val MEMORY_AND_DISK_SER_2 = new StorageLevel(true, true, false, false, 2)
  val OFF_HEAP = new StorageLevel(false, false, true, false)
```

缓存有可能丢失，或者由于内存不足而被删除。RDD 的缓存容错机制在缓存丢失时也能保证计算的正确执行。通过基于 RDD 的一系列转换，丢失的数据会被重算。由于 RDD 的各个 Partition 之间相对独立，因此只需要计算丢失的部分，并不需要重算全部 Partition。

补充：

启动 spark：/sbin/start-master.sh

启动 spark-shell：/bin/spark-shell –master spark://mini1:7077

val rdd1 = sc.textFile(“hdfs://mini1:9000/logs/itcast.log”)

```
val rdd2 = rdd1.map(_.split("\t")).map(arr => arr(1))
rdd2.cache()
rdd2.count ## 从 hdfs 读数据，计算较慢
rdd2.count ## 从内存读数据，计算较快
rdd2.take(10) ## 从内存中取数据
rdd2.unpersist ## 释放内存，任务结束也会从内存中释放资源
```

注：

（1）触发第一个 Action 时将数据 cache 到内存，在进行其他计算时从内存读取数据；

（2）若缓存数据量非常大，内存不足，则放不下的仍从 hdfs 中读取；

（3）cache()调用 persist()进行持久化，cache()默认持久化到内存；

（4）持久化机制包括：内存、磁盘、内存+磁盘；

（5）频繁使用上游已整理好的数据时可选择使用缓存机制；

（6）Driver 端不存在多线程问题。

Executor 在什么时候启动？

Driver 与 Master 建立连接后，Master 分配资源时启动 Executor。默认情况下，一个 Worker 启动一个 Executor。