1、Spark 有几种部署方式？

1 Local 模式

2 standalone 模式

3 spark on yarn 模式

4 spark on mesos 模式

2、Spark 提交作业重要参数

executor-cores —— 每个 executor 使用的内核数，默认为 1，官方建议 2-5 个

num-executors —— 启动 executors 的数量，默认为 2

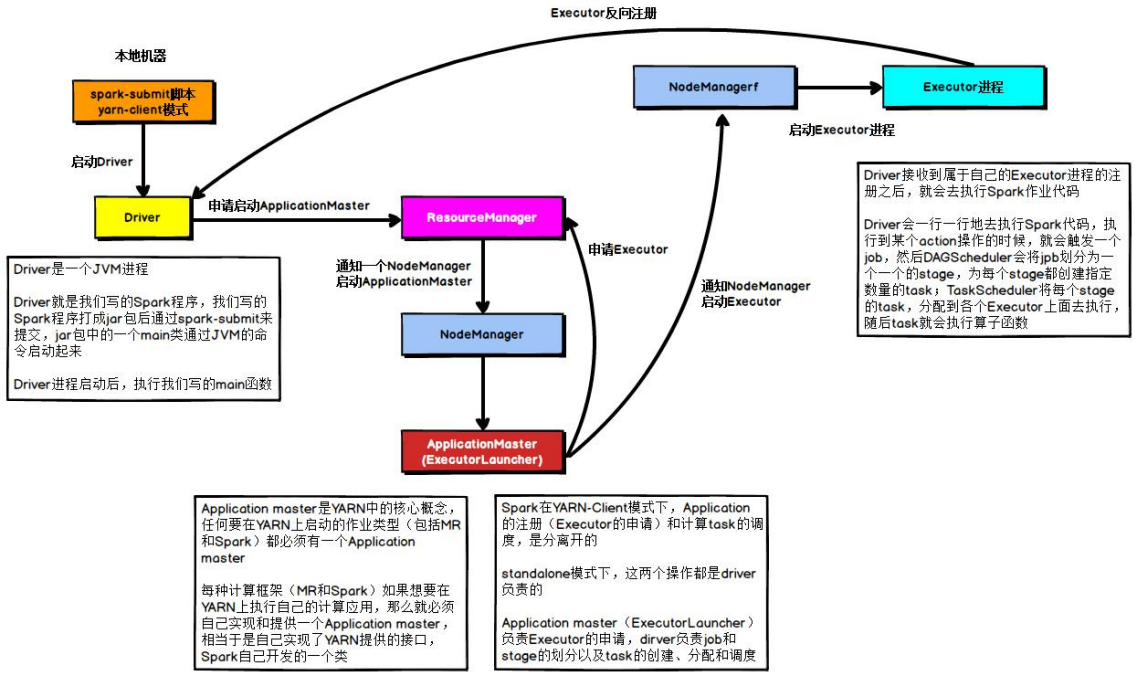
executor-memory —— executor 内存大小，默认 1G

driver-cores —— driver 使用内核数，默认为 1

driver-memory —— driver 内存大小，默认 512M

3、简述 Spark on yarn 的作业提交流程

YARN Client 模式

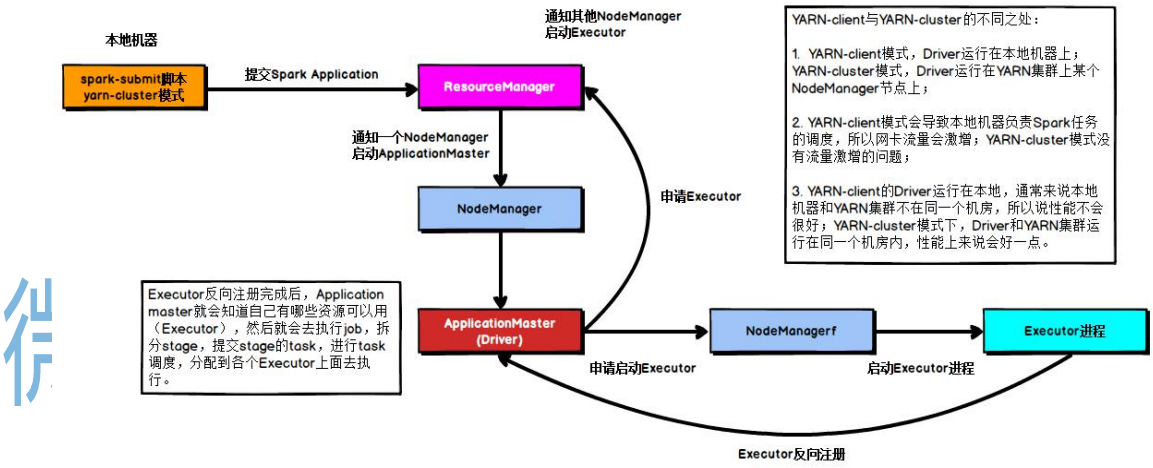


YARN Client 模式

在 YARN Client 模式下，Driver 在任务提交的本地机器上运行，Driver 启动后会和 ResourceManager 通讯申请启动 ApplicationMaster，随后 ResourceManager 分配 container，在合适的 NodeManager 上启动 ApplicationMaster，此时的 ApplicationMaster 的功能相当于一个 ExecutorLaucher，只负责向 ResourceManager 申请 Executor 内存。

ResourceManager 接到 ApplicationMaster 的资源申请后会分配 container，然后 ApplicationMaster 在资源分配指定的 NodeManager 上启动 Executor 进程，Executor 进程启动后会向 Driver 反向注册，Executor 全部注册完成后 Driver 开始执行 main 函数，之后执行到 Action 算子时，触发一个 job，并根据宽依赖开始划分 stage，每个 stage 生成对应的 taskSet，之后将 task 分发到各个 Executor 上执行。

YARN Cluster 模式



YARN Cluster 模式

在 YARN Cluster 模式下，任务提交后会和 ResourceManager 通讯申请启动 ApplicationMaster，随后 ResourceManager 分配 container，在合适的 NodeManager 上启动 ApplicationMaster，此时的 ApplicationMaster 就是 Driver。

Driver 启动后向 ResourceManager 申请 Executor 内存，ResourceManager 接到 ApplicationMaster 的资源申请后会分配 container，然后在合适的 NodeManager 上启动 Executor 进程，Executor 进程启动后会向 Driver 反向注册，Executor 全部注册完成后 Driver 开始执行 main 函数，之后执行到 Action 算子时，触发一个 job，并根据宽依赖开始划分 stage，每个 stage 生成对应的 taskSet，之后将 task 分发到各个 Executor 上执行。

4、Spark 的两种核心 Shuffle

spark 的 Shuffle 有 Hash Shuffle 和 Sort Shuffle 两种。

5、简述 RDD、DataFrame、DataSet 三者的区别

RDD 弹性分布式数据集；不可变、可分区、元素可以并行计算的集合。

DataFrame 以 RDD 为基础的分布式数据集。

DataFrame 带有元数据 schema，每一列都带有名称和类型。

DataFrame=RDD+schema

Dataset 和 DataFrame 拥有完全相同的成员函数，区别只是每一行的数据类型不同

6、Repartition 和 Coalesce 关系与区别

1）关系：

两者都是用来改变 RDD 的 partition 数量的，repartition 底层调用的就是 coalesce 方法：coalesce(numPartitions, shuffle = true)

2）区别：

repartition 一定会发生 shuffle，coalesce 根据传入的参数来判断是否发生 shuffle

一般情况下增大 rdd 的 partition 数量使用 repartition，减少 partition 数量时使用 coalesce

7、Spark 中 cache 默认缓存级别是什么？

DataFrame 的 cache 默认采用 MEMORY_AND_DISK 这和 RDD 的默认方式不一样 RDD cache 默认采用 MEMORY_ONLY

10、SparkSQL 中 left semi join 操作与 left join 操作的区别？

left semi join 是 in(keySet) 的关系，遇到右表重复记录，左表会跳过,性能更高，而 left join 则会一直遍历。但是 left semi join 中最后 select 的结果中只许出现左表中的列名，因为右表只有 join key 参与关联计算了。

11、Spark 常用算子 reduceByKey 与 groupByKey 的区别，哪一种更具优势？

reduceByKey：按照 key 进行聚合，在 shuffle 之前有 combine（预聚合）操作，返回结果是 RDD[k,v]。

groupByKey：按照 key 进行分组，直接进行 shuffle。

开发指导：reduceByKey 比 groupByKey，建议使用。但是需要注意是否会影响业务逻辑。

12、Spark Shuffle 默认并行度是多少？

参数 spark.sql.shuffle.partitions 决定 默认并行度 200

13、简述 Spark 中共享变量（广播变量和累加器）

Spark 为此提供了两种共享变量，一种是 Broadcast Variable（广播变量），另一种是 Accumulator（累加变量）。

14、SparkStreaming 有哪几种方式消费 Kafka 中的数据，它们之间的区别是什么？

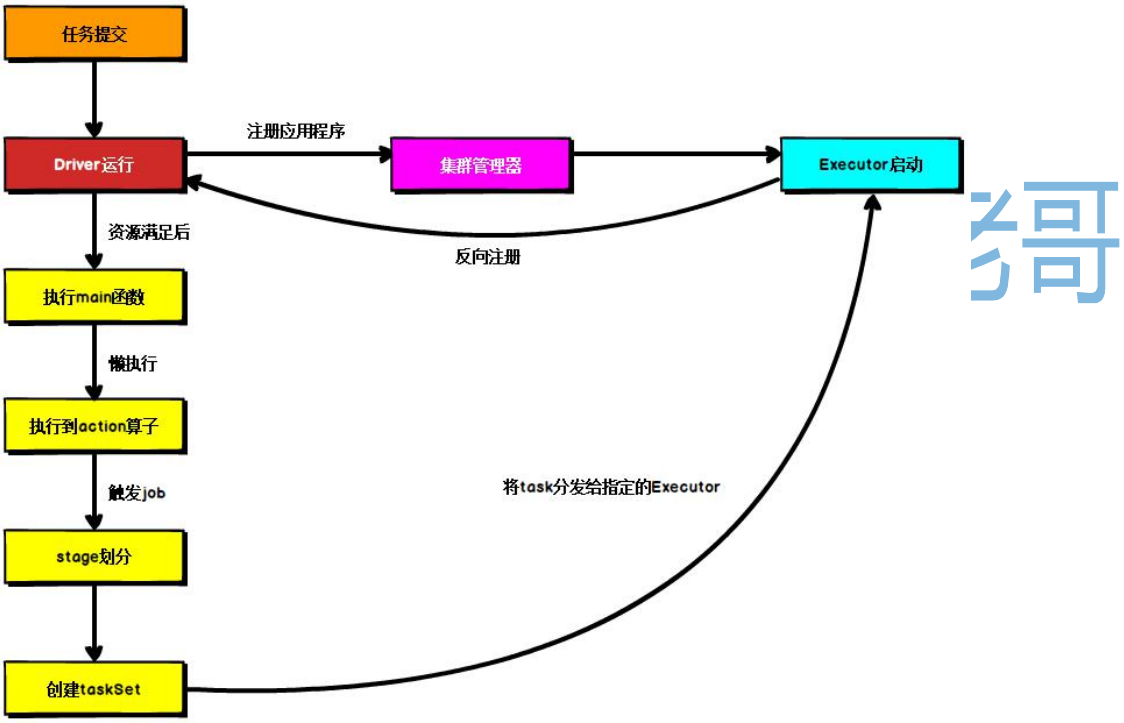
基于 receiver 的方式，是使用 Kafka 的高阶 API 来在 ZooKeeper 中保存消费过的 offset 的。这是消费 Kafka 数据的传统方式。这种方式配合着 WAL 机制可以保证数据零丢失的高可靠性，但是却无法保证数据被处理一次且仅一次，可能会处理两次。因为 Spark 和 ZooKeeper 之间可能是不同步的。

基于 direct 的方式，使用 kafka 的简单 api，Spark Streaming 自己就负责追踪消费的 offset，并保存在 checkpoint 中。Spark 自己一定是同步的，因此可以保证数据是消费一次且仅消费一次。

15、请简述一下 SparkStreaming 的窗口函数中窗口宽度和滑动距离的关系？

窗口宽度=滑动距离：刚刚好，不重复读取数据，也不丢失数据（此方式替换）
数据接入时将数据的切分宽度放大(不使用窗口函数)
窗口宽度>滑动距离：有部分数据被重复计算
窗口宽度<滑动距离：会有部分数据丢失（企业一般不会使用这种方式）

16、Spark 通用运行流程概述？



不论 Spark 以何种模式进行部署，任务提交后，都会先启动 Driver 进程，随后 Driver 进程向集群管理器注册应用程序，之后集群管理器根据此任务的配置文件分配 Executor 并启动，当 Driver 所需的资源全部满足后，Driver 开始执行 main 函数，Spark 查询为懒执行，当执行到 action 算子时开始反向推算，根据宽依赖进行 stage 的划分，随后每一个 stage 对应一个 taskset，taskset 中有多个 task，根据本地化原则，task 会被分发到指定的 Executor 去执行，在任务执行的过程中，Executor 也会不断与 Driver 进行通信，报告任务运行情况。

17、Standalone 集群有四个重要组成部分

1） Driver：是一个进程，我们编写的 Spark 应用程序就运行在 Driver 上，由 Driver 进程执行；

2） Master(RM)：是一个进程，主要负责资源的调度和分配，并进行集群的监控等职责；

3） Worker(NM)：是一个进程，一个 Worker 运行在集群中的一台服务器上，主要负责两个职责，一个是用自己的内存存储 RDD 的某个或某些 partition；另一个是启动其他进程和线程（Executor），对 RDD 上的 partition 进行并行的处理和计算。

4） Executor：是一个进程，一个 Worker 上可以运行多个 Executor，Executor 通过启动多个线程（task）来执行对 RDD 的 partition 进行并行计算，也就是执行我们对 RDD 定义的例如 map、flatMap、reduce 等算子操作。

18、Transformation 和 action 的区别

1、transformation 是得到一个新的 RDD，方式很多，比如从数据源生成一个新的 RDD，从 RDD 生成一个新的 RDD

2、action 是得到一个值，或者一个结果（直接将 RDDcache 到内存中）

19、Map 和 FlatMap 区别 对结果集的影响有什么不同

map：对 RDD 每个元素转换，文件中的每一行数据返回一个数组对象。

flatMap：对 RDD 每个元素转换，然后再扁平化。

20、driver 的功能是什么？

负责向集群申请资源，向 master 注册信息，负责了作业的调度，负责作业的解析、生成 Stage 并调度 Task 到 Executor 上。

21、Spark 中 Work 的主要工作是什么？

管理当前节点内存，CPU 的使用状况，接收 master 分配过来的资源指令，通过 ExecutorRunner 启动程序分配任务。

22、Spark 为什么比 mapreduce 快？

24.1、spark 是基于内存进行数据处理的，MapReduce 是基于磁盘进行数据处理的

24.2、Spark 计算比 MapReduce 快的根本原因在于 DAG 计算模型。

24.3、spark 是粗粒度资源申请，MapReduce 是细粒度资源申请。

23、Mapreduce 和 Spark 的都是并行计算，那么他们有什么相同和区别？

Mapreduce 的 job 只有 map 和 reduce 操作，表达能力比较欠缺而且在 mr 过程中会重复的读写 hdfs，造成大量的 io 操作，多个 job 需要自己管理关系。

spark 的迭代计算都是在内存中进行的，API 中提供了大量的 RDD 操作如 join，groupby 等，而且通过 DAG 图可以实现良好的容错。

24、Spark 应用程序的执行过程是什么？

1、构建 Spark Application 的运行环境（启动 SparkContext），SparkContext 向资源管理器（可以是 Standalone、Mesos 或 YARN）注册并申请运行 Executor 资源；

2、资源管理器分配 Executor 资源并启动 StandaloneExecutorBackend，

Executor 运行情况将随着心跳发送到资源管理器上；

3、SparkContext 构建成 DAG 图，将 DAG 图分解成 Stage，并把 Taskset 发送给 Task Scheduler。Executor 向 SparkContext 申请 Task，Task Scheduler 将 Task 发放给 Executor 运行同时 SparkContext 将应用程序代码发放给 Executor。

4、Task 在 Executor 上运行，运行完毕释放所有资源。

25、spark on yarn Cluster 模式下，ApplicationMaster 和 driver 是在同一个进程么？

是，driver 位于 ApplicationMaster 进程中。该进程负责申请资源，还负责监控程序、资源的动态情况。

26、Spark on Yarn 模式有哪些优点？

1、与其他计算框架共享集群资源

2、Yarn 的资源分配更加细致

3、Application 部署简化

4、Yarn 通过队列的方式，实现资源弹性管理。

27、spark 中 cache 和 persist 的区别？

cache：缓存数据，默认是缓存在内存中，其本质还是调用 persist

persist:缓存数据，有丰富的数据缓存策略。数据可以保存在内存也可以保存在磁盘中，使用的时候指定对应的缓存级别就可以了

28、Spark 中创建 RDD 的方式总结 3 种

1、从集合中创建 RDD；2、从外部存储创建 RDD；3、从其他 RDD 创建。

1、从集合中创建 RDD，Spark 主要提供了两种函数：parallelize 和 makeRDD

```
val rdd = sc.parallelize(Array(1,2,3,4,5,6,7,8))
val rdd = sc.makeRDD(Array(1,2,3,4,5,6,7,8))
```

2、由外部存储系统的数据集创建 RDD

```
val rdd= sc.textFile("hdfs://node01:8020/data/test")
```

3、从其他 RDD 创建

```
val rdd1 = sc.parallelize(Array(1,2,3,4))
val rdd2 =rdd.map(x=>x.map(_*2))
```

29、常用算子

Transformation 算子

29.1 map(func)案例

1. 作用：返回一个新的 RDD，该 RDD 由每一个输入元素经过 func 函数转换后组成

2. 需求：创建一个 1-10 数组的 RDD，将所有元素*2 形成新的 RDD

（1）创建

```
scala> var source  = sc.parallelize(1 to 10)
source: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[8] at parallelize at <console>:24
```

（2）打印

```
scala> source.collect()
res7: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

（3）将所有元素*2

```
scala> val mapadd = source.map(_ * 2)
mapadd: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[9] at map at <console>:26
```

（4）打印最终结果

```
scala> mapadd.collect()
res8: Array[Int] = Array(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
```

29.2 mapPartitions(func) 案例

1. 作用：类似于 map，但独立地在 RDD 的每一个分片上运行，因此在类型为 T 的 RDD 上

运行时，func 的函数类型必须是 Iterator[T] => Iterator[U]。假设有 N 个元素，有 M 个分区，那么 map 的函数的将被调用 N 次,而 mapPartitions 被调用 M 次,一个函数一次处理所有分区。

2. 需求：创建一个 RDD，使每个元素*2 组成新的 RDD

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(Array(1,2,3,4))
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[4] at parallelize at <console>:24
```

（2）使每个元素*2 组成新的 RDD

```
scala> rdd.mapPartitions(x=>x.map(_*2))
res3: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[6] at mapPartitions at <console>:27
```

（3）打印新的 RDD

```
scala> res3.collect
res4: Array[Int] = Array(2, 4, 6, 8)
```

29.3 mapPartitionsWithIndex(func) 案例

1. 作用：类似于 mapPartitions，但 func 带有一个整数参数表示分片的索引值，因此在类型为 T 的 RDD 上运行时，func 的函数类型必须是(Int, Interator[T]) => Iterator[U]；

2. 需求：创建一个 RDD，使每个元素跟所在分区形成一个元组组成一个新的 RDD

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(Array(1,2,3,4))
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[4] at parallelize at <console>:24
```

（2）使每个元素跟所在分区形成一个元组组成一个新的 RDD

```
scala> val indexRdd = rdd.mapPartitionsWithIndex((index,items)=>(items.map((index, ))))
indexRdd: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[5] at mapPartitionsWithIndex at <console>:26
```

（3）打印新的 RDD

```
scala> indexRdd.collect
res2: Array[(Int, Int)] = Array((0,1), (0,2), (1,3), (1,4))
```

29.4 map()和 mapPartition()的区别

1. map()：每次处理一条数据。

2. mapPartition()：每次处理一个分区的数据，这个分区的数据处理完后，原 RDD 中分区的数据才能释放，可能导致 OOM。

3. 开发指导：当内存空间较大的时候建议使用 mapPartition()，以提高处理效率。

29.5 flatMap(func) 案例

1. 作用：类似于 map，但是每一个输入元素可以被映射为 0 或多个输出元素（所以 func 应该返回一个序列，而不是单一元素）

2. 需求：创建一个元素为 1-5 的 RDD，运用 flatMap 创建一个新的 RDD，新的 RDD 为原 RDD 的每个元素的扩展（1->1,2->1,2……5->1,2,3,4,5）

（1）创建

```
scala> val sourceFlat = sc.parallelize(1 to 5)
sourceFlat: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[12] at parallelize at <console>:24
```

（2）打印

```
scala> sourceFlat.collect()
res11: Array[Int] = Array(1, 2, 3, 4, 5)
```

（3）根据原 RDD 创建新 RDD（1->1,2->1,2……5->1,2,3,4,5）

```
scala> val flatMap = sourceFlat.flatMap(1 to _)
flatMap: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[13] at flatMap at <console>:26
```

（4）打印新 RDD

```
scala> flatMap.collect()
res12: Array[Int] = Array(1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5)
```


29.6 sortBy(func,[ascending], [numTasks]) 案例

1. 作用：使用 func 先对数据进行处理，按照处理后的数据比较结果排序，默认为正序。

2. 需求：创建一个 RDD，按照不同的规则进行排序

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(List(2,1,3,4))
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[21] at parallelize at <console>:24
```

（2）按照自身大小排序

```
scala> rdd.sortBy(x => x).collect()
res11: Array[Int] = Array(1, 2, 3, 4)
```

（3）按照与 3 余数的大小排序

```
scala> rdd.sortBy(x => x%3).collect()
res12: Array[Int] = Array(3, 4, 1, 2)
```

29.7 groupBy(func)案例

1. 作用：分组，按照传入函数的返回值进行分组。将相同的 key 对应的值放入一个迭代器。

2. 需求：创建一个 RDD，按照元素模以 2 的值进行分组。

（1）创建

```
scala> val rdd = sc.parallelize(1 to 4)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[65] at parallelize at <console>:24
```

（2）按照元素模以 2 的值进行分组

```
scala> val group = rdd.groupBy( %2)
group: org.apache.spark.rdd.RDD[(Int, Iterable[Int])] = ShuffledRDD[2] at groupBy at <console>:26
```

（3）打印结果

```
scala> group.collect
res0: Array[(Int, Iterable[Int])] = Array((0,CompactBuffer(2, 4)), (1,CompactBuffer(1, 3)))
```

29.8 filter(func) 案例

1. 作用：过滤。返回一个新的 RDD，该 RDD 由经过 func 函数计算后返回值为 true 的输入元素组成。

2. 需求：创建一个 RDD（由字符串组成），过滤出一个新 RDD（包含”xiao”子串）

（1）创建

```
scala> var sourceFilter = sc.parallelize(Array("xiaoming","xiaojiang","xiaohe","dazhi"))
sourceFilter: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[10] at parallelize at <console>:24
```


（2）打印

```
scala> sourceFilter.collect()
res9: Array[String] = Array(xiaoming, xiaojiang, xiaohe, dazhi)
```

（3）过滤出含” xiao”子串的形成一个新的 RDD

```
scala> val filter = sourceFilter.filter(_.contains("xiao"))
filter: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[11] at filter at <console>:26
```

（4）打印新 RDD

```
scala> filter.collect()
res10: Array[String] = Array(xiaoming, xiaojiang, xiaohe)
```

29.9 sample(withReplacement, fraction, seed) 案例

1. 作用：以指定的随机种子随机抽样出数量为 fraction 的数据，withReplacement 表示是抽出的数据是否放回，true 为有放回的抽样，false 为无放回的抽样，seed 用于指定随机数生成器种子。

2. 需求：创建一个 RDD（1-10），从中选择放回和不放回抽样

（1）创建 RDD

```
scala> val rdd = sc.parallelize(1 to 10)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[20] at parallelize at <console>:24
```

（2）打印

```
scala> rdd.collect()
```

```
res15: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

（3）放回抽样

```
scala> var sample1 = rdd.sample(true,0.4,2)
sample1: org.apache.spark.rdd.RDD[Int] = PartitionwiseSampledRDD[21] at sample at <console>:26
```

（4）打印放回抽样结果

```
scala> sample1.collect()
res16: Array[Int] = Array(1, 2, 2, 7, 7, 8, 9)
```

（5）不放回抽样

```
scala> var sample2 = rdd.sample(false,0.2,3)
sample2: org.apache.spark.rdd.RDD[Int] = PartitionwiseSampledRDD[22] at sample at <console>:26
```

（6）打印不放回抽样结果

```
scala> sample2.collect()
res17: Array[Int] = Array(1, 9)
```

29.10 distinct([numTasks])) 案例

1. 作用：对源 RDD 进行去重后返回一个新的 RDD。

2. 需求：创建一个 RDD，使用 distinct()对其去重。

（1）创建一个 RDD

```
scala> val distinctRdd = sc.parallelize(List(1,2,1,5,2,9,6,1))
distinctRdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[34] at parallelize at <console>:24
```

（2）对 RDD 进行去重

```
scala> val unionRDD = distinctRdd.distinct()
unionRDD: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[37] at distinct at <console>:26
```

（3）打印去重后生成的新 RDD

```
scala> unionRDD.collect()
res20: Array[Int] = Array(1, 9, 5, 6, 2)
```

（4）对 RDD

```
scala> val unionRDD = distinctRdd.distinct(2)
unionRDD: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[40] at distinct at <console>:26
```

（5）打印去重后生成的新 RDD

```
scala> unionRDD.collect()
res21: Array[Int] = Array(6, 2, 1, 9, 5)
```

29.11 coalesce(numPartitions) 案例

1. 作用：缩减分区数，用于大数据集过滤后，提高小数据集的执行效率。

2. 需求：创建一个 4 个分区的 RDD，对其缩减分区

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(1 to 16,4)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[54] at parallelize at <console>:24
```

（2）查看 RDD 的分区数

```
scala> rdd.partitions.size
res20: Int = 4
```

（3）对 RDD 重新分区

```
scala> val coalesceRDD = rdd.coalesce(3)
coalesceRDD: org.apache.spark.rdd.RDD[Int] = CoalescedRDD[55] at coalesce at <console>:26
```

（4）查看新 RDD 的分区数

```
scala> coalesceRDD.partitions.size
res21: Int = 3
```

29.12 repartition(numPartitions) 案例

1. 作用：根据分区数，重新通过网络随机洗牌所有数据。

2. 需求：创建一个 4 个分区的 RDD，对其重新分区

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(1 to 16,4)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[56] at parallelize at <console>:24
```


（2）查看 RDD 的分区数

```
scala> rdd.partitions.size
res22: Int = 4
```

（3）对 RDD 重新分区

```
scala> val rerdd = rdd.repartition(2)
rerdd: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[60] at repartition at <console>:26
```

（4）查看新 RDD 的分区数

```
scala> rerdd.partitions.size
res23: Int = 2
```

coalesce 和 repartition 的区别

1. coalesce 重新分区，可以选择是否进行 shuffle 过程。由参数 shuffle: Boolean = false/true 决定。

2. repartition 实际上是调用的 coalesce，进行 shuffle。源码如下：

```
def repartition(numPartitions: Int)(implicit ord: Ordering[T] = null): RDD[T] = withScope {
  coalesce(numPartitions, shuffle = true)
}
```

29.13 partitionBy 案例

1. 作用：对 pairRDD 进行分区操作，如果原有的 partionRDD 和现有的 partionRDD 是一致的话就不进行分区， 否则会生成 ShuffleRDD，即会产生 shuffle 过程。

2. 需求：创建一个 4 个分区的 RDD，对其重新分区

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(Array((1,"aaa"),(2,"bbb"),(3,"ccc"),(4,"ddd")),4)
rdd: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[44] at parallelize at <console>:24
```

（2）查看 RDD 的分区数

```
scala> rdd.partitions.size
res24: Int = 4
```

（3）对 RDD 重新分区

```
scala> var rdd2 = rdd.partitionBy(new org.apache.spark.HashPartitioner(2))
rdd2: org.apache.spark.rdd.RDD[(Int, String)] = ShuffledRDD[45] at partitionBy at <console>:26
```

（4）查看新 RDD 的分区数

```
scala> rdd2.partitions.size
res25: Int = 2
```


29.14 reduceByKey(func, [numTasks]) 案例

1. 在一个(K,V)的 RDD 上调用，返回一个(K,V)的 RDD，使用指定的 reduce 函数，将相同 key 的值聚合到一起，reduce 任务的个数可以通过第二个可选的参数来设置。

2. 需求：创建一个 pairRDD，计算相同 key 对应值的相加结果

（1）创建一个 pairRDD

```
scala> val rdd = sc.parallelize(List(("female",1),("male",5),("female",5),("male",2)))
rdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[46] at parallelize at <console>:24
```

（2）计算相同 key 对应值的相加结果

```
scala> val reduce = rdd.reduceByKey((x,y) => x+y)
reduce: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[47] at reduceByKey at <console>:26
```

（3）打印结果

```
scala> reduce.collect()
res29: Array[(String, Int)] = Array((female,6), (male,7))
```

29.15 groupByKey 案例

1. 作用：groupByKey 也是对每个 key 进行操作，但只生成一个 seq。

2. 需求：创建一个 pairRDD，将相同 key 对应值聚合到一个 seq 中，并计算相同 key 对

应值的相加结果。

（1）创建一个 pairRDD

```
scala> val words = Array("one", "two", "two", "three", "three", "three")
words: Array[String] = Array(one, two, two, three, three, three)

scala> val wordPairsRDD = sc.parallelize(words).map(word => (word, 1))
wordPairsRDD: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[4] at map at <console>:26
```

（2）将相同 key 对应值聚合到一个 Seq 中

```
scala> val group = wordPairsRDD.groupByKey()
group: org.apache.spark.rdd.RDD[(String, Iterable[Int])] = ShuffledRDD[5] at groupByKey at <console>:28
```

（3）打印结果

```
scala> group.collect()
res1: Array[(String, Iterable[Int])] = Array((two,CompactBuffer(1, 1)), (one,CompactBuffer(1)), (three,CompactBuffer(1, 1, 1)))
```

（4）计算相同 key 对应值的相加结果

```
scala> group.map(t => (t._1, t._2.sum))
res2: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[6] at map at <console>:31
```

（5）打印结果

```
scala> res2.collect()
res3: Array[(String, Int)] = Array((two,2), (one,1), (three,3))
```


29.16 reduceByKey 和 groupByKey 的区别

1. reduceByKey：按照 key 进行聚合，在 shuffle 之前有 combine（预聚合）操作，返回结果是 RDD[k,v]。

2. groupByKey：按照 key 进行分组，直接进行 shuffle。

3. 开发指导：reduceByKey 比 groupByKey，建议使用。但是需要注意是否会影响业务逻辑。

29.17join(otherDataset, [numTasks]) 案例

1. 作用：在类型为(K,V)和(K,W)的 RDD 上调用，返回一个相同 key 对应的所有元素对在一起的(K,(V,W))的 RDD

2. 需求：创建两个 pairRDD，并将 key 相同的数据聚合到一个元组。

（1）创建第一个 pairRDD

```
scala> val rdd = sc.parallelize(Array((1,"a"),(2,"b"),(3,"c")))
rdd: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[32] at parallelize at <console>:24
```

（2）创建第二个 pairRDD

```
scala> val rdd1 = sc.parallelize(Array((1,4),(2,5),(4,6)))
rdd1: org.apache.spark.rdd.RDD[(Int, Int)] = ParallelCollectionRDD[33] at parallelize at
```

```
<console>:24
```

（3）join 操作并打印结果

```
scala> rdd.join(rdd1).collect()
res13: Array[(Int, (String, Int))] = Array((1,(a,4)), (2,(b,5)), (3,(c,6)))
```

Action 算子

29.18 reduce(func)案例

1. 作用：通过 func 函数聚集 RDD 中的所有元素，先聚合分区内数据，再聚合分区间数据。

2. 需求：创建一个 RDD，将所有元素聚合得到结果

（1）创建一个 RDD[Int]

```
scala> val rdd1 = sc.makeRDD(1 to 10,2)
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[85] at makeRDD at <console>:24
```

（2）聚合 RDD[Int]所有元素

```
scala> rdd1.reduce(_+_)
res50: Int = 55
```

（3）创建一个 RDD[String]

```
scala> val rdd2 = sc.makeRDD(Array(("a",1),("a",3),("c",3),("d",5)))
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[86] at makeRDD at <console>:24
```

（4）聚合 RDD[String]所有数据

```
scala> rdd2.reduce((x,y)=>(x._1 + y._1,x._2 + y._2))
res51: (String, Int) = (adca,12)
```

29.19 countByKey()案例

1. 作用：针对(K,V)类型的 RDD，返回一个(K,Int)的 map，表示每一个 key 对应的元素个数。

2. 需求：创建一个 PairRDD，统计每种 key 的个数

（1）创建一个 PairRDD

```
scala> val rdd = sc.parallelize(List((1,3),(1,2),(1,4),(2,3),(3,6),(3,8)),3)
rdd: org.apache.spark.rdd.RDD[(Int, Int)] = ParallelCollectionRDD[95] at parallelize at <console>:24
```

（2）统计每种 key 的个数

```
scala> rdd.countByKey
res63: scala.collection.Map[Int,Long] = Map(3 -> 2, 1 -> 3, 2 -> 1)
```

29.20 foreach(func)案例

1. 作用：在数据集的每一个元素上，运行函数 func 进行更新。

2. 需求：创建一个 RDD，对每个元素进行打印

（1）创建一个 RDD

```
scala> var rdd = sc.makeRDD(1 to 5,2)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[107] at makeRDD at <console>:24
```

（2）对该 RDD 每个元素进行打印

```
scala> rdd.foreach(println(_))
3
4
5
1
2
```

29.21 sortByKey([ascending], [numTasks]) 案例

1. 作用：在一个(K,V)的 RDD 上调用，K 必须实现 Ordered 接口，返回一个按照 key 进行排序的(K,V)的 RDD

2. 需求：创建一个 pairRDD，按照 key 的正序和倒序进行排序

（1）创建一个 pairRDD

```
scala> val rdd = sc.parallelize(Array((3,"aa"),(6,"cc"),(2,"bb"),(1,"dd")))
rdd: org.apache.spark.rdd.RDD[(Int, String)] = ParallelCollectionRDD[14] at parallelize at <console>:24
```

（2）按照 key 的正序

```
scala> rdd.sortByKey(true).collect()
res9: Array[(Int, String)] = Array((1,dd), (2,bb), (3,aa), (6,cc))
```

（3）按照 key 的倒序

```
scala> rdd.sortByKey(false).collect()
res10: Array[(Int, String)] = Array((6,cc), (3,aa), (2,bb), (1,dd))
```

29.22 collect()案例

1. 作用：在驱动程序中，以数组的形式返回数据集的所有元素。

2. 需求：创建一个 RDD，并将 RDD 内容收集到 Driver 端打印

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(1 to 10)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24
```

（2）将结果收集到 Driver 端

```
scala> rdd.collect
```

```
res0: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

29.23 count()案例

1. 作用：返回 RDD 中元素的个数
2. 需求：创建一个 RDD，统计该 RDD 的条数

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(1 to 10)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24
```

（2）统计该 RDD 的条数

```
scala> rdd.count
res1: Long = 10
```

29.24 first()案例

1. 作用：返回 RDD 中的第一个元素
2. 需求：创建一个 RDD，返回该 RDD 中的第一个元素

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(1 to 10)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24
```


（2）统计该 RDD 的条数

```
scala> rdd.first
res2: Int = 1
```

29.25 take(n)案例

1. 作用：返回一个由 RDD 的前 n 个元素组成的数组
2. 需求：创建一个 RDD，统计该 RDD 的条数

（1）创建一个 RDD

```
scala> val rdd = sc.parallelize(Array(2,5,4,6,8,3))
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[2] at parallelize at <console>:24
```

（2）统计该 RDD 的条数

```
scala> rdd.take(3)
res10: Array[Int] = Array(2, 5, 4)
```

30、什么是宽窄依赖

窄依赖指的是每一个父 RDD 的 Partition 最多被子 RDD 的一个 Partition 使用。

宽依赖指的是多个子 RDD 的 Partition 会依赖同一个父 RDD 的 Partition，会引起 shuffle。

31、任务划分的几个重要角色

RDD 任务切分中间分为：Application、Job、Stage 和 Task

1）Application：初始化一个 SparkContext 即生成一个 Application；

2）Job：一个 Action 算子就会生成一个 Job；

3）Stage：根据 RDD 之间的依赖关系的不同将 Job 划分成不同的 Stage，遇到一个宽依赖则划分一个 Stage；

4）Task：Stage 是一个 TaskSet，将 Stage 划分的结果发送到不同的 Executor 执行即为一个 Task。

32、spark 中自定义函的过程

自定义 UDF 的过程

1）创建 DataFrame

2）打印数据

3）注册 UDF，功能为在数据前添加字符串

4）创建临时表

5）应用 UDF

33、Spark 有哪两种算子？

Transformation（转化）算子和 Action（执行）算子。

34、Spark 并行度怎么设置比较合适？

官方建议：每个 core 承载 2~4 个 partition,

说明：如，32 个 core，那么 64~128 之间的并行度，也就是设置 64~128 个 partion

35、collect 功能是什么？

driver 通过 collect 把集群中各个节点的内容收集过来汇总

36、cache 后面能不能接其他算子，它是不是 action 操作？

cache 可以接其他算子，但是接了算子之后，起不到缓存应有的效果，因为会重新触发 cache。cache 不是 action 操作。

37、reduceByKey 是不是 action？

不是，很多人都会以为是 action，reduce rdd 是 action

38、Spark 的数据本地性有哪几种？

Spark 中的数据本地性有三种：

1）PROCESS_LOCAL 是指读取缓存在本地节点的数据

2）NODE_LOCAL 是指读取本地节点硬盘数据

3）ANY 是指读取非本地节点数据

39、Spark 使用 parquet 文件存储格式能带来哪些好处？

1）采用 parquet 可以极大的优化 spark 的调度和执行。

2）速度更快

3）parquet 的压缩技术非常稳定

4）极大的减少磁盘 I/o,通常情况下能够减少 75%的存储空间

40、不需要排序的 hash shuffle 是否一定比需要排序的 sort shuffle 速度快？

不一定。

当数据规模小，Hash shuffle 快于 Sorted Shuffle 数据规模大的时候。

当数据量大，sorted Shuffle 会比 Hash shuffle 快很多，因为 Hash shuffle 在数量大的时候有很多小文件，不均匀，甚至出现数据倾斜，消耗内存大。

41、spark 中的数据为什么要进行序列化？

可以减少数据的体积，减少存储空间，高效存储和传输数据，不好的是使用的时候要反序列化，非常消耗 CPU。

42、不启动 Spark 集群 Master 和 work 服务，可不可以运行 Spark 程序？

可以，只要资源管理器第三方管理就可以，如由 yarn 管理。

43、spark on yarn Cluster 模式下，ApplicationMaster 和 driver 是在同一个进程么？

是，driver 位于 ApplicationMaster 进程中。该进程负责申请资源，还负责监控程序、资源的动态情况。

44、Executor 启动时，资源通过哪几个参数指定？

1）num-executors 是 executor 的数量

2）executor-memory 是每个 executor 使用的内存

3）executor-cores 是每个 executor 分配的 CPU

45、什么是 shuffle，以及为什么需要 shuffle？

shuffle 中文翻译为洗牌，需要 shuffle 的原因是：某种具有共同特征的数据汇聚到一个计算节点上进行计算。

46、一个 task 的 map 数量由谁来决定？

一般情况下，在输入源是文件的时候，一个 task 的 map 数量由 splitSize 来决定的。

一个 task 的 reduce 数量，由 partition 决定。

47、列出你所知道的调度器？

1）FiFo schedular 默认的调度器 先进先出

2）Capacity schedular 容量调度器

3）Fair schedular 公平调度器

48、union 操作是产生宽依赖还是窄依赖？

产生窄依赖。

49、窄依赖父 RDD 的 partition 和子 RDD 的 parition 是不是都是一对一的关系？

不一定

50、Hadoop 中，Mapreduce 操作的 mapper 和 reducer 阶段相当于 spark 中的哪几个算子？

相当于 spark 中的 map 算子和 reduceByKey 算子。

51、Spark 中的 HashShufle 的有哪些不足？

1）shuffle 产生海量的小文件在磁盘上，此时会产生大量耗时的、低效的 IO 操作；

2）容易导致内存不够用，由于内存需要保存海量的文件操作句柄和临时缓存信息，如果数据处理规模比较大的话，容易出现 OOM；

3）容易出现数据倾斜，导致 OOM。

52、spark.default.parallelism 这个参数有什么意义，实际生产中如何设置？

1）参数用于设置每个 stage 的默认 task 数量。

2）很多人都不会设置这个参数，会使得集群非常低效，你的 cpu，内存再多，如果 task 始终为 1，那也是浪费， spark 官网建议 task 个数为 CPU 的核数*executor 的个数的 2~3 倍。

53、Spark 中 standalone 模式特点，有哪些优点和缺点？

1）特点：

（1）standalone 是 master/slave 架构，集群由 Master 与 Worker 节点组成，程序通过与 Master 节点交互申请资　源，Worker 节点启动 Executor 运行；

（2）standalone 调度模式使用 FIFO 调度方式；

（3）无依赖任何其他资源管理系统，Master 负责管理集群资源。

2）优点：

（1）部署简单；

（2）不依赖其他资源管理系统。

3）缺点：

（1）默认每个应用程序会独占所有可用节点的资源，当然可以通过 spark.cores.max 来决定一个应用可以申请的 CPU cores 个数；

（2）可能有单点故障，需要自己配置 master HA。

54、Spark sql 为什么比 hive 快呢？

计算引擎不一样，一个是 spark 计算模型，一个是 mapreudce 计算模型。

55、RDD 的基本属性？

一个 RDD 对象，包含如下 5 个核心属性。

1）一个分区列表，每个分区里是 RDD 的部分数据（或称数据块）。

2）一个依赖列表，存储依赖的其他 RDD。

3）一个名为 compute 的计算函数，用于计算 RDD 各分区的值。

4）分区器（可选），用于键/值类型的 RDD，比如某个 RDD 是按散列来分区。

5）计算各分区时优先的位置列表（可选）