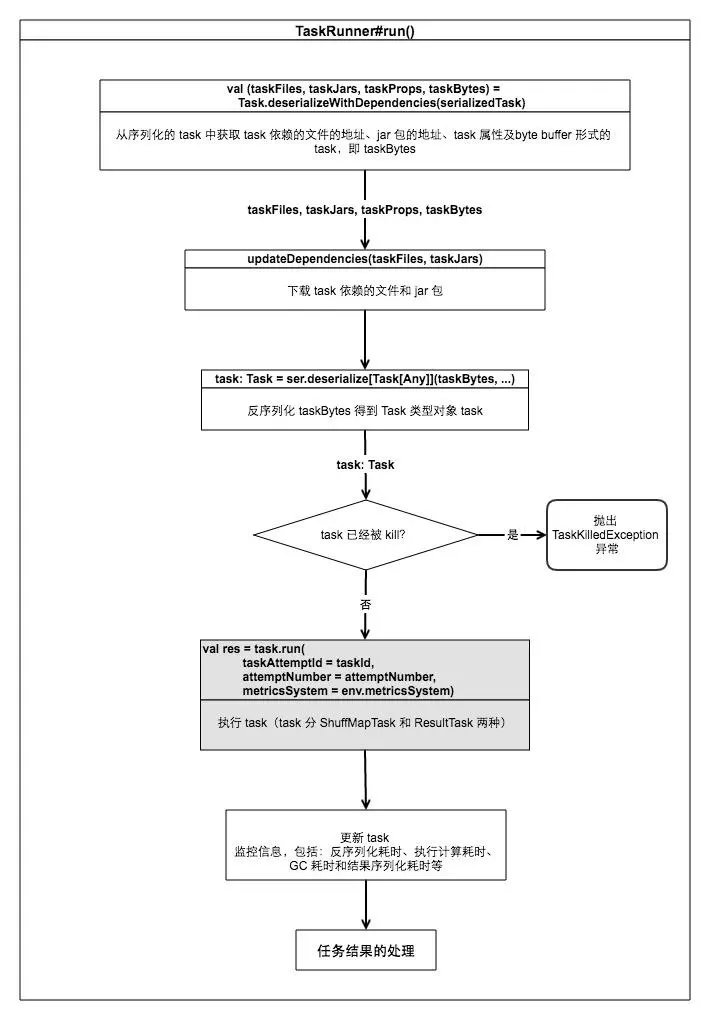
## Spark Task的执行流程--执行task

TaskRunner（继承于 Runnable） 对象最终会被提交到 Executor 的线程池中去执行， 该执行过程封装在 TaskRunner#run() 中，搞懂该函数就搞懂了 task 是如何执行的，该函数的核心实现如下：



需要注意的是，上图的流程都是在 Executor 的线程池中的某条线程中执行的。上图中最复杂和关键的是 task.run(...) 以及任务结果的处理，也即怎么把各个 partition 计算结果汇报到 driver 端。

task 结果处理这一块内容将另写一篇文章进行说明，下文主要对 task.run(...) 进行分析。Task 类共有两种实现：

* ResultTask：对于 DAG 图中最后一个 Stage（也就是 ResultStage），会生成与该 DAG 图中最后一个 RDD（DAG 图中最后边）partition 个数相同的 ResultTask
* ShuffleMapTask：对于非最后的 Stage（也就是 ShuffleMapStage），会生成与该 Stage 最后的 RDD partition 个数相同的 ShuffleMapTask

在 Task#run(...) 方法中最重要的是调用了 Task#runTask(context: TaskContext) 方法，来分别看看 ResultTask 和 ShuffleMapTask 的实现：

### ResultTask#runTask(context: TaskContext)

|  |
| --- |
| override def runTask(context: TaskContext): U = {  // Deserialize the RDD and the func using the broadcast variables.  val deserializeStartTime = System.currentTimeMillis()  val ser = SparkEnv.get.closureSerializer.newInstance()  //< 反序列化得到 rdd 及 func  val (rdd, func) = ser.deserialize[(RDD[T], (TaskContext, Iterator[T]) => U)](ByteBuffer.wrap(taskBinary.value), Thread.currentThread.getContextClassLoader)  \_executorDeserializeTime = System.currentTimeMillis() - deserializeStartTime  //< 对 rdd 指定 partition 的迭代器执行 func 函数  func(context, rdd.iterator(partition, context))  } |

实现代码如上，主要做了两件事：

1. 反序列化得到 rdd 及 func
2. 对rdd指定partition的迭代器执行func函数并返回结果

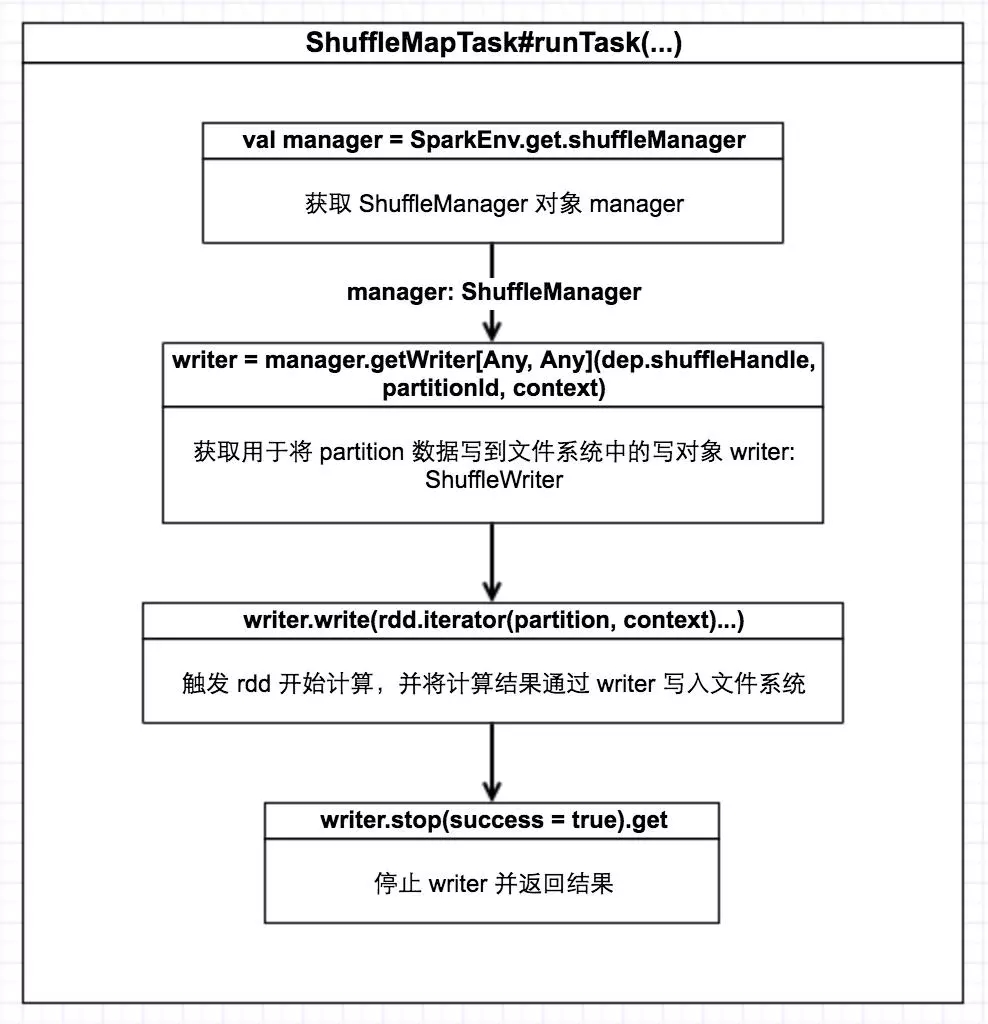
func 函数是什么呢？举几个例子就很容易明白：

* 对于RDD#count()的ResultTask 这里的func真正执行的是def getIteratorSize[T](iterator: Iterator[T]): Long，即计算该partition对应的迭代器的数据条数
* 对于RDD#take(num: Int): Array[T]的ResultTask这里的func真正执行的是 (it: Iterator[T]) => it.take(num).toArray，即取该partition对应的迭代器的前num条数据

也就是说，func是对已经计算获得的 RDD 的某个 partition 的迭代器执行在 RDD action 中预定义好的操作，具体的操作根据不同的 action 不同而不同。而这个 partition 对应的迭代器的获取是通过调动 RDD#iterator(split: Partition, context: TaskContext): Iterator[T] 去获取的，会通过计算或从 cache 或 checkpoint 中获取。

### ShuffleMapTask#runTask(context: TaskContext)

与 ResultTask 对 partition 数据进行计算得到计算结果并汇报给 driver 不同，ShuffleMapTask 的职责是为下游的 RDD 计算出输入数据。更具体的说，ShuffleMapTask 要计算出 partition 数据并通过 shuffle write 写入磁盘（由 BlockManager 来管理）来等待下游的 RDD 通过 shuffle read 读取，其核心流程如下：



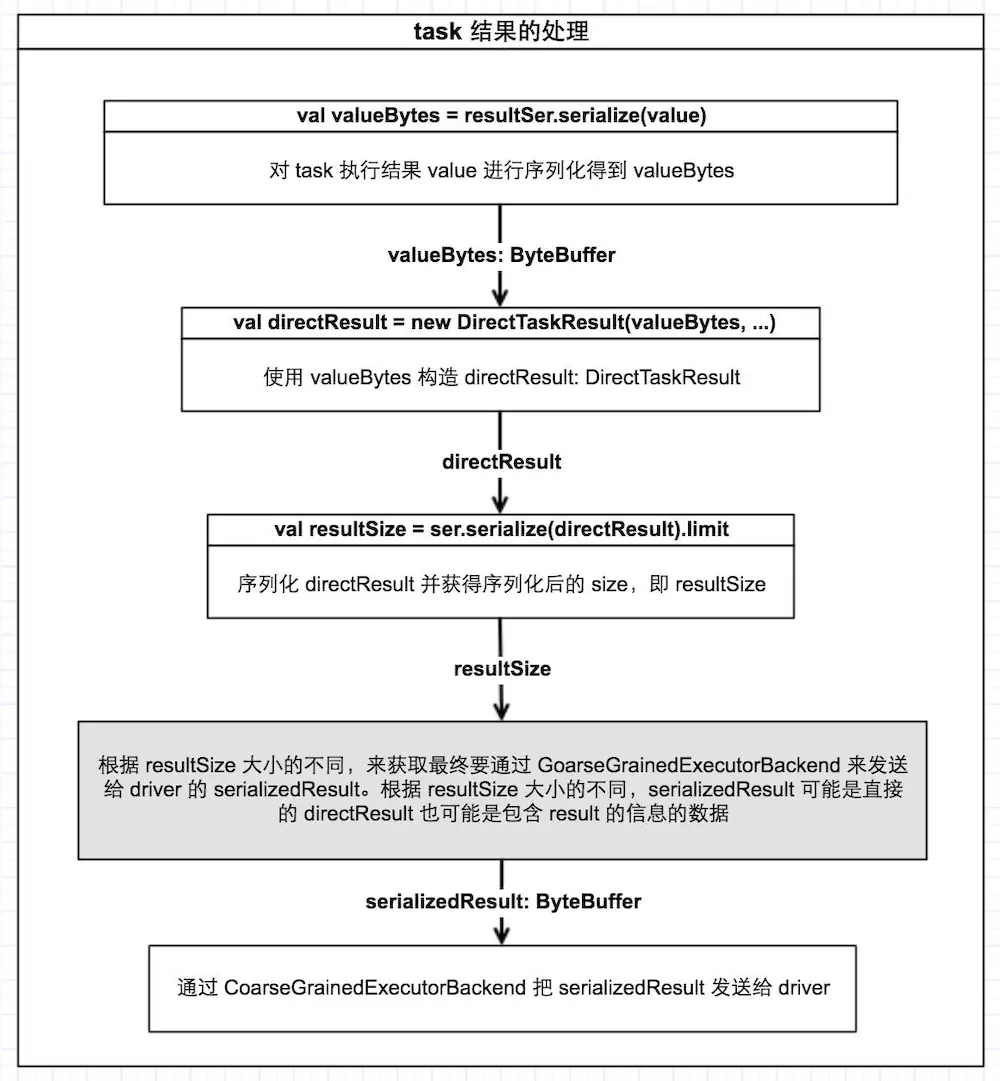
共分为四步：

1. 从 SparkEnv 中获取 ShuffleManager 对象，当前支持 Hash、Sort Based、Tungsten-sort Based 以及自定义的 Shuffle（关于 shuffle 之后会专门写文章说明）
2. 从 ShuffleManager 中获取 ShuffleWriter 对象 writer
3. 得到对应 partition 的迭代器后，通过 writer 将数据写入文件系统中
4. 停止 writer 并返回结果

## Spark Task的执行流程--task结果的处理

### worker 端的处理

处理 task 的结果是在 TaskRunner#run() 中进行的，紧接着 task 执行步骤，结果处理的核心流程如下：



我们进一步展开上图中浅灰色背景步骤，根据 resultSize（序列化后的 task 结果大小） 大小的不同，共有三种情况：

* resultSize > spark.driver.maxResultSize 配置值（默认1G）：直接丢弃，若有必要需要修改 spark.driver.maxResultSize 的值。此时，serializedResult 为序列化的 IndirectTaskResult 对象，driver 之后通过该对象是获得不到结果的
* resultSize > maxDirectResultSize 且 resultSize <= spark.driver.maxResultSize 配置值：maxDirectResultSize 为配置的 spark.rpc.message.maxSize 与 spark.task.maxDirectResultSize 更小的值；这种情况下，会将结果存储到 BlockManager 中。此时，serializedResult 为序列化的 IndirectTaskResult 对象，driver 之后可以通过该对象在 BlockManager 系统中拉取结果
* resultSize <= maxDirectResultSize：serializedResult 直接就是 serializedDirectResult

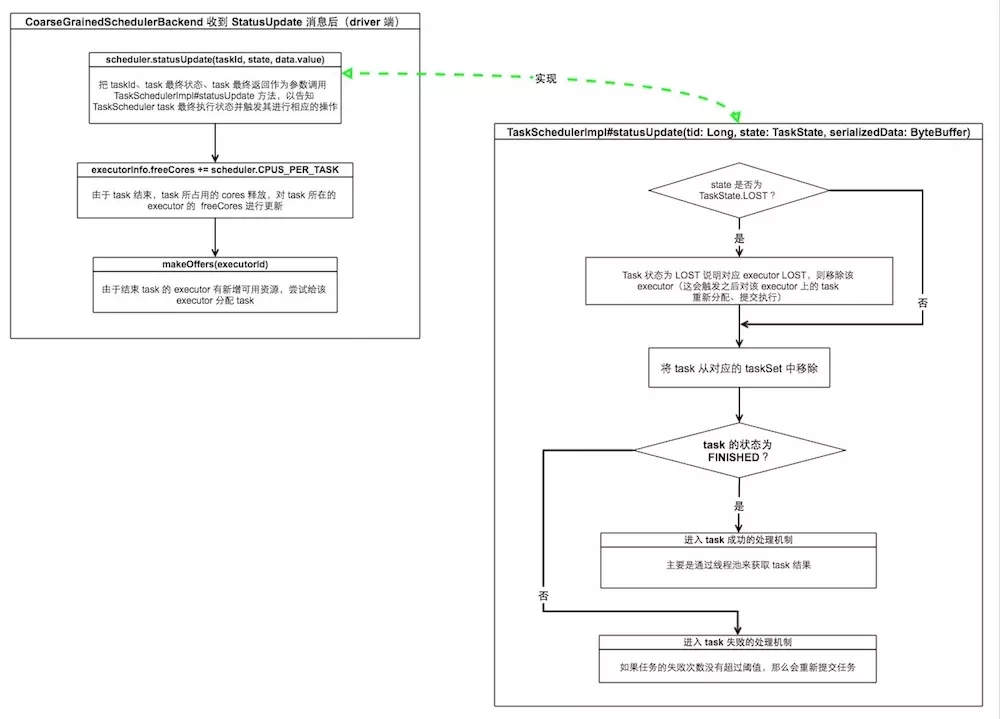
在拿到 serializedResult 之后，调用 CoarseGrainedExecutorBackend#statusUpdate 方法，如下：

|  |
| --- |
| execBackend.statusUpdate(taskId, TaskState.FINISHED, serializedResult) |

该方法会使用 driverRpcEndpointRef 发送一条包含 serializedResult 的 StatusUpdate 消息给 driver (更具体说是其中的 CoarseGrainedSchedulerBackend 对象)

### driver 端的处理

driver 端的 CoarseGrainedSchedulerBackend 在收到 worker 端发送的 StatusUpdate 消息后，会进行一系列的处理，包括调用 TaskScheduler 方法以做通知，主要流程如下：



其中，需要说明的是 Task 的状态只有为 FINISHED 时才成功，其他值（FAILED, KILLED, LOST）均为失败。