一:Task执行原理流程图:

1. 当Driver中的CoarseGrainedSchedulerBackend给CoarseGrainedExecutorBackend发送LuanchTask之后，CoarseGrainedExecutorBackend在收到LaunchTask消息后，首先会反序列化TaskDescription；

**val** taskDesc = *ser*.deserialize[TaskDescription](data.value)

1. Executor会通过launchTask来执行Task；
2. TaskRunner在ThreadPool来运行具体的Task，在TaskRunner的run方法中首先会调用statusUpdate给Driver发信息汇报自己的状态说明自己是Running状态；

execBackend.statusUpdate(taskId, TaskState.*RUNNING*, *EMPTY\_BYTE\_BUFFER*)

1. TaskRunner内部会做一些准备工作：例如反序列Task的依赖；

**val** (taskFiles, taskJars, taskBytes) = Task.*deserializeWithDependencies*(serializedTask)

1. 然后通过网络来获取需要的文件、Jar等；

updateDependencies(taskFiles, taskJars)

1. 然后是反序列Task本身；

task = ser.deserialize[Task[Any]](taskBytes, Thread.*currentThread*.getContextClassLoader)

1. 调用反序列化后的Task.run方法来执行任务并获得执行结果：

**val** (value, accumUpdates) = **try** {  
 **val** res = *task*.run(  
 taskAttemptId = taskId,  
 attemptNumber = attemptNumber,  
 metricsSystem = env.metricsSystem)  
 threwException = **false** res  
}

其中Task的run方法调用给的是时候会导致Task的抽象方法RunTask的调用，在Task的runTask内部会调用gRDD的iterator()方法，该方法就是我们针对当前Task所对应的partition进行计算的关键之所在，在处理的内部会迭代partition的元素并交给我们自定义的function进行处理！

对于ShuffleMapTask，首先要对RDD以及其依赖关系进行发序列化：

**val** (rdd, dep) = ser.deserialize[(RDD[\_], ShuffleDependency[\_, \_, \_])]

最终会调用RDD的compute方法：

**def** compute(split: Partition, context: TaskContext): Iterator[T]

具体计算的时候有具体的RDD，例如MapPartitionsRDD的compute：

**override def** compute(split: Partition, context: TaskContext): Iterator[U] =  
 f(context, split.index, firstParent[T].iterator(split, context))

其中的f就是我们在前面的stage中计算具体Partition的业务逻辑代码

对于resultTask：

**val** ser = SparkEnv.*get*.closureSerializer.newInstance()  
**val** (rdd, func) = ser.deserialize[(RDD[T], (TaskContext, Iterator[T]) => U)](  
 ByteBuffer.*wrap*(taskBinary.value), Thread.*currentThread*.getContextClassLoader)  
*\_executorDeserializeTime* = System.*currentTimeMillis*() - deserializeStartTime  
  
*metrics* = *Some*(context.taskMetrics)  
func(context, rdd.iterator(partition, context))

1. 把执行结果序列化：

**val** resultSer = env.serializer.newInstance()

并根据大小判断不同的结果传回给Driver的方式；

1. CoarseGrainedExecutorBackend给DriverEndppoint发送statusUpdate来传输执行结果，DriverEndpoint会把执行结果传递给TaskSchedulerImpl处理，然后交给TaskResultGetter内部通过线程分别处理Task执行结果成功和失败时候的不同情况，然后告诉DAGScheduler任务处理结束的状况。