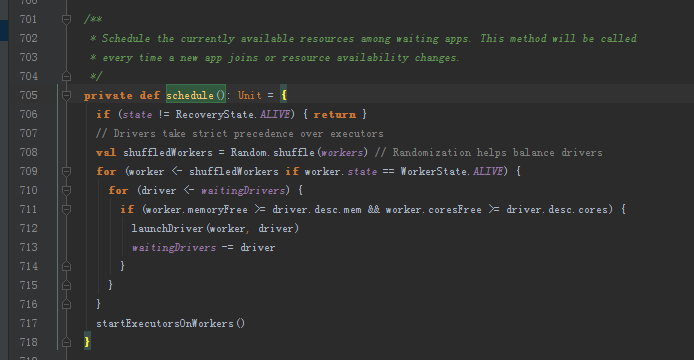
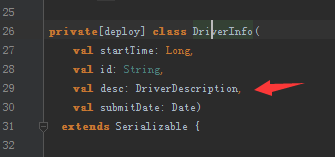
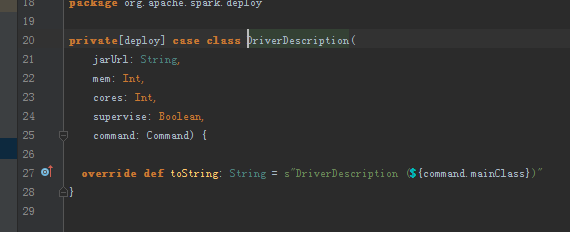
一：任务调度与资源调度：

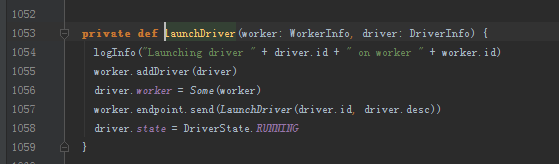
1. 任务调度室通过DAGScheduler，TaskScheduler，SchedulerBackend等进行的作业调度。
2. 资源调度是指应用程序如何获得资源。
3. 任务调度室在资源调度的基础上进行的。

二：资源调度:

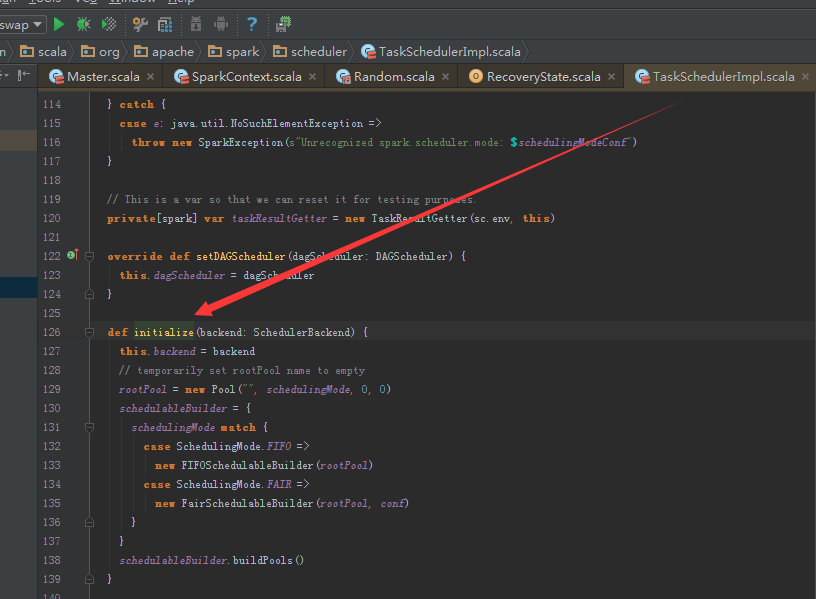
1. 因为Master负责资源调度和管理，所以资源调度的方法schedule位于Master.scala这个类中。
2. Schedule调用的时机：每次新的应用程序提交或者资源状况发生改变的时候（包括Executor增加或者减少，Worker增加或者减少等）。
3. **if** (*state* != RecoveryState.*ALIVE*) { **return** }当前Master必须是Alive的方式才进行资源的调度，如果不是Alive的状态会直接返回，也就是standbyMaster不会进行Application的资源调度。
4. **val** shuffledWorkers = Random.shuffle(*workers*) // Randomization helps balance drivers 使用Random.Shuffle把Master中保留的集群中所有Worker的信息随机打乱，其算法内部是循环交换所有Worker在master缓存数据结构中的位置。
5. **for** (worker <- shuffledWorkers **if** worker.*state* == WorkerState.*ALIVE*) 接下来要判断所有Worker中哪些是Alive级别的Worker，Alive才能够参与资源的分配工作。
6. 当[SparkSubmit](submit.docx)指定Driver在Cluster模式的情况下，此时Driver会加入waitingDrivers等待列表中，在每个DiverInfo的DriverDescription中有要启动Driver时候对Worker的内存及cores的要求的内容。在符合资源要求的情况下然后采用随机打乱后的一个Worker来启动Driver，Master发指令给Worker，[让远程的Worker启动Driver（具体启动过程）](Master启动Driver和Executor.docx)。



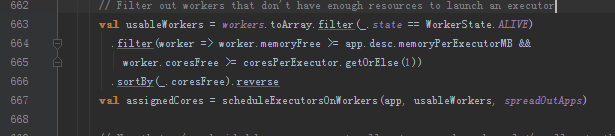




1. 先启动Driver才会发生后续的一切的资源调度的模式。
2. Spark默认为应用程序启动Executor的方式是FIFO的方式，也就是说所有提交的应用程序都是放在调度的等待队列中，先进先出，只有满足了前面应用程序的资源分配的基础上才能够满足下一个应用程序资源的分配。

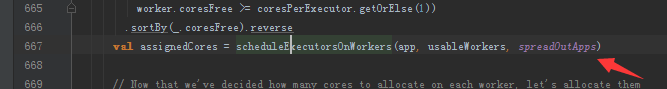


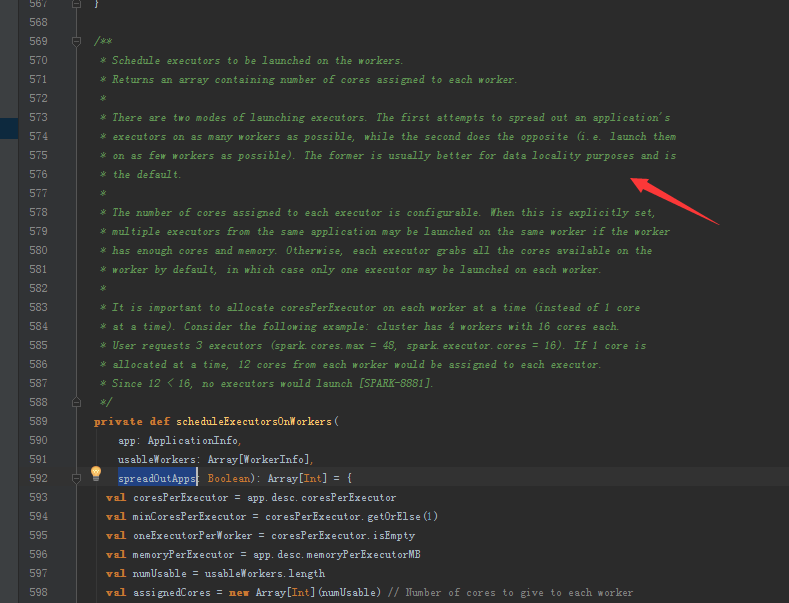
1. 为应用程序具体分配Executor之前要判断应用程序是否还需要分配core，如果不需要则不会为应用程序分配zExecutor；
2. 具体分配Exetutor之前有队要求Worker必须是Alive的状态且必须满足Application对每个Executor的内存和cores的要求，并且在此基础上进行排序：



产生计算资源由小到大的usable workers数据结构。

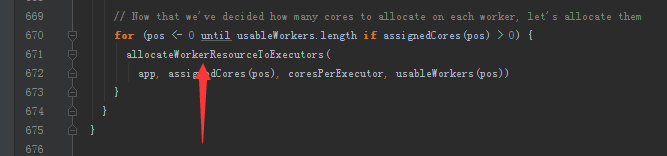
在FIFI的情况下默认是spreadOutApps来让应用程序尽可能多的运行在所有的Node上。

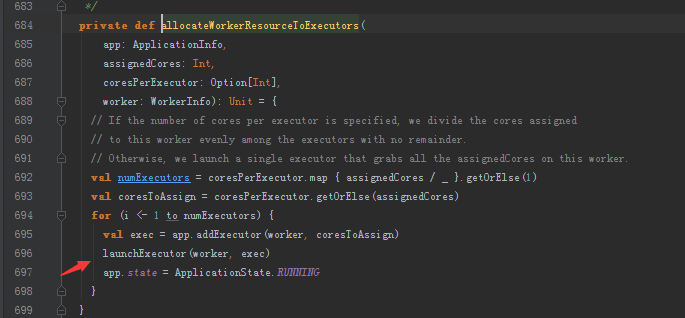


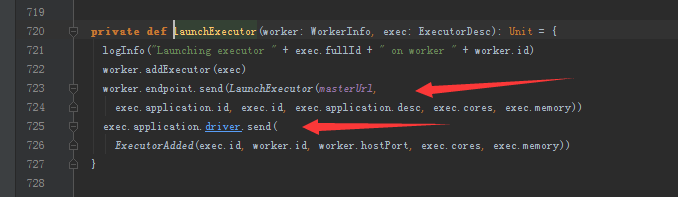
1. 

为应用程序分配Executors有两种方式，第一种方式是尽可能在集群的所有worker上分配Executor，这种方式旺旺会带来潜在的更好的数据本地性。

1. 准备具体要为当前应用程序的Executor信息后，Master要 通过进程通信发送指令给Worker来具体启动ExecutorBackend进程。







1. worker.endpoint.send(*LaunchExecutor*(*masterUrl*,  
    exec.application.id, exec.id, exec.application.desc, exec.cores, exec.memory))

[发送LaunchExecutor消息给Worker（具体启动过程）](Master启动Driver和Executor.docx)