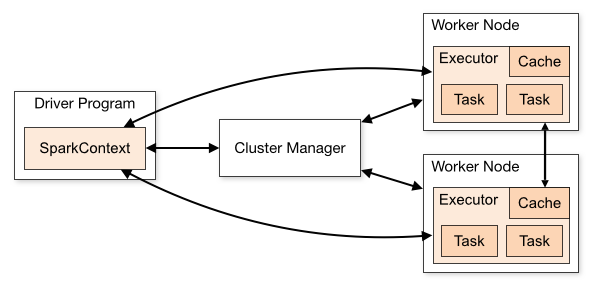
# Spark工作机制

* 集群架构



1. **Driver Program**

**在集群模式下，用户编写的Spark程序称为Driver Program。**

**程序的执行从Driver Program开始，中间会执行一系列的RDD操作，这些操作通过Cluster Manager来调度执行**

1. **SparkContext**

**每个Driver Program包含一个SparkContext对象，负责与Cluster进行沟通**

1. **Cluster Manager**

**负责集群资源调度。**

**三种集群部署模式**

1. **Standalone模式**
2. **Hadoop Yarn模式**
3. **Apache Mesos模式**
4. **Worker Node**

**计算节点**

1. **Cache**

**RDD缓存**

1. **Executor**

**每个Spark Program会在多个Worker Node下启动一个Executor进程，这些Executor专属某个Spark Program，负责Spark在节点上启动的Task、管理内存和磁盘**

1. **Task**

**在Executor上执行的最小单元。比如RDD Transformation操作**

* 作业调度

**在集群上运行时，每个Spark应用程序都会获得一组独立的Executor JVM，这些JVM只运行任务并存储该应用程序的数据。如果多个用户需要共享群集，则根据群集管理器的不同，有不同的选项来管理分配。**

**静态分区**

**对所有集群管理器可用的最简单的选项是资源的静态分区。通过这种方法，每个应用程序都会得到它可以使用的最大数量的资源，并在整个过程中保存它们。**

**Standalone mode**

**默认情况下，提交到独立模式集群的应用程序将以FIFO(先进先出)顺序运行，每个应用程序将尝试使用所有可用节点。**

spark.cores.max

spark.executor.memory

**YARN**

  --num-executors

--executor-memory

--executor-cores

**动态资源分配**

**Spark提供了一种机制，可以根据工作负载动态调整应用程序占用的资源。**

**这意味着，如果不再使用资源，应用程序可能会将资源还给集群，并在有需求时再次请求资源。**

**默认情况下，此功能是禁用的。**

**配置与启动**

1. **为应用设置**spark.dynamicAllocation.enabled true
2. **在每个工作节点设置**spark.shuffle.service.enabled true

YARN mode

1. **将spark/lib目录下的spark-<version>-yarn-shuffle.jar拷贝到所有 NodeManager机器下的hadoop/lib下**
2. **在每个NodeManager的yarn-site.xml中yarn.nodemanager.aux-services配置下新增spark\_shuffle**
3. **在每个NodeManager的yarn-site.xml中设置yarn.nodemanager.aux-services.spark\_shuffle.class为org.apache.spark.network.yarn.YarnShuffleService**
4. **重启所有的NodeManager**

**资源分配策略**

**在一个更好的层次上，Spark应在不使用时释放Executor，需要时再获取Executor。由于没有好的方法来预知Spark何时释放何时需要，因此我们需要一些方法来确定何时需要何时释放。**

**请求策略**

**在开启动态资源分配后，当Spark程序有Task处于排队待调度时，会请求额外的 Executor资源。这种状况说明现有的Executor资源不足以让同时满足所有已提交的**

**任务。**

**Spark会一直循环请求，该策略的触发实际是当排队Task已经等待了spark.dynamicAllocation.schedulerBacklogTimeout秒**

**只要队列中还有排队Task，就每隔spark.dynamicAllocation.sustainedSchedulerBacklogTimeout秒触发一次**

**此外，每一轮请求的Executor数量呈指数级增长，例如第一轮添加一个执行器，**

**然后在随后的几轮添加2、4、8等数量的Executor**

**指数增长政策的动机有两方面。**

**首先，应用程序应该在开始时谨慎地请求执行程序，以防只需要几个额外的执行器就足够了。**

**其次，如果实际需要许多执行者，应用程序应该能够及时增加其资源使用。**

**移除策略**

**移除Executor的策略要简单很多。**

**Spark会在应用程序空闲时间超过spark.dynamicAllocation.executorIdleTimeout秒时，**

**将Executor移除。**

**应用程序中的调度**

**在Spark程序中，不同线程提交的作业可以并行执行。**

**默认情况下，Spark的调度器以FIFO的方式Job。每个Job又分为多个Stage(比如map和reduce)。**

**如果最前面的Job的Stage有Task要运行，则优先获取所有资源。然后才是第二个Job，一次类推。**

**如果队列中的第一个Job不需要使用集群的所有资源，那么第二个Job可以马上运行。**

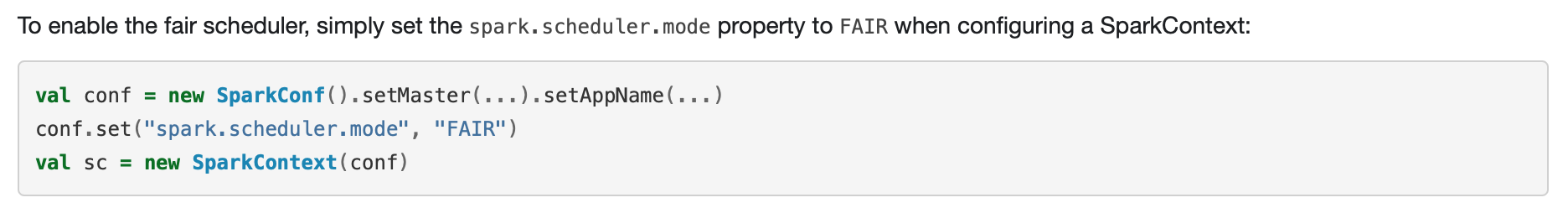
**但如果第一个Job很大，则后面的Job将会被延迟。**

**从Spark 0.8开始，可以配置Job之间的公平共享资源。**

**在公平共享模式下，Spark采用”循环”的方式为不同的Job分配资源，这样所有的Job可以获得差不**

**多相同的资源。这意味着在长作业运行时提交的短作业可以立即开始接收资源，并且仍然可以获得良好**

**的响应时间，而无需等待长作业完成。**

****

**公平调度池**

**公平调度还支持将多个Job分组为池。并对每个池设置不同的选项(例如：权重)。**

**很多时候，我们可以为更重要的任务池设置更高的优先级。**

**在不做任何设置的情况下，每个Job会自动进入默认调度池。**

**我们也可以指定让Job进行哪个池。**

****

**每个线程提交Job后，所有的Job都会使用这个池。**

**也可以清除线程所关联的池。**

****

**调度池的默认行为**

**默认情况下，所有调度池平均共享集群的资源，默认调度池也是。**

**每个调度池中的Job按FIFO的顺序执行。也就说多个Job不会出现资源抢占的情况。**

**调度池配置**

**每个调度池有如下3个属性**

1. schedulingMode

可以是FIFO或FAIR，控制调度池内Job是排队执行还是平均共享资源。

1. weight

控制调度池的权重，是一个相对值。默认为1，表示平均共享资源。

如果某个调度池的权重为2，它就拥有权重为1的调度池2倍的资源。

如果设置了一个非常高的值，比如1000，相当于配置了各个调度池的优先级。

3. minShare

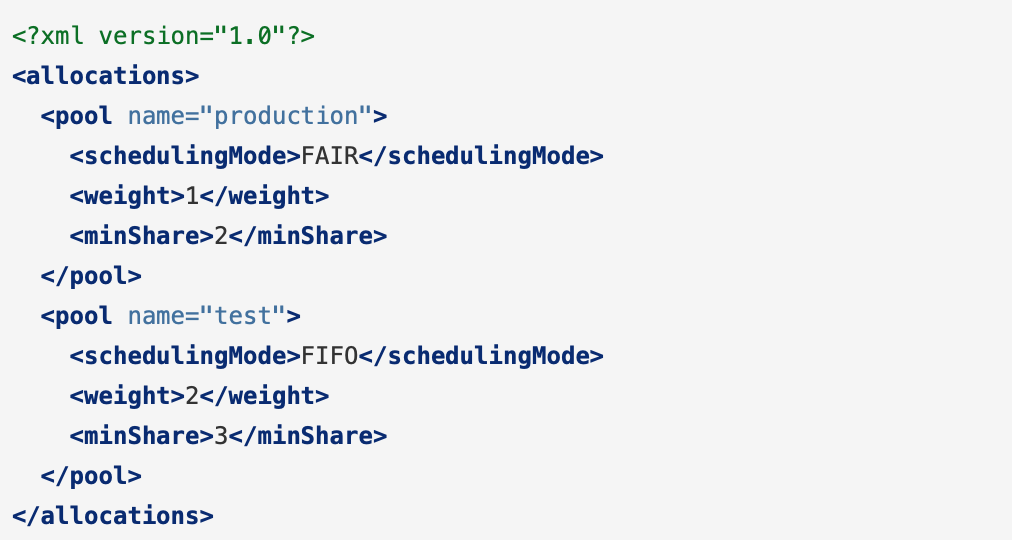
最小资源值(CPU cores数量)，公平调度器在跟中权重重新分配资源之前，

公平调度器总是尝试满足所有池的最小资源。这样可以保证池总会获取一些资源，

而不会被其他优先级高的池抢走。minShare默认为0。

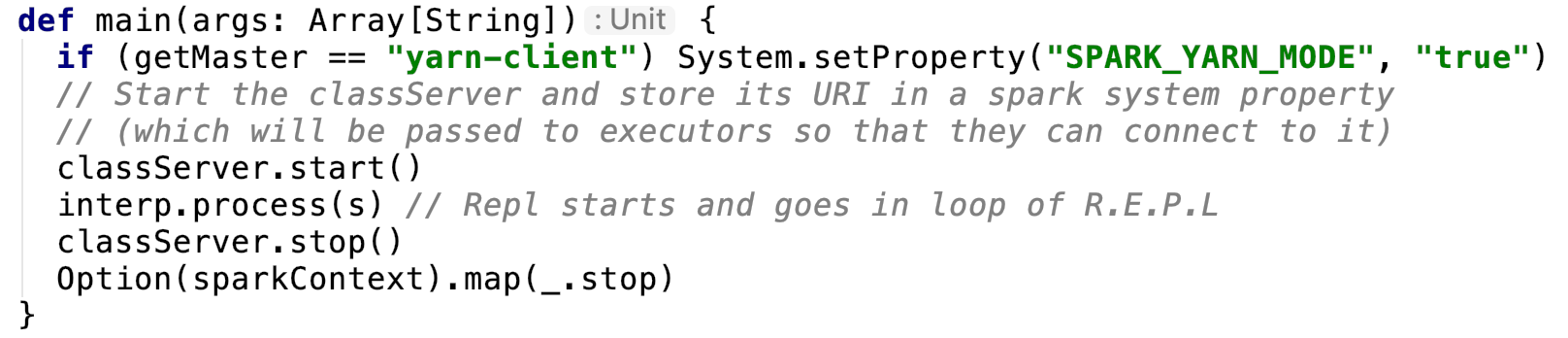
通过创建一个XML来配置池属性，类似conf/fairscheduler.xml.template



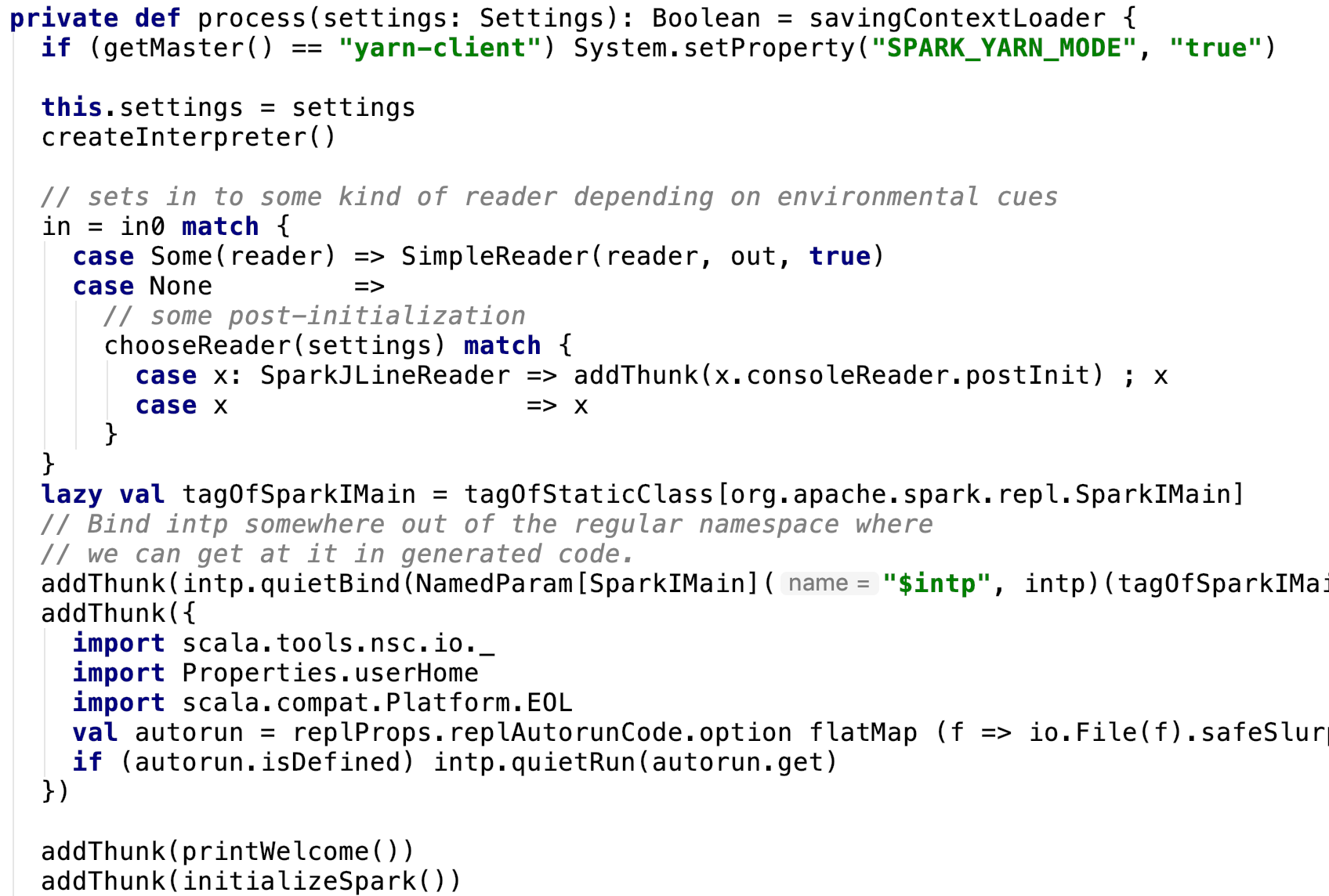


* spark-shell执行原理

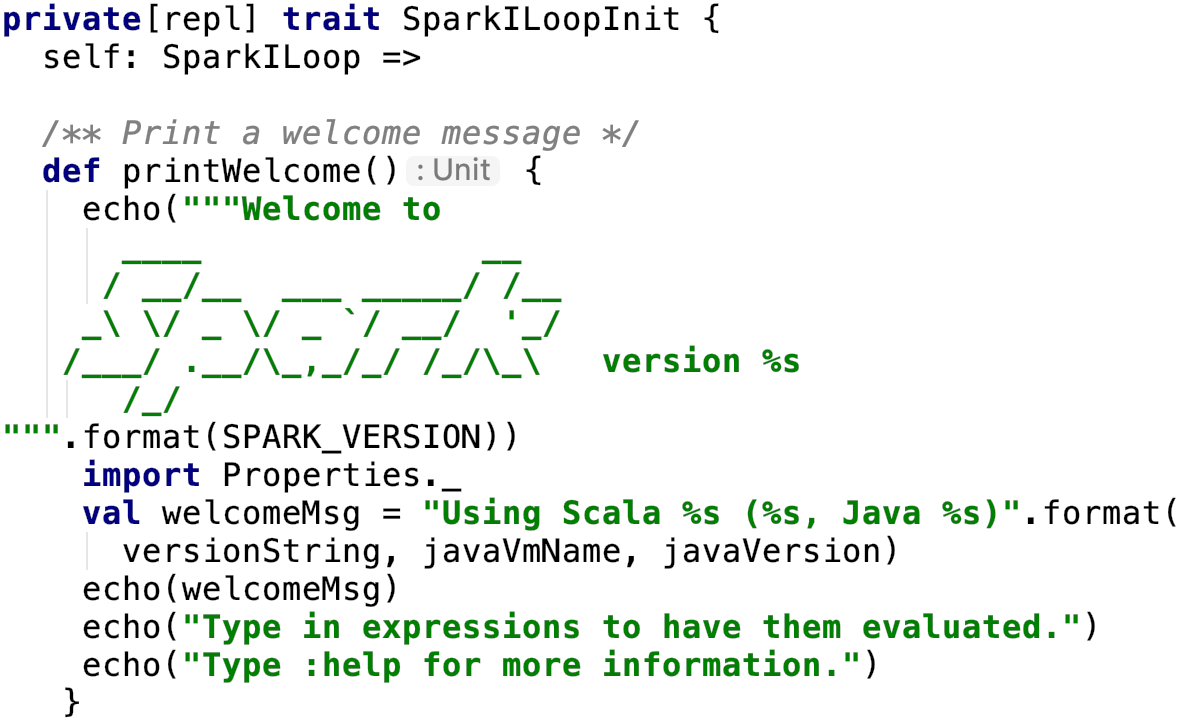
**Main.scala**

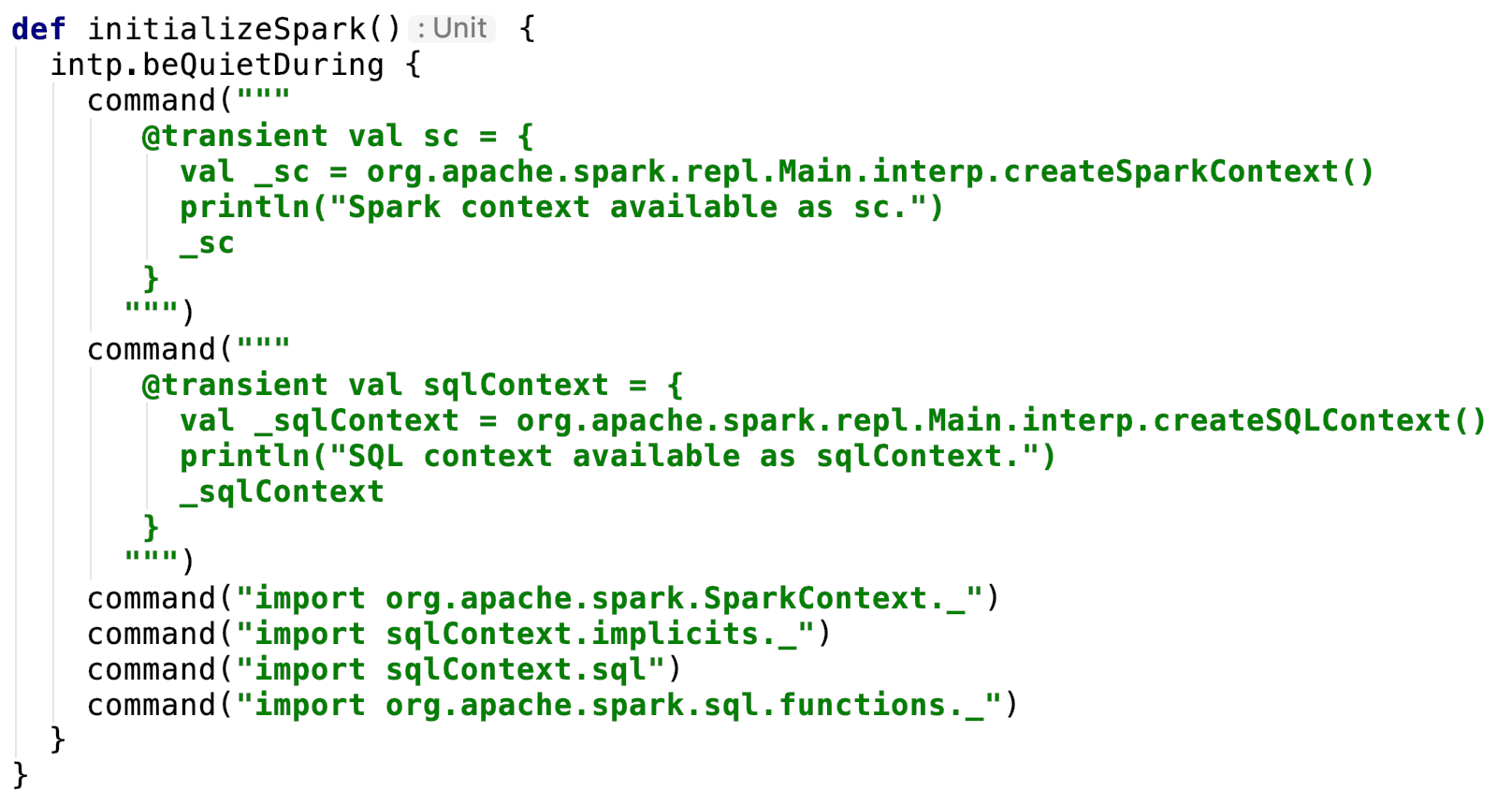
****

**SparkILoop.scala**

****

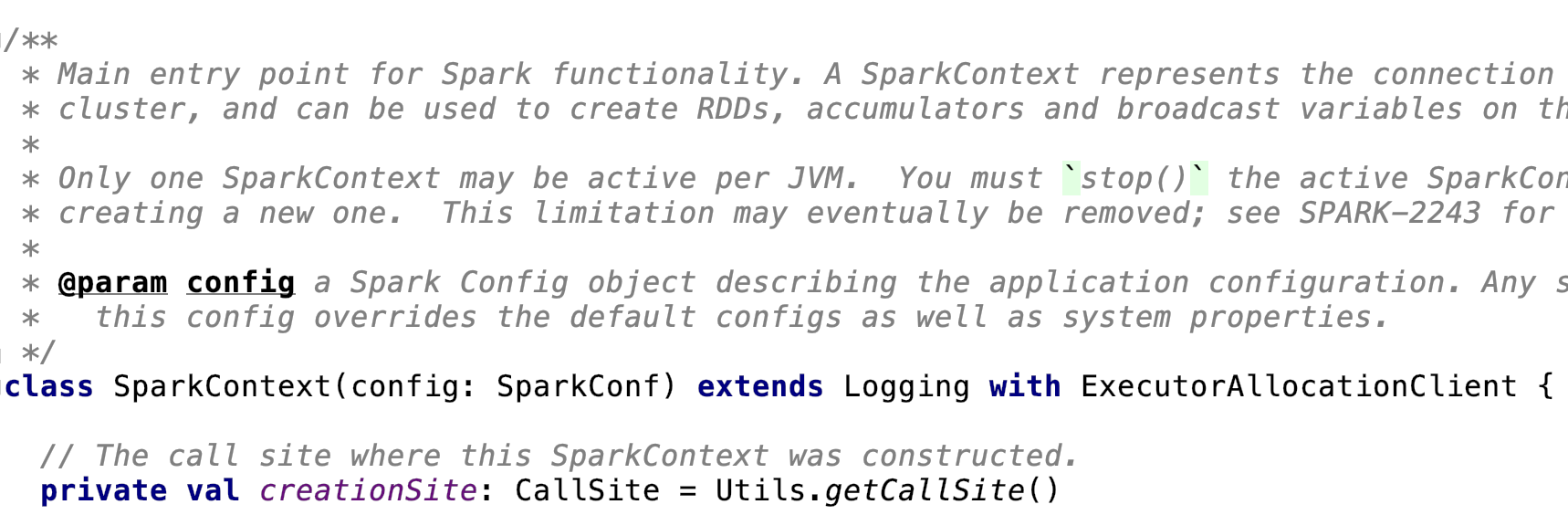
**SparkILoopInit**

****

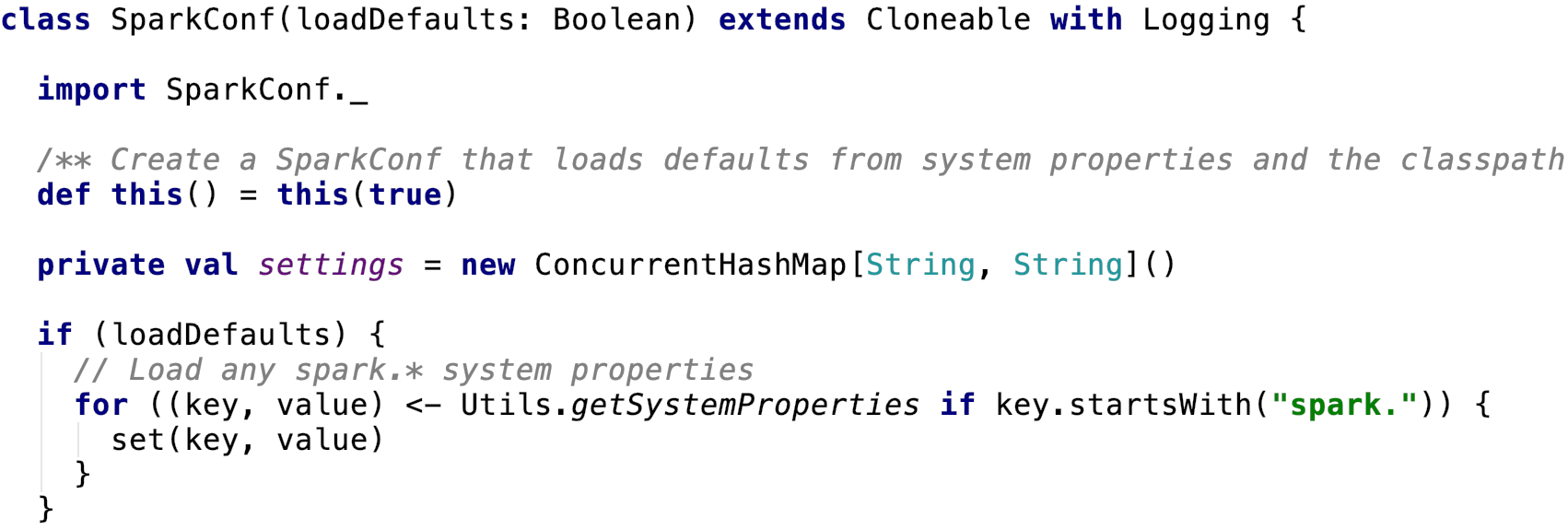
****

* SparkContext初始化之TaskScheduler

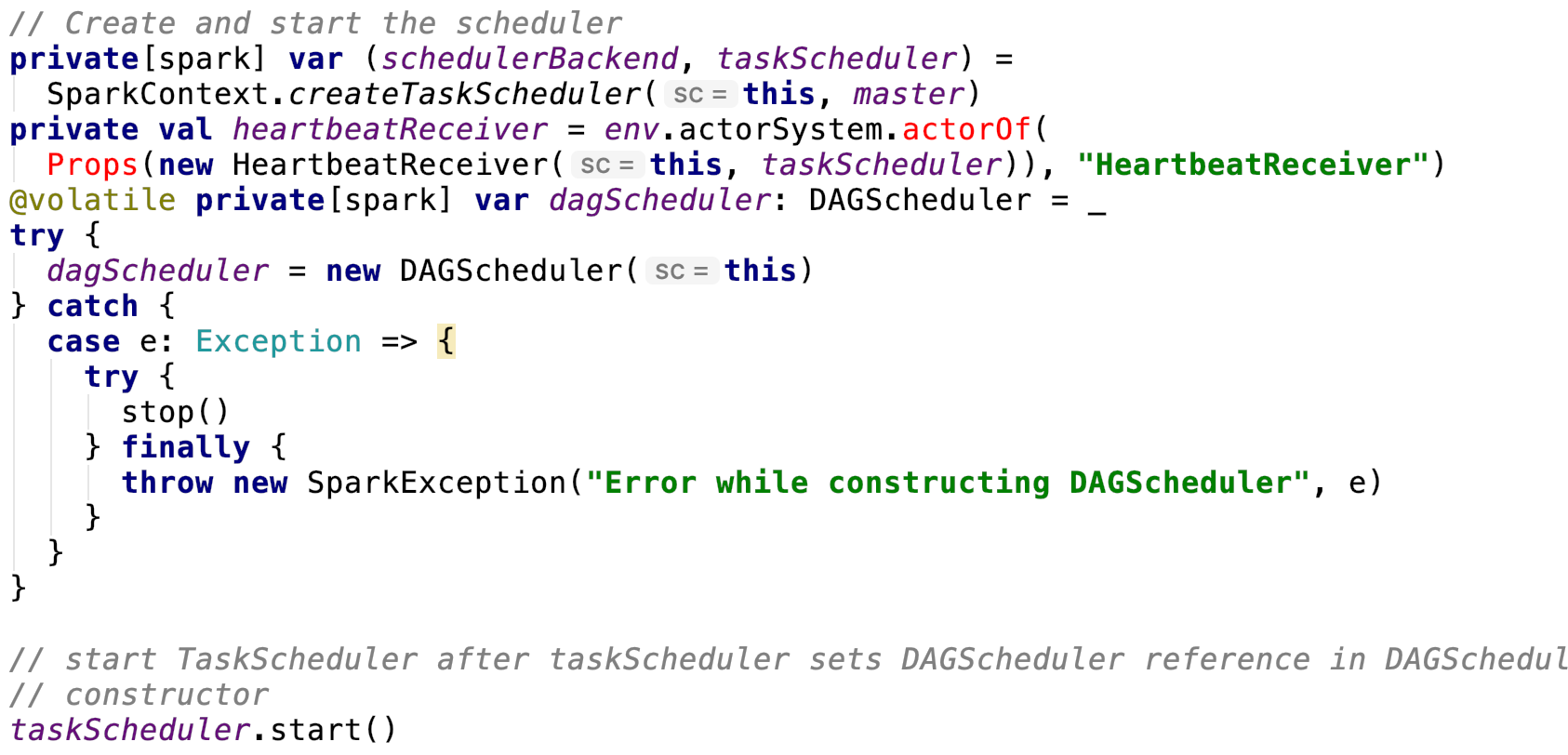
**SparkContext.scala**

****

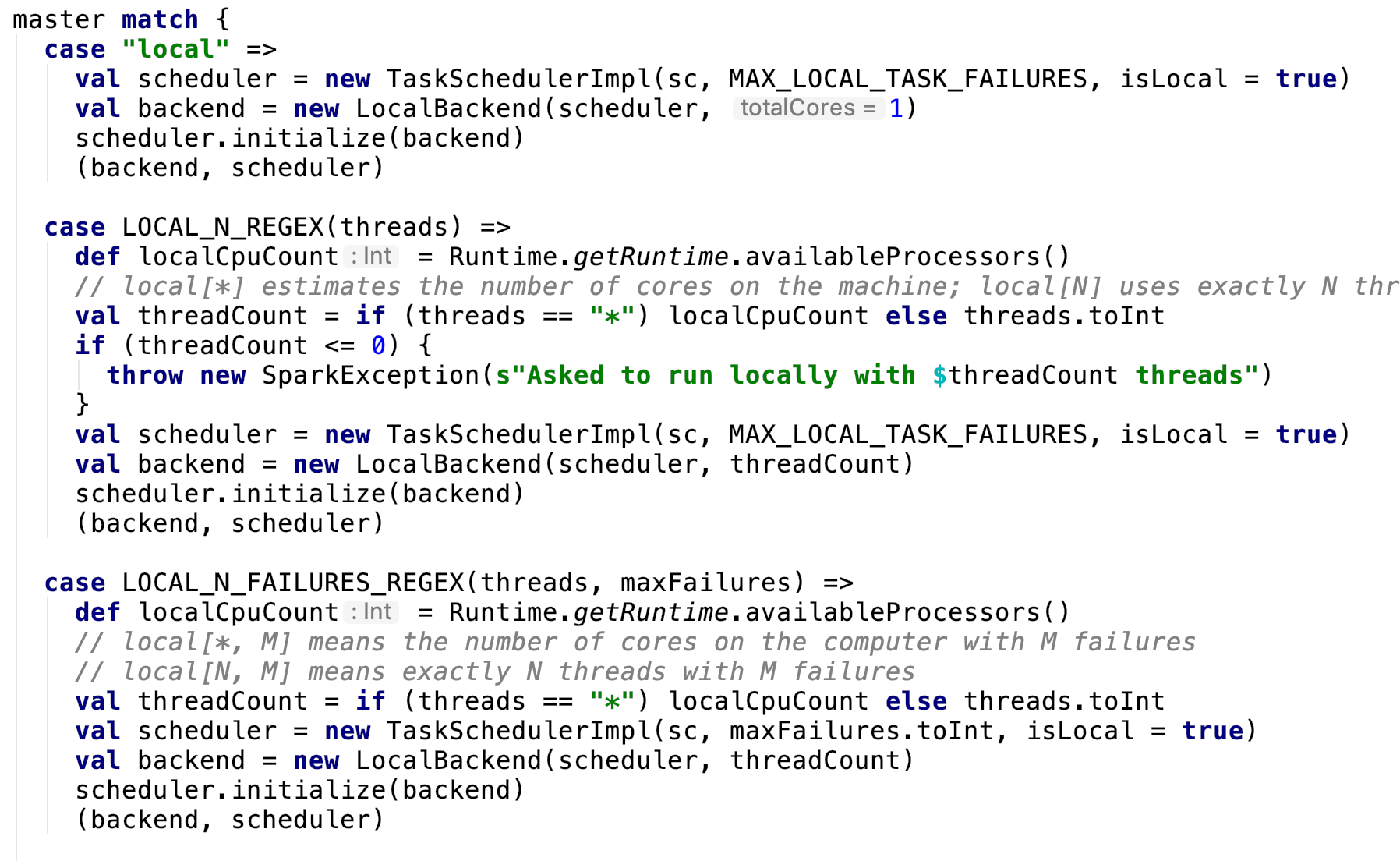
**SparkConf.scala**

****

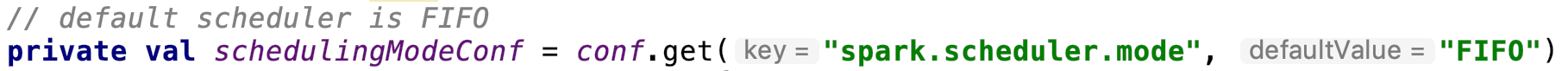
**createTaskScheduler()**

****

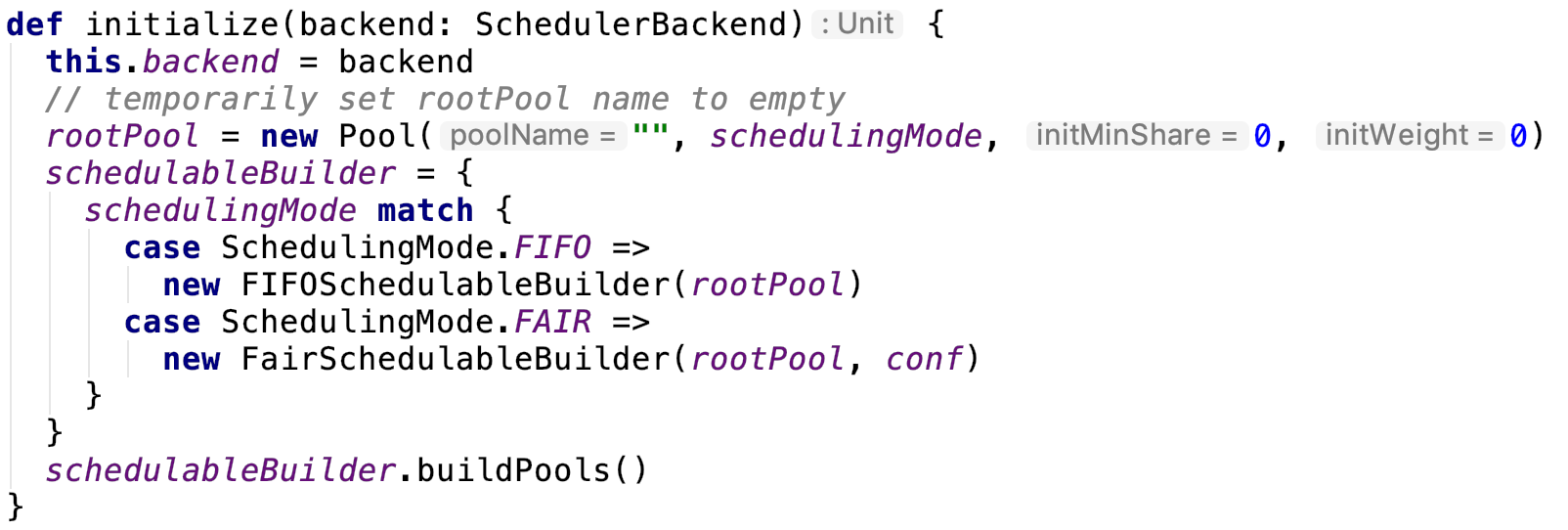
**根据不同的部署模式创建不同的TaskSchedulerImpl和SchedulerBackend**

****

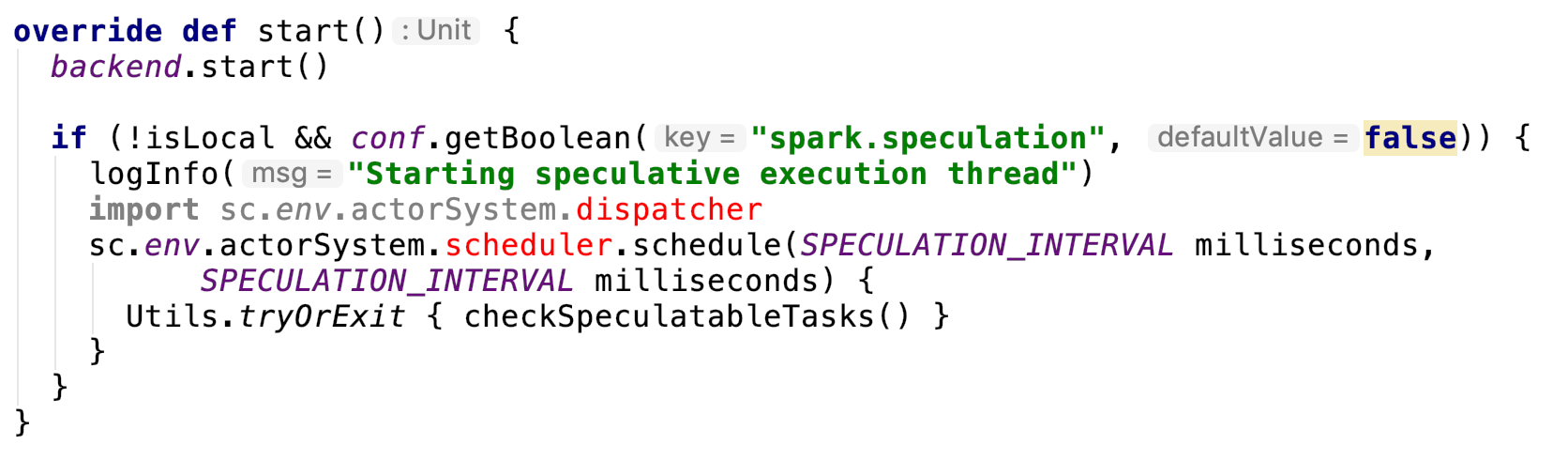
**default scheduler mode**

****

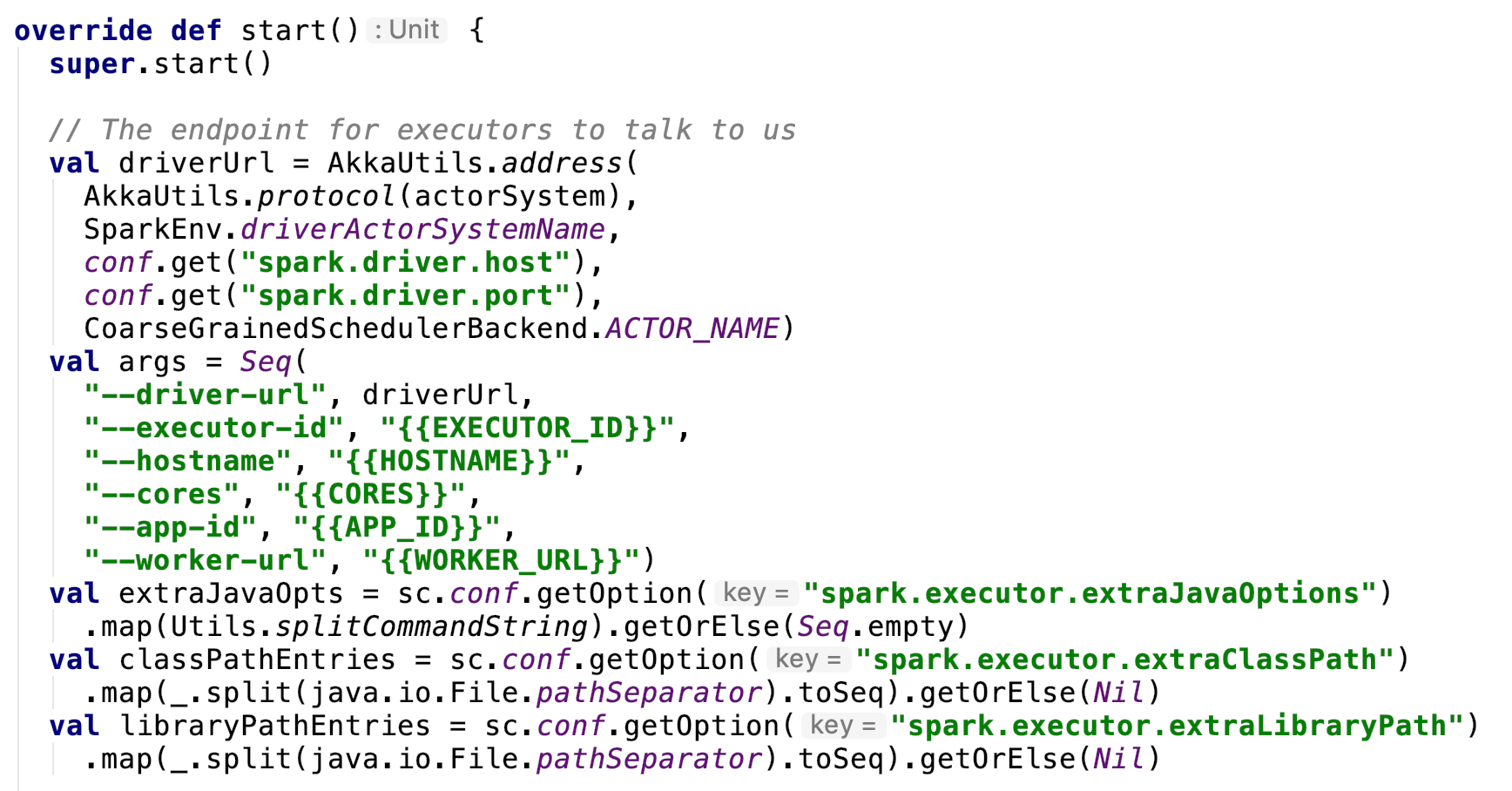
**task scheduler initialize**

****

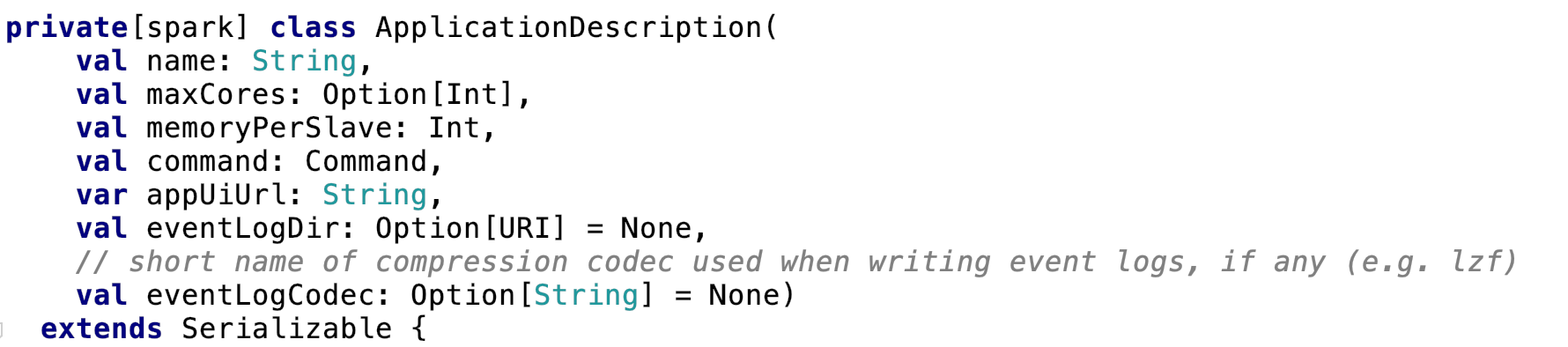
**task scheduler start**

****

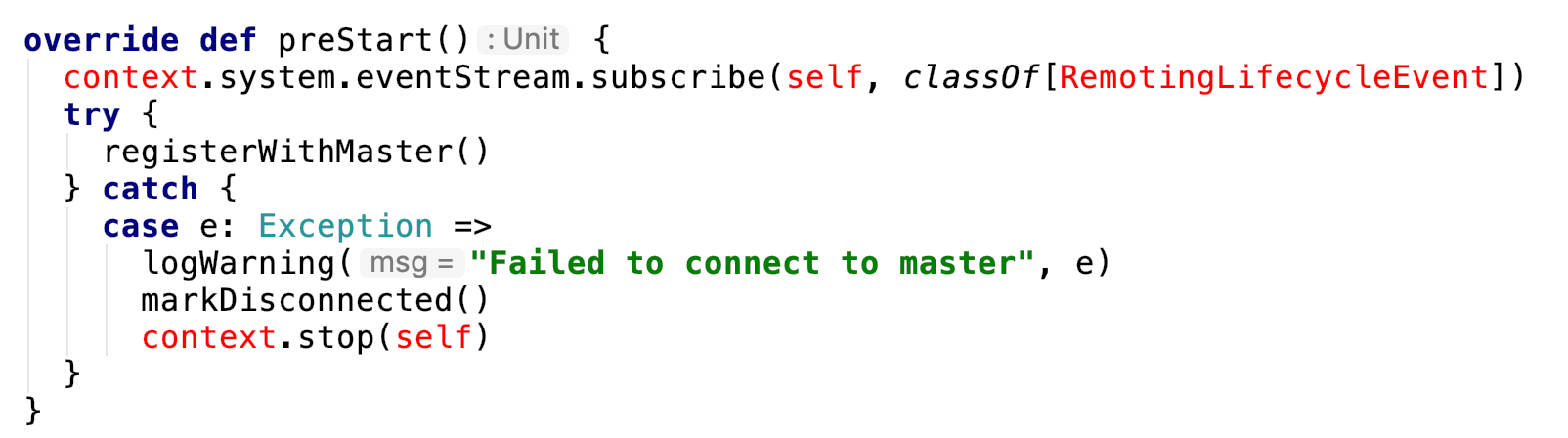
**backend start**

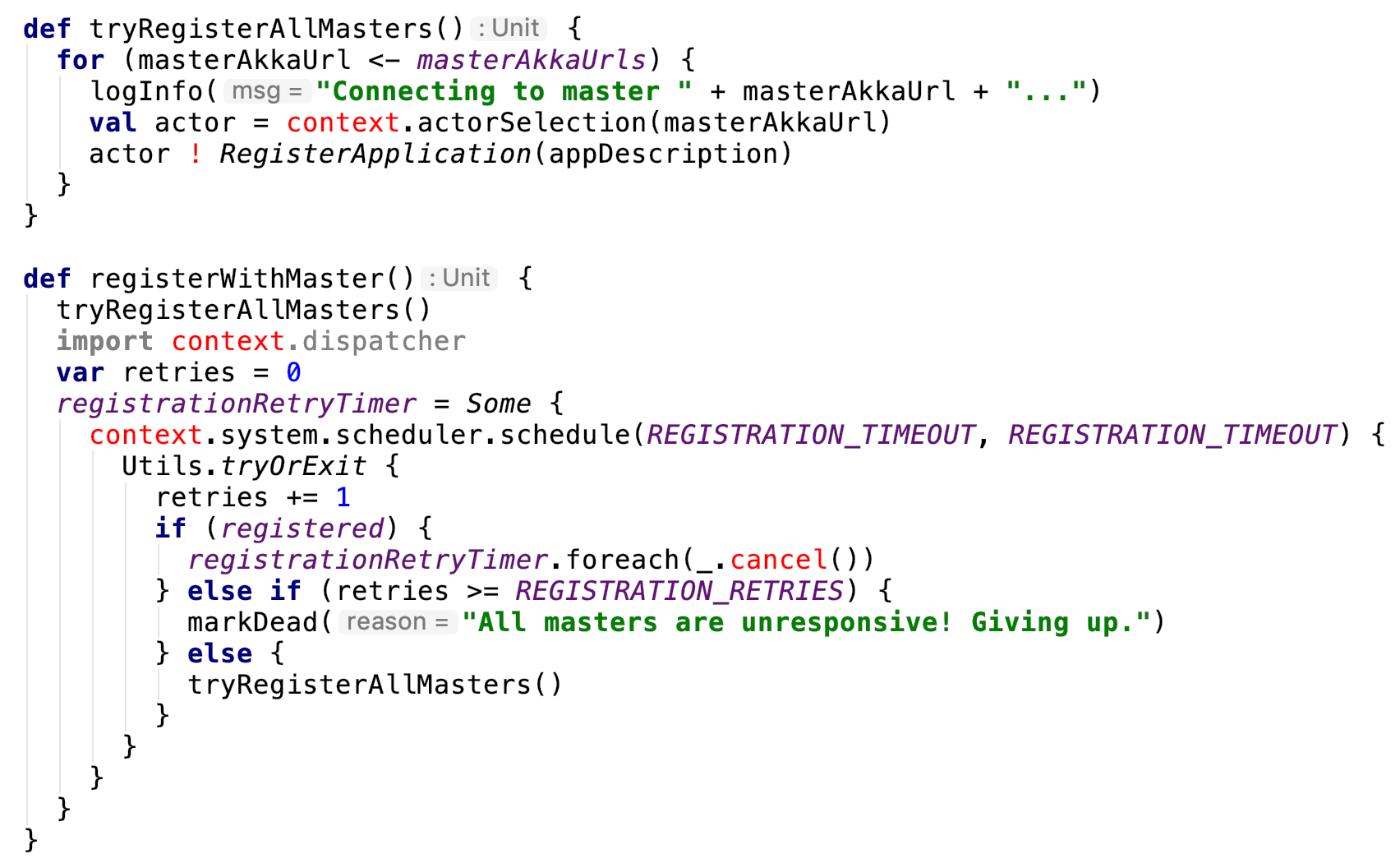
****

**ApplicationDescription**

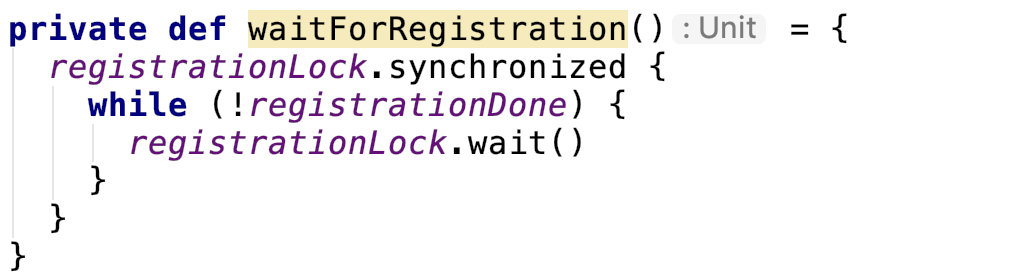
****

**AppClient**

****

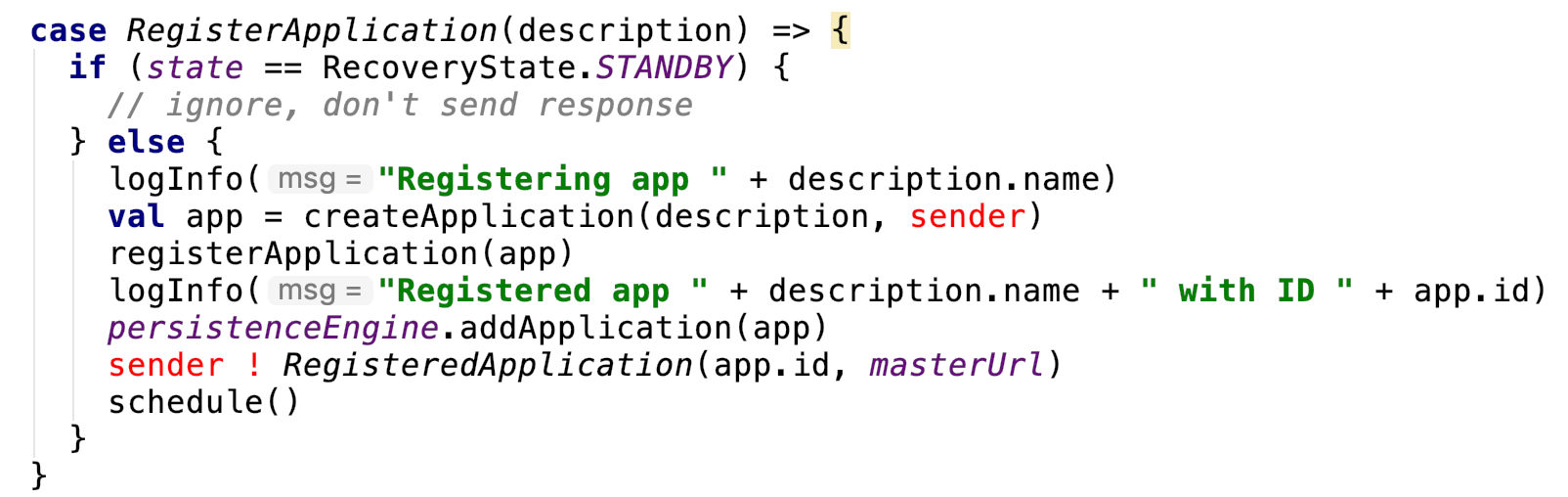
****

**waitForRegistration**

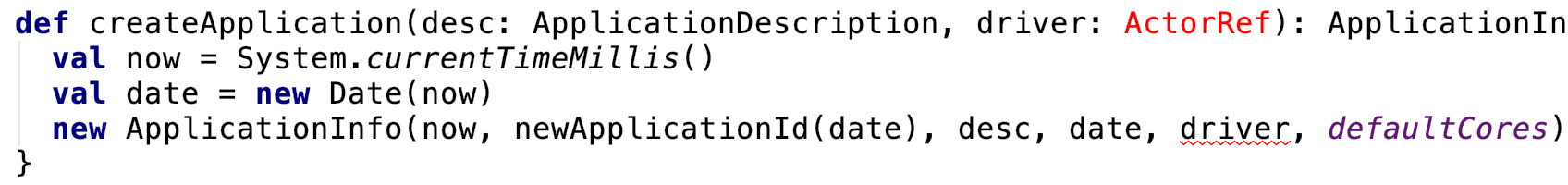
****

* SparkContext初始化之application注册

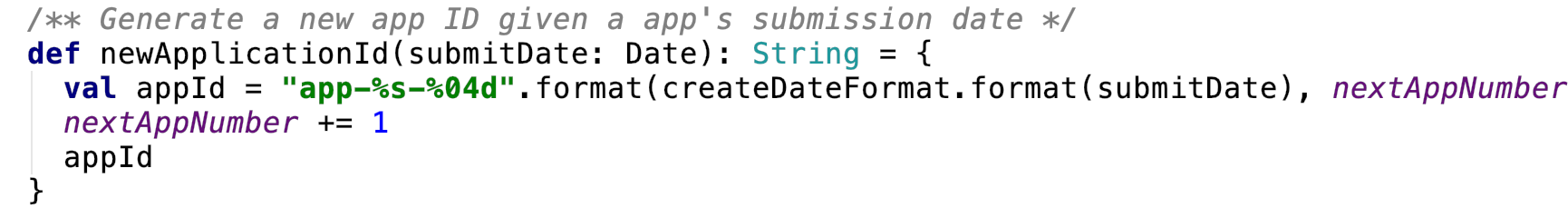
**Master.scala**

****

**createApplication**

****

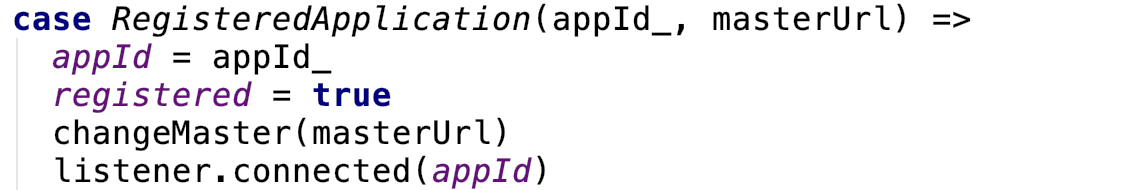
**Generate a new app ID given a app's submission date**

****

**registerApplication**

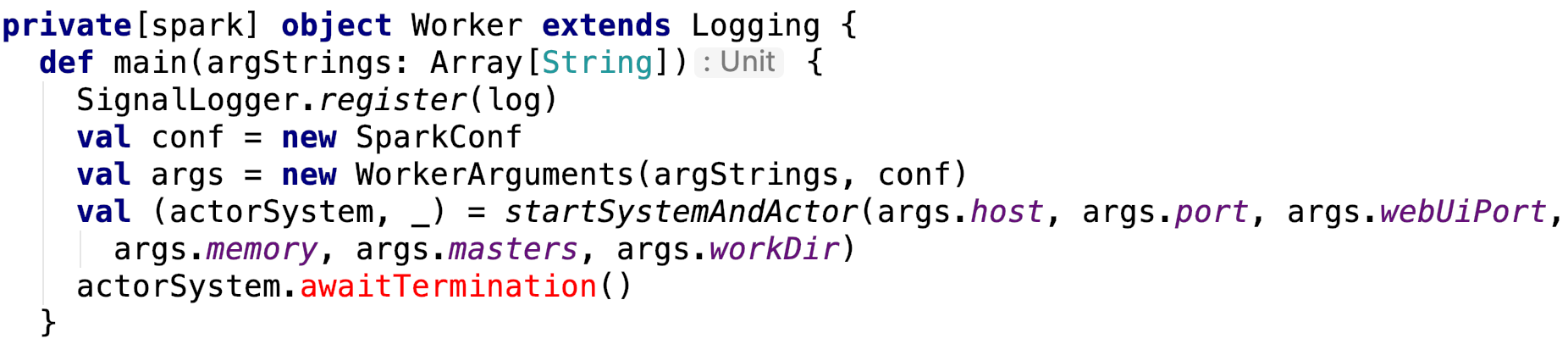
****

**RegisteredApplication**

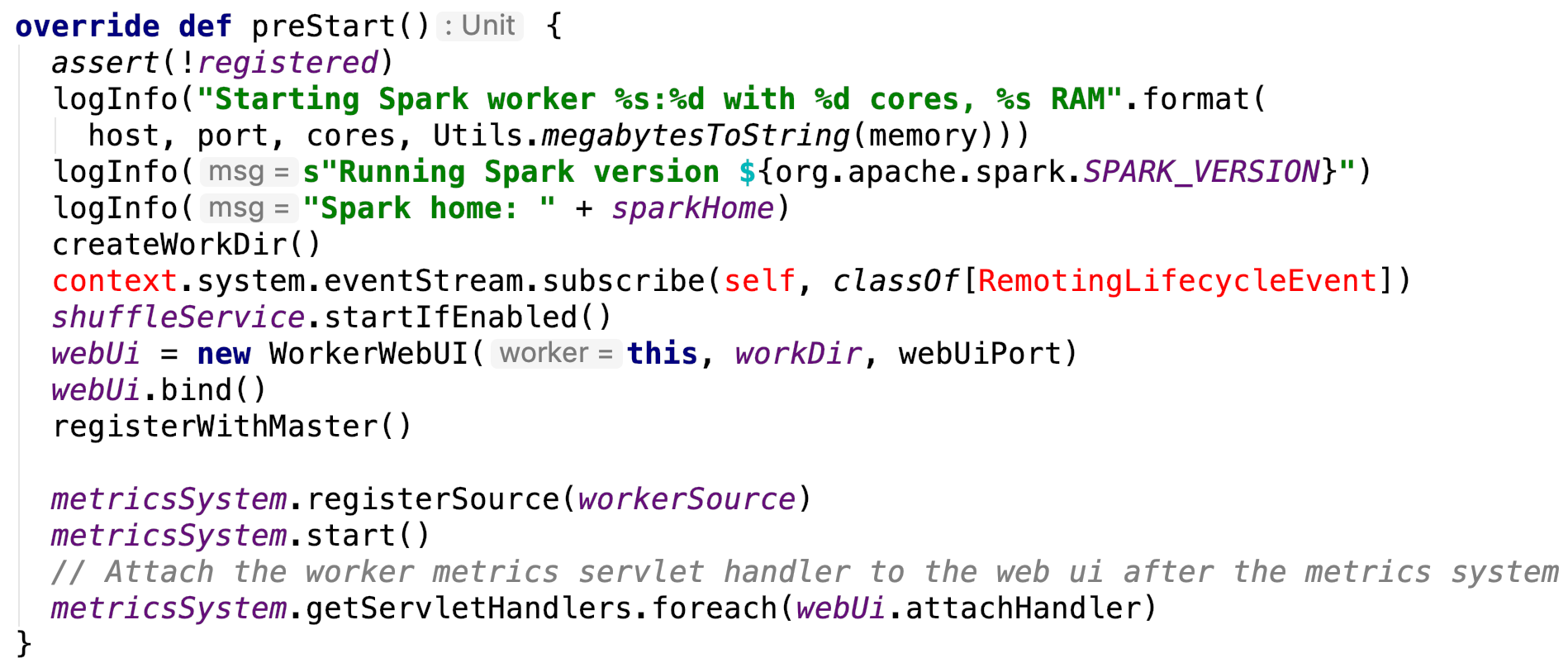
****

* Worker注册

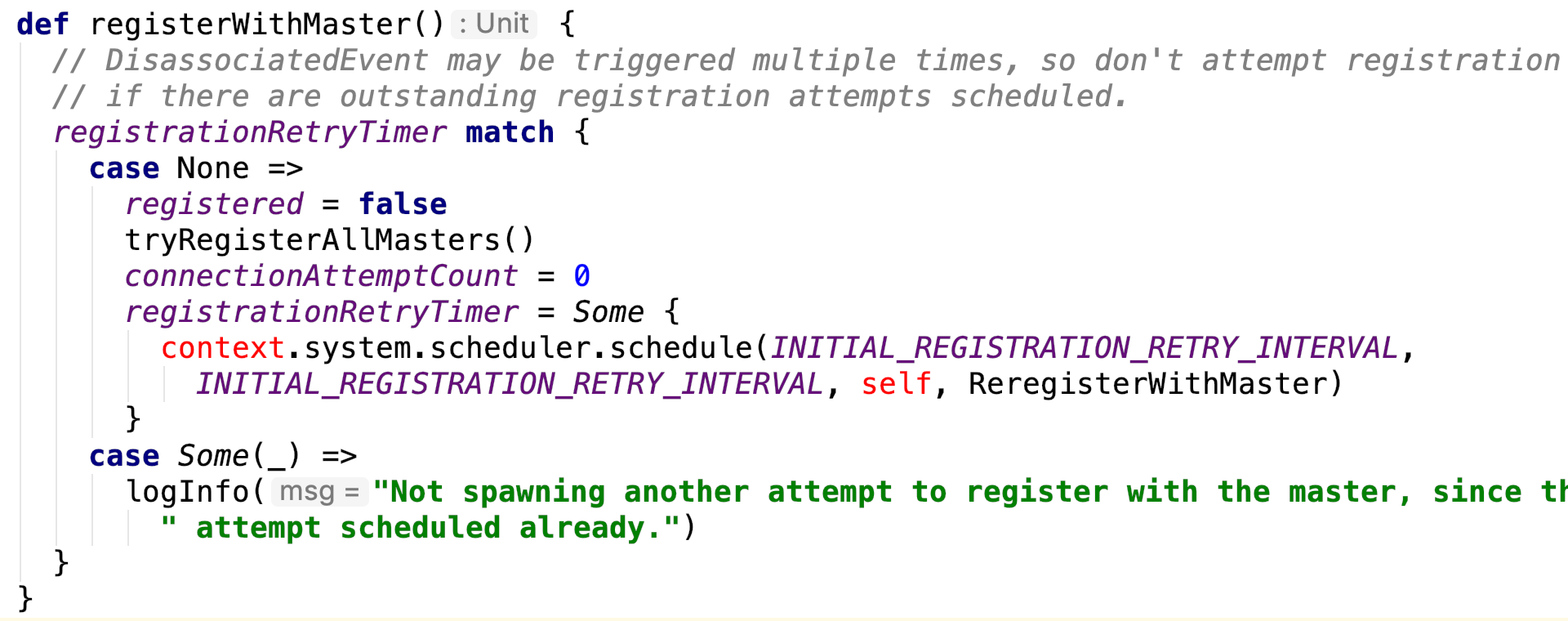
**Worker.scala**

****

**registerWithMaster**

****

**tryRegisterAllMasters**

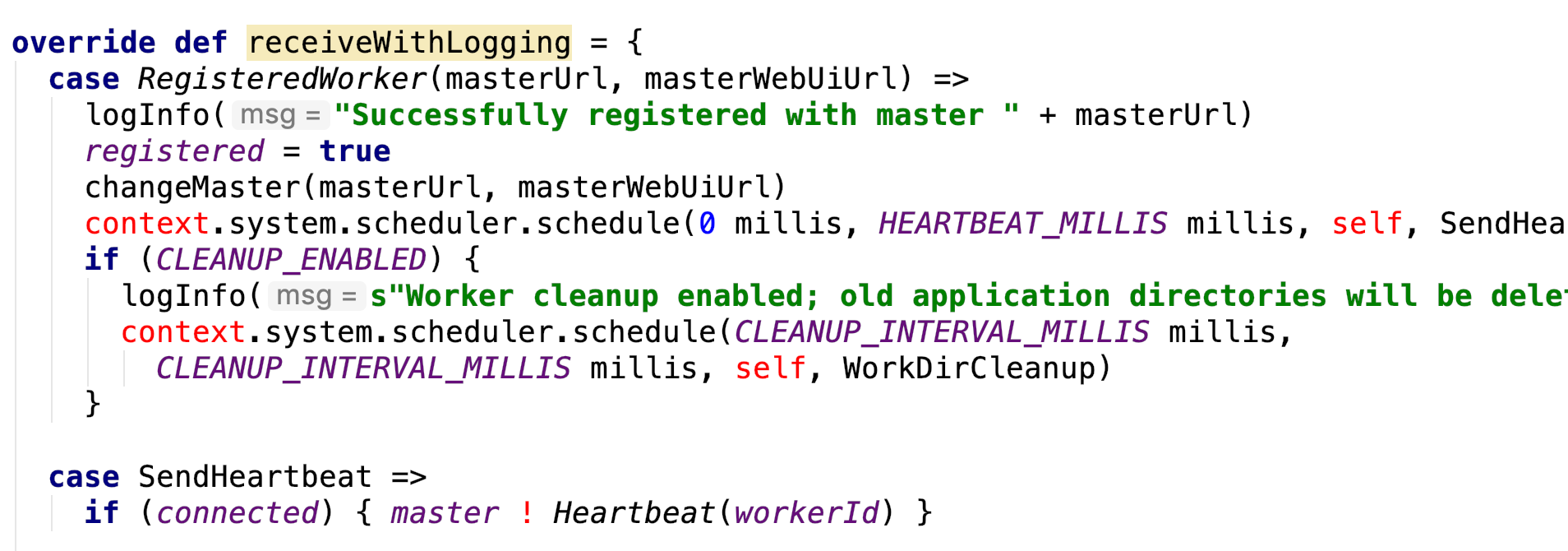
****

****

**Master.scala**

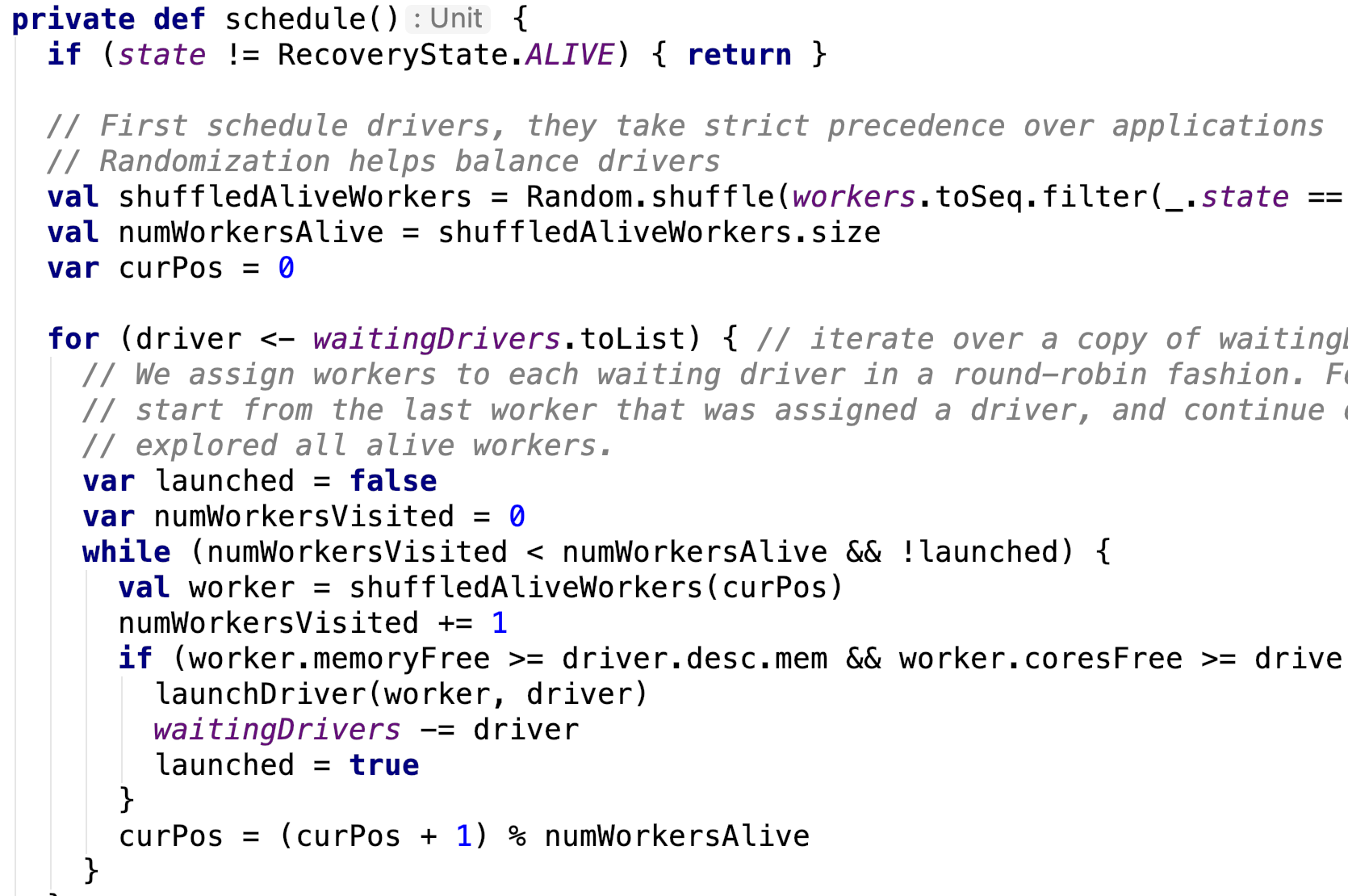
****

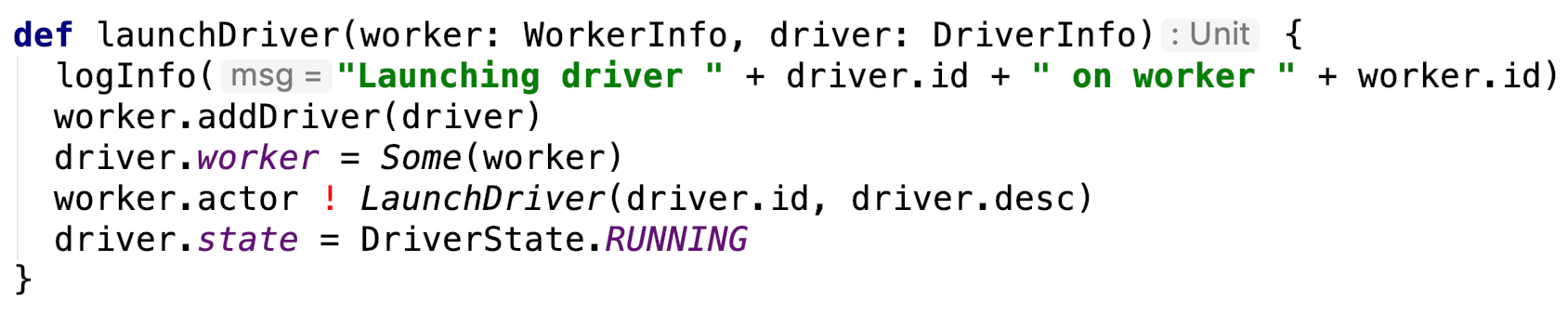
**Worker.scala**

****

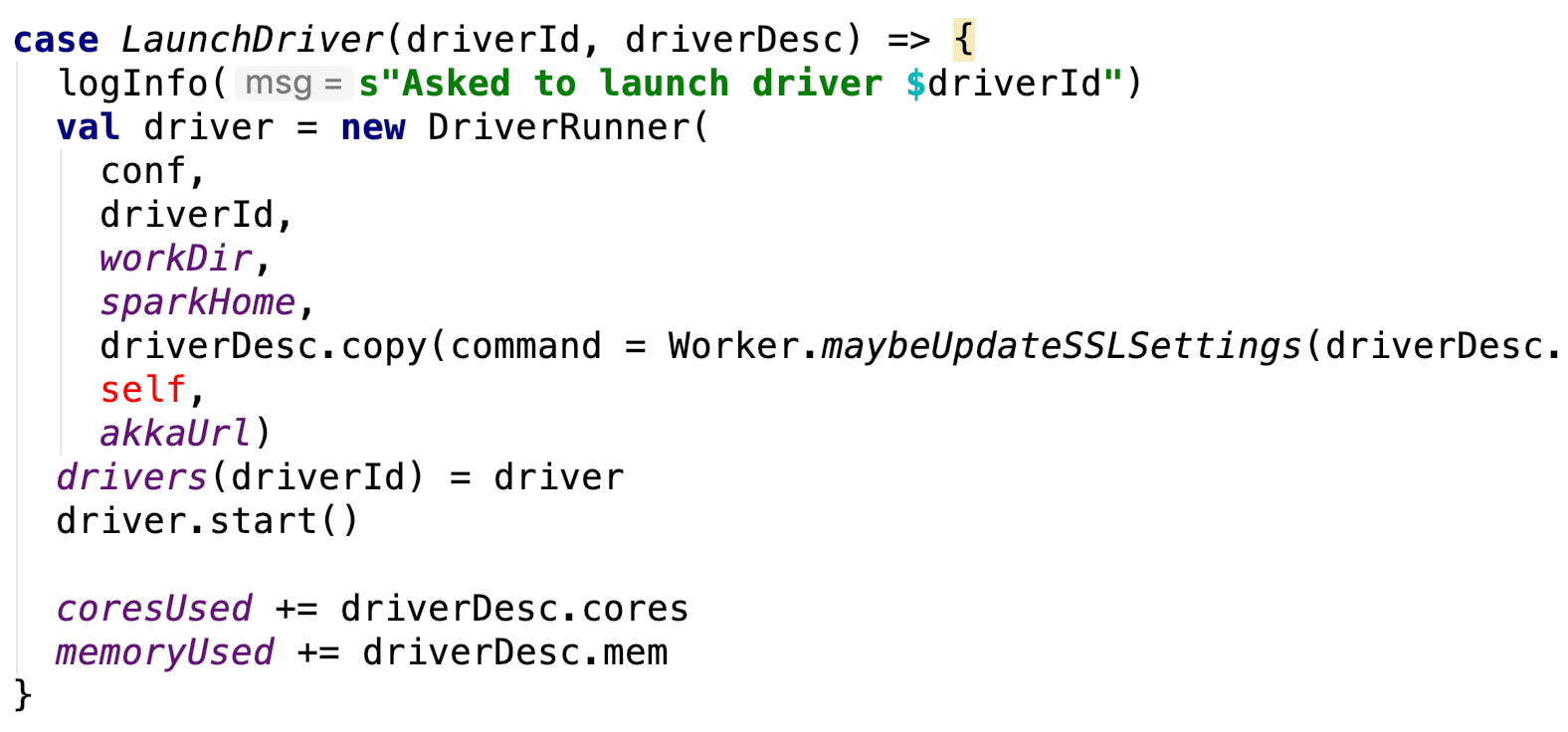
* SparkContext初始化之资源调度算法

**Master.scala**

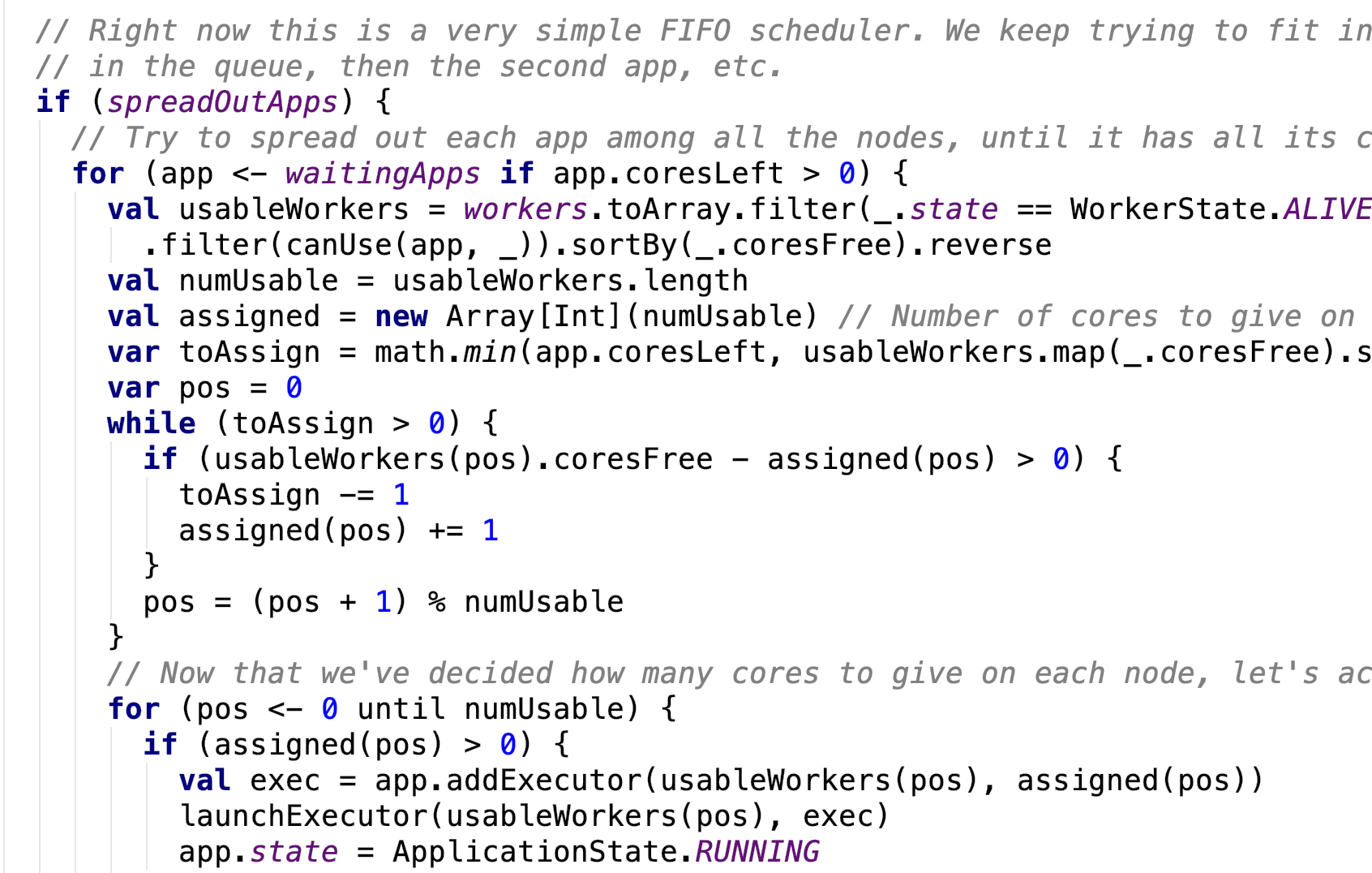


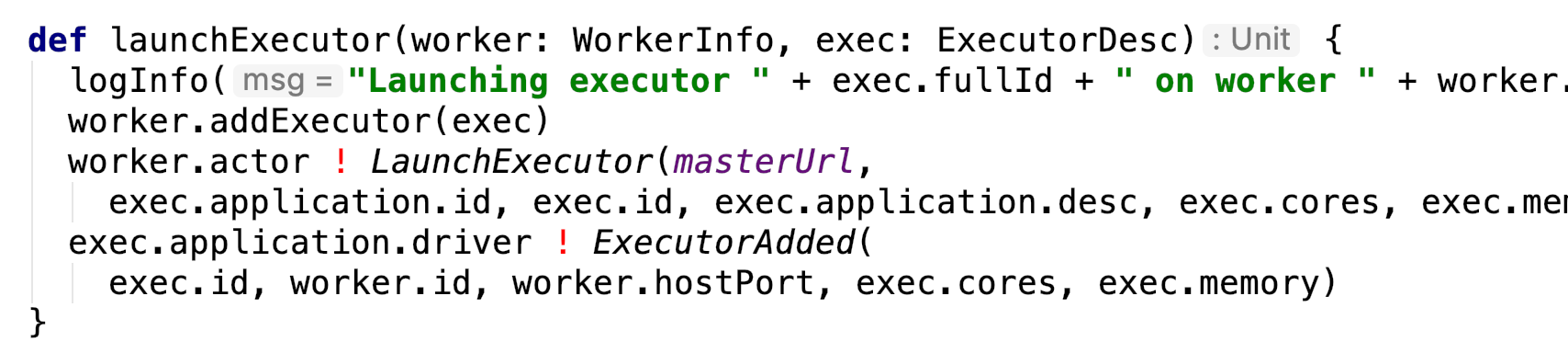


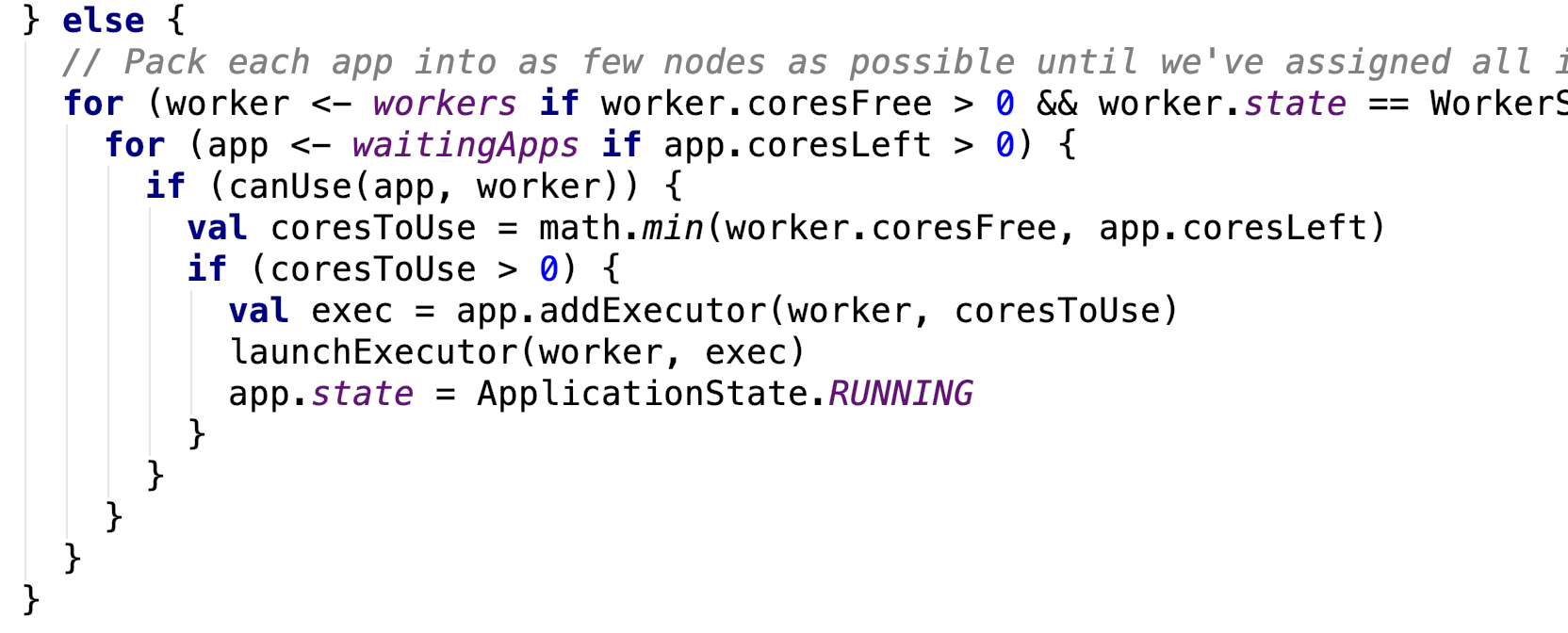
**Worker.scala**

****

**Master.scala**

****

****

****