官网的说明

```
Inside a given Spark application (SparkContext instance), multiple
parallel jobs can run simultaneously if they were submitted from
separate threads. By “job”, in this section, we mean a Spark action
(e.g. save, collect) and any tasks that need to run to evaluate that
action. Spark’s scheduler is fully thread-safe and supports this use
case to enable applications that serve multiple requests (e.g. queries
for multiple users).
By default, Spark’s scheduler runs jobs in FIFO fashion. Each job is
divided into “stages” (e.g. map and reduce phases), and the first job
gets priority on all available resources while its stages have tasks
to launch, then the second job gets priority, etc. If the jobs at the
head of the queue don’t need to use the whole cluster, later jobs can
start to run right away, but if the jobs at the head of the queue are
large, then later jobs may be delayed significantly.
Starting in Spark 0.8, it is also possible to configure fair sharing
between jobs. Under fair sharing, Spark assigns tasks between jobs in
a “round robin” fashion, so that all jobs get a roughly equal share
of cluster resources. This means that short jobs submitted while a
long job is running can start receiving resources right away and
still get good response times, without waiting for the long job to
finish. This mode is best for multi-user settings.
```

在一个spark application(sc 实例)中，多个并行job可以同时运行，如果被一个单独的线程提交的话。

默认情况下，spark的调度器以FIFO运行job，每一个job被切
为“stage”(ShuffleMapStage和ReduceStage)，并且当第一个job的stage有task启动时，优先获取所有可用资源，然后第二个job再优先获取，等等。如果第一个job不需要获取整个集群资源，第二个job可以立即运行，但是如果第一个job是很长的，后续的job有非常明显的延迟。

从spark0.8开始，可以在把job配置为FAIR，在FAIR模式下，spark以“轮询”的方式分配
job之间的task，以便所有的job获取大致相等的集群资源，这意味着长job开始接收资源的时，
提交的短job可以立即执行并获得良好的相应时间，无需等待长job完成。

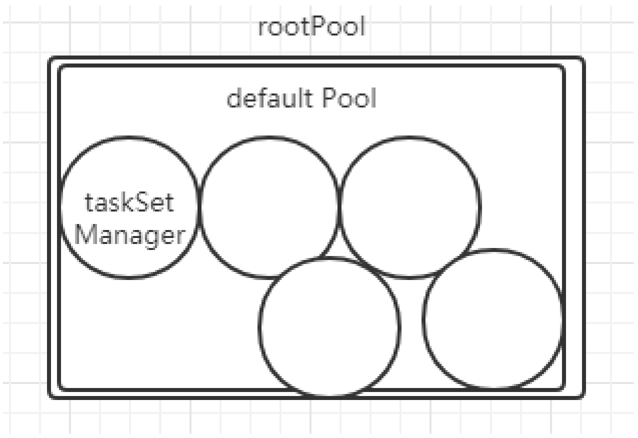
如果启用公平调度的话，只需要将spark.scheduler.mode设为FAIR

```
val conf = new SparkConf().setMaster(...).setAppName(...)
conf.set("spark.scheduler.mode", "FAIR")
```

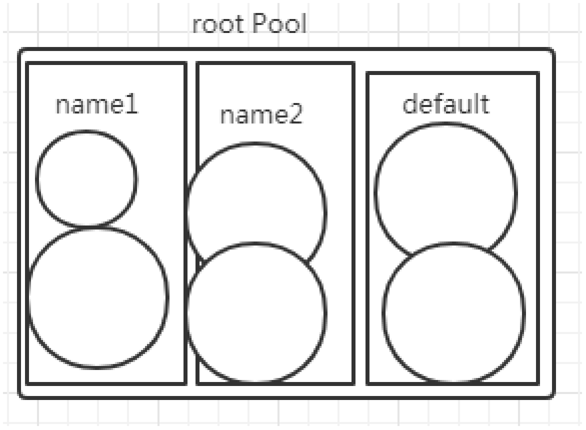
job提交过程中的“资源池pool”

spark job提交的时候，如果未配置fairScheduler.xml文件(可以去文档末尾查看)，代码未配置spark.scheduler.pool(资源池名)，job会提交的rootPool下的默认资源池，如果配置了，会根据配置参数在rootPool在new Pool，假设我们的application触发了3个job，1个未配置，则构建的调度树为rootPool下有一个默认资源池，两个自定义有name的资源池，当每个job内划分完stage，提交taskSet时，会new taskSetManager管理这组task，并将taskSetManager添加到各自的资源池下，下面用图示来表示该流程。

假设job都未自定义，则会在root Pool下new 默认Pool，所有job的所有taskSetManager都
会添加到默认Pool来调度，如下图



假设application触发3个job，2个自定义了name，如下图，未自定义job的taskSetManager添加到默认Pool下，自定义的添加到自己的Pool下



```
//直接从taskScheduler.submitTasks()开始，之前的在别的小模块细说
taskScheduler.submitTasks(new TaskSet(
 ..., properties))

taskSchedulerImpl.submitTasks(taskSet)

manager = createTaskSetManager(taskSet, maxTaskFailures)

```

```
schedulableBuilder.addTaskSetManager(manager,
 manager.taskSet.properties)

case FIFO
 rootPool.addSchedulable(manager)

case FAIR
 会先判断用户是否自定义了资源池，如果未自定义，则new默认资源池，否则根据
配置new资源池

```

文字描述：将task打包成taskSet提交时，会给当前taskSet创建一个taskSetManager(持续跟踪每个task，在失败次数内，如果task失败则重新提交，并且通过延迟调度尽量保证数据本地性),然后schedulableBuilder将此taskSetManager添加到调度树里，如果是FIFO，则直接添加
该taskSetManager到根池下。

FAIR是重点，下面是详细的match到FAIR的源码

```
override def addTaskSetManager(manager: Schedulable,
 properties: Properties) {
 val poolName = if (properties != null) {//判断用户是否自定义资源池
 //获取配置的资源池名，如果获取不到，取默认值"default"
 properties.getProperty(FAIR_SCHEDULER_PROPERTIES,
 DEFAULT_POOL_NAME)
 } else {
 DEFAULT_POOL_NAME
 }
  //根据资源池名获取资源池
  var parentPool = rootPool.getSchedulableByName(poolName)
  if (parentPool == null) {
  //如果为空，则new 资源池
  parentPool = new Pool(poolName, DEFAULT_SCHEDULING_MODE,
  DEFAULT_MINIMUM_SHARE, DEFAULT_WEIGHT)
  //添加到根池下
  rootPool.addSchedulable(parentPool)
  }
  //不为空，添加taskSetManager到该资源池
  parentPool.addSchedulable(manager)
}
```

资源池涉及的源码

#trait Schedulable

可调度的实体，有资源池pool和taskSetManager

```
var parent: Pool//父资源池

// child queues 资源池下面添加的子队列
//ConcurrentLinkedQueue并发编程中，保证线程安全的
def schedulableQueue: ConcurrentLinkedQueue[Schedulable]

def schedulingMode: SchedulingMode//调度模式
def weight: Int//权重
def minShare: Int//需要最小的core数
def runningTasks: Int//需要运行的task数
def priority: Int//job ID
def stageId: Int//stage ID
def name: String//资源池的名字

//添加可调度的实体
def addSchedulable(schedulable: Schedulable): Unit
//移除可调度的实体
def removeSchedulable(schedulable: Schedulable): Unit
//通过资源池name获取资源池
def getSchedulableByName(name: String): Schedulable
```

#class FIFOSchedulingAlgorithm

//FIFO调度策略

```
override def comparator(s1: Schedulable, s2: Schedulable): Boolean = {
 val priority1 = s1.priority//获取jobID，越早start的job jobID越小
 val priority2 = s2.priority

 //signum 大于0 返回1 小于0 返回-1 等于0 返回 0
 var res = math.signum(priority1 - priority2)
 if (res == 0) {//如果jobID相同，同一job，比较stageID
 val stageId1 = s1.stageId//获取stageID
 val stageId2 = s2.stageId
  res = math.signum(stageId1 - stageId2)
  }
  res < 0 //再比较res是否小于0
 }
```

#class FairSchedulingAlgorithm

//公平调度策略

```
override def comparator(s1: Schedulable, s2: Schedulable): Boolean = {
```

```
val minShare1 = s1.minShare//调度池要求最少的cpu core数
val minShare2 = s2.minShare
val runningTasks1 = s1.runningTasks//需要运行的task数
val runningTasks2 = s2.runningTasks

//是否需要运行的task数小于要求cpu core数
val s1Needy = runningTasks1 < minShare1
val s2Needy = runningTasks2 < minShare2

 //求task数/cpu数，如果需要task数越少，cpu越大，值越小
 val minShareRatio1 = runningTasks1.toDouble
 / math.max(minShare1, 1.0)
 val minShareRatio2 = runningTasks2.toDouble
 / math.max(minShare2, 1.0)

 //task数/权重 权重越高，值越小
 val taskToWeightRatio1 = runningTasks1.toDouble
 / s1.weight.toDouble
 val taskToWeightRatio2 = runningTasks2.toDouble
 / s2.weight.toDouble

 var compare = 0
 if (s1Needy && !s2Needy) {//如果s1 task数小于cpu数，先运行s1
 return true
 } else if (!s1Needy && s2Needy) {
 return false
 } else if (s1Needy && s2Needy) {//如果都小于
 //比较比值，小的先运行
 compare = minShareRatio1.compareTo(minShareRatio2)
 } else {
 //还相等，比较给的权重
 compare = taskToWeightRatio1.compareTo(taskToWeightRatio2)
 }
 if (compare < 0) {
 true
 } else if (compare > 0) {
 false
 } else {
 s1.name < s2.name//上述如果都相等，比较给定的资源池name
 }
```

```
```

#class Pool

//可调度的资源池

```
//调度队列
val schedulableQueue = new ConcurrentLinkedQueue[Schedulable]
#该类的主要方法有:
1.根据传入的schedulingMode来匹配是FAIR或FIFO
2.添加/移除可调度的实体, 如果是根池, 可以添加资源池,
 如果不是根池, 只能添加tasksetmanager
3.根据传入的task数, 增加或减少要运行的task数
```

#trait SchedulableBuilder

//构建调度树的接口

```
def rootPool: Pool//返回根资源池
def buildPools(): Unit//构建树节点
def addTaskSetManager(manager: Schedulable, properties: Properties)
  : Unit//构建叶节点
```

#class FIFOSchedulableBuilder

//构建FIFO调度树

```
override def addTaskSetManager(manager: Schedulable,
 properties: Properties) {
 rootPool.addSchedulable(manager)//在根池子里添加tasksetmanager
}
```

#class FairSchedulableBuilder

//构建FAIR调度树

```
#三个主要的方法
def buildPools()
如果用户创建了fairScheduler.xml, 则取读取内容, 根据配置创建资源池, 如果配置
有误, 则创建默认资源池

def buildDefaultPool()
先判断是否有name为default的资源池(大池子/根池子), 没有则创建默认资源池,
有的话将tasksetmanager添加到下面

def buildFairSchedulerPool()
构建公平调度的池子, 加载fairScheduler.xml配置文件, 遍历结构, 用
获取到的参数new 资源池,

rootPool.addSchedulable(new Pool(poolName, schedulingMode,
 minShare, weight))
```

fairScheduler.xml的配置如下(官网案例)

可以配置资源池名，调度模式，权重，需要的cpu core数，可以在job触发前设置spark.scheduler.pool，spark会根据设置的name去xml文件找对应配置参数

```
 <?xml version="1.0"?>
<allocations>
 <pool name="production">
 <schedulingMode>FAIR</schedulingMode>
 <weight>1</weight>
 <minShare>2</minShare>
 </pool>
 <pool name="test">
 <schedulingMode>FIFO</schedulingMode>
 <weight>2</weight>
 <minShare>3</minShare>
 </pool>
</allocations>
```