# Fork/Join框架学习笔记

## 前言

熟悉或了解过Hadoop的同学理解起来会比较容易，思路跟MapReduce是一样的，将一个大的任务拆分成合适粒度的小任务——fork，然后再将结果合并起来（如果需要返回结果的话）——join。

举一个经典的例子，从1加到10000。单线程的处理方式肯定是一个循环，不停地累加，最后返回结果。

如果数小还没问题，如果要处理的数据比较大，内存装不下怎么办？那么大的数，只用一个for循环，一个线程，一个CPU去处理，没有把硬件资源全发挥出来，是不是很浪费？

多线程就是比较好的选择，而经过大牛们不懈地努力，Fork/Join就是一个比较合适的框架。

大家先记住下面这段伪代码：

***if(待处理的数据数量<阈值){***

***//处理数据***

***}else{***

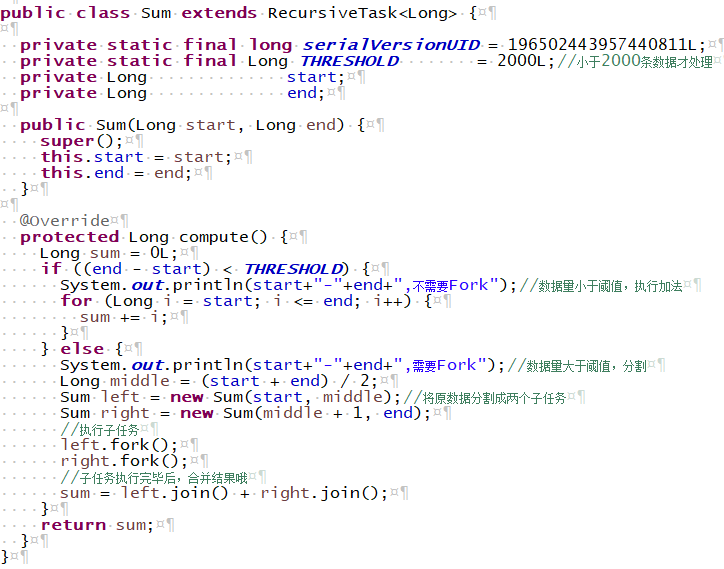
***//拆分数据，构建子任务***

***//执行子任务***

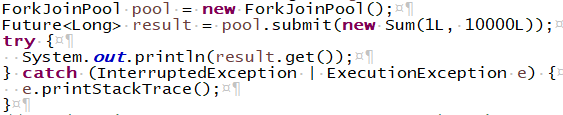
***}***

## 例1：数相加

这是一个经典的例子，要求与逻辑很简单，直接上代码不解释。



测试类：



结果：

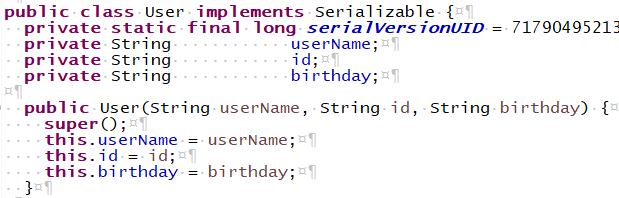


## 例2：不需要返回值

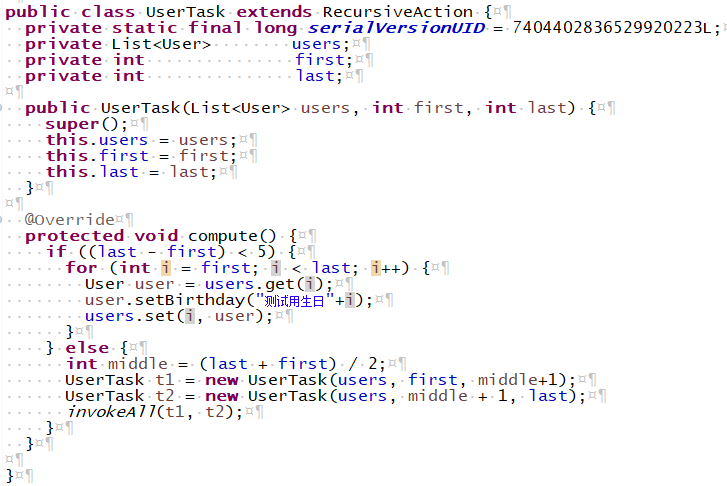
在上面的例子中，Sum这个任务类继承了RecursiveTask，用于有返回结果的任务。而没有返回结果的任务，则应该使用RecursiveAction。

比如我们有N个人员信息的列表，只有姓名和身份证，出生年月日为空。我们想根据身份证上的信息，给每个人补上生日。

用户类：

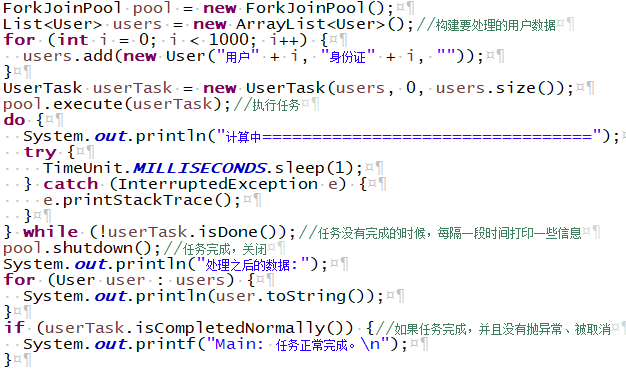


任务类：

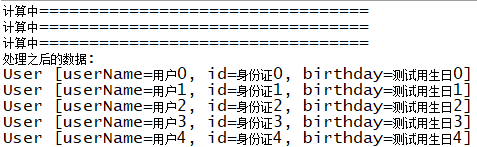


为了简化，处理生日的代码直接写死了。

测试类：



结果：



剩下的就不截图了。

## 一些知识点

### 原理

举个例子，1个学校有10个老师，有1000份卷子需要批改，有几种做法？

1. 指定一个老师批改，这是单线程。



1. 每个老师批改100份卷子，这是多线程。这种情况很美好，但是请允许我举一个极端的场景，大家思考一下：1000份卷子里面有900份是白卷，10个老师里面有1位老师身体不舒服，看东西看不清楚，脑子蒙蒙的导致批改卷子很耗时间。而这时候，900份白卷分配给了另外9位健康的老师，100份正常的卷子分配给了这位得病的老师。是不是这种情况也很不好，怎么办？
2. 1000份卷子按照编号分配给老师们，A老师负责1-100，B老师负责101-200，以此类推。如果A老师速度快把手里的卷子全批改完了，B老师才批改了200-150卷子，那么A老师把第101份卷子拿过来批改，是不是整体效率更高了。

Fork/Join使用的就是第三种解决方案——Work-Stealing（任务窃取）算法。

每一个工作线程（老师）都会维护自己的工作队列（编号1-100的卷子等等），这个队列是双向的，你可以从前向后拿数据（FIFO），也可以从后向前拿数据（LIFO）。

工作线程处理自己的任务队列遵从LIFO原则，最后入列放在最上面的卷子最先批改。

工作线程空闲下来，从别处“窃取”任务的时候遵从FIFO原则，拿一摞卷子最下面的进行批改，很合情合理吧。

采用双向队列，结合FIFO、LIFO原则，在自己和其他工作线程获取任务的时候，减少了竞争。

结合Fork/Join来看，排在前面的操作多是Fork——对任务进行分解。即一个大任务里面就天然的包含很多子任务，“窃取”时采用FIFO原则，等于一次“窃取”就打包拿到了很多的子任务，够执行一阵子了，不用总是去“窃取”。

### 一些类

先说最重要的两个：

#### ForkJoinPool池类

用来执行任务（ForkJoinTask），生成新的工作线程（ForkJoinWorkerThread）。

负责工作线程间的工作窃取。

线程池的个数一般是CPU核数。

执行任务有三个方法：

invoke：同步执行带返回值的任务。ForkJoinTask.invoke。

submit：异步执行带返回值（Future）的任务。ForkJoinTask.fork(ForkJoinTasks are Futures)。

execute：异步执行不带返回值的任务。ForkJoinTask.fork。

（大家可以看前文的两个例子）

#### ForkJoinTask任务类

ForkJoinTask就是用来执行的任务类，主要是工作是大任务的分割（Fork）以及结果的合并（Join），结合之前的例子可看，这是我们主要的编码区。

不过真正工作中我们是不用这个类的，而是用它的两个子类：

RecursiveAction，没有返回值。

RecursiveTask，有返回值。

ForkJoinTask有几个重要方法：

fork()，异步执行当前Task。

invoke()，同步执行，等待任务完成并返回结果。

invokeAll()，同步执行给的Task，等待所有任务完成并返回结果。

join()，获取返回值。不可中断，否则会抛出InterruptedException。如果运行的任务抛出异常，返回RuntimeException。

get()，获取返回值。如果运行的任务抛出异常，返回ExecutionException。

说说invokeAll(task1,task2)这个方法。具体代码不再附上。其步骤是：

1. 执行task2的fork()。fork()代码步骤不再附上，步骤如下：
2. 先通过push ()将任务放入当前线程的工作队列的Top位置。注意之前提到的“工作窃取”，放在Top位置可以优先让本线程处理。

额外再说一句，有的资料上说用pushTask()。我看的是JDK8源码，用push()。

1. 如果队列满了，扩容。
2. 如果队列中的任务比较少（<=1），就执行ForkJoinPool.signalWork()方法，唤醒一个空闲的线程，或新建一个线程去立即执行。

由上可知，如果队列中任务较多，Task就会存在workQueue中，可供其他工作线程进行“工作窃取”。

1. task1调用doInvoke()方法，直接同步执行。先判断任务状态，如果状态为<0（正常、取消、异常等），则说明任务执行完毕，返回状态。否则就执行任务，然后判断是否结束，如果结束就把状态置为“正常”，否则就等待任务完成。
2. task1执行完，task2调用doJoin()方法。join()方法也主要是调用doJoin()，其主要工作就是判断任务的状态，如果状态为<0（正常、取消、异常等），则说明任务已经执行完毕，返回状态。否则就通过tryUnpush()从队尾（队列的Top位置）取任务执行。