目录

[Java并行编程 1](#_Toc514765317)

[Fork/Join 1](#_Toc514765318)

[ForkJoinTask类 1](#_Toc514765319)

[RecursiveAction类和RecursiveTask类是ForkJoinTask的两个子类 1](#_Toc514765320)

[ForkJoinTask类中的两个方法 2](#_Toc514765321)

[Fork（）方法 2](#_Toc514765322)

[Join（）方法 2](#_Toc514765323)

[一种可行的编程模式 2](#_Toc514765324)

# Java并行编程

## Fork/Join

JDK7中引入的，分为ForkJoinTask类和ForkJoinPool类。

### ForkJoinTask类

是一个为执行任务二设计的抽象类，因为任务的大量数据和子任务可以被ForkJoinPool中世纪的更小的线程执行一个任务

Fork分离成若干子任务，并且被多处理器系统并行处理

当所有的处理完毕后通过Join再“合成”最后的结果

Fork/Join架构对程序的并行化是分治思想的一种体现。（也叫做映射规约），思路是将算法要处理的数据空间拆分成较小的独立块：“映射”阶段。一旦块集处理完后，将各个部分结果收集起来形成最终结果：“规约”阶段。

#### RecursiveAction类和RecursiveTask类是ForkJoinTask的两个子类

1. 用来定义一个具体的任务类
2. 用户自己新建的类必须继承自RecursiveAction或者RecursiveTask
3. RecursiveAction用于无返回值的任务
4. RecursiveTask用于有返回值的任务

RecursiveTask类使用的机会多，因为相当一部分分而治之的算法均需要返回数据集的计算值。

1. 然后再重写compute（）方法来具体描述任务的操作。

#### ForkJoinTask类中的两个方法

##### Fork（）方法

1. 允许计划ForkJoinTask异步执行
2. 从现有的ForkJoinTask启动新的ForkJoinTask
3. Fork（）仅在ForkJoinPool中调度一个新任务，不创建子java虚拟机

##### Join（）方法

1. 允许ForkJoinTask等待另一个ForkJoinTask完成
2. 与java多线程编程中的Join（）方法类似

### 一种可行的编程模式

If(这个任务足够小{

执行要做的任务

)else{

将任务分割成若干小部分

执行若干小部分并等待执行结果

}

## 如何使用Fork/Join框架

1. 第0步：如何分割任务？

这个任务是否值得分割？如果值得分割要如何分割？

一个if…else来判断是否值得分割。常用给的分割方法是二等分。

1. 第1步：首先需要从RecursiveTask extends出来一个类A，在这个类中初始化阈值、start、end，其中阈值想设置可以设置，也可以不设置，主要是为了在大于阈值的时候把整个任务分开。接着需要重写构造函数用来传入start和end参数。
2. 第2步：重写上述类中的compute方法，先用第0步里说的if…else判断一下是否要并行。Else里面写并行代码块。实例化两个子任务（类名A 实例1；类名A实例2……）。执行子任务（实例1.fork()；实例2.fork()…...）.用（实例名.join()）把分开的执行结果合并到一起。
3. 第3步：ForkJoinPool生成一个任务。

ForkJoinPool ex = new ForkJoinPool();//生成一个任务

类名A task = new 类名（参数1，参数2…）;//执行一个任务

Future result = ex.submit(task);//结果，返回的是compute return 的结果。