

23 | Future：如何用多线程实现最优的“烧水泡茶”程序？

2019-04-20 王宝令

Java并发编程实战

[进入课程 >](#)



讲述：王宝令

时长 06:52 大小 6.31M




在上一篇文章 [《22 | Executor 与线程池：如何创建正确的线程池？》](#) 中，我们详细介绍了如何创建正确的线程池，那创建完线程池，我们该如何使用呢？在上一篇文章中，我们仅仅介绍了 `ThreadPoolExecutor` 的 `void execute(Runnable command)` 方法，利用这个方法虽然可以提交任务，但是却没有办法获取任务的执行结果（`execute()` 方法没有返回值）。而很多场景下，我们又都是需要获取任务的执行结果的。那 `ThreadPoolExecutor` 是否提供了相关功能呢？必须的，这么重要的功能当然需要提供了。

下面我们就来介绍一下使用 `ThreadPoolExecutor` 的时候，如何获取任务执行结果。


如何获取任务执行结果

Java 通过 `ThreadPoolExecutor` 提供的 3 个 `submit()` 方法和 1 个 `FutureTask` 工具类来支持获得任务执行结果的需求。下面我们先来介绍这 3 个 `submit()` 方法，这 3 个方法的方法签名如下。

 复制代码

```
1 // 提交 Runnable 任务
2 Future<?>
3     submit(Runnable task);
4 // 提交 Callable 任务
5 <T> Future<T>
6     submit(Callable<T> task);
7 // 提交 Runnable 任务及结果引用
8 <T> Future<T>
9     submit(Runnable task, T result);
```


你会发现它们的返回值都是 `Future` 接口，`Future` 接口有 5 个方法，我都列在下面了，它们分别是**取消任务的方法 `cancel()`**、**判断任务是否已取消的方法 `isCancelled()`**、**判断任务是否已结束的方法 `isDone()`**以及**2 个获得任务执行结果的 `get()` 和 `get(timeout, unit)`**，其中最后一个 `get(timeout, unit)` 支持超时机制。通过 `Future` 接口的这 5 个方法你会发现，我们提交的任务不但能够获取任务执行结果，还可以取消任务。不过需要注意的是：这两个 `get()` 方法都是阻塞式的，如果被调用的时候，任务还没有执行完，那么调用 `get()` 方法的线程会阻塞，直到任务执行完才会被唤醒。

 复制代码

```
1 // 取消任务
2 boolean cancel(
3     boolean mayInterruptIfRunning);
4 // 判断任务是否已取消
5 boolean isCancelled();
6 // 判断任务是否已结束
7 boolean isDone();
8 // 获得任务执行结果
9 get();
10 // 获得任务执行结果，支持超时
11 get(long timeout, TimeUnit unit);
```


这 3 个 `submit()` 方法之间的区别在于方法参数不同，下面我们简要介绍一下。

1. 提交 Runnable 任务 `submit(Runnable task)`：这个方法的参数是一个 Runnable 接口，Runnable 接口的 `run()` 方法是没有返回值的，所以 `submit(Runnable task)` 这个方法返回的 Future 仅可以用来断言任务已经结束了，类似于 `Thread.join()`。
2. 提交 Callable 任务 `submit(Callable<T> task)`：这个方法的参数是一个 Callable 接口，它只有一个 `call()` 方法，并且这个方法是有返回值的，所以这个方法返回的 Future 对象可以通过调用其 `get()` 方法来获取任务的执行结果。
3. 提交 Runnable 任务及结果引用 `submit(Runnable task, T result)`：这个方法很有意思，假设这个方法返回的 Future 对象是 `f`，`f.get()` 的返回值就是传给 `submit()` 方法的参数 `result`。这个方法该怎么用呢？下面这段示例代码展示了它的经典用法。需要你注意的是 Runnable 接口的实现类 `Task` 声明了一个有参构造函数 `Task(Result r)`，创建 `Task` 对象的时候传入了 `result` 对象，这样就能在类 `Task` 的 `run()` 方法中对 `result` 进行各种操作了。`result` 相当于主线程和子线程之间的桥梁，通过它主子线程可以共享数据。

 复制代码


```
1 ExecutorService executor
2   = Executors.newFixedThreadPool(1);
3 // 创建 Result 对象 r
4 Result r = new Result();
5 r.setAAA(a);
6 // 提交任务
7 Future<Result> future =
8   executor.submit(new Task(r), r);
9 Result fr = future.get();
10 // 下面等式成立
11 fr === r;
12 fr.getAAA() === a;
13 fr.getXXX() === x
14
15 class Task implements Runnable{
16   Result r;
17   // 通过构造函数传入 result
18   Task(Result r){
19     this.r = r;
20   }
21   void run() {
22     // 可以操作 result
23     a = r.getAAA();
24     r.setXXX(x);
25   }
26 }
```

下面我们再来介绍 FutureTask 工具类。前面我们提到的 Future 是一个接口，而 FutureTask 是一个实实在在的工具类，这个工具类有两个构造函数，它们的参数和前面介绍的 submit() 方法类似，所以这里我就不再赘述了。

 复制代码

```
1 FutureTask(Callable<V> callable);
2 FutureTask(Runnable runnable, V result);
```

那如何使用 FutureTask 呢？其实很简单，FutureTask 实现了 Runnable 和 Future 接口，由于实现了 Runnable 接口，所以可以将 FutureTask 对象作为任务提交给 ThreadPoolExecutor 去执行，也可以直接被 Thread 执行；又因为实现了 Future 接口，所以也能用来获得任务的执行结果。下面的示例代码是将 FutureTask 对象提交给 ThreadPoolExecutor 去执行。

 复制代码

```
1 // 创建 FutureTask
2 FutureTask<Integer> futureTask
3   = new FutureTask<>(() -> 1+2);
4 // 创建线程池
5 ExecutorService es =
6   Executors.newCachedThreadPool();
7 // 提交 FutureTask
8 es.submit(futureTask);
9 // 获取计算结果
10 Integer result = futureTask.get();
```

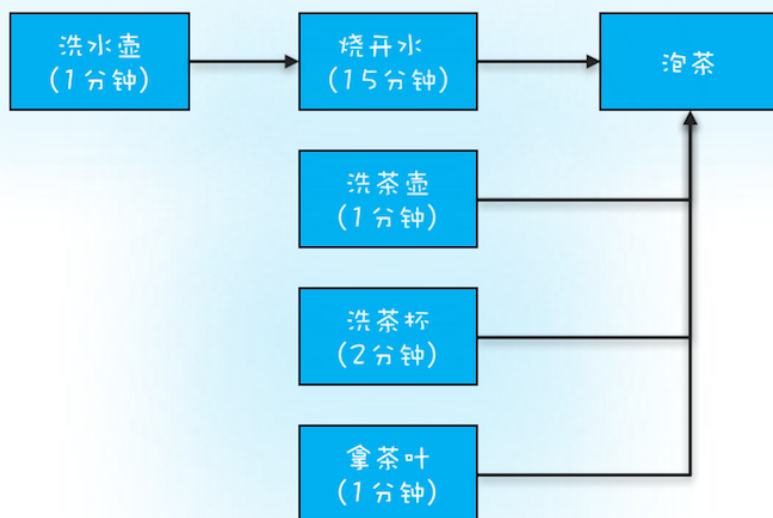
FutureTask 对象直接被 Thread 执行的示例代码如下所示。相信你已经发现了，利用 FutureTask 对象可以很容易获取子线程的执行结果。

 复制代码

```
1 // 创建 FutureTask
2 FutureTask<Integer> futureTask
3   = new FutureTask<>(() -> 1+2);
4 // 创建并启动线程
5 Thread T1 = new Thread(futureTask);
6 T1.start();
7 // 获取计算结果
8 Integer result = futureTask.get();
```

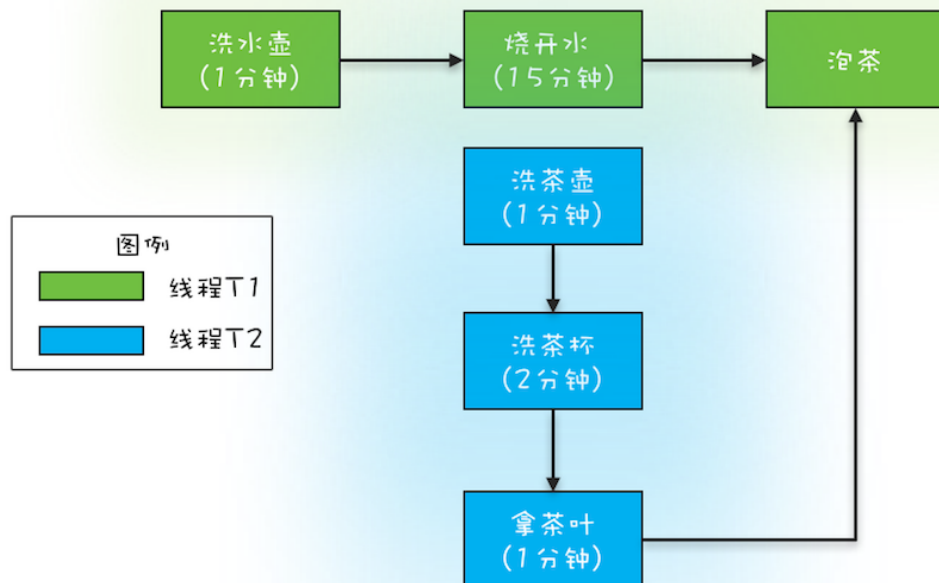
实现最优的“烧水泡茶”程序

记得以前初中语文课文里有一篇著名数学家华罗庚先生的文章《统筹方法》，这篇文章里介绍了一个烧水泡茶的例子，文中提到最优的工序应该是下面这样：



烧水泡茶最优工序

下面我们用程序来模拟一下这个最优工序。我们专栏前面曾经提到，并发编程可以总结为三个核心问题：分工、同步和互斥。编写并发程序，首先要做的就是分工，所谓分工指的是如何高效地拆解任务并分配给线程。对于烧水泡茶这个程序，一种最优的分工方案可以是下图所示的这样：用两个线程 T1 和 T2 来完成烧水泡茶程序，T1 负责洗水壶、烧开水、泡茶这三道工序，T2 负责洗茶壶、洗茶杯、拿茶叶三道工序，其中 T1 在执行泡茶这道工序时需要等待 T2 完成拿茶叶的工序。对于 T1 的这个等待动作，你应该可以想出很多种办法，例如 `Thread.join()`、`CountDownLatch`，甚至阻塞队列都可以解决，不过今天我们用 `Future` 特性来实现。



烧水泡茶最优分工方案

下面的示例代码就是用这一章提到的 Future 特性来实现的。首先，我们创建了两个 FutureTask——ft1 和 ft2，ft1 完成洗水壶、烧开水、泡茶的任务，ft2 完成洗茶壶、洗茶杯、拿茶叶的任务；这里需要注意的是 ft1 这个任务在执行泡茶任务前，需要等待 ft2 把茶叶拿来，所以 ft1 内部需要引用 ft2，并在执行泡茶之前，调用 ft2 的 get() 方法实现等待。

复制代码

```
1 // 创建任务 T2 的 FutureTask
2 FutureTask<String> ft2
3   = new FutureTask<>(new T2Task());
4 // 创建任务 T1 的 FutureTask
5 FutureTask<String> ft1
6   = new FutureTask<>(new T1Task(ft2));
7 // 线程 T1 执行任务 ft1
8 Thread T1 = new Thread(ft1);
9 T1.start();
10 // 线程 T2 执行任务 ft2
11 Thread T2 = new Thread(ft2);
12 T2.start();
13 // 等待线程 T1 执行结果
14 System.out.println(ft1.get());
15
16 // T1Task 需要执行的任务：
17 // 洗水壶、烧开水、泡茶
18 class T1Task implements Callable<String>{
19     FutureTask<String> ft2;
```

```
20 // T1 任务需要 T2 任务的 FutureTask
21 T1Task(FutureTask<String> ft2){
22     this.ft2 = ft2;
23 }
24 @Override
25 String call() throws Exception {
26     System.out.println("T1: 洗水壶...");
27     TimeUnit.SECONDS.sleep(1);
28
29     System.out.println("T1: 烧开水...");
30     TimeUnit.SECONDS.sleep(15);
31     // 获取 T2 线程的茶叶
32     String tf = ft2.get();
33     System.out.println("T1: 拿到茶叶:"+tf);
34
35     System.out.println("T1: 泡茶...");
36     return " 上茶:" + tf;
37 }
38 }
39 // T2Task 需要执行的任务:
40 // 洗茶壶、洗茶杯、拿茶叶
41 class T2Task implements Callable<String> {
42     @Override
43     String call() throws Exception {
44         System.out.println("T2: 洗茶壶...");
45         TimeUnit.SECONDS.sleep(1);
46
47         System.out.println("T2: 洗茶杯...");
48         TimeUnit.SECONDS.sleep(2);
49
50         System.out.println("T2: 拿茶叶...");
51         TimeUnit.SECONDS.sleep(1);
52         return " 龙井 ";
53     }
54 }
55 // 一次执行结果:
56 T1: 洗水壶...
57 T2: 洗茶壶...
58 T1: 烧开水...
59 T2: 洗茶杯...
60 T2: 拿茶叶...
61 T1: 拿到茶叶: 龙井
62 T1: 泡茶...
63 上茶: 龙井
```


总结

利用 Java 并发包提供的 Future 可以很容易获得异步任务的执行结果，无论异步任务是通过线程池 ThreadPoolExecutor 执行的，还是通过手工创建子线程来执行的。Future 可以类比为现实世界里的提货单，比如去蛋糕店订生日蛋糕，蛋糕店都是先给你一张提货单，你拿到提货单之后，没有必要一直在店里等着，可以先去干点其他事，比如看场电影；等看完电影后，基本上蛋糕也做好了，然后你就可以凭提货单领蛋糕了。

利用多线程可以快速将一些串行的任务并行化，从而提高性能；如果任务之间有依赖关系，比如当前任务依赖前一个任务的执行结果，这种问题基本上都可以用 Future 来解决。在分析这种问题的过程中，建议你用有向图描述一下任务之间的依赖关系，同时将线程的分工也做好，类似于烧水泡茶最优分工方案那幅图。对照图来写代码，好处是更形象，且不易出错。

课后思考

不久前听说小明要做一个询价应用，这个应用需要从三个电商询价，然后保存在自己的数据库里。核心示例代码如下所示，由于是串行的，所以性能很慢，你来试着优化一下吧。

 复制代码

```
1 // 向电商 S1 询价，并保存
2 r1 = getPriceByS1();
3 save(r1);
4 // 向电商 S2 询价，并保存
5 r2 = getPriceByS2();
6 save(r2);
7 // 向电商 S3 询价，并保存
8 r3 = getPriceByS3();
9 save(r3);
```

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

Java 并发编程实战

全面系统提升你的并发编程能力

王宝令

资深架构师



新版升级：点击「👤 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 22 | Executor与线程池：如何创建正确的线程池？

下一篇 24 | CompletableFuture：异步编程没那么难

精选留言 (23)

写留言



vector

2019-04-21

8

最近使用CompletableFuture工具方法以及lamda表达式比较多，语言语法的变化带来编码效率的提升真的很大。



aroll

2019-04-20

6

建议并发编程课程中的Demo代码，尽量少使用System.out.println, 因为其实有使用隐式锁，一些情况还会有锁粗化产生

作者回复: 好建议



Asanz

2019-04-21

👍 4

不是不建议使用 Executors 创建线程池了吗???

展开 ▾



linqw

2019-04-22

👍 2

课后习题，老师帮忙看下哦

```
public class ExecutorExample {
```

```
    private static final ExecutorService executor;
```

```
    static {executor = new ThreadPoolExecutor(4, 8, 1, TimeUnit.SECONDS, new  
        ArrayBlockingQueue<Runnable>(1000), runnable -> null, (r, executor) -> {//根据...
```

展开 ▾

作者回复: 没问题，就是有点复杂，代码还可以精简一下



圆圆

2019-04-22

👍 2

你这个不对啊，应该是executeservice.submit t2futuretask，不能直接提交t2



undefined

2019-04-20

👍 2

课后题：

可以用 Future

```
ExecutorService threadPoolExecutor = Executors.newFixedThreadPool(3);
```

```
Future<R> future1 = threadPoolExecutor.submit(Test::getPriceByS1);
```

```
Future<R> future2 = threadPoolExecutor.submit(Test::getPriceByS2);...
```

展开 ▾



liu

2019-04-25

👍 1

future是阻塞的等待。发起任务后，做其他的工作。做完后，从future获取处理结果，继续进行后面的任务



捞鱼的搬砖...

2019-04-21

👍 1

Future的get()是拿到任务的执行结果不吧。为什么又说是拿到方法的入参了。

展开 ▾



QQ怪

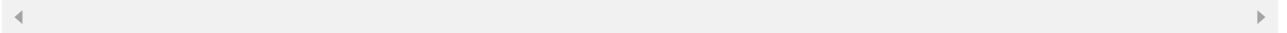
2019-04-20

👍 1

老师，在提交 Runnable 任务及结果引用的例子里面的x变量是什么？

展开 ▾

作者回复: 任意的东西，想成数字0也行



QQ怪

2019-04-20

👍 1

在实际项目中应用已经应用到了Future,但没有使用线程池，没有那么优雅，所以算是get到了👍



张三

2019-04-20

👍 1

打卡。感觉很神奇，之前完全不会用。学的知识太陈旧了，继续学习。

展开 ▾



张天屹

2019-04-20

👍 1

我不知道是不是理解错老师意思了，先分析依赖有向图，可以看到三条线，没有入度>1的节点

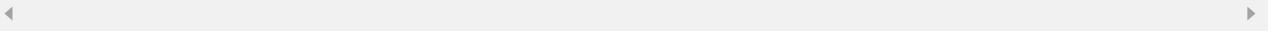
那么启动三个线程即可。

图：

s1询价 -> s1保存 ...

展开 ▾

作者回复: 用线程池就用到了



Sunsun

2019-05-28



```
public class ExecutorExample {
```

```
    private static final ExecutorService executor = Executors.newFixedThreadPool(4);
```

```
    public static void main(String[] args) {...
```

展开 ▾



三木禾

2019-05-18



这个可以用生产者消费者模式啊

展开 ▾



2019-04-30



分别提交三个futuretask给线程池，然后最后分别get出结果，统一进行保存数据库



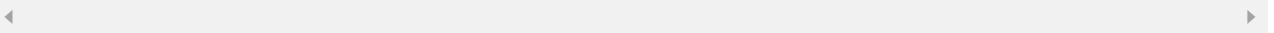
Sunqc

2019-04-30



老师，你所说的订蛋糕，我这样理解对吗，把任务提交给线程池就是让蛋糕店做蛋糕；去看电影就是主线程做其他事，提货单是对应调用future的get

作者回复: 理解的对



右耳听海

2019-04-26



回答思考，这三个任务如果没有结果依赖，直接用线程池提交三个任务应该就可以并行了吧



Cancer

2019-04-24



```
static class S1QueryTask implements Callable<Double>{  
    public Double call() throws Exception {  
        Thread.sleep(1000);//模拟查询时间  
        return Double.valueOf(10f);  
    }...  
}
```

展开 ▾



jeeker

2019-04-23



课后问题: S2和S3应该引用S1的结果吧,S2,3通过s1.get来做业务逻辑.

展开 ▾



张德

2019-04-21



我也同意张天屹同学的观点 这个询价操作如果之间没有联系的话 直接起三个线程就可以了 老师能不能讲一下 用线程池怎么就有关联了?

作者回复: 如果每分钟询价1万次，还能直接创建线程吗？联系指的是线程池和future，不是三个查询操作

