# JUC中的Executor框架详解2

## 本文内容

1. ExecutorCompletionService出现的背景
2. 介绍CompletionService接口及常用的方法
3. 介绍ExecutorCompletionService类及其原理
4. 示例：执行一批任务，然后消费执行结果
5. 示例【2种方式】：异步执行一批任务，有一个完成立即返回，其他取消

## **需要解决的问题**

还是举个例子说明更好理解一些。

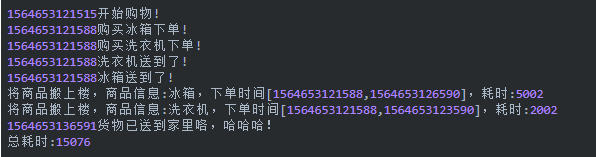
买新房了，然后在网上下单买冰箱、洗衣机，电器商家不同，所以送货耗时不一样，然后等他们送货，快递只愿送到楼下，然后我们自己将其搬到楼上的家中。

用程序来模拟上面的实现。示例代码如下：





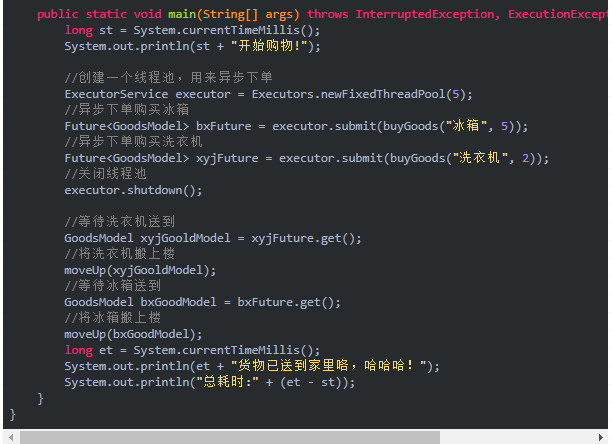
输出：



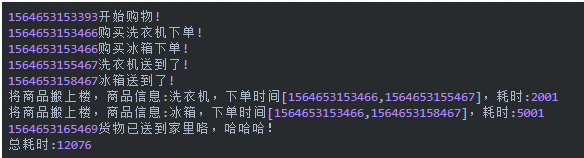
从输出中我们可以看出几个时间：

1. 购买冰箱耗时5秒
2. 购买洗衣机耗时2秒
3. 将冰箱送上楼耗时5秒
4. 将洗衣机送上楼耗时5秒
5. 共计耗时15秒

购买洗衣机、冰箱都是异步执行的，我们先把冰箱送上楼了，然后再把洗衣机送上楼了。上面大家应该发现了一个问题，洗衣机先到的，洗衣机到了，我们并没有去把洗衣机送上楼，而是在等待冰箱到货（bxFuture.get();），然后将冰箱送上楼，中间导致浪费了3秒，现实中应该是这样的，先到的先送上楼，修改一下代码，洗衣机先到的，先送洗衣机上楼，代码如下：



输出：



耗时12秒，比第一种少了3秒。

问题来了，上面是我们通过调整代码达到了最优效果，实际上，购买冰箱和洗衣机具体哪个耗时时间长我们是不知道的，怎么办呢，有没有什么解决办法？

## **CompletionService接口**

CompletionService相当于一个执行任务的服务，通过submit丢任务给这个服务，服务内部去执行任务，可以通过服务提供的一些方法获取服务中已经完成的任务。

**接口内的几个方法：**

Future<V> submit(Callable<V> task);

用于向服务中提交有返回结果的任务，并返回Future对象

Future<V> submit(Runnable task, V result);

用户向服务中提交有返回值的任务去执行，并返回Future对象

Future<V> take() throws InterruptedException;

从服务中返回并移除一个已经完成的任务，如果获取不到，会一致阻塞到有返回值为止。此方法会响应线程中断。

Future<V> poll();

从服务中返回并移除一个已经完成的任务，如果内部没有已经完成的任务，则返回空，此方法会立即响应。

Future<V> poll(long timeout, TimeUnit unit) throws InterruptedException;

尝试在指定的时间内从服务中返回并移除一个已经完成的任务，等待的时间超时还是没有获取到已完成的任务，则返回空。此方法会响应线程中断

通过submit向内部提交任意多个任务，通过take方法可以获取已经执行完成的任务，如果获取不到将等待。

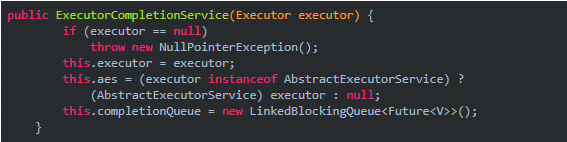
## **ExecutorCompletionService类**

ExecutorCompletionService类是CompletionService接口的具体实现。

说一下其内部原理，ExecutorCompletionService创建的时候会传入一个线程池，调用submit方法传入需要执行的任务，任务由内部的线程池来处理；ExecutorCompletionService内部有个阻塞队列，任意一个任务完成之后，会将任务的执行结果（Future类型）放入阻塞队列中，然后其他线程可以调用它take、poll方法从这个阻塞队列中获取一个已经完成的任务，获取任务返回结果的顺序和任务执行完成的先后顺序一致，所以最先完成的任务会先返回。

**关于阻塞队列的知识后面会专门抽几篇来讲，大家可以关注一下后面的文章。**

看一下构造方法：



构造方法需要传入一个Executor对象，这个对象表示任务执行器，所有传入的任务会被这个执行器执行。

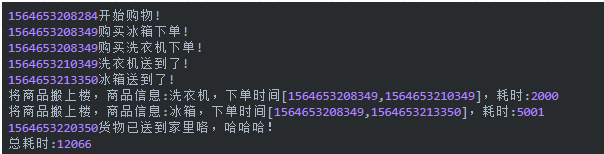
completionQueue是用来存储任务结果的阻塞队列，默认用采用的是LinkedBlockingQueue，也支持开发自己设置。通过submit传入需要执行的任务，任务执行完成之后，会放入completionQueue中，有兴趣的可以看一下原码，还是很好理解的。

## **使用ExecutorCompletionService解决文章开头的问题**

代码如下：



输出：



从输出中可以看出和我们希望的结果一致，代码中下单顺序是：冰箱、洗衣机，冰箱送货耗时5秒，洗衣机送货耗时2秒，洗衣机先到的，然后被送上楼了，冰箱后到被送上楼，总共耗时12秒，和期望的方案一样。

## **示例：执行一批任务，然后消费执行结果**

代码如下：



输出：



代码中传入了一批任务进行处理，最终将所有处理完成的按任务完成的先后顺序传递给Consumer进行消费了。

## **示例：异步执行一批任务，有一个完成立即返回，其他取消**

这个给大家讲解2种方式。

### 方式1

使用ExecutorCompletionService实现，ExecutorCompletionService提供了获取一批任务中最先完成的任务结果的能力。

**代码如下：**



程序输出厦门结果然后停止了：



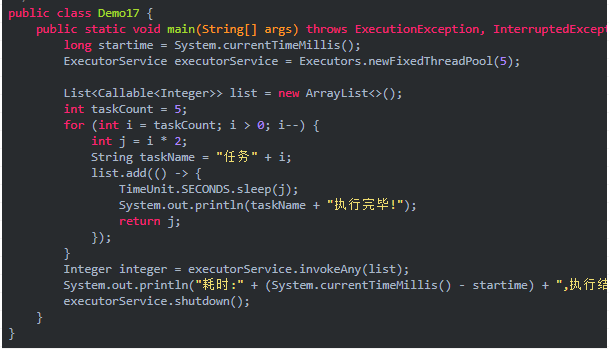
代码中执行了5个任务，使用CompletionService执行任务，调用take方法获取最先执行完成的任务，然后返回。在finally中对所有任务发送取消操作（future.cancel(true);），从输出中可以看出只有任务1执行成功，其他任务被成功取消了，符合预期结果。

### 方式2

其实ExecutorService已经为我们提供了这样的方法，方法声明如下：

<T> T invokeAny(Collection<? extends Callable<T>> tasks)  
        throws InterruptedException, ExecutionException;

**示例代码：**



输出下面结果之后停止：



输出结果和方式1中结果类似。