## 一：Callable接口

Callable的接口定义如下

public interface Callable<V> {

V call() throws Exception;

}

该接口声明了一个名称为call()的方法，同时这个方法可以有返回值V，也可以抛出异常。嗯，对该接口我们先了解这么多就行，下面我们来说明如何使用，前篇文章我们说过，无论是Runnable接口的实现类还是Callable接口的实现类，都可以被ThreadPoolExecutor或ScheduledThreadPoolExecutor执行，ThreadPoolExecutor或ScheduledThreadPoolExecutor都实现了ExcutorService接口，而因此Callable需要和Executor框架中的ExcutorService结合使用，我们先看看ExecutorService提供的方法：

<T> Future<T> submit(Callable<T> task);

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

Future<?> submit(Runnable task);

第一个方法：submit提交一个实现Callable接口的任务，并且返回封装了异步计算结果的Future。

第二个方法：submit提交一个实现Runnable接口的任务，并且指定了在调用Future的get方法时返回的result对象。（不常用）

第三个方法：submit提交一个实现Runnable接口的任务，并且返回封装了异步计算结果的Future。

因此我们只要创建好我们的线程对象（实现Callable接口或者Runnable接口），然后通过上面3个方法提交给线程池去执行即可。还有点要注意的是，除了我们自己实现Callable对象外，我们还可以使用工厂类Executors来把一个Runnable对象包装成Callable对象。Executors工厂类提供的方法如下：

public static Callable<Object> callable(Runnable task)

public static <T> Callable<T> callable(Runnable task, T result)

## 二：Future<V>接口

Future<V>接口是用来获取异步计算结果的，说白了就是对具体的Runnable或者Callable对象任务执行的结果进行获取(get()),取消(cancel()),判断是否完成等操作。我们看看Future接口的源码：

public interface Future<V> {

boolean cancel(boolean mayInterruptIfRunning);

boolean isCancelled();

boolean isDone();

V get() throws InterruptedException, ExecutionException;

V get(long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException;

}

方法解析：

V get() ：获取异步执行的结果，如果没有结果可用，此方法会阻塞直到异步计算完成。

V get(Long timeout , TimeUnit unit) ：获取异步执行结果，如果没有结果可用，此方法会阻塞，但是会有时间限制，如果阻塞时间超过设定的timeout时间，该方法将抛出异常。

boolean isDone() ：如果任务执行结束，无论是正常结束或是中途取消还是发生异常，都返回true。

boolean isCanceller() ：如果任务完成前被取消，则返回true。

boolean cancel(boolean mayInterruptRunning) ：如果任务还没开始，执行cancel(...)方法将返回false；如果任务已经启动，执行cancel(true)方法将以中断执行此任务线程的方式来试图停止任务，如果停止成功，返回true；当任务已经启动，执行cancel(false)方法将不会对正在执行的任务线程产生影响(让线程正常执行到完成)，此时返回false；当任务已经完成，执行cancel(...)方法将返回false。mayInterruptRunning参数表示是否中断执行中的线程。

通过方法分析我们也知道实际上Future提供了3种功能：（1）能够中断执行中的任务（2）判断任务是否执行完成（3）获取任务执行完成后额结果。

但是我们必须明白Future只是一个接口，我们无法直接创建对象，因此就需要其实现类FutureTask登场啦。

## 三：FutureTask类

我们先来看看FutureTask的实现

public class FutureTask<V> implements RunnableFuture<V> {

FutureTask类实现了RunnableFuture接口，我们看一下RunnableFuture接口的实现：

public interface RunnableFuture<V> extends Runnable, Future<V> {

void run();

}

分析：FutureTask除了实现了Future接口外还实现了Runnable接口（即可以通过Runnable接口实现线程，也可以通过Future取得线程执行完后的结果），因此FutureTask也可以直接提交给Executor执行。最后我们给出FutureTask的两种构造函数：

public FutureTask(Callable<V> callable) {

}

public FutureTask(Runnable runnable, V result) {

}

案例：使用Callable+Future获取执行结果

package com.simple.javathreaddemo.callable.CallableFuture;  
  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
*/\*\*  
 \** ***@author*** *Simple  
 \** ***@date*** *2021/3/11 16:18  
 \*/*public class CallableTest {  
 public static void main(String[] args) {  
 //创建线程池  
 ExecutorService executorService = Executors.*newSingleThreadExecutor*();  
 //创建Callable  
 }  
}

package com.simple.javathreaddemo.callable.CallableFutureandTask;  
  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.Future;  
  
*/\*\*  
 \** ***@author*** *Simple  
 \** ***@date*** *2021/3/11 16:18  
 \*/*public class CallableFutureTest {  
 public static void main(String[] args) {  
 //创建线程池  
 ExecutorService executorService = Executors.*newSingleThreadExecutor*();  
 //创建Callable对象任务  
 CallableDemo caltask = new CallableDemo();  
 //提交任务并获取执行结果  
 Future<Integer> future = executorService.submit(caltask);  
 //关闭线程池  
 executorService.shutdown();  
  
 try {  
 Thread.*sleep*(2000);  
 System.*out*.println("主线程在执行其他任务");  
  
 if (future.get() != null){  
 //输出获取到的结果  
 System.*out*.println("future.get()--->"+future.get());  
 }else {  
 System.*out*.println("未获取到结果!");  
 }  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 System.*out*.println("主线程执行完成");  
 }  
}

案例：使用Callable+FutureTask获取执行结果

package com.simple.javathreaddemo.callable.CallableFutureandTask;  
  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.FutureTask;  
  
*/\*\*  
 \** ***@author*** *Simple  
 \** ***@date*** *2021/3/12 8:47  
 \*/*public class CallableFutureTaskTest {  
 public static void main(String[] args) {  
 //创建线程池  
 ExecutorService es = Executors.*newSingleThreadExecutor*();  
 //创建Callable对象任务  
 CallableDemo calTask = new CallableDemo();  
 //创建FutureTask  
 FutureTask<Integer> futureTask = new FutureTask<>(calTask);  
 //执行任务  
 es.submit(futureTask);  
 es.shutdown();  
  
 try {  
 Thread.*sleep*(2000);  
 System.*out*.println("主线程在执行其他任务");  
  
 if (futureTask.get() != null){  
 //输出获取到的结果  
 System.*out*.println("future.get()--->"+futureTask.get());  
 }else {  
 System.*out*.println("未获取到结果!");  
 }  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 System.*out*.println("主线程执行完成");  
 }  
}