[Java多线程-Callable的Future返回值的使用](https://www.cnblogs.com/syp172654682/p/9788051.html)

一般使用线程池执行任务都是调用的execute方法，这个方法定义在Executor接口中：

public interface Executor {

　　void execute(Runnable command);

}

这个方法是没有返回值的，而且只接受Runnable。

那么像得到线程的返回值怎嘛办呢？

在ExecutorService接口中能找到这个方法：

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

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

Future<?> submit(Runnable task);

这个方法接收两种参数，Callable和Runnable。返回值是Future。

下面具体看一下这些是什么东西。

**Callable和Runnable**  
先看一下两个接口的定义：

Callable

public interface Callable<V> {

　　V call() throws Exception;

}

Runnable

interface Runnable {

　　public abstract void run();

}

和明显能看到区别：

1.Callable能接受一个泛型，然后在call方法中返回一个这个类型的值。而Runnable的run方法没有返回值  
2.Callable的call方法可以抛出异常，而Runnable的run方法不会抛出异常。

**Future**  
返回值Future也是一个接口，通过他可以获得任务执行的返回值。

定义如下：

[复制代码](javascript:void(0);)

public interface Future<V> {

　　boolean cancel(boolean var1);

　　boolean isCancelled();

　　boolean isDone();

　　V get() throws InterruptedException, ExecutionException;

　　V get(long var1, TimeUnit var3) throws InterruptedException, ExecutionException, TimeoutException;

}

[复制代码](javascript:void(0);)

其中的get方法获取的就是返回值。

来个例子

submit(Callable task)

[复制代码](javascript:void(0);)

public class Main {

　　public static void main(String[] args) throws InterruptedException, ExecutionException {

　　ExecutorService executor = Executors.newFixedThreadPool(2);

　　//创建一个Callable，3秒后返回String类型

　　Callable myCallable = new Callable() {

　　　　@Override

　　　　public String call() throws Exception {

　　　　　　Thread.sleep(3000);

　　　　　　System.out.println("calld方法执行了");

　　　　　　return "call方法返回值";

　　　　}

　　};

　　System.out.println("提交任务之前 "+getStringDate());

　　Future future = executor.submit(myCallable);

　　System.out.println("提交任务之后，获取结果之前 "+getStringDate());

　　System.out.println("获取返回值: "+future.get());

　　System.out.println("获取到结果之后 "+getStringDate());

　　}

　　public static String getStringDate() {

　　　　Date currentTime = new Date();

　　　　SimpleDateFormat formatter = new SimpleDateFormat("HH:mm:ss");

　　　　String dateString = formatter.format(currentTime);

　　　　return dateString;

　　　　}

　　}

[复制代码](javascript:void(0);)

通过executor.submit提交一个Callable，返回一个Future，然后通过这个Future的get方法取得返回值。

看一下输出：

提交任务之前 12:13:01

提交任务之后，获取结果之前 12:13:01

calld方法执行了

获取返回值: call方法返回值

获取到结果之后 12:13:04

**get()方法的阻塞性**  
通过上面的输出可以看到，在调用submit提交任务之后，主线程本来是继续运行了。但是运行到future.get()的时候就阻塞住了，一直等到任务执行完毕，拿到了返回的返回值，主线程才会继续运行。

这里注意一下，他的阻塞性是因为调用get()方法时，任务还没有执行完，所以会一直等到任务完成，形成了阻塞。

任务是在调用submit方法时就开始执行了，如果在调用get()方法时，任务已经执行完毕，那么就不会造成阻塞。

下面在调用方法前先睡4秒，这时就能马上得到返回值。

[复制代码](javascript:void(0);)

System.out.println("提交任务之前 "+getStringDate());

Future future = executor.submit(myCallable);

System.out.println("提交任务之后 "+getStringDate());

Thread.sleep(4000);

System.out.println("已经睡了4秒,开始获取结果 "+getStringDate());

System.out.println("获取返回值: "+future.get());

System.out.println("获取到结果之后 "+getStringDate());

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

提交任务之前 12:36:04

提交任务之后 12:36:04

calld方法执行了

已经睡了4秒,开始获取结果 12:36:08

获取返回值: call方法返回值

获取到结果之后 12:36:08

[复制代码](javascript:void(0);)

可以看到吗，因为睡了4秒，任务已经执行完毕，所以get方法立马就得到了结果。

同样的原因，submit两个任务时，总阻塞时间是最长的那个。

例如，有两个任务，一个3秒，一个5秒。

[复制代码](javascript:void(0);)

Callable myCallable = new Callable() {

　　@Override

　　public String call() throws Exception {

　　Thread.sleep(5000);

　　System.out.println("calld方法执行了");

　　return "call方法返回值";

　　}

};

Callable myCallable2 = new Callable() {

　　@Override

　　public String call() throws Exception {

　　Thread.sleep(3000);

　　System.out.println("calld2方法执行了");

　　return "call2方法返回值";

　　}

};

System.out.println("提交任务之前 "+getStringDate());

Future future = executor.submit(myCallable);

Future future2 = executor.submit(myCallable2);

System.out.println("提交任务之后 "+getStringDate());

System.out.println("开始获取第一个返回值 "+getStringDate());

System.out.println("获取返回值: "+future.get());

System.out.println("获取第一个返回值结束，开始获取第二个返回值 "+getStringDate());

System.out.println("获取返回值2: "+future2.get());

System.out.println("获取第二个返回值结束 "+getStringDate());

[复制代码](javascript:void(0);)

输出

[复制代码](javascript:void(0);)

提交任务之前 14:14:47

提交任务之后 14:14:48

开始获取第一个返回值 14:14:48

calld2方法执行了

calld方法执行了

获取返回值: call方法返回值

获取第一个返回值结束，开始获取第二个返回值 14:14:53

获取返回值2: call2方法返回值

获取第二个返回值结束 14:14:53

[复制代码](javascript:void(0);)

获取第一个结果阻塞了5秒，所以获取第二个结果立马就得到了。

**submit(Runnable task)**  
因为Runnable是没有返回值的，所以如果submit一个Runnable的话，get得到的为null：

[复制代码](javascript:void(0);)

Runnable myRunnable = new Runnable() {

　　@Override

　　public void run() {

　　try {

　　　　Thread.sleep(2000);

　　　　System.out.println(Thread.currentThread().getName() + " run time: " + System.currentTimeMillis());

　　} catch (InterruptedException e) {

　　　　e.printStackTrace();

　　}

　　}

};

Future future = executor.submit(myRunnable);

System.out.println("获取的返回值： "+future.get());

[复制代码](javascript:void(0);)

输出为：

pool-1-thread-1 run time: 1493966762524

获取的返回值： null

**submit(Runnable task, T result)**  
虽然submit传入Runnable不能直接返回内容，但是可以通过submit(Runnable task, T result)传入一个载体，通过这个载体获取返回值。这个其实不能算返回值了，是交给线程处理一下。

先新建一个载体类Data：

[复制代码](javascript:void(0);)

public static class Data {

　　String name;

　　String sex;

　　public String getName() {

　　　　return name;

　　}

　　public void setName(String name) {

　　　　this.name = name;

　　}

　　public String getSex() {

　　　　return sex;

　　}

　　public void setSex(String sex) {

　　　　this.sex = sex;

　　}

}

[复制代码](javascript:void(0);)

然后在Runnable的构造方法中传入：

[复制代码](javascript:void(0);)

static class MyThread implements Runnable {

　　Data data;

　　public MyThread(Data name) {

　　　　this.data = name;

　　}

　　@Override

　　public void run() {

　　　　try {

　　　　　　Thread.sleep(2000);

　　　　　　System.out.println("线程 执行:");

　　　　　　data.setName("新名字");

　　　　　　data.setSex("新性别");

　　　} catch (InterruptedException e) {

　　　　　　e.printStackTrace();

　　　　}

　　}

}

[复制代码](javascript:void(0);)

然后调用：

Data data = new Data();

Future<Data> future = executor.submit(new MyThread(data), data);

System.out.println("返回的结果 name: " + future.get().getName()+", sex: "+future.get().getSex());

System.out.println("原来的Data name: " + data.getName()+", sex: "+data.getSex());

输出：

线程 执行:

返回的结果 name: 新名字, sex: 新性别

原来的Data name: 新名字, sex: 新性别

发现原来的data也变了。

**get(long var1, TimeUnit var3)**  
前面都是用的get()方法获取返回值，那么因为这个方法是阻塞的，有时需要等很久。所以有时候需要设置超时时间。

get(long var1, TimeUnit var3)这个方法就是设置等待时间的。

如下面的任务需要5秒才能返回结果：

[复制代码](javascript:void(0);)

Callable myCallable = new Callable() {

　　@Override

　　public String call() throws Exception {

　　　　Thread.sleep(5000);

　　　　return "我是结果";

　　}

};

[复制代码](javascript:void(0);)

使用get:

Future future1 = executor.submit(myCallable);

System.out.println("开始拿结果 "+getStringDate());

System.out.println("返回的结果是： "+future1.get()+ " "+getStringDate());

System.out.println("结束拿结果 "+getStringDate());

输出是：

开始拿结果 16:00:43

返回的结果是： 我是结果 16:00:48

结束拿结果 16:00:48

现在要求最多等3秒，拿不到返回值就不要了，所以用get(long var1, TimeUnit var3)这个方法

方法的第一个参数是长整形数字，第二个参数是单位，跟线程池ThreadPoolExecutor的构造方法里一样的。

[复制代码](javascript:void(0);)

Future future1 = executor.submit(myCallable);

System.out.println("开始拿结果 "+getStringDate());

try {

　　System.out.println("返回的结果是： "+future1.get(3, TimeUnit.SECONDS)+ " "+getStringDate());

} catch (TimeoutException e) {

　　e.printStackTrace();

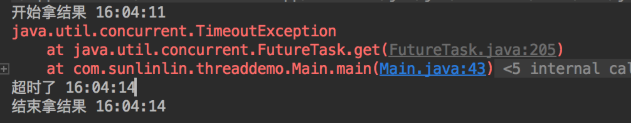
　　System.out.println("超时了 "+getStringDate());

}

System.out.println("结束拿结果 "+getStringDate());

[复制代码](javascript:void(0);)

然后输出是



过了三秒就抛出超时异常了，主线程继续运行，不会再继续阻塞。

**异常**  
使用submit方法还有一个特点就是，他的异常可以在主线程中catch到。

而使用execute方法执行任务是捕捉不到异常的。

用下面这个Runnable来说,这个 里面一定会抛出一个异常

[复制代码](javascript:void(0);)

Runnable myRunnable = new Runnable() {

　　@Override

　　public void run() {

　　　　executor.execute(null);

　　}

};

[复制代码](javascript:void(0);)

使用execute  
这里如果捕捉到异常，只打印一行异常信息。

[复制代码](javascript:void(0);)

try {

　　executor.execute(myRunnable);

} catch (Exception e) {

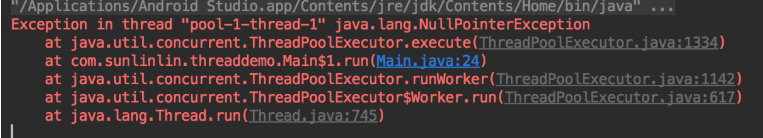
　　e.printStackTrace();

　　System.out.println("抓到异常 "+e.getMessage());

}

[复制代码](javascript:void(0);)

输出



并没有出现抓到异常哪行日志。而且这个异常输出是在线程pool-1-thread-1中，并不是在主线程中。说明主线程的catch不能捕捉到这个异常。

使用submit

[复制代码](javascript:void(0);)

try {

　　Future future1= executor.submit(myCallable);

　　future1.get();

} catch (Exception e) {

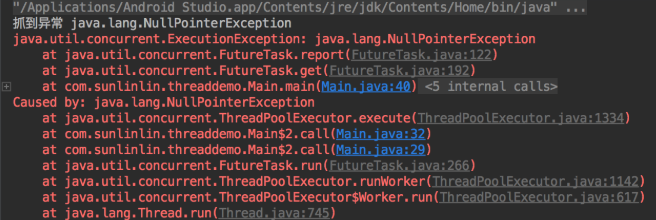
　　e.printStackTrace();

System.out.println("抓到异常 "+e.getMessage());

}

[复制代码](javascript:void(0);)

输出



这个就能抓到异常了。