Java 回调机制解读

五月的仓颉 [程序员之家](javascript:void(0);) 2017-04-25

作者：五月的仓颉

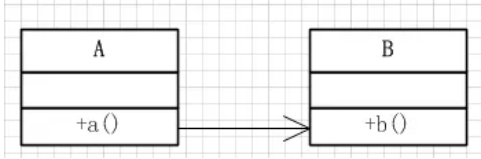
原文：www.cnblogs.com/xrq730/p/6424471.html

（点击文末阅读原文即可前往）

**模块间调用**

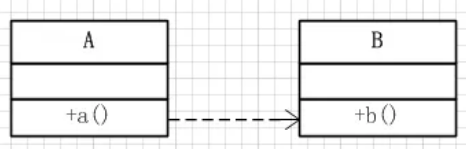
在一个应用系统中，无论使用何种语言开发，必然存在模块之间的调用，调用的方式分为几种：

**（1）同步调用**



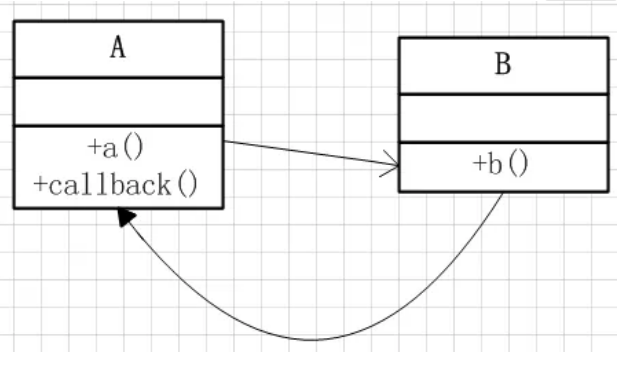
同步调用是最基本并且最简单的一种调用方式，类A的方法a()调用类B的方法b()，一直等待b()方法执行完毕，a()方法继续往下走。**这种调用方式适用于方法b()执行时间不长的情况**，因为b()方法执行时间一长或者直接阻塞的话，a()方法的余下代码是无法执行下去的，这样会造成整个流程的阻塞。

**（2）异步调用**



异步调用是为了解决同步调用可能出现阻塞，导致整个流程卡住而产生的一种调用方式。**类A的方法方法a()通过新起线程的方式调用类B的方法b()，代码接着直接往下执行**，这样无论方法b()执行时间多久，都不会阻塞住方法a()的执行。但是这种方式，由于方法a()不等待方法b()的执行完成，在方法a()需要方法b()执行结果的情况下（视具体业务而定，有些业务比如启异步线程发个微信通知、刷新一个缓存这种就没必要），必须通过一定的方式对方法b()的执行结果进行监听。在Java中，可以使用Future+Callable的方式做到这一点，具体做法可以参见我的这篇文章《 Java多线程21：多线程下其他组件之CyclicBarrier、Callable、Future和FutureTask 》*（注1）*。

**（3）回调**



最后是回调，回调的思想是：

* 类A的a()方法调用类B的b()方法
* 类B的b()方法执行完毕主动调用类A的callback()方法

这样一种调用方式组成了上图，也就是一种双向的调用方式。

**代码示例**

接下来看一下回调的代码示例，代码模拟的是这样一种场景：老师问学生问题，学生思考完毕回答老师。

首先定义一个回调接口，只有一个方法tellAnswer(int answer)，即学生思考完毕告诉老师答案：

/\*\*

 \* 回调接口，原文出处http://www.cnblogs.com/xrq730/p/6424471.html

 \*/

public interface Callback {

    public void tellAnswer(int answer);

}

定义一个老师对象，实现Callback接口：

/\*\*

 \* 老师对象，原文出处http://www.cnblogs.com/xrq730/p/6424471.html

 \*/

public class Teacher implements Callback {

    private Student student;

    public Teacher(Student student) {

        this.student = student;

    }

    public void askQuestion() {

        student.resolveQuestion(this);

    }

    @Override

    public void tellAnswer(int answer) {

        System.out.println("知道了，你的答案是" + answer);

    }

}

老师对象有两个public方法：

（1）回调接口tellAnswer(int answer)，即学生回答完毕问题之后，老师要做的事情

（2）问问题方法askQuestion()，即向学生问问题

接着定义一个学生接口，学生当然是解决问题，但是接收一个Callback参数，这样学生就知道解决完毕问题向谁报告：

/\*\*

 \* 学生接口，原文出处http://www.cnblogs.com/xrq730/p/6424471.html

 \*/

public interface Student {

    public void resolveQuestion(Callback callback);

}

最后定义一个具体的学生叫Ricky：

/\*\*

 \* 一个名叫Ricky的同学解决老师提出的问题，原文出处http://www.cnblogs.com/xrq730/p/6424471.html

 \*/

public class Ricky implements Student {

    @Override

    public void resolveQuestion(Callback callback) {

        // 模拟解决问题

        try {

            Thread.sleep(3000);

        } catch (InterruptedException e) {

        }

        // 回调，告诉老师作业写了多久

        callback.tellAnswer(3);

    }

}

在解决完毕问题之后，第16行向老师报告答案。

写一个测试类，比较简单：

/\*\*

 \* 回调测试，原文出处http://www.cnblogs.com/xrq730/p/6424471.html

 \*/

public class CallbackTest {

    @Test

    public void testCallback() {

        Student student = new Ricky();

        Teacher teacher = new Teacher(student);

        teacher.askQuestion();

    }

}

代码运行结果就一行：

知道了，你的答案是3

简单总结、分析一下这个例子就是：

（1）老师调用学生接口的方法resolveQuestion，向学生提问

（2）学生解决完毕问题之后调用老师的回调方法tellAnswer

这样一套流程，构成了一种双向调用的关系。

**代码分析**

分析一下上面的代码，上面的代码我这里做了两层的抽象：

**（1）将老师进行抽象**

将老师进行抽象之后，对于学生来说，就不需要关心到底是哪位老师询问我问题，只要我根据询问的问题，得出答案，然后告诉提问的老师就可以了，即使老师换了一茬又一茬，对我学生而言都是没有任何影响的。

**（2）将学生进行抽象**

将学生进行抽象之后，对于老师这边来说就非常灵活，因为老师未必对一个学生进行提问，可能同时对Ricky、Jack、Lucy三个学生进行提问，这样就可以将成员变量Student改为List<Student>，这样在提问的时候遍历Student列表进行提问，然后得到每个学生的回答即可。

这个例子是一个典型的体现接口作用的例子，之所以这么说是因为我想到有些朋友可能不太明白接口的好处，不太明白接口好处的朋友可以重点看一下这个例子，多多理解。

总结起来，**回调的核心就是回调方将本身即this传递给调用方**，这样调用方就可以在调用完毕之后告诉回调方它想要知道的信息。回调是一种思想、是一种机制，至于具体如何实现，如何通过代码将回调实现得优雅、实现得可扩展性比较高，一看开发者的个人水平，二看开发者对业务的理解程度。

**同步回调与异步回调**

上面的例子，可能有人会提出这样的疑问：

这个例子需要用什么回调啊，使用同步调用的方式，学生对象回答完毕问题之后直接把回答的答案返回给老师对象不就好了？

这个问题的提出没有任何问题，可以从两个角度去理解这个问题。

首先，老师不仅仅想要得到学生的答案怎么办？可能这个老师是个更喜欢听学生解题思路的老师，在得到学生的答案之前，老师更想先知道学生姓名和学生的解题思路，当然有些人可以说，那我可以定义一个对象，里面加上学生的姓名和解题思路不就好了。这个说法在我看来有两个问题：

（1）如果老师想要的数据越来越多，那么返回的对象得越来越大，而使用回调则可以进行数据分离，将一批数据放在回调方法中进行处理，至于哪些数据依具体业务而定，如果需要增加返回参数，直接在回调方法中增加即可

（2）无法解决老师希望得到学生姓名、学生解题思路先于学生回答的答案的问题

因此我认为简单的返回某个结果确实没有必要使用回调而可以直接使用同步调用，但是如果**有多种数据需要处理且数据有主次之分**，使用回调会是一种更加合适的选择，优先处理的数据放在回调方法中先处理掉。

另外一个理解的角度则更加重要，就是标题说的同步回调和异步回调了。例子是一个同步回调的例子，意思是老师向Ricky问问题，Ricky给出答案，老师问下一个同学，得到答案之后继续问下一个同学，这是一种正常的场景，但是如果我把场景改一下：

老师并不想One-By-One这样提问，而是同时向Ricky、Mike、Lucy、Bruce、Kate五位同学提问，让同学们自己思考，哪位同学思考好了就直接告诉老师答案即可。

这种场景相当于是说，同学思考完毕完毕问题要有一个办法告诉老师，有两个解决方案：

（1）使用Future+Callable的方式，等待异步线程执行结果，这相当于就是同步调用的一种变种，因为其本质还是方法返回一个结果，即学生的回答

（2）使用异步回调，同学回答完毕问题，调用回调接口方法告诉老师答案即可。由于老师对象被抽象成了Callback接口，因此这种做法的扩展性非常好，就像之前说的，即使老师换了换了一茬又一茬，对于同学来说，只关心的是调用Callback接口回传必要的信息即可