著作权归作者所有。  
商业转载请联系作者获得授权，非商业转载请注明出处。  
作者：futeng  
链接：http://www.zhihu.com/question/19801131/answer/26586203  
来源：知乎

前言

在Java社区的各种开源工具中，回调方法的使用俯拾即是。所以熟悉回调方法无疑能加速自己对开源轮子的掌握。  
网上搜了一些文章，奈何对回调方法的介绍大多只停留在什么是回调方法的程度上。本篇文章尝试从回调方法怎么来的、为什么要使用回调方法以及在实际项目中如何使用等方面来介绍下。

场景

场景选择的得当与否，很影响读者的继续阅读的兴趣甚至理解的主动性（长期作为互联网技术博文读者的我，深有感触）。  
好场景私以为是：熟悉且简单。

本例小心翼翼选择的场景是：写作业。（hope you like）

自己写

注：写作业这个动作至少交代三个方面：谁，什么动作（写），写什么。  
下面先从（有个学生，写，作业）开始。

# 1. 有个学生Student student = new Student();

# 2. 该学生有写作业这个动作需要执行student.doHomeWork(someHomeWork);

# 3. 注意到这个写作业这个动作是需要得到入参“作业”的后才能进行的。所以给这个学生new了个简单的题目做。

String aHomeWork = "1+1=?";student.doHomeWork(aHomeWork);

至此，完成写作业的动作。

*完整代码*

public class Student {

public void doHomeWork(String homeWork) {

System.out.println("作业本");

if("1+1=?".equals(homeWork)) {

System.out.println("作业："+homeWork+" 答案："+"2");

} else {

System.out.println("作业："+homeWork+" 答案："+"不知道~~");

}

}

public static void main(String[] args) {

Student student = new Student();

String aHomeWork = "1+1=?";

student.doHomeWork(aHomeWork);

}}

*程序执行*

作业本

作业：1+1=? 答案：2

我们一定要把焦点聚焦到，”写作业“这个需求上面。  
该学生写作业的方法是现成的，但是需要有作业作为入参，怎么获取作业才是完成动作的关键。希望这点能深深印入我们的脑海。

让室友帮忙解答

上面的例子中该同学自己调用自己的方法，把收到的homework直接写了。  
但是现实可能会出现各种各样的问题导致该同学不能（xiang）自己来做。比如他想玩游戏或者有约会。所以他拜托了他的好室友（roommate）来帮忙写下。该怎么实现呢。

#1. 因为室友帮忙写，所以在doHomeWork动作里面，就不需要有逻辑判断的代码，因为舍友会直接把答案写进来。改成：

student.doHomeWork(aHomeWork, theAnswer);

#上句中做作业的动作支持传入“作业”和“答案”，有了这两个，就说明能做好作业了。

#其中aHomeWork作业是已知的，但是theAnswer这个答案却是室友提供的。

#室友怎么才能提供答案呢，最简单是，室友这个对象直接提供一个传入作业，传出答案的方法，这样该同学就可以直接调用了。

RoomMate roomMate = new RoomMate();String theAnswer = roomMate.getAnswer(aHomeWork);student.doHomeWork(aHomeWork,theAnswer);

*完整代码*

public class Student {

public void doHomeWork(String homeWork, String answer) {

System.out.println("作业本");

if(answer != null) {

System.out.println("作业："+homeWork+" 答案："+ answer);

} else {

System.out.println("作业："+homeWork+" 答案："+ "(空白)");

}

}

public static void main(String[] args) {

Student student = new Student();

String aHomeWork = "1+1=?";

RoomMate roomMate = new RoomMate();

String theAnswer = roomMate.getAnswer(aHomeWork);

student.doHomeWork(aHomeWork, theAnswer);

}}

public class RoomMate {

public String getAnswer(String homework) {

if("1+1=?".equals(homework)) {

return "2";

} else {

return null;

}

}}

*程序执行*

作业本

作业：1+1=? 答案：2

**怒，说好的回调方法呢~~**

因为到目前为止，不需要使用回调方法。  
技术总是伴随新的需求出现的。

**好，给你新的需求。**  
注意重点来了  
我们回顾下这两行代码

#室友写好作业String theAnswer = roomMate.getAnswer(aHomeWork);

#该同学直接抄答案，完成作业student.doHomeWork(aHomeWork, theAnswer);

该同学想了想，你给了答案有屁用，还得要我自己誊写到作业本上面去（执行自己的做作业方法）。  
你就不能**直接调用我的做作业方法**帮我把答案写好，把作业做完得了。

让室友直接把作业写了

经不住该同学的软磨硬泡，“中国好室友”答应了。怎么实现呢。  
再回顾下做作业的全过程

#待解决的作业String aHomeWork = "1+1=?";#室友写出答案String theAnswer = roomMate.getAnswer(aHomeWork);

#该同学调用，自己把答案写到作业本。（也即是这个步骤不给调用了） student.doHomeWork(aHomeWork, theAnswer);

#做作业必须得调用这个方法，而根据需求这个方法必须由室友去调用。那很显 然，该室友得保持一个该同学的引用，才能正常调用啊。

#灯灯灯~

#室友说，那你在调用getAnswer方法的时候，除了传入作业，还需要把自己的

引用放里面。这样我做完了，直接调用你的做作业方法就行了。 roomMate.getAnswer(aHomeWork,student);

*完整代码*

public class Student {

public void doHomeWork(String homeWork, String answer) {

System.out.println("作业本");

if(answer != null) {

System.out.println("作业："+homeWork+" 答案："+ answer);

} else {

System.out.println("作业："+homeWork+" 答案："+ "(空白)");

}

}

public static void main(String[] args) {

Student student = new Student();

String aHomeWork = "1+1=?";

RoomMate roomMate = new RoomMate();

roomMate.getAnswer(aHomeWork,student);

}}

public class RoomMate {

public void getAnswer(String homework, Student student) {

if("1+1=?".equals(homework)) {

student.doHomeWork(homework, "2");

} else {

student.doHomeWork(homework, "(空白)");

}

}

}

*执行程序*

作业本

作业：1+1=? 答案：2

回调方法

在上述“让室友直接把作业写了”的例子中，其实已经体现了回调的意思。  
场景的核心在于这位学生要把作业给做了。  
简单点描述：这位学生告诉室友要做什么作业，并把自己的引用也给了室友。该室友得到作业，做完后直接引用该学生并调用其做作业的方法，完成代写作业的任务。  
**稍微复杂点描述：**  
该学生做作业的方法有两个入参，一个是作业题目（已知），一个是作业答案（未知）。  
室友为了帮助他写作业提供了一个方法，该方法有两个入参，一个是作业题目，一个是该学生的引用（解出答案得知道往哪写）。  
程序执行时，该学生只要调用室友的代写作业方法就行了。一旦室友得到答案，因为有该学生的引用，所以直接找到对应方法，帮助其完成作业。  
**再复杂点描述：**  
学生调用室友的替写作业方法，注册了题目和自己的引用。室友的替写作业方法被调用，则会根据题目完成作业后，再回调该同学写作业方法，完成作业。  
**再抽象点描述：**  
类A调用类B的方法b（传入相关信息），类B的方法在执行完后，会将结果写到（再回调）类A的方法a，完成动作。（其实方法a就是传说中的回调方法啦）  
**最抽象的描述：**  
调用，回调。

接口方式的回调方法

常常使用回调方法的同学可能会说，我从来也没见过直接把对象的引用写到第一次调用方法里面的。  
嗯，是的，下面就来填上述例子留下的“天坑”（实际项目中常见到）。

**问题：在调用方法中直接传对象引用进去有什么不好？**  
只说一点，只是让别人代写个方法，犯得着把自己全部暴露给别人吗。万一这个别人是竞争对手的接口咋办。这就是传说中的后面代码吗（/tx）。  
总之这样做是非常不安全的。

因此，最常规的《调用，回调》实现，是（你已经猜到了）使用接口作为引用（说的不严谨）传入调用的方法里面。

我承认，怎么将思路跳转到使用接口的花了我好长时间。  
我们再看RoomMate类的getAnswer方法。

public class RoomMate {

public void getAnswer(String homework, Student student) {

if("1+1=?".equals(homework)) {

student.doHomeWork(homework, "2");

} else {

student.doHomeWork(homework, "(空白)");

}

}}

关键在于，该方法的用途是来解决某学生提出的某个问题。答案是通过学生的doHomeWork方法回调传回去的。那假设有个工人也有问题，这位室友该怎么解决呢。再开个方法，专门接收工人的引用作为传参？当然不用，只要你这个引用包含了doHomeWork()方法，那么不论你是工人、警察还是环卫工人，直接调用getAnswer()方法就能解决你提的问题。  
至此我们的思路达到了：所有的对象要有同一个方法，所以自热而然就引出了接口概念。只要这些对象都实现了某个接口就行了，这个接口的作用，仅仅是用来规定那个做作业的方法长什么样。这样工人实现了该接口，那么就有了默认继承的做作业方法。工人再把自己的引用抛给该室友的时候，这个室友就不需要改动任何代码，直接接触答案，完成任务了。

创建一个做作业的接口，专门规定，需要哪些东西（问题和答案）就能做作业.

public interface DoHomeWork {

void doHomeWork(String question, String answer);}

改动下中国好室友的解答方法。任意一个实现了DoHomeWork 接口的someone，都拥有doHomeWork(String question,String answer)的方法。这个方法就是上面已经提到的“回调方法”。someone先调用下好室友的getAnswer()方法，把问题和自己传进来（此为调用），好室友把问题解答出之后，调用默认提供的方法，写完作业。  
思考下，因为是以接口作为参数类型的约定，在普通对象upcast向上转型之后将只暴露接口描述的那个方法，别人获取到这个引用，也只能使用这个（回调）方法。至此，遗留的重大安全隐患重要解决。

*完整代码*

public class RoomMate {

public void getAnswer(String homework, DoHomeWork someone) {

if("1+1=?".equals(homework)) {

someone.doHomeWork(homework, "2");

} else {

someone.doHomeWork(homework, "(空白)");

}

}}

package org.futeng.designpattern.callback.test1;

public class Worker implements DoHomeWork {

@Override

public void doHomeWork(String question, String answer) {

System.out.println("作业本");

if(answer != null) {

System.out.println("作业："+question+" 答案："+ answer);

} else {

System.out.println("作业："+question+" 答案："+ "(空白)");

}

}

public static void main(String[] args) {

Worker worker = new Worker();

String question = "1+1=?";

new RoomMate().getAnswer(question, worker);

}}

*执行程序*

作业本

作业：1+1=? 答案：2

至此，调用+回调的文章是不是写完了呢。  
咳咳，还木有。大家喝点茶再忍耐下。（我都写一天了 - -）

常规使用之匿名内部类

作为平凡的屁民，实用主义是我们坚持的生存法则。  
所以凡事用不到的技术都可以不学，凡事学了却不用的技术等于白学。

我们之前已经定性，中国好室友RoomMate类拥有接受任何人任何问题挑战的潜质。  
自从好室友出名之后，有个不知道什么工作（类型）的人也来问问题。反正只要实现了回调接口，好室友都能调用你默认继承的回调方法，那就放马过来吧。

package org.futeng.designpattern.callback.test1;

public class RoomMate {

public void getAnswer(String homework, DoHomeWork someone) {

if("1+1=?".equals(homework)) {

someone.doHomeWork(homework, "2");

} else {

someone.doHomeWork(homework, "(空白)");

}

}

public static void main(String[] args) {

RoomMate roomMate = new RoomMate();

roomMate.getAnswer("1+1=?", new DoHomeWork() {

@Override

public void doHomeWork(String question, String answer) {

System.out.println("问题："+question+" 答案："+answer);

}

});

}}

看到稍显奇怪的roomMate.getAnswer("1+1=?", new DoHomeWork() {的哪一行，其实这里new的是DoHomeWork接口的一个匿名内部类。这里我想大家应该自己动脑想想，调用+反调，这个过程是怎么实现的了。  
至于是否使用匿名内部类是根据具体使用场景决定的。普通类不够直接，匿名内部类的语法似乎也不够友好。

开源工具中对回调方法的使用

上述匿名内部类的示例才是开源工具中常见到的使用方式。

调用roomMate解答问题的方法（传进去自己的引用），roomMate解决问题，回调引用里面包含的回调方法，完成动作。  
roomMate就是个工具类，“调用”这个方法你传进去两个参数（更多也是一个道理），什么问题，问题解决了放哪，就行了。该“调用”方法根据问题找到答案，就将结果写到你指定放置的位置（作为回调方法的入参）。

试想下，“中国好室友”接收的问题是SQL语句，接收的放置位置是我的引用。你解决问题（执行完SQL），将答案（SQL的反馈结果）直接写入我的回调方法里面。回调方法里面可能包括一个个的字段赋值。但是在调用层面隐藏了很多细节的处理。这是回调方法的一个小小的优势。再换句话说，不需要拿到执行完SQL之后的返回结果一个个来赋值。

*SQL的例子*

public static List<Person> queryPerson() {

QueryRunner queryRunner = new QueryRunner(DataSourceSupport.getDataSource());

return queryRunner.query(" select t.name, t.age from person t ", new ResultSetHandler<List<Person>>(){

List list = new ArrayList<Person>();

@Override

public List<Person> handle(ResultSet rs) throws SQLException {

while(rs.next()) {

Person person = new Person();

person.setName(rs.getString(0));

person.setAge(rs.getInt(1));

list.add(person);

}

return list;

}

});

}

回调方法的优势

回调方法最大的优势在于，异步回调，这样是其最被广为使用的原因。  
下面将沿用“中国好室友” 来对回调方法做异步实现。

回调接口不用变

public interface DoHomeWork {

void doHomeWork(String question, String answer);}

为了体现异步的意思，我们给好室友设置了个较难的问题，希望好室友能多好点时间思考。

Student student = new Student();String homework = "当x趋向于0，sin(x)/x =?";

#给学生新建个ask方法，该方法中另开一个线程，来等待回调方法的结果反馈。student.ask(homework, new RoomMate());

#ask方法如下public void ask(final String homework, final RoomMate roomMate) {

new Thread(new Runnable() {

@Override

public void run() {

roomMate.getAnswer(homework, Student.this);

}

}).start();

goHome();

}#新开的线程纯粹用来等待好室友来写完作用。由于在好室友类中设置了3秒的等待时间，所以可以看到goHome方法将先执行。#意味着该学生在告知好室友做作用后，就可以做自己的事情去了，不需要同步阻塞去等待结果。#一旦好室友完成作用，写入作业本，该现场也就结束运行了。

*完整代码*

public class Student implements DoHomeWork{

@Overridepublic void doHomeWork(String question, String answer) {

System.out.println("作业本");

if(answer != null) {

System.out.println("作业："+question+" 答案："+ answer);

} else {

System.out.println("作业："+question+" 答案："+ "(空白)");

}}

public void ask(final String homework, final RoomMate roomMate) {

new Thread(new Runnable() {

@Override

public void run() {

roomMate.getAnswer(homework, Student.this);

}

}).start();

goHome();}

public void goHome(){

System.out.println("我回家了……好室友，帮我写下作业。");}

public static void main(String[] args) {

Student student = new Student();

String homework = "当x趋向于0，sin(x)/x =?";

student.ask(homework, new RoomMate());

}}

public class RoomMate {

public void getAnswer(String homework, DoHomeWork someone) {

if ("1+1=?".equals(homework)) {

someone.doHomeWork(homework, "2");

} else if("当x趋向于0，sin(x)/x =?".equals(homework)) {

System.out.print("思考：");

for(int i=1; i<=3; i++) {

System.out.print(i+"秒 ");

try {

TimeUnit.SECONDS.sleep(1);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println();

someone.doHomeWork(homework, "1");

} else {

someone.doHomeWork(homework, "(空白)");

}

}

}

完结

至此回调方法的介绍告一段落。