

《JAVA与模式》之命令模式

在阎宏博士的《JAVA与模式》一书中开头是这样描述命令（Command）模式的：

命令模式属于对象的行为模式。命令模式又称为行动(Action)模式或交易(Transaction)模式。

命令模式把一个请求或者操作封装到一个对象中。命令模式允许系统使用不同的请求把客户端参数化，对请求排队或者记录请求日志，可以提供命令的撤销和恢复功能。

命令模式的结构

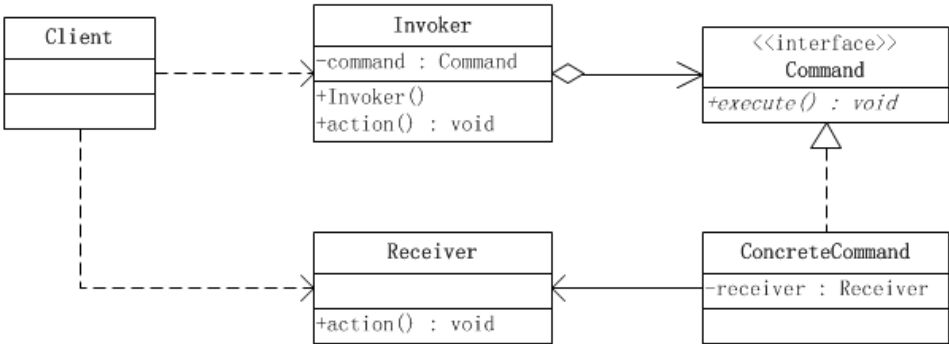
命令模式是对命令的封装。命令模式把发出命令的责任和执行命令的责任分割开，委派给不同的对象。

每一个命令都是一个操作：请求的一方发出请求要求执行一个操作；接收的一方收到请求，并执行操作。命令模式允许请求的一方和接收的一方独立开来，使得请求的一方不必知道接收请求的一方的接口，更不必知道请求是怎么被接收，以及操作是否被执行、何时被执行，以及是怎么被执行的。

命令允许请求的一方和接收请求的一方能够独立演化，从而具有以下优点：

- (1) 命令模式使新的命令很容易地被加入到系统里。
- (2) 允许接收请求的一方决定是否要否决请求。
- (3) 能较容易地设计一个命令队列。
- (4) 可以容易地实现对请求的撤销和恢复。
- (5) 在需要的情况下，可以较容易地将命令记入日志。

下面以一个示意性的系统，说明命令模式的结构。



命令模式涉及到五个角色，它们分别是：

- 客户端(Client)角色：创建一个具体命令(ConcreteCommand)对象并确定其接收者。
- 命令(Command)角色：声明了一个给所有具体命令类的抽象接口。

导航

[博客园](#)  
[首页](#)  
[新随笔](#)  
[联系](#)  
[订阅](#) [XML](#)  
[管理](#)

统计

随笔 - 28  
文章 - 0  
评论 - 301  
引用 - 0

公告

昵称：[java\\_my\\_life](#)  
园龄：5年11个月  
粉丝：1170  
关注：0  
[+加关注](#)

搜索

找找看

谷歌搜索

常用链接

[我的随笔](#)  
[我的评论](#)  
[我的参与](#)  
[最新评论](#)  
[我的标签](#)

我的标签

[设计模式\(1\)](#)

[随笔档案\(28\)](#)

[2012年8月 \(3\)](#)  
[2012年6月 \(6\)](#)  
[2012年5月 \(8\)](#)  
[2012年4月 \(7\)](#)  
[2012年3月 \(4\)](#)

[积分与排名](#)

积分 - 95140  
排名 - 2321

最新评论

[1. Re: 《JAVA与模式》之适配器模式](#)  
博主讲的很透彻，看之前还对

- 具体命令(ConcreteCommand)角色：定义一个接收者和行为之间的弱耦合；实现execute()方法，负责调用接收者的相应操作。execute()方法通常叫做执行方法。
- 请求者(Invoker)角色：负责调用命令对象执行请求，相关的方法叫做行动方法。
- 接收者(Receiver)角色：负责具体实施和执行一个请求。任何一个类都可以成为接收者，实施和执行请求的方法叫做行动方法。

源代码

接收者角色类

```
public class Receiver {  
  
    /**  
     * 真正执行命令相应的操作  
     */  
  
    public void action(){  
  
        System.out.println("执行操作");  
  
    }  
  
}
```

抽象命令角色类

```
public interface Command {  
  
    /**  
     * 执行方法  
     */  
  
    void execute();  
  
}
```

具体命令角色类

```
public class ConcreteCommand implements Command {  
  
    //持有相应的接收者对象  
  
    private Receiver receiver = null;  
  
    /**  
     * 构造方法  
     */  
  
    public ConcreteCommand(Receiver receiver){  
  
        this.receiver = receiver;  
  
    }  
  
    @Override
```

适配器模式一知半解，看了之后就就很明白了，谢谢博主。  
--、soul。小叶子  
2. Re: 《JAVA与模式》之工厂方法模式  
很不错，正在学习中  
--realshow  
3. Re: 《JAVA与模式》之装饰模式  
IO类图那块感觉不是非常直观，博主看看这个图如何？

讲解鞭辟入里，非常感谢  
--北门煎饼东门串川  
4. Re: 《JAVA与模式》之单例模式  
最近在看设计模式，讲道理：非常棒！有个小小的疑问，static 放在 volatile之后还是之前呢？  
--Fighting4Family  
5. Re: 《JAVA与模式》之策略模式  
调调的  
--持之以恒，贵在坚持

阅读排行榜

1. 《JAVA与模式》之观察者模式(60074)
2. 《JAVA与模式》之适配器模式(54691)
3. 《JAVA与模式》之责任链模式(43878)
4. 《JAVA与模式》之装饰模式(41248)
5. 《JAVA与模式》之模板方法模式(38378)

评论排行榜

1. JAVA虚拟机体系结构(19)
2. 《JAVA与模式》之抽象工厂模式(18)
3. 《JAVA与模式》之适配器模式(15)
4. 《JAVA与模式》之装饰模式(14)
5. 《JAVA与模式》之策略模式(13)

推荐排行榜

1. 《JAVA与模式》之装饰模式(28)
2. JAVA虚拟机体系结构(26)
3. 《JAVA与模式》之策略模式(22)
4. 《JAVA与模式》之适配器模式(21)
5. 《JAVA与模式》之观察者模式(19)

```
public void execute() {  
    //通常会转调接收者对象的相应方法，让接收者来真正执行功能  
    receiver.action();  
}  
  
}
```



#### 请求者角色类

```
public class Invoker {  
    /**  
     * 持有命令对象  
     */  
    private Command command = null;  
    /**  
     * 构造方法  
     */  
    public Invoker(Command command) {  
        this.command = command;  
    }  
    /**  
     * 行动方法  
     */  
    public void action() {  
        command.execute();  
    }  
}
```



#### 客户端角色类

```
public class Client {  
  
    public static void main(String[] args) {  
        //创建接收者  
        Receiver receiver = new Receiver();  
        //创建命令对象，设定它的接收者  
        Command command = new ConcreteCommand(receiver);  
        //创建请求者，把命令对象设置进去  
        Invoker invoker = new Invoker(command);  
    }  
}
```



```
//执行方法
invoker.action();

}

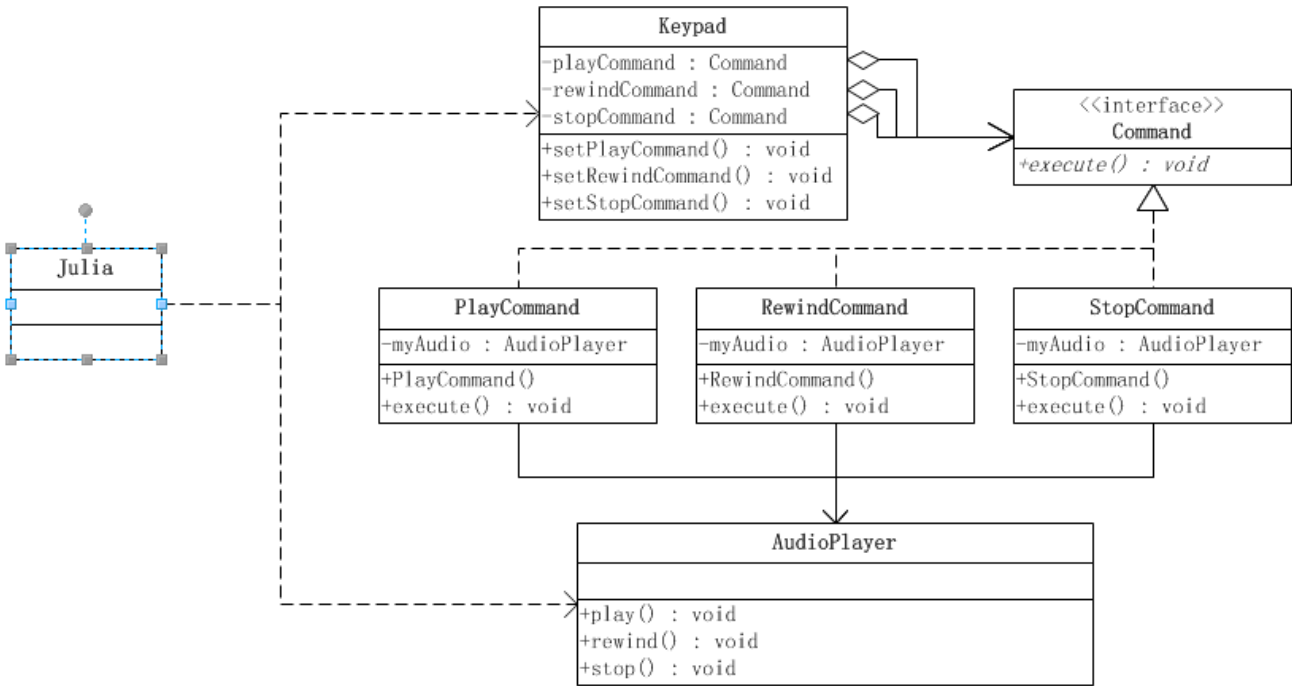
}
```



## AudioPlayer系统

小女孩茱丽(Julia)有一个盒式录音机，此录音机有播音(Play)、倒带(Rewind)和停止(Stop)功能，录音机的键盘便是请求者(Invoker)角色；茱丽(Julia)是客户端角色，而录音机便是接收者角色。Command类扮演抽象命令角色，而PlayCommand、StopCommand和RewindCommand便是具体命令类。茱丽(Julia)不需要知道播音(play)、倒带(rewind)和停止(stop)功能是怎么具体执行的，这些命令执行的细节全都由键盘(Keypad)具体实施。茱丽(Julia)只需要在键盘上按下相应的键便可以了。

录音机是典型的命令模式。录音机按键把客户端与录音机的操作细节分割开来。



### 源代码

接收者角色，由录音机类扮演

```
public class AudioPlayer {

    public void play() {

        System.out.println("播放...");

    }

}
```



```
public void rewind() {  
    System.out.println("倒带...");  
}  
  
public void stop() {  
    System.out.println("停止...");  
}  
}
```



### 抽象命令角色类

```
public interface Command {  
    /**  
     * 执行方法  
     */  
    public void execute();  
}
```



### 具体命令角色类

```
public class PlayCommand implements Command {  
  
    private AudioPlayer myAudio;  
  
    public PlayCommand(AudioPlayer audioPlayer) {  
        myAudio = audioPlayer;  
    }  
    /**  
     * 执行方法  
     */  
    @Override  
    public void execute() {  
        myAudio.play();  
    }  
}
```



```
public class RewindCommand implements Command {

    private AudioPlayer myAudio;

    public RewindCommand(AudioPlayer audioPlayer){

        myAudio = audioPlayer;

    }

    @Override

    public void execute() {

        myAudio.rewind();

    }

}
```



```
public class StopCommand implements Command {

    private AudioPlayer myAudio;

    public StopCommand(AudioPlayer audioPlayer){

        myAudio = audioPlayer;

    }

    @Override

    public void execute() {

        myAudio.stop();

    }

}
```



请求者角色，由键盘类扮演

```
public class Keypad {

    private Command playCommand;

    private Command rewindCommand;

    private Command stopCommand;

    public void setPlayCommand(Command playCommand) {

        this.playCommand = playCommand;

    }

    public void setRewindCommand(Command rewindCommand) {

        this.rewindCommand = rewindCommand;

    }

}
```



```
public void setStopCommand(Command stopCommand) {  
    this.stopCommand = stopCommand;  
}  
  
/**  
 * 执行播放方法  
 */  
public void play(){  
    playCommand.execute();  
}  
  
/**  
 * 执行倒带方法  
 */  
public void rewind(){  
    rewindCommand.execute();  
}  
  
/**  
 * 执行播放方法  
 */  
public void stop(){  
    stopCommand.execute();  
}  
}
```



客户端角色，由茱丽小女孩扮演



```
public class Julia {  
    public static void main(String[] args){  
        //创建接收者对象  
        AudioPlayer audioPlayer = new AudioPlayer();  
        //创建命令对象  
        Command playCommand = new PlayCommand(audioPlayer);  
        Command rewindCommand = new RewindCommand(audioPlayer);  
        Command stopCommand = new StopCommand(audioPlayer);  
        //创建请求者对象  
        Keypad keypad = new Keypad();  
        keypad.setPlayCommand(playCommand);  
        keypad.setRewindCommand(rewindCommand);  
        keypad.setStopCommand(stopCommand);  
        //测试  
        keypad.play();  
        keypad.rewind();  
        keypad.stop();  
    }  
}
```

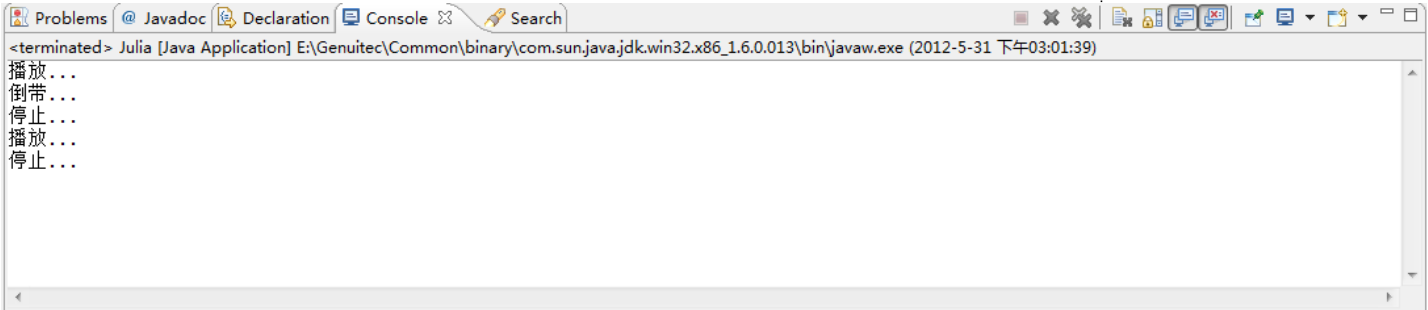
```
        keypad.play();

        keypad.stop();

    }

}
```

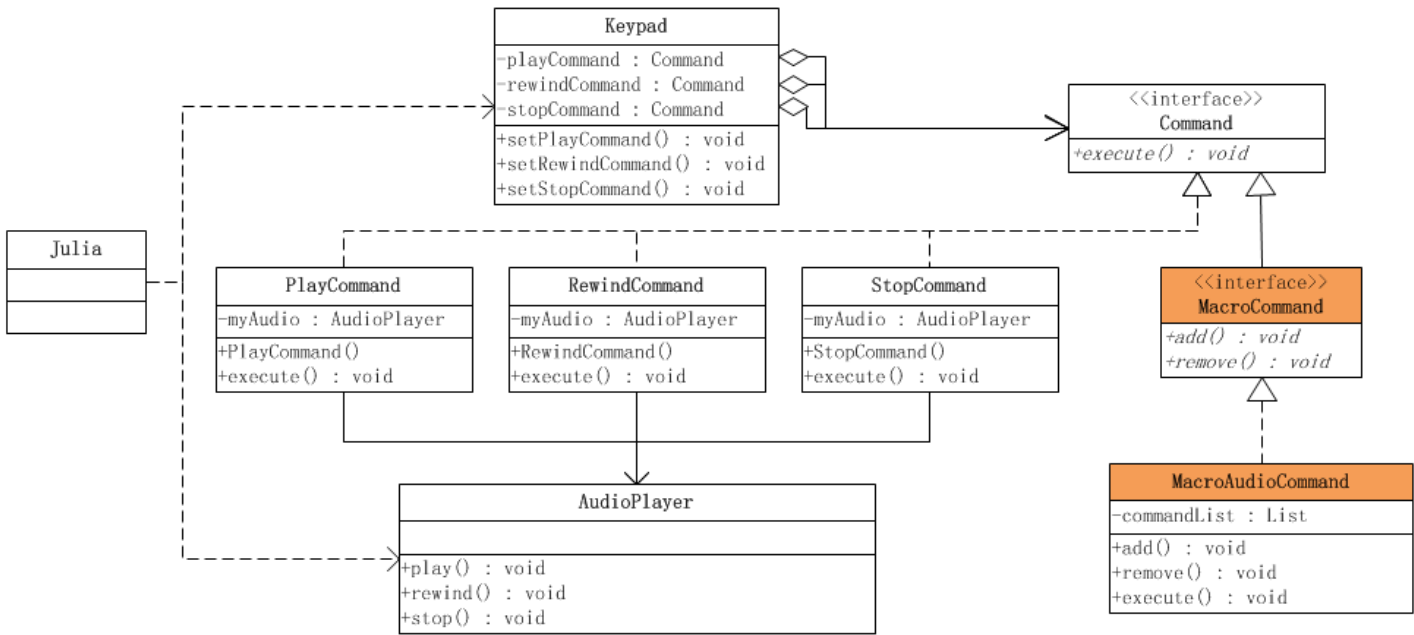
运行结果如下：



# 宏命令

所谓宏命令简单点说就是包含多个命令的命令，是一个命令的组合。

设想茱丽的录音机有一个记录功能，可以把一个一个的命令记录下来，再在任何需要的时候重新把这些记录下来的命令一次性执行，这就是所谓的宏命令集功能。因此，茱丽的录音机系统现在有四个键，分别为播音、倒带、停止和宏命令功能。此时系统的设计与前面的设计相比有所增强，主要体现在Julia类现在有了一个新方法，用以操作宏命令键。



## 源代码

系统需要一个代表宏命令的接口，以定义出具体宏命令所需要的接口。

```
public interface MacroCommand extends Command {

    /**
```



```
    * 宏命令聚集的管理方法
    * 可以添加一个成员命令
    */

    public void add(Command cmd);

    /**
    * 宏命令聚集的管理方法
    * 可以删除一个成员命令
    */

    public void remove(Command cmd);
}
```



具体的宏命令MacroAudioCommand类负责把个别的命令合成宏命令。

```
public class MacroAudioCommand implements MacroCommand {

    private List<Command> commandList = new ArrayList<Command>();

    /**
    * 宏命令聚集管理方法
    */

    @Override
    public void add(Command cmd) {
        commandList.add(cmd);
    }

    /**
    * 宏命令聚集管理方法
    */

    @Override
    public void remove(Command cmd) {
        commandList.remove(cmd);
    }

    /**
    * 执行方法
    */

    @Override
    public void execute() {
        for(Command cmd : commandList){
            cmd.execute();
        }
    }
}
```



## 客户端类Julia

```
public class Julia {

    public static void main(String[] args) {

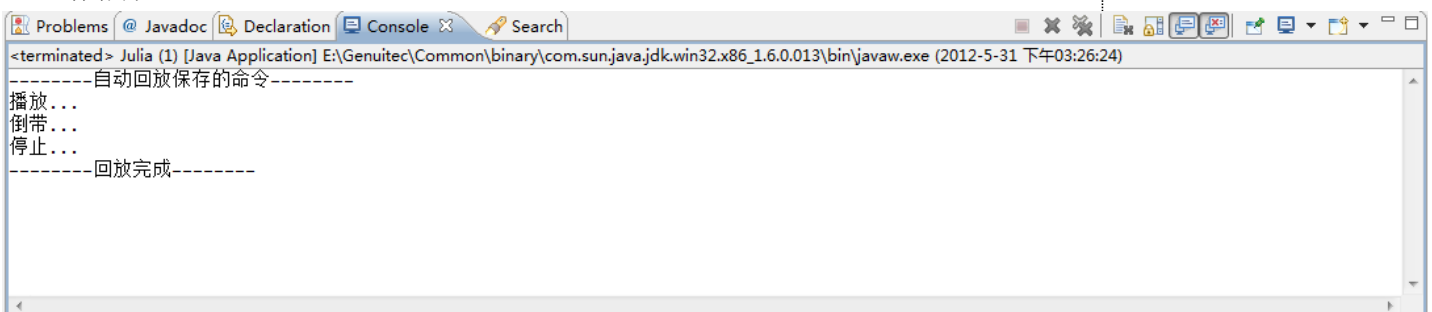
        //创建接收者对象
        AudioPlayer audioPlayer = new AudioPlayer();

        //创建命令对象
        Command playCommand = new PlayCommand(audioPlayer);
        Command rewindCommand = new RewindCommand(audioPlayer);
        Command stopCommand = new StopCommand(audioPlayer);

        MacroCommand marco = new MacroAudioCommand();

        marco.add(playCommand);
        marco.add(rewindCommand);
        marco.add(stopCommand);
        marco.execute();
    }
}
```

运行结果如下：



## 命令模式的优点

- 更松散的耦合

命令模式使得发起命令的对象——客户端，和具体实现命令的对象——接收者对象完全解耦，也就是说发起命令的对象完全不知道具体实现对象是谁，也不知道如何实现。

- 更动态的控制

命令模式把请求封装起来，可以动态地对它进行参数化、队列化和日志化等操作，从而使得系统更灵活。

- 很自然的复合命令

命令模式中的命令对象能够很容易地组合成复合命令，也就是宏命令，从而使系统操作更简单，功能更强大。

● 更好的扩展性

由于发起命令的对象和具体的实现完全解耦，因此扩展新的命令就很容易，只需要实现新的命令对象，然后在装配的时候，把具体的实现对象设置到命令对象中，然后就可以使用这个命令对象，已有的实现完全不用变化。

好文要顶

关注我

收藏该文



java\_my\_life  
关注 - 0  
粉丝 - 1170

+加关注

« 上一篇: [《JAVA与模式》之责任链模式](#)  
» 下一篇: [《JAVA与模式》之备忘录模式](#)

100

posted on 2012-06-01 10:13 [java\\_my\\_life](#) 阅读(16723) 评论(5) [编辑](#) [收藏](#)

评论

#1楼 2012-06-04 00:23 码魂

不错,但...简单了点,还有...  
执行的时候居然是for循环来调用的...  
扩展成最长用的应该是  
add("commandName",command);  
execution("commandName",args);

支持(0) 反对(0)

#2楼 2013-03-18 16:15 掸尘

由于项目应用设计模式较少，自己研究设计模式不知道为什么这么写，很难共鸣！

支持(2) 反对(0)

#3楼 2014-12-11 20:12 带妳心菲

写的很不错，简单易懂！谢谢！

支持(1) 反对(0)

#4楼 2016-06-22 10:34 chiansxjdy

你好，我想问下，这个如果想要得到命令的执行结果怎么弄啊？

支持(0) 反对(0)

#5楼 2016-08-31 10:11 码农先森

很不错，实例通俗易懂，谢谢楼主。

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

- 【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库
- 【活动】优达学城正式发布“无人驾驶车工程师”课程

【推荐】移动直播百强八成都在用融云即时通讯云

【推荐】别再闷头写代码！找对工具，事半功倍，全能开发工具包用起来

【推荐】网易这群程序员1年撸了10万+IM开发者，一天让APP接入一个微信



#### 最新IT新闻:

- 用于Microsoft Azure BaaS的Bletchley项目第一版发布
- 王川——雷军之臂膀，小米之变量
- 齐家、土巴兔、爱空间老板倾述，互联网家装痛到心房、“碍”到深处
- 当华为的奋斗者也输给炒房者，中国实体经济拿什么对抗这个时代？
- Prisma推出视频版的艺术滤镜，自制GIF的时代要来啦！

» 更多新闻...



#### 最新知识库文章:

- 陈皓：什么是工程师文化？
- 没那么难，谈CSS的设计模式
- 程序猿媳妇儿注意事项
- 可是姑娘，你为什么要编程呢？
- 知其所以然（以算法学习为例）

» 更多知识库文章...

Powered by:

博客园

Copyright © java\_my\_life