

49 | 桥接模式：如何实现支持不同类型和渠道的消息推送系统？

2020-02-24 王争

设计模式之美

[进入课程 >](#)



讲述：冯永吉

时长 09:17 大小 7.44M



上一节课我们学习了第一种结构型模式：代理模式。它在不改变原始类（或者叫被代理类）代码的情况下，通过引入代理类来给原始类附加功能。代理模式在平时的开发经常被用到，常用在业务系统中开发一些非功能性需求，比如：监控、统计、鉴权、限流、事务、幂等、日志。

今天，我们再学习另外一种结构型模式：桥接模式。桥接模式的代码实现非常简单，但是理解起来稍微有点难度，并且应用场景也比较局限，所以，相当于代理模式来说，桥接模式在实际的项目中并没有那么常用，你只需要简单了解，见到能认识就可以，并不是我们重点。☆

话不多说，让我们正式开始今天的学习吧！

桥接模式的原理解析

桥接模式，也叫作**桥梁模式**，英文是 **Bridge Design Pattern**。这个模式可以说是 23 种设计模式中最难理解的模式之一了。我查阅了比较多的书籍和资料之后发现，对于这个模式有两种不同的理解方式。

当然，这其中“最纯正”的理解方式，当属 GoF 的《设计模式》一书中对桥接模式的定义。毕竟，这 23 种经典的设计模式，最初就是由这本书总结出来的。在 GoF 的《设计模式》一书中，桥接模式是这么定义的：“Decouple an abstraction from its implementation so that the two can vary independently。”翻译成中文就是：“将抽象和实现解耦，让它们可以独立变化。”

关于桥接模式，很多书籍、资料中，还有另外一种理解方式：“一个类存在两个（或多个）独立变化的维度，我们通过组合的方式，让这两个（或多个）维度可以独立进行扩展。”通过组合关系来替代继承关系，避免继承层次的指数级爆炸。这种理解方式非常类似于，我们之前讲过的“组合优于继承”设计原则，所以，这里我就不多解释了。我们重点看下 GoF 的理解方式。

GoF 给出的定义非常的简短，单凭这一句话，估计没几个人能看懂是什么意思。所以，我们通过 JDBC 驱动的例子来解释一下。JDBC 驱动是桥接模式的经典应用。我们先来看一下，如何利用 JDBC 驱动来查询数据库。具体的代码如下所示：

 复制代码

```
1 Class.forName("com.mysql.jdbc.Driver");//加载及注册JDBC驱动程序
2 String url = "jdbc:mysql://localhost:3306/sample_db?user=root&password=your_pa
3 Connection con = DriverManager.getConnection(url);
4 Statement stmt = con.createStatement();
5 String query = "select * from test";
6 ResultSet rs=stmt.executeQuery(query);
7 while(rs.next()) {
8     rs.getString(1);
9     rs.getInt(2);
10 }
```

如果我们想要把 MySQL 数据库换成 Oracle 数据库，只需要把第一行代码中的 `com.mysql.jdbc.Driver` 换成 `oracle.jdbc.driver.OracleDriver` 就可以了。当然，也有更灵活的实现方式，我们可以把需要加载的 Driver 类写到配置文件中，当程序启动的时候，自

动从配置文件中加载，这样在切换数据库的时候，我们都不需要修改代码，只需要修改配置文件就可以了。

不管是改代码还是改配置，在项目中，从一个数据库切换到另一种数据库，都只需要改动很少的代码，或者完全不需要改动代码，那如此优雅的数据库切换是如何实现的呢？

源码之下无秘密。要弄清楚这个问题，我们先从 `com.mysql.jdbc.Driver` 这个类的代码看起。我摘抄了部分相关代码，放到了这里，你可以看一下。

 复制代码

```
1 package com.mysql.jdbc;
2 import java.sql.SQLException;
3
4 public class Driver extends NonRegisteringDriver implements java.sql.Driver {
5     static {
6         try {
7             java.sql.DriverManager.registerDriver(new Driver());
8         } catch (SQLException E) {
9             throw new RuntimeException("Can't register driver!");
10        }
11    }
12
13    /**
14     * Construct a new driver and register it with DriverManager
15     * @throws SQLException if a database error occurs.
16     */
17    public Driver() throws SQLException {
18        // Required for Class.forName().newInstance()
19    }
20 }
```

结合 `com.mysql.jdbc.Driver` 的代码实现，我们可以发现，当执行 `Class.forName(“com.mysql.jdbc.Driver”)` 这条语句的时候，实际上是做了两件事情。第一件事情是要求 JVM 查找并加载指定的 `Driver` 类，第二件事情是执行该类的静态代码，也就是将 `MySQL Driver` 注册到 `DriverManager` 类中。

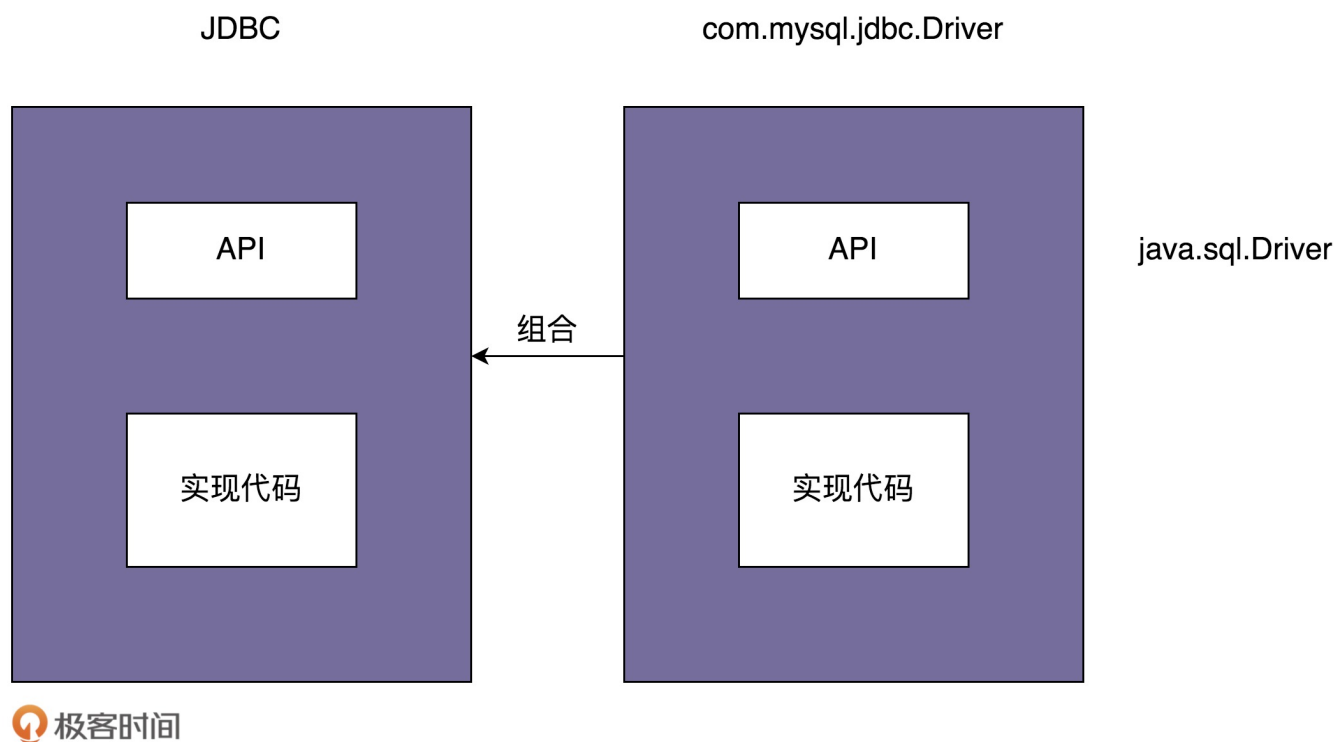
现在，我们再来看一下，`DriverManager` 类是干什么用的。具体的代码如下所示。当我们把具体的 `Driver` 实现类（比如，`com.mysql.jdbc.Driver`）注册到 `DriverManager` 之后，后续所有对 JDBC 接口的调用，都会委派到对具体的 `Driver` 实现类来执行。而 `Driver` 实现类都实现了相同的接口（`java.sql.Driver`），这也是可以灵活切换 `Driver` 的原因。

```
1 public class DriverManager {
2     private final static CopyOnWriteArrayList<DriverInfo> registeredDrivers = new
3
4     //...
5     static {
6         loadInitialDrivers();
7         println("JDBC DriverManager initialized");
8     }
9     //...
10
11     public static synchronized void registerDriver(java.sql.Driver driver) throws
12         if (driver != null) {
13             registeredDrivers.addIfAbsent(new DriverInfo(driver));
14         } else {
15             throw new NullPointerException();
16         }
17     }
18
19     public static Connection getConnection(String url, String user, String password,
20         java.util.Properties info = new java.util.Properties();
21         if (user != null) {
22             info.put("user", user);
23         }
24         if (password != null) {
25             info.put("password", password);
26         }
27         return (getConnection(url, info, Reflection.getCallerClass()));
28     }
29     //...
30 }
```

桥接模式的定义是“将抽象和实现解耦，让它们可以独立变化”。那弄懂定义中“抽象”和“实现”两个概念，就是理解桥接模式的关键。那在 JDBC 这个例子中，什么是“抽象”？什么是“实现”呢？

实际上，JDBC 本身就相当于“抽象”。注意，这里所说的“抽象”，指的并非“抽象类”或“接口”，而是跟具体的数据库无关的、被抽象出来的一套“类库”。具体的 Driver（比如，com.mysql.jdbc.Driver）就相当于“实现”。注意，这里所说的“实现”，也并非指“接口的实现类”，而是跟具体数据库相关的一套“类库”。JDBC 和 Driver 独立开发，通过对象之间的组合关系，组装在一起。JDBC 的所有逻辑操作，最终都委托给 Driver 来执行。

我画了一张图帮助你理解，你可以结合着我刚才的讲解一块看。



桥接模式的应用举例

在 [第 16 节](#) 中，我们讲过一个 API 接口监控告警的例子：根据不同的告警规则，触发不同类型的告警。告警支持多种通知渠道，包括：邮件、短信、微信、自动语音电话。通知的紧急程度有多种类型，包括：SEVERE（严重）、URGENCY（紧急）、NORMAL（普通）、TRIVIAL（无关紧要）。不同的紧急程度对应不同的通知渠道。比如，SERVE（严重）级别的消息会通过“自动语音电话”告知相关人员。

在当时的代码实现中，关于发送告警信息那部分代码，我们只给出了粗略的设计，现在我们来一块实现一下。我们先来看最简单、最直接的一种实现方式。代码如下所示：

```
1 public enum NotificationEmergencyLevel {
2     SEVERE, URGENCY, NORMAL, TRIVIAL
3 }
4
5 public class Notification {
6     private List<String> emailAddresses;
7     private List<String> telephones;
8     private List<String> wechatIds;
9
10    public Notification() {}
```

复制代码


```

11     public void setEmailAddress(List<String> emailAddress) {
12         this.emailAddresses = emailAddress;
13     }
14
15     public void setTelephones(List<String> telephones) {
16         this.telephones = telephones;
17     }
18
19     public void setWechatIds(List<String> wechatIds) {
20         this.wechatIds = wechatIds;
21     }
22
23     public void notify(NotificationEmergencyLevel level, String message) {
24         if (level.equals(NotificationEmergencyLevel.SEVERE)) {
25             //...自动语音电话
26         } else if (level.equals(NotificationEmergencyLevel.URGENCY)) {
27             //...发微信
28         } else if (level.equals(NotificationEmergencyLevel.NORMAL)) {
29             //...发邮件
30         } else if (level.equals(NotificationEmergencyLevel.TRIVIAL)) {
31             //...发邮件
32         }
33     }
34 }
35
36 //在API监控告警的例子中，我们如下方式来使用Notification类：
37 public class ErrorAlertHandler extends AlertHandler {
38     public ErrorAlertHandler(AlertRule rule, Notification notification){
39         super(rule, notification);
40     }
41
42
43     @Override
44     public void check(ApiStatInfo apiStatInfo) {
45         if (apiStatInfo.getErrorCount() > rule.getMatchedRule(apiStatInfo.getApi()))
46             notification.notify(NotificationEmergencyLevel.SEVERE, "...");
47     }
48 }
49 }
50

```

Notification 类的代码实现有一个最明显的问题，那就是有很多 if-else 分支逻辑。实际上，如果每个分支中的代码都不复杂，后期也没有无限膨胀的可能（增加更多 if-else 分支判断），那这样的设计问题并不大，没必要非得一定要摒弃 if-else 分支逻辑。

不过，Notification 的代码显然不符合这个条件。因为每个 if-else 分支中的代码逻辑都比较复杂，发送通知的所有逻辑都扎堆在 Notification 类中。我们知道，类的代码越多，就

越难读懂，越难修改，维护的成本也就越高。很多设计模式都是试图将庞大的类拆分成更细小的类，然后再通过某种更合理的结构组装在一起。

针对 Notification 的代码，我们将不同渠道的发送逻辑剥离出来，形成独立的消息发送类（MsgSender 相关类）。其中，Notification 类相当于抽象，MsgSender 类相当于实现，两者可以独立开发，通过组合关系（也就是桥梁）任意组合在一起。所谓任意组合的意思就是，不同紧急程度的消息和发送渠道之间的对应关系，不是在代码中固定写死的，我们可以动态地去指定（比如，通过读取配置来获取对应关系）。

按照这个设计思路，我们对代码进行重构。重构之后的代码如下所示：

 复制代码

```
1 public interface MsgSender {
2     void send(String message);
3 }
4
5 public class TelephoneMsgSender implements MsgSender {
6     private List<String> telephones;
7
8     public TelephoneMsgSender(List<String> telephones) {
9         this.telephones = telephones;
10    }
11
12    @Override
13    public void send(String message) {
14        //...
15    }
16
17 }
18
19 public class EmailMsgSender implements MsgSender {
20     // 与TelephoneMsgSender代码结构类似，所以省略...
21 }
22
23 public class WechatMsgSender implements MsgSender {
24     // 与TelephoneMsgSender代码结构类似，所以省略...
25 }
26
27 public abstract class Notification {
28     protected MsgSender msgSender;
29
30     public Notification(MsgSender msgSender) {
31         this.msgSender = msgSender;
32     }
33
34     public abstract void notify(String message);
```

```
35 }
36
37 public class SevereNotification extends Notification {
38     public SevereNotification(MsgSender msgSender) {
39         super(msgSender);
40     }
41
42     @Override
43     public void notify(String message) {
44         msgSender.send(message);
45     }
46 }
47
48 public class UrgencyNotification extends Notification {
49     // 与SevereNotification代码结构类似，所以省略...
50 }
51 public class NormalNotification extends Notification {
52     // 与SevereNotification代码结构类似，所以省略...
53 }
54 public class TrivialNotification extends Notification {
55     // 与SevereNotification代码结构类似，所以省略...
56 }
```

重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

总体上来讲，桥接模式的原理比较难理解，但代码实现相对简单。

对于这个模式有两种不同的理解方式。在 GoF 的《设计模式》一书中，桥接模式被定义为：“将抽象和实现解耦，让它们可以独立变化。”在其他资料和书籍中，还有另外一种更加简单的理解方式：“一个类存在两个（或多个）独立变化的维度，我们通过组合的方式，让这两个（或多个）维度可以独立进行扩展。”

对于第一种 GoF 的理解方式，弄懂定义中“抽象”和“实现”两个概念，是理解它的关键。定义中的“抽象”，指的并非“抽象类”或“接口”，而是被抽象出来的一套“类库”，它只包含骨架代码，真正的业务逻辑需要委派给定义中的“实现”来完成。而定义中的“实现”，也并非“接口的实现类”，而是的一套独立的“类库”。“抽象”和“实现”独立开发，通过对象之间的组合关系，组装在一起。

对于第二种理解方式，它非常类似我们之前讲过的“组合优于继承”设计原则，通过组合关系来替代继承关系，避免继承层次的指数级爆炸。

课堂讨论

在桥接模式的第二种理解方式的第一段代码实现中，Notification 类中的三个成员变量通过 set 方法来设置，但是这样的代码实现存在一个明显的问题，那就是 emailAddresses、telephones、wechatIds 中的数据有可能在 Notification 类外部被修改，那如何重构代码才能避免这种情况的发生呢？

 复制代码

```
1 public class Notification {
2     private List<String> emailAddresses;
3     private List<String> telephones;
4     private List<String> wechatIds;
5
6     public Notification() {}
7
8     public void setEmailAddress(List<String> emailAddress) {
9         this.emailAddresses = emailAddress;
10    }
11
12    public void setTelephones(List<String> telephones) {
13        this.telephones = telephones;
14    }
15
16    public void setWechatIds(List<String> wechatIds) {
17        this.wechatIds = wechatIds;
18    }
19    //...
20 }
```

欢迎留言和我分享你的思考和疑惑。如果有收获，也欢迎你把这篇文章分享给你的朋友。

关注极客时间服务号 每日学习签到

月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>



© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 48 | 代理模式：代理在RPC、缓存、监控等场景中的应用

下一篇 加餐二 | 设计模式、重构、编程规范等相关书籍推荐

精选留言 (23)

写留言



下雨天

2020-02-24

课后题：可以考虑使用建造者模式来重构！参见46讲中

建造者使用场景：

- 1.构造方法必填属性很多，需要检验
- 2.类属性之间有依赖关系或者约束条件...

展开 ▾



5



忆水寒

2020-02-24

参数不多的情况可以在构造函数初始化，如果参数较多 就可以使用建造者模式初始化。



5



小晏子

2020-02-24

“emailAddresses、telephones、wechatIds 中的数据有可能在 Notification 类外部被修改”的原因是对外暴露了修改接口set*，如果不想被修改那么就不要暴露set接口，这样的话初始化这些email，telephone和wechat的工作就放到构造函数里，用构造函数去初始化这些变量，这样初始化之后正常情况下外面没法修改。

展开 ▾



👍 2



陈尧东

2020-02-24

老师，有个疑问，重构后SevereNotification类依赖的都是接口MessageSender，没有依赖具体的实现，哪其它几个XxxNotification实现与其有何区别？

展开 ▾



👍 1



黄林晴

2020-02-24

打卡

展开 ▾



👍 1



每天晒白牙

2020-02-24

建造者模式

展开 ▾



👍 1



松花皮蛋me

2020-02-24

这个模式和策略模式的区别是？

展开 ▾



👍 1



test

2020-02-25

返回不可变对象

展开 ▾



👍



冰激凌的眼泪

2020-02-25

在桥接模式中，所谓抽象就是要干什么，所谓实现就是怎么去干，但是这俩是没有抽象与实现的意义。



小兵

2020-02-25

最简单的是加一个非空判断，其次是在构造函数中赋值，这里并没有很复杂的逻辑，通过构造函数就可以了，如果逻辑复杂，可以考虑建造者模式。

展开 ∨



sunnywhy

2020-02-25

课后题：首先去掉public的setter方法是必须的，然后可以用构造方法来初始化（没有太多参数的情况下，没有必要用builder模式）。另外，还要考虑是不是需要暴露getter方法，如果需要的话，可以返回一个不可修改的list（本文中list中的元素为String，所以不用担心单个元素被修改的情况，没有必要返回一个copy）

展开 ∨



平风造雨

2020-02-25

Builder模式隐藏目标对象的set方法，或者在set方法里深拷贝构建一个新的只读容器。



Frank

2020-02-24

1. 桥接模式是将“抽象”和“实现”解耦，让它们可以独立变化。重点在于理解“抽象”和“实现”。这个模式理解难度挺大的。
2. 对于第二种理解“一个类存在两个（或多个）独立变化的维度，我们通过组合的方式，让这两个（或多个）维度可以独立进行扩展”不能理解清楚，一个类的不同维度怎么独立扩展？是像例子中的Notification类，抽离出发送渠道类，Notification演变成抽象，只...

展开 ∨



李小四

2020-02-24

设计模式_49:

作业

使用Builder模式

感想...

展开 ▾



吴帆

2020-02-24

这个模式挺精妙的，不知道最终使用的时候是不是通过传入等级再反射来生成Notification的实现类，类似于JDBC的用法。这样确实可以完成抽象和实现的解耦，丢掉那些讨厌的if else语句



不似旧日

2020-02-24

简单的问题将复杂了， 建议看一看大话设计模式

展开 ▾



Jxin

2020-02-24

1.防止引用类型成员变量内的属性或元素被外部程序修改。可以在set时赋值 目标参数的深拷贝对象，以保证当前引用类型成员变量的作用范围尽在当前类（同时，对引用类型成员变量的所有修改操作，也应以对象方法的方式，限定在当前类的对象上）。

2.防止成员变量本身被修改。为成员变量加final标识（增强语意），如此一来，其赋值操...

展开 ▾



webmin

2020-02-24

```
public enum NotificationTargetAddressType {  
    EMAIL,  
    TELEPHONE,  
    WECHAT  
}...
```

展开 ▾





2020-02-24

课后题:我们项目有短信通知服务模块,我们使用的方式是每次调用短信服务时,请求体都需要携带发送的对象集合,短信模板,然后交给短信服务校验和发送

如果需要邮件通知,微信通知的话,我的想法是这样的:

将短信通知模块扩展成消息通知模块,将短信通知,邮件通知,微信通知等分别以不同的Sender封装起来,同时编写SMSService,EmailService来调用,同样,上层封装不同的接口开放出来...

展开 ∨



峰

2020-02-24

set 方法里拷贝一份值，而不是直接赋值。

展开 ∨

