原 一个示例让你明白适配器模式

2014年01月26日 00:08:28 张纪刚 阅读数：51232 标签： 设计模式 适配器 适配器模式 Adapter Design Pattern 更多

个人分类： 设计模式与架构

现实生活中的适配器

本文讨论适配器模式。适配器模式是23中设计模式之一，它的主要作用是在新接口和老接口之间进行适配。它非常像我们出国旅行时带的电源转换器。为了举这个例子，我还特意去京东上搜了一下电源转[illegible]确实看到了很多地方的标准不一样。我们国家的电器使用普通的扁平两项或三项插头，而去外国的话，使用的标准就不一样了，比如德国，使用的是德国标准，是两项圆头的插头。如果去德国旅游，那么我们使用的手机充电器插头无[illegible]德国的插排中去，那就意味着我们无法给手机充电。怎样解决这个问题呢？只要使用一个电源转化器就行了。如下图所示：

该适配器下面的插头符合德国标准，可以插到德国的插排中去，上面提供的接口符合国标，可以供我们的手机充电器使用。

实现电源适配器

下面我们使用代码来表述适配器模式：

代码中有两个接口，分别为德标接口和国标接口，分别命名为DBSocketInterface和GBSocketInterface，此外还有两个实现类，分别为德国插座和中国插座，分别为DBSocket和GBSocket。为了提供两套接口之间的适配，我们提供了一个适配器，叫做SocketAdapter。除此之外，还有一个客户端，比如是我们去德国旅游时住的一家宾馆，叫Hotel，在这个德国旅馆中使用德国接口。

德标接口：

复制

```
/**
 * 德标接口
 */
public interface DBSocketInterface {

        /**
         * 这个方法的名字叫做：使用两项圆头的插口供电
         * 本人英语就这个水平
         */
        void powerWithTwoRound();
}
```

德国插座实现德标接口

```
/**
 * 德国插座
 */
public class DBSocket implements DBSocketInterface{

        public void powerWithTwoRound(){
                System.out.println("使用两项圆头的插孔供电");
        }
}
```

德国旅馆是一个客户端，它里面有德标的接口，可以使用这个德标接口给手机充电：

```
/**
 * 德国宾馆
 */
public class Hotel {

        //旅馆中有一个德标的插口
        private DBSocketInterface dbSocket;

        public Hotel(){}

        public Hotel(DBSocketInterface dbSocket) {
                this.dbSocket = dbSocket;
        }

        public void setSocket (DBSocketInterface dbSocket){
                this.dbSocket = dbSocket;
```

```
19          //旅馆中有一个充电的功能20          public void charge(){
21
22                  //使用德标插口充电
23                  dbSocket.powerWithTwoRound();
24          }
25  }
```

现在写一段代码进行测试：

```
1   public class Test {
2
3           public static void main(String[] args) {
4
5                   //初始化一个德国插座对象，  用一个德标接口引用它
6                   DBSocketInterface dbSoket = new DBSocket();
7
8                   //创建一个旅馆对象
9                   Hotel hotel = new Hotel(dbSoket);
10
11                  //在旅馆中给手机充电
12                  hotel.charge();
13          }
14  }
```

运行程序，打印出以下结果： 使用两项圆头的插孔供电

现在我去德国旅游，带去的三项扁头的手机充电器。如果没有带电源适配器，我是不能充电的，因为不可能为了我一个旅客而为我更改墙上的插座，更不可能为我专门盖一座使用中国国标插座的宾馆。因为人家德国人一直这么使用，并且用的挺好，俗话说入乡随俗，我就要自己想办法来解决问题。对应到我们的代码中，也就是说，上面的Hotel类，DBSocket类，DBSocketInterface接口都是不可变的（由德国的客户提供），如果我想使用这一套API，那么只能自己写代码解决。

下面是国标接口和中国插座的代码。

国标接口：

```
1   /**
2    * 国标接口
3    */
4   public interface GBSocketInterface {
5
6           /**
7            * 这个方法的名字叫做：使用三项扁头的插口供电
8            * 本人英语就这个水平，从有道词典查得，  flat意思好像是：  扁的
9            */
10          void powerWithThreeFlat();
11  }
```

中国插座实现国标接口：

```
1   /**
2    * 中国插座
3    */
4   public class GBSocket implements GBSocketInterface{
5
6           @Override
7           public void powerWithThreeFlat() {
8                   System.out.println("使用三项扁头插孔供电");
9           }
10  }
```

可以认为这两个东西是我带到德国去的，目前他们还不能使用，因为接口不一样。那么我必须创建一个适配器，这个适配器必须满足以下条件：

1　必须符合德国标准的接口，否则的话还是没办法插到德国插座中；
2　在调用上面实现的德标接口进行充电时，提供一种机制，将这个调用转到对国标接口的调用 。

这就要求：
1 适配器必须实现原有的旧的接口
2 适配器对象中持有对新接口的引用，当调用旧接口时，将这个调用委托给实现新接口的对象来处理，也就是在适配器对象中组合一个新接口。

下面给出适配器类的实现：

```
1   public class SocketAdapter
2                   implements DBSocketInterface{    //实现旧接口
3
4           //组合新接口
5           private GBSocketInterface gbSocket;
6
7           /**
8            * 在创建适配器对象时，必须传入一个新街口的实现类
9            * @param gbSocket
10           */
11          public SocketAdapter(GBSocketInterface gbSocket) {
12                  this.gbSocket = gbSocket;
13          }
14
15
```

```
        /**
         * 将对就接口的调用适配到新接口
         */
        @Override
        public void powerWithTwoRound() {

                gbSocket.powerWithThreeFlat();
        }

}
```

这个适配器类满足了上面的两个要求。下面写一段测试代码来验证一下适配器能不能工作，我们按步骤一步步的写出代码，以清楚的说明适配器是如何使用的。

1　我去德国旅游，带去的充电器是国标的（可以将这里的GBSocket看成是充电器）

```
GBSocketInterface gbSocket = new GBSocket();
```

2　来到德国后， 找到一家德国宾馆住下 (这个宾馆还是上面代码中的宾馆，使用的依然是德国标准的插口)

```
Hotel hotel = new Hotel();
```

3　由于没法充电，我拿出随身带去的适配器，并且将我带来的充电器插在适配器的上端插孔中。这个上端插孔是符合国标的，我的充电器完全可以插进去。

```
SocketAdapter socketAdapter = new SocketAdapter(gbSocket);
```

4　再将适配器的下端插入宾馆里的插座上

```
hotel.setSocket(socketAdapter);
```

5　可以在宾馆中使用适配器进行充电了

```
hotel.charge();
```

上面的五个步骤就是适配器的使用过程，下面是完整的测试代码。

```
public class TestAdapter {

        public static void main(String[] args) {

                GBSocketInterface gbSocket = new GBSocket();

                Hotel hotel = new Hotel();

                SocketAdapter socketAdapter = new SocketAdapter(gbSocket);

                hotel.setSocket(socketAdapter);

                hotel.charge();
        }
}
```

运行上面的程序，打印出以下结果：
使用三项扁头插孔供电

这说明适配器起作用了，上一个实例中打印的是：使用两项圆头的插孔供电。 现在可以使用三项扁头插孔供电了。我们并没有改变宾馆中的德标插口，提供了一个适配器就能使用国标的插口充电。这就是适配器模式的魅力：不改变原有接口，却还能使用新接口的功能。

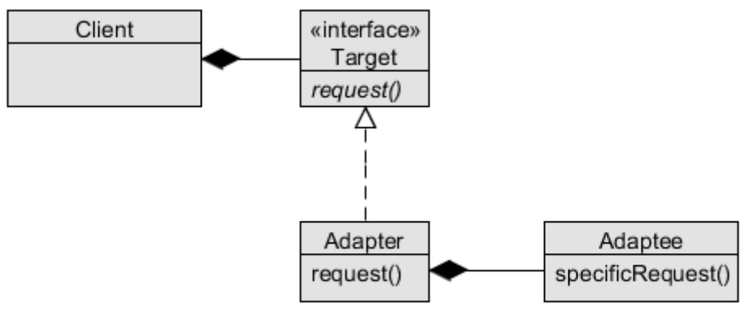
由于上面的代码都是分片的，没有完整的项目源码，为了使读者对示例中的类和接口更清晰，下面给出UML类图：

总结

根据上面的示例，想必读者应该能比较深入的了解到了适配器模式的魔力。下面给出适配器模式的定义（该定义来自于《Head First 设计模式》）：

适配器模式将一个类的接口转换成客户期望的另一个接口，让原本不兼容的接口可以合作无间。

下面给出适配器模式的类图（该类图同样来自于《Head First 设计模式》）：



适配器模式的三个特点：

1　适配器对象实现原有接口
2　适配器对象组合一个实现新接口的对象（这个对象也可以不实现一个接口，只是一个单纯的对象）
3　对适配器原有接口方法的调用被委托给新接口的实例的特定方法

有人认为讲解设计模式的例子都太简单，看着感觉是那么回事，但是要是真想在项目开发中使用，还真是应用不到。其实我们不必在项目中刻意使用设计模式，而是应该从实际的设计问题出发，看哪个模式能解决我们的问题，就使用哪个模式。不要为了使用模式而使用模式，那样就舍本逐末了，一般情况下，只要遵循一定的设计原则就可以了，设计模式也是根据这些原则被总结出来的，熟悉了这些原则，模式自然而然就有了。

其实只要平时善于思考了感悟，在项目中是可以用到设计模式的，并且如果用的合理的话，会为此而受益良多。在下一篇博客中，我会以项目中遇到的一个真实的需求来解析适配器模式的使用，敬请期待。

关于上面提到的适配器模式实际应用的文章已经完成并发表， 感兴趣的朋友可以看一下， 如果有不合理的地方还请指正。

文章链接 运用适配器模式应对项目中的变化