elegant_shadow的专栏

个人资料

elegant_shadow

访问： 38174次
积分： 311
等级：
排名： 千里之外

原创： 3篇 转载： 0篇
译文： 0篇 评论： 65条

Java模式(适配器模式)

标签： java class interface string 任务 语言

2009-12-14 20:11 35731人阅读 评论(65) 收藏 举报

分类： Java开发（2）

今天看了下Java中的适配器模式，以下就来小做下总结和谈谈感想，以便日后使用。

首先，先来先讲讲适配器。适配就是由“源”到“目标”的适配，而当中链接两者的关系就是适配器。它负责把“源”过度到“目标”。举个简单的例子，比如有一个“源”是一个对象人，他拥有2种技能分别是说日语和说英语，而某个岗位（目标）需要你同时回说日语、英语、和法语，好了，现在我们的任务就是要将人这个“源”适配的这个岗位中，如何适配呢？显而易见地我们需要为人添加一个说法语的方法，这样才能满足目标的需要。

接着讨论如何加说法语这个方法，也许你会说，为什么不直接在“源”中直接添加方法，我的理解是，适配是为了实现某种目的而为一个源类暂时性的加上某种方法，所以不能破坏原类的结构。同时不这么做也符合Java的高内聚，低耦合的原理。既然不能直接加，接着我们就来说该怎么来实现为人这个“源”添加一个方法，而又不破坏“源”的本身结构。

适配器模式有2种，第一种是“面向类的适配器模式”，第二种是“面向对象的适配器模式”。

先说“面向类的适配器模式”。顾名思义，这类适配器模式就是主要用于，单一的为某个类而实现适配的这样一种模式，为什么说只为某个类去实现，一会提到，我们先展示这种类适配模式的代码实现。

源的代码如下：

```c-sharp
public class Person {

    private String name;
    private String sex;
    private int age;

    public void speakJapanese(){
        System.out.println("I can speak Japanese!");
    }

    public void speakEnglish(){
        System.out.println("I can speak English!");
    }
    ...//以下省略成员变量的get和set方法
}
```

目标接口的代码如下：

```c-sharp
public interface Job {

    public abstract void speakJapanese();
    public abstract void speakEnglish();
    public abstract void speakFrench();

}
```

适配器的代码如下：

```c-sharp
public class Adapter extends Person implements Job{

    public void speakFrench() {

    }

}
```

好了，代码看完然后要做一些说明了，之前遗留的一个问题，为什么称其为类适配模式呢？很显然的，Adapter类继承了Person类，而在Java这种单继承的语言中也就意味着，他不可能再去继承其他的类了，这样也就是这个适配器只为Person这一个类服务。所以称其为类适配模式。

说完类的适配模式，我们要开始说第2种对象的适配器模式了。对象适配器模式是把“源”作为一个对象聚合到适配器类中。同样的话不多说，贴上代码：

源的代码以及目标代码同上，再次不再赘述。

仅贴出适配器代码：

```c-sharp
public class Adapter implements Job {

    Person person;

    public Adapter(Person person) {
        this.person = person;
    }

    public void speakEnglish() {
        person.speakEnglish();
    }

    public void speakJapanese() {
        person.speakJapanese();
    }

    //new add
    public void speakFrench() {

    }

}
```

对象的适配器模式，把“源”作为一个构造参数传入适配器，然后执行接口所要求的方法。这种适配模式可以为多个源进行适配。弥补了类适配模式的不足。

现在来对2种适配模式做个分析：

1.类的适配模式用于单一源的适配，由于它的源的单一话，代码实现不用写选择逻辑，很清晰；而对象的适配模式则可用于多源的适配，弥补了类适配模式的不足，使得原本用类适配模式需要写很多适配器的情况不复存在，弱点是，由于源的数目可以较多，所以具体的实现条件选择分支比较多，不太清晰。

2.适配器模式主要用于几种情况：（1）系统需要使用现有的类，但现有的类不完全符合需要。（2）讲彼此没有太大关联的类引进来一起完成某项工作（指对象适配）。

最后，再来顺带谈谈默认适配器模式：这种模式的核心归结如下：当你想实现一个接口但又不想实现所有接口方法，只想去实现一部分方法时，就用中默认的适配器模式，他的方法是在接口和具体实现类中添加一个抽象类，而用抽象类去空实现目标接口的所有方法。而具体的实现类只需要覆盖其需要完成的方法即可。代码如下：

接口类：

```c-sharp
public interface Job {

    public abstract void speakJapanese();
    public abstract void speakEnglish();
    public abstract void speakFrench();
    public abstract void speakChinese();

}
```

抽象类：

```c-sharp
public abstract class JobDefault implements Job{

    public void speakChinese() {

    }

    public void speakEnglish() {

    }

    public void speakFrench() {

    }

    public void speakJapanese() {

    }

}
```

实现类：

```c-sharp
public class JobImpl extends JobDefault{

    public void speakChinese(){
        System.out.println("I can speak Chinese!");
    }

}
```

好了，适配器模式就先说到这了，希望对自己和大家都有一个提高。