

elegant_shadow的专栏

 目录视图

 摘要视图

 订阅

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

文章搜索

文章分类

Java开发 (3)

文章存档

2009年12月 (3)

阅读排行

Java模式(适配器模式) (35724)

应用启动时加载XML文档 (1295)

java.util 中的Properties (735)

评论排行

Java模式(适配器模式) (65)

应用启动时加载XML文档 (0)

java.util 中的Properties (0)

推荐文章

* 2016 年最受欢迎的编程语言是什么？

* Chromium扩展（Extension）的页面（Page）加载过程分析

* Android Studio 2.2 来啦

* 手把手教你做音乐播放器（二）技术原理与框架设计

深度学习代码专栏 攒课--我的学习我做主 开启你的知识管理，知识库个人图谱上线

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)**

版权声明：本文为博主原创文章，未经博主允许不得转载。

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]
01. public class Person {
02.
03.     private String name;
04.     private String sex;
05.     private int age;
06.
07.     public void speakJapanese(){
08.         System.out.println("I can speak Japanese!");
09.     }
10.
11.     public void speakEnglish(){
12.         System.out.println("I can speak English!");
13.     }
14.     ...//以下省略成员变量的get和set方法
15. }
```

* JVM 性能调优实战之：使用阿里开源工具 **TProfiler** 在海量业务代码中精确定位性能代码

最新评论

Java模式(适配器模式)
halfsuccess: 通俗易懂，确实不错

Java模式(适配器模式)
lxf_2015: 默认的适配器哪里，实现类继承了那个抽象类，那这个实现类也要把方法补全啊

Java模式(适配器模式)
Stack_mz: 真的挺好的，疑惑了很久，今天第一次看懂啦

Java模式(适配器模式)
bisubisu: 确实易懂。

Java模式(适配器模式)
iris19920918: 大神，怎么测试啊！

Java模式(适配器模式)
baidu_31759869: 默认适配器一定是抽象类吗？

Java模式(适配器模式)
u013034640: 总结得很棒，说英语和中文的翻译就是一个很好的例子。

Java模式(适配器模式)
Barcelona_Lu: 好

Java模式(适配器模式)
a11767483924: 看了好几个，这个看明白了好多东西，

Java模式(适配器模式)
cy297179121: 我居然看懂了

目标接口的代码如下：

```
[c-sharp]
01. public interface Job {
02.
03.     public abstract void speakJapanese();
04.     public abstract void speakEnglish();
05.     public abstract void speakFrench();
06.
07. }
```

适配器的代码如下：

```
[c-sharp]
01. public class Adapter extends Person implements Job{
02.
03.     public void speakFrench() {
04.
05.     }
06.
07. }
```

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]
01. public class Adapter implements Job {
02.
03.     Person person;
04.
05.     public Adapter(Person person) {
06.         this.person = person;
07.     }
08.
09.     public void speakEnglish() {
10.         person.speakEnglish();
11.     }
12.
13.     public void speakJapanese() {
14.         person.speakJapanese();
15.     }
16.
17.     //new add
18.     public void speakFrench() {
19.
20.     }
21.
22. }
```

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]
01. public interface Job {
02.
03.     public abstract void speakJapanese();
04.     public abstract void speakEnglish();
05.     public abstract void speakFrench();
06.     public abstract void speakChinese();
07.
08. }
```

抽象类：

```
[c-sharp]
01. public abstract class JobDefault implements Job{
02.
03.     public void speakChinese() {
04.
05.     }
06.
07.     public void speakEnglish() {
08.
09.     }
10.
11.     public void speakFrench() {
12.
13.     }
14.
15.     public void speakJapanese() {
16.
17.     }
18.
19. }
```

实现类：

```
[c-sharp]
01. public class JobImpl extends JobDefault{
02.
03.     public void speakChinese(){
04.         System.out.println("I can speak Chinese!");
05.     }
06.
07. }
```

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。

顶 踩
5 0