

       最近学习java 基础知识，学习适配器的时候疑惑很多，上网查下资料，对于 Adapter的资料很多，但是比较下本篇不错，这里记录下，大家需要的可以看下。

**适配器模式就是把一个类的接口转换成客户端所期待的另外一种接口，从而使原接口不匹配而无法在一起工作的的两个类能在一起工作。**从功能上说，这些接口不兼容的类一般具有相同或相似的功能。通常我们通过修改该类的接口来解决这种接口不兼容的情形，但是如果我们不愿意为了一个应用而修改各原接口，或者我们压根儿就没有对象的源代码的时候，适配器模式就会派上用场了。

适配器的优点：

1、将目标类和适配者类解耦

2、增加了类的透明性和复用性，将具体的实现封装在适配者类中，对于客户端类来说是透明的，而且提高了适配者的复用性

3、灵活性和扩展性都非常好，符合开闭原则

适配器所涉及的角色包括下面几种：

目标（Target）：定义一个客户端使用的特定接口。

客户端（Client）：使用目标接口，与和目标接口一致的对象合作。

被适配者（Adaptee）：一个现存需要适配的接口。

适配器（Adapter）：**负责将Adaptee的接口转换为Target的接口。**适配器是一个具体的类，这是该模式的核心。

适配器分为**类适配器和对象适配器**两种，下面将详细讲述。

**类适配器**

所谓类适配器，**指的是适配器Adapter继承我们的被适配者Adaptee，并实现目标接口Target**。由于Java中是单继承，所以这个适配器仅仅只能服务于所继承的被适配者Adaptee。

**对象适配器**

所谓对象适配器，简单的说就是适配器实现我们的目标接口，但是并不继承需要被适配的类。**而是通过在适配器的构造函数中将需要被适配的类传递进来从而进行适配。**

类适配器的优点：

1、由于适配器类是适配者类的子类，因此可以再适配器类中置换一些适配者的方法，使得适配器的灵活性更强。

类适配器的缺点：

1、对于Java、C#等不支持多重继承的语言，一次最多只能适配一个适配者类，而且目标抽象类只能为接口，不能为类，其使用有一定的局限性，不能将一个适配者类和他的子类同时适配到目标接口。

对象适配器的优点：

1、把多个不同的适配者适配到同一个目标，也就是说，同一个适配器可以把适配者类和他的子类都适配到目标接口。

对象适配器的缺点：

1、与类适配器模式相比，要想置换适配者类的方法就不容易。

现在把两种实现方式一起写出来：

//////////////////////////////类适配器/////////////////////

**package** adapter;  
  
**public interface** Target {  
 **void** handleRequest();  
}

**package** adapter;  
  
**public class** Adaptee {  
 **public void** request() {  
 System.***out***.println(**"鼠标键盘等被适配器调用了"**);  
 }  
}

**package** adapter;  
  
**public class** AdapterClass **extends** Adaptee **implements** Target {  
  
 @Override  
 **public void** handleRequest() {  
 **this**.request();  
 }  
}

//////////////////////////////对象适配器/////////////////////

**package** adapter;  
  
**public interface** Target {  
 **void** handleRequest();  
}

**package** adapter;  
  
**public class** Adaptee {  
 **public void** request() {  
 System.***out***.println(**"鼠标键盘等被适配器调用了"**);  
 }  
}

**package** adapter;  
  
**public class** Adaptee2 {  
 **public void** request() {  
 System.***out***.println(**"主机内存等被适配器调用了"**);  
 }  
}

**package** adapter;  
  
**public class** AdapterObject **implements** Target {  
  
 **private** Adaptee **adaptee**;  
  
 **private** Adaptee2 **adaptee2**;  
  
 **public** AdapterObject(Adaptee adaptee, Adaptee2 adaptee2) {  
 **this**.**adaptee** = adaptee;  
 **this**.**adaptee2** = adaptee2;  
 }  
  
 @Override  
 **public void** handleRequest() {  
 **adaptee**.request();  
 **adaptee2**.request();  
 }  
}

////////test///////

**package** adapter;  
  
**public class** Client {  
  
 **public static void** main(String[] args) {  
 */\*\*  
 \* 类适配器  
 \*/* System.***out***.println(**"+++++++++++++类适配器,适配器直接继承实现类，只能适配一个类+++++++++++++++"**);  
 Target adapterClass = **new** AdapterClass();*//客户直接调用了USBTarget接口，实现了调用鼠标键盘等方法。类适配器* adapterClass.handleRequest();  
  
  
 */\*\*  
 \* 对象适配器  
 \*/* System.***out***.println(**"+++++++++++++对象适配器,适配器通过构造函数将实现类传递进适配器，可以适配多个类+++++++++++++++"**);  
 Adaptee adaptee = **new** Adaptee();  
 Adaptee2 adaptee2 = **new** Adaptee2();  
 Target adaptObject = **new** AdapterObject(adaptee,adaptee2);  
 adaptObject.handleRequest();  
 }  
  
}

结果：

+++++++++++++类适配器,适配器直接继承实现类，只能适配一个类+++++++++++++++

鼠标键盘等被适配器调用了

+++++++++++++对象适配器,适配器通过构造函数将实现类传递进适配器，可以适配多个类+++++++++++++++

鼠标键盘等被适配器调用了

主机内存等被适配器调用了