adapter模式

适配器模式（Adapter）：将一个类的接口转换成客户希望的另外一个接口。Adapter 模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

适用场景：

1、已经存在的类的接口不符合我们的需求；

2、创建一个可以复用的类，使得该类可以与其他不相关的类或不可预见的类（即那些接口可能不一定兼容的类）协同工作；

3、在不对每一个都进行子类化以匹配它们的接口的情况下，使用一些已经存在的子类。

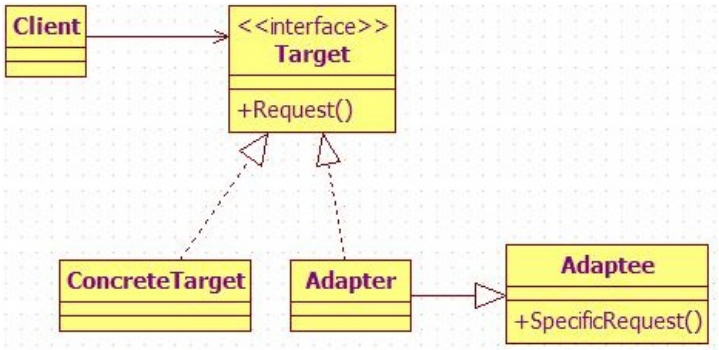
其实现方式主要有两种：

1.类的适配器模式（采用继承实现）

2.对象适配器（采用对象组合方式实现）

2. 类适配器

我们生活中常常听到的是电源适配器，它是用于电流变换（整流）的设备。适配器的存在，就是为了将已存在的东西（接口）转换成适合我们的需要、能被我们所利用。在现实生活中，适配器更多的是作为一个中间层来实现这种转换作用。



其中：

• Target

— 定义Client使用的与特定领域相关的接口。

• Client

— 与符合Target接口的对象协同。

• Adaptee

— 定义一个已经存在的接口，这个接口需要适配。

• Adapter

— 对Adaptee的接口与Target接口进行适配

在上面的通用类图中，Cient 类最终面对的是 Target 接口（或抽象类），它只能够使用符合这一目标标准的子类；而 Adaptee 类则是被适配的对象（也称 源角色），因为它包含specific （特殊的）操作、功能等，所以我们想要在自己的系统中使用它，将其转换成符合我们标准的类，使得 Client 类可以在透明的情况下任意选择使用 ConcreteTarget 类或是具有特殊功能的 Adatee 类。

代码实现如下：

[java] view plain copy

```
1. // 已存在的、具有特殊功能、但不符合我们既有的标准接口的类
2. class Adaptee {
3.     public void specificRequest() {
4.         System.out.println("被适配类具有 特殊功能...");
5.     }
6. }
7.
8.
9. // 目标接口，或称为标准接口
```

```java
interface Target {
    public void request();
}

// 具体目标类，只提供普通功能
class ConcreteTarget implements Target {
    public void request() {
        System.out.println("普通类 具有 普通功能...");
    }
}

// 适配器类，继承了被适配类，同时实现标准接口
class Adapter extends Adaptee implements Target{
    public void request() {
        super.specificRequest();
    }
}

// 测试类
public class Client {
    public static void main(String[] args) {
        // 使用普通功能类
        Target concreteTarget = new ConcreteTarget();
        concreteTarget.request();

        // 使用特殊功能类，即适配类
        Target adapter = new Adapter();
        adapter.request();
    }
}
```

测试结果：

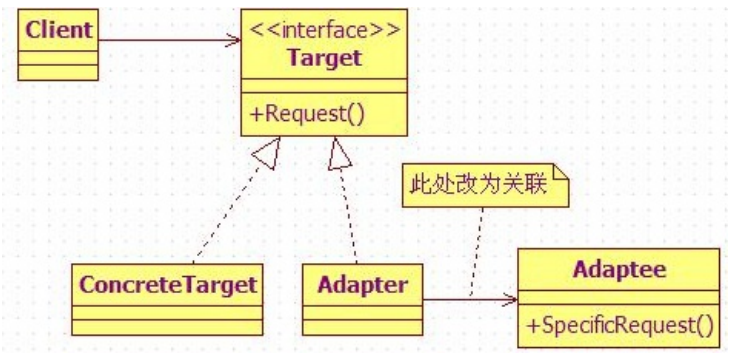
普通类 具有 普通功能...

被适配类具有 特殊功能...

上面这种实现的适配器称为类适配器，因为 Adapter 类既继承了 Adaptee （被适配类），也实现了 Target 接口（因为 Java 不支持多继承，所以这样来实现），在 Client 类中我们可以根据需要选择并创建任一种符合需求的子类，来实现具体功能。

3. 对象适配器

另外一种适配器模式是对象适配器，它不是使用多继承或继承再实现的方式，而是使用直接关联，或者称为委托的方式，类图如下：



代码实现如下：

[java] view plain copy

```
// 适配器类，直接关联被适配类，同时实现标准接口
class Adapter implements Target{
    // 直接关联被适配类
    private Adaptee adaptee;

    // 可以通过构造函数传入具体需要适配的被适配类对象
    public Adapter (Adaptee adaptee) {
        this.adaptee = adaptee;
    }

    public void request() {
        // 这里是使用委托的方式完成特殊功能
        this.adaptee.specificRequest();
    }
}


// 测试类
public class Client {
    public static void main(String[] args) {
        // 使用普通功能类
        Target concreteTarget = new ConcreteTarget();
        concreteTarget.request();

        // 使用特殊功能类，即适配类，
        // 需要先创建一个被适配类的对象作为参数
        Target adapter = new Adapter(new Adaptee());
        adapter.request();
    }
}
```

测试结果与上面的一致。从类图中我们也知道需要修改的只不过就是 Adapter 类的内部结构，即 Adapter 自身必须先拥有一个被适配类的对象，再把具体的特殊功能委托给这个对象来实现。使用对象适配器模式，可以使得 Adapter 类（适配类）根据传入的 Adaptee 对象达到适配多个不同被适配类的功能，当然，此时我们可以为多个被适配类提取出一个接口或抽象类。这样看起来的话，似乎对象适配器模式更加灵活一点。

4. 小结

1、适配器模式也是一种包装模式，与之前的 Decorator 装饰模式同样具有包装的功能；此外，对象适配器模式还具有显式委托的意思在里面（其实类适配器也有这种意思，只不过比较隐含而已），那么我在认为它与 Proxy 代理模式也有点类似；

2、从上面一点对比来看， Decorator 、 Proxy、 Adapter 在实现了自身的最主要目的（这个得看各个模式的最初动机、描述）之外，都可以在包装的前后进行额外的、特殊的功能上的增减，因为我认为它们都有委托的实现意思在里面；

3、我所看的书中说适配器模式不适合在详细设计阶段使用它，它是一种补偿模式，专用来在系统后期扩展、修改时所用。