适配器模式

> *整理自：《java与模式》之适配器模式*

在阎宏博士的《JAVA与模式》一书中开头是这样描述适配器（Adapter）模式的：

适配器模式把一个类的接口变换成客户端所期待的另一种接口，从而使原本因接口不匹配而无法在一起工作的两个类能够在一起工作。

适配器模式的用途

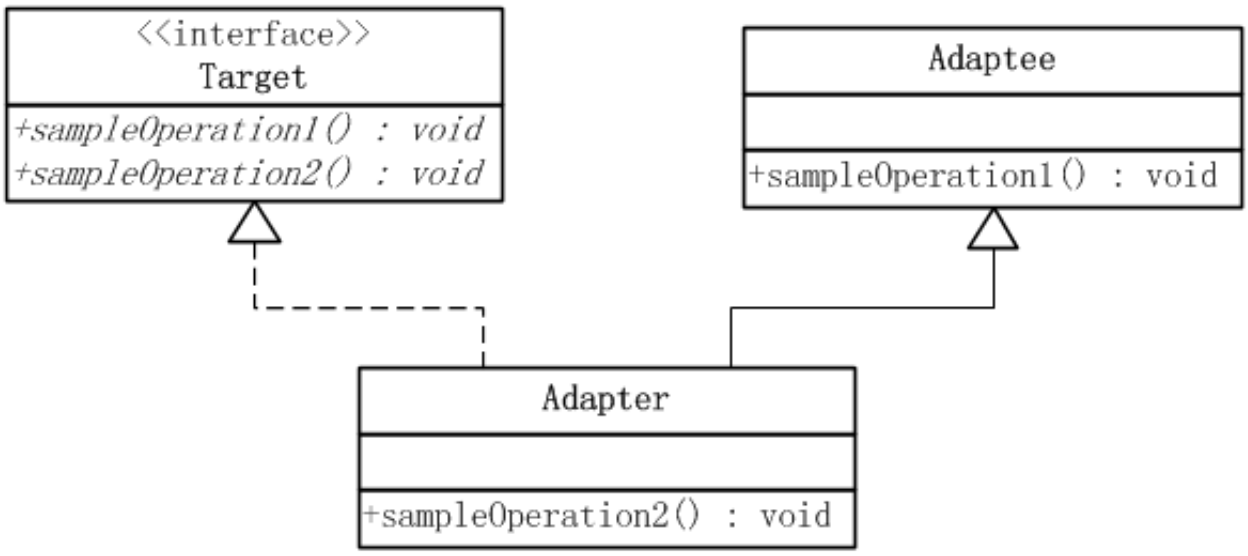
用电器做例子，笔记本电脑的插头一般都是三相的，即除了阳极、阴极外，还有一个地极。而有些地方的电源插座却只有两极，没有地极。电源插座与笔记本电脑的电源插头不匹配使得笔记本电脑无法使用。这时候一个三相到两相的转换器（适配器）就能解决此问题，而这正像是本模式所做的事情。

适配器模式的结构

适配器模式有 **类的适配器模式** 和 **对象的适配器模式** 两种不同的形式。

类适配器模式

类的适配器模式把适配的类的API转换成为目标类的API。



在上图中可以看出，Adaptee类并没有sampleOperation2()方法，而客户端则期待这个方法。为使客户端能够使用Adaptee类，提供一个中间环节，即类Adapter，把Adaptee的API与Target类的API衔接起来。Adapter与Adaptee是继承关系，这决定了这个适配器模式是类的：

模式所涉及的角色有：

- **目标(Target)角色：**这就是所期待得到的接口。注意：由于这里讨论的是类适配器模式，因此目标不可以是类。
- **源(Adapee)角色：**现在需要适配的接口。
- **适配器(Adaper)角色：**适配器类是本模式的核心。适配器把源接口转换成目标接口。显然，这一角色不可以是接口，而必须是具体类。

源代码

```
public interface Target {
    /**
     * 这是源类Adaptee也有的方法
     */
    public void sampleOperation1();
    /**
     * 这是源类Adapteee没有的方法
     */
    public void sampleOperation2();
}
```

上面给出的是目标角色的源代码，这个角色是以一个JAVA接口的形式实现的。可以看出，这个接口声明了两个方法：sampleOperation1()和sampleOperation2()。而源角色Adaptee是一个具体类，它有一个sampleOperation1()方法，但是没有sampleOperation2()方法。

```
public class Adaptee {

    public void sampleOperation1(){}

}
```

适配器角色Adapter扩展了Adaptee,同时又实现了目标(Target)接口。由于Adaptee没有提供sampleOperation2()方法，而目标接口又要求这个方法，因此适配器角色Adapter实现了这

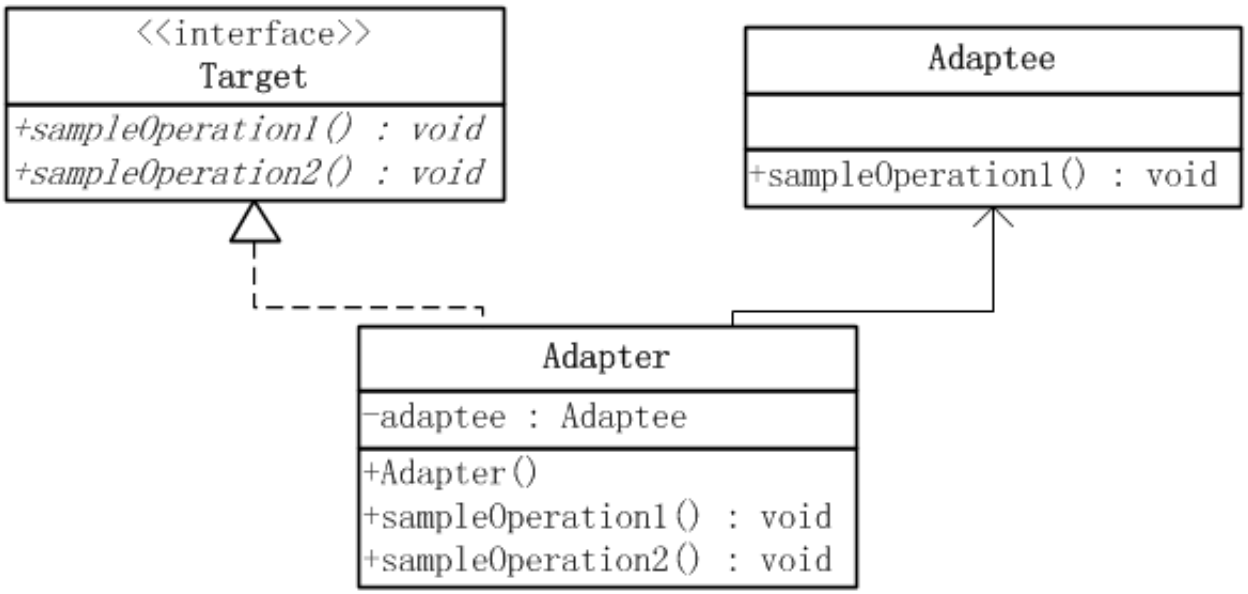
个方法。

```
public class Adapter extends Adaptee implements Target {
    /**
     * 由于源类Adaptee没有方法sampleOperation2()
     * 因此适配器补充上这个方法
     */
    @Override
    public void sampleOperation2() {
        //写相关的代码
    }

}
```

对象适配器模式

与类的适配器模式一样，对象的适配器模式把被适配的类的API转换成为目标类的API，与类的适配器模式不同的是，对象的适配器模式不是使用继承关系连接到Adaptee类，而是使用委派关系连接到Adaptee类。



从上图可以看出，Adaptee类并没有sampleOperation2()方法，而客户端则期待这个方法。为使客户端能够使用Adaptee类，需要提供一个包装(Wrapper)类Adapter。这个包装类包装了一个Adaptee的实例，从而此包装类能够把Adaptee的API与Target类的API衔接起来。Adapter与Adaptee是委派关系，这决定了适配器模式是对象的。

源代码

```
public interface Target {
    /**
     * 这是源类Adaptee也有的方法
     */
    public void sampleOperation1();
    /**
     * 这是源类Adapteee没有的方法
     */
    public void sampleOperation2();
}
```

```
public class Adaptee {

    public void sampleOperation1(){}

}
```

```
public class Adapter {
    private Adaptee adaptee;

    public Adapter(Adaptee adaptee){
        this.adaptee = adaptee;
    }
    /**
     * 源类Adaptee有方法sampleOperation1
     * 因此适配器类直接委派即可
     */
    public void sampleOperation1(){
        this.adaptee.sampleOperation1();
    }
    /**
     * 源类Adaptee没有方法sampleOperation2
     * 因此由适配器类需要补充此方法
     */
    public void sampleOperation2(){
        //写相关的代码
    }
```

```
}
```

类适配器和对象适配器的权衡

- **类适配器** 使用对象继承的方式，是静态的定义方式；而 **对象适配器** 使用对象组合的方式，是动态组合的方式。
- **对于类适配器**，由于适配器直接继承了Adaptee，使得适配器不能和Adaptee的子类一起工作，因为继承是静态的关系，当适配器继承了Adaptee后，就不可能再去处理Adaptee的子类了。

对于对象适配器，一个适配器可以把多种不同的源适配到同一个目标。换言之，同一个适配器可以把源类和它的子类都适配到目标接口。因为对象适配器采用的是对象组合的关系，只要对象类型正确，是不是子类都无所谓。

- **对于类适配器**，适配器可以重定义Adaptee的部分行为，相当于子类覆盖父类的部分实现方法。

对于对象适配器，要重定义Adaptee的行为比较困难，这种情况下，需要定义Adaptee的子类来实现重定义，然后让适配器组合子类。虽然重定义Adaptee的行为比较困难，但是想要增加一些新的行为则方便的很，而且新增加的行为可同时适用于所有的源。

- **对于类适配器**，仅仅引入了一个对象，并不需要额外的引用来间接得到Adaptee。

对于对象适配器，需要额外的引用来间接得到Adaptee。

建议尽量使用对象适配器的实现方式，多用合成/聚合、少用继承。当然，具体问题具体分析，根据需要来选用实现方式，最适合的才是最好的。

适配器模式的优点

更好的复用性

系统需要使用现有的类，而此类的接口不符合系统的需要。那么通过适配器模式就可以让这些功能得到更好的复用。

更好的扩展性

在实现适配器功能的时候，可以调用自己开发的功能，从而自然地扩展系统的功能。

适配器模式的缺点

过多的使用适配器，会让系统非常零乱，不易整体进行把握。比如，明明看到调用的是A接口，其实内部被适配成了B接口的实现，一个系统如果太多出现这种情况，无异于一场灾难。因此如果不是很有必要，可以不使用适配器，而是直接对系统进行重构。

**缺省适配模式

缺省适配(Default Adapter)模式为一个接口提供缺省实现，这样子类型可以从这个缺省实现进行扩展，而不必从原有接口进行扩展。作为适配器模式的一个特例，缺省是适配模式在JAVA语言中有着特殊的应用。

鲁智深的故事

和尚要做什么呢？吃斋、念经、打坐、撞钟、习武等。如果设计一个和尚接口，给出所有的和尚都需要实现的方法，那么这个接口应当如下：

```
public interface 和尚 {
    public void 吃斋();
    public void 念经();
    public void 打坐();
    public void 撞钟();
    public void 习武();
    public String getName();
}
```

显然，所有的和尚类都应当实现接口所定义的全部方法，不然就根本通不过JAVA语言编辑器。像下面的鲁智深类就不行。

```
public class 鲁智深 implements 和尚{
    public void 习武(){
        拳打镇关西;
        大闹五台山;
        大闹桃花村;
        火烧瓦官寺;
        倒拔垂杨柳;
    }
    public String getName(){
        return "鲁智深";
    }
}
```

由于鲁智深只实现了getName()和习武()方法，而没有实现任何其他的方法。因此，它根本就通不过Java语言编译器。鲁智深类只有实现和尚接口的所有的方法才可以通过Java语言编译器，但是这样一来鲁智深就不再是鲁智深了。以史为鉴，可以知天下。研究一下几百年前鲁智深是怎么剃度成和尚的，会对Java编程有很大的启发。不错，当初鲁达剃度，众僧说：“此人形容丑恶、相貌凶顽，不可剃度他”,但是长老却说：”此人上应天星、心地刚直。虽然时下凶顽，命中驳杂，久后却得清净。证果非凡，汝等皆不及他。”

原来如此！看来只要这里也应上一个天星的话，问题就解决了！使用面向对象的语言来说，“应”者，实现也；“天星”者，抽象类也。

```
public abstract class 天星 implements 和尚 {
    public void 吃斋(){}
    public void 念经(){}
    public void 打坐(){}
    public void 撞钟(){}
    public void 习武(){}
    public String getName(){
        return null;
    }
}
```

鲁智深类继承抽象类“天星”

```
public class 鲁智深 extends 和尚{
    public void 习武(){
        拳打镇关西;
        大闹五台山;
        大闹桃花村;
        火烧瓦官寺;
        倒拔垂杨柳;
    }
    public String getName(){
        return "鲁智深";
    }
}
```

这个抽象的天星类便是一个适配器类，鲁智深实际上借助于适配器模式达到了剃度的目的。此适配器类实现了和尚接口所要求的所有方法。但是与通常的适配器模式不同的是，此

适配器类给出的所有的方法的实现都是“平庸”的。这种“平庸化”的适配器模式称作缺省适配模式。

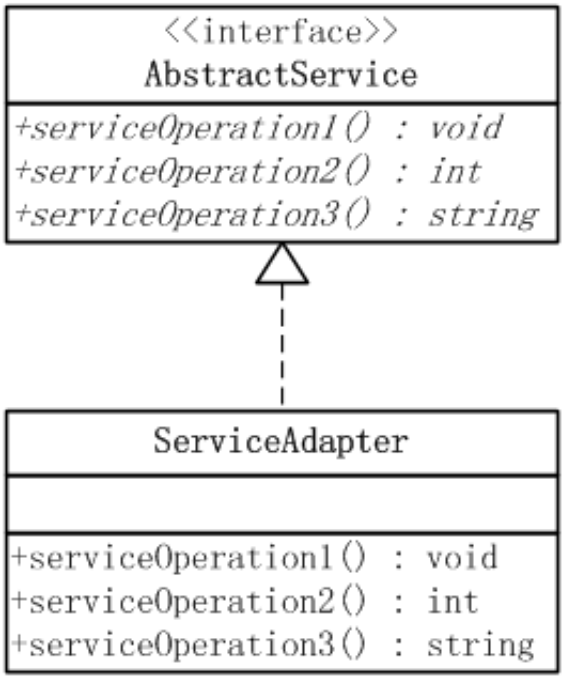
在很多情况下，必须让一个具体类实现某一个接口，但是这个类又用不到接口所规定的所有的方法。通常的处理方法是，这个具体类要实现所有的方法，那些有用的方法要有实现，那些没有用的方法也要有空的、平庸的实现。

这些空的方法是一种浪费，有时也是一种混乱。除非看过这些空方法的代码，程序员可能会以为这些方法不是空的。即便他知道其中有一些方法是空的，也不一定知道哪些方法是空的，哪些方法不是空的，除非看过这些方法的源代码或是文档。

缺省适配模式可以很好的处理这一情况。可以设计一个抽象的适配器类实现接口，此抽象类要给接口所要求的每一种方法都提供一个空的方法。就像帮助了鲁智深的“上应天星”一样，此抽象类可以使它的具体子类免于被迫实现空的方法。

缺省适配模式的结构

缺省适配模式是一种“平庸”化的适配器模式。




```
public interface AbstractService {
    public void serviceOperation1();
    public int serviceOperation2();
    public String serviceOperation3();
```

```
}
```

```
public class ServiceAdapter implements AbstractService{

    @Override
    public void serviceOperation1() {
    }

    @Override
    public int serviceOperation2() {
        return 0;
    }

    @Override
    public String serviceOperation3() {
        return null;
    }

}
```

可以看到，接口AbstractService要求定义三个方法，分别是serviceOperation1()、serviceOperation2()、serviceOperation3()；而抽象适配器类ServiceAdapter则为这三种方法都提供了平庸的实现。因此，任何继承自抽象类ServiceAdapter的具体类都可以选择它所需要的方法实现，而不必理会其他的不需要的方法。

适配器模式的用意是要改变源的接口，以便于目标接口相容。**缺省适配的用意**稍有不同，它是为了方便建立一个不平庸的适配器类而提供的一种平庸实现。

在任何时候，如果不准备实现一个接口的所有方法时，就可以使用“缺省适配模式”制造一个抽象类，给出所有方法的平庸的具体实现。这样，从这个抽象类再继承下去的子类就不必实现所有的方法了。