在Adapter模式中有以下登场角色。

【加个“适配器”以便复用】

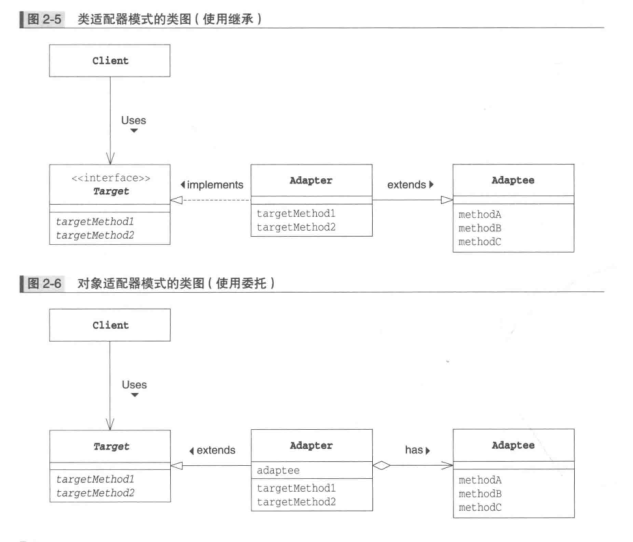
◆Target（对象）该角色负责定义所需的方法。以本章开头的例子来说，即让笔记本电脑正常工作所需的直流12伏特电源。在示例程序中，由Print接口（使用继承时）和Print类（使用委托时）扮演此角色。

◆Client（请求者）该角色负责使用Target角色所定义的方法进行具体处理。以本章开头的例子来说，即直流12伏特电源所驱动的笔记本电脑。在示例程序中，由Main类扮演此角色。

◆Adaptee（被适配）注意不是Adapt-er（适配）角色，而是Adapt-ee（被适配）角色。Adaptee是一个持有既定方法的角色。以本章开头的例子来说，即交流100伏特电源。在示例程序中，由Banner类扮演此角色。如果Adaptee角色中的方法与Target角色的方法相同（也就是说家庭使用的电压就是12伏特直流电压），就不需要接下来的Adapter角色了。

◆Adapter（适配）Adapter模式的主人公。使用Adaptee角色的方法来满足Target角色的需求，这是Adapter模式的目的，也是Adapter角色的作用。以本章开头的例子来说，Adapter角色就是将交流100伏特电压转换为直流12伏特电压的适配器。在示例程序中，由PrintBanner类扮演这个角色。

在类适配器模式中，Adapter角色通过继承来使用Adaptee角色，而在对象适配器模式中，Adapter 角色通过委托来使用Adaptee角色。图2-5和图2-6展示了这两种Adapter模式的类图。



|2.5拓展思路的要点

|什么时候使用Adapter模式

一定会有读者认为“如果某个方法就是我们所需要的方法，那么直接在程序中使用不就可以了吗？为什么还要考虑使用Adapter模式呢？”那么，究竟应当在什么时候使用Adapter模式呢？

很多时候，我们并非从零开始编程，经常会用到现有的类。特别是当现有的类已经被充分测试过了，Bug很少，而且已经被用于其他软件之中时，我们更愿意将这些类作为组件重复利用。

Adapter模式会对现有的类进行适配，生成新的类。通过该模式可以很方便地创建我们需要的方法群。当出现Bug时，由于我们很明确地知道Bug不在现有的类（Adaptee角色）中，所以只需调查扮演Adapter角色的类即可。这样一来，代码问题的排查就会变得非常简单。

如果没有现成的代码

让现有的类适配新的接口（APl）时，使用Adapter模式似乎是理所当然的。不过实际上，我们在让现有的类适配新的接口时，常常会有“只要将这里稍微修改下就可以了”的想法，一不留神就会修改现有的代码。但是需要注意的是，如果要对已经测试完毕的现有代码进行修改，就必须在修改后重新进行测试。

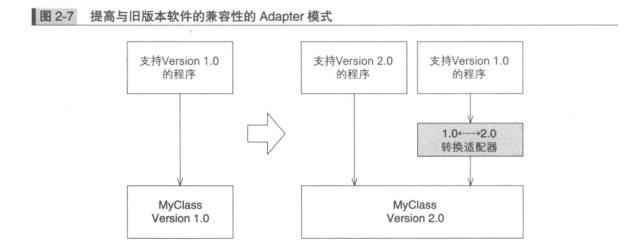
使用Adapter模式可以在完全不改变现有代码的前提下使现有代码适配于新的接口（APl）。此外，在Adapter模式中，并非一定需要现成的代码。只要知道现有类的功能，就可以编写出新的类。

|版本升级与兼容性

软件的生命周期总是伴随着版本的升级，而在版本升级的时候经常会出现“与旧版本的兼容性”问题。如果能够完全抛弃旧版本，那么软件的维护工作将会轻松得多，但是现实中往往无法这样做。这时，可以使用Adapter模式使新旧版本兼容，帮助我们轻松地同时维护新版本和旧版本。

例如，假设我们今后只想维护新版本。这时可以让新版本扮演Adaptee角色，旧版本扮演Target角色。接着编写一个扮演Adapter角色的类，让它使用新版本的类来实现旧版本的类中的方法。

图2-7展示了这些关系的类图（请注意它并非UML图）。



功能完全不同的类

当然，当Adaptee角色和Target角色的功能完全不同时，Adapter模式是无法使用的。就如同我们无法用交流100伏特电压让自来水管出水一样。

|2.6相关的设计模式

◆Bridge模式（第9章）Adapter模式用于连接接口（APl）不同的类，而Bridge模式则用于连接类的功能层次结构与实现层次结构。

◆Decorator模式（第12章）Adapter模式用于填补不同接口（API）之间的缝隙，而Decorator模式则是在不改变接口（API）的前提下增加功能。