如何让20V的笔记本电脑能够在220V的电压下工作？

答案是引入一个电源适配器(AC Adapter)，俗称充电器或变压器，有了这个电源适配器，生活用电和笔记本电脑即可兼容。在软件开发中，有时也存在类似这种不兼容的情况，我们也可以像引入一个电源适配器一样引入一个称之为适配器的角色来协调这些存在不兼容的结构，这种设计方案即为适配器模式。

与电源适配器相似，在适配器模式中引入了一个被称为**适配器(Adapter)的包装类**，而它所包装的对象称为适配者(Adaptee)，即被适配的类。适配器的实现就是把客户类的请求转化为对适配者的相应接口的调用。也就是说：当客户类调用适配器的方法时，在适配器类的内部将调用适配者类的方法，而这个过程对客户类是透明的，客户类并不直接访问适配者类。因此，适配器让那些由于接口不兼容而不能交互的类可以一起工作。

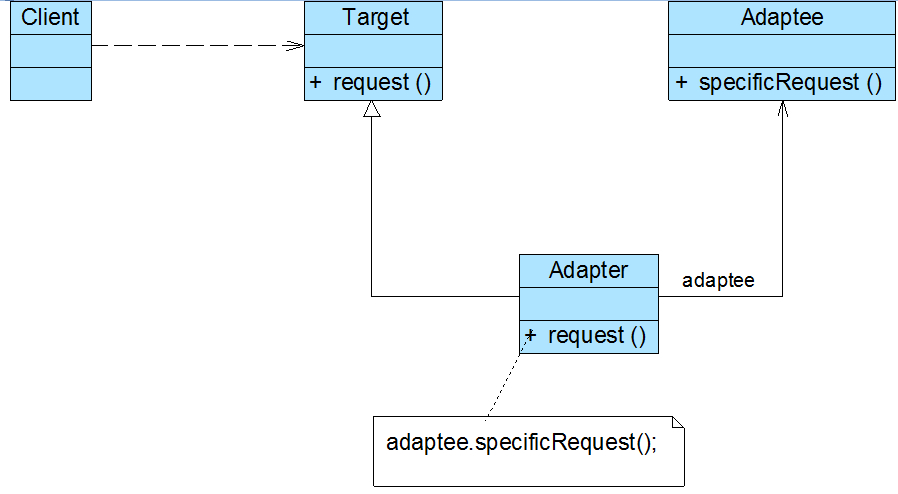
       适配器模式可以将一个类的接口和另一个类的接口匹配起来，而无须修改原来的适配者接口和抽象目标类接口。适配器模式定义如下：

**适配器模式(Adapter Pattern)**：将一个接口转换成客户希望的另一个接口，使接口不兼容的那些类可以一起工作，其别名为包装器(Wrapper)。适配器模式既可以作为类结构型模式，也可以作为对象结构型模式。

注：在适配器模式定义中所提及的接口是指广义的接口，它可以表示一个方法或者方法的集合。

根据适配器类与适配者类的关系不同，适配器模式可分为对象适配器和类适配器两种，在**对象适配器模式**中，适配器与适配者之间是关联关系；在**类适配器模式**中，适配器与适配者之间是继承（或实现）关系。在实际开发中，对象适配器的使用频率更高。

对象适配器模式结构如下图。



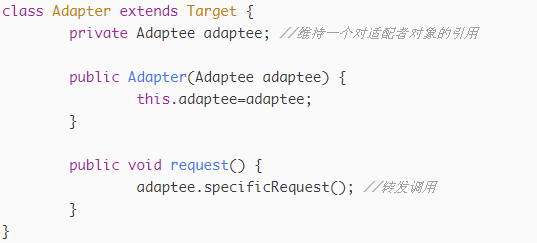
在对象适配器模式结构图中包含如下几个角色：

       ● **Target（目标抽象类）**：目标抽象类定义客户所需接口，可以是一个抽象类或接口，也可以是具体类。

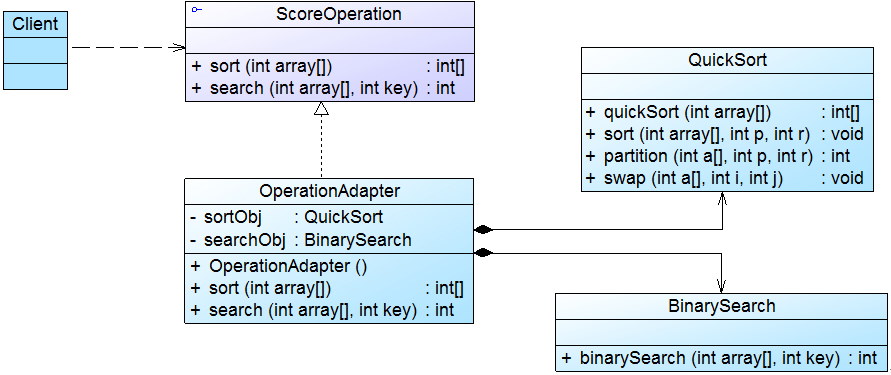
       ● **Adapter（适配器类）**：适配器可以调用另一个接口，作为一个转换器，对Adaptee和Target进行适配，适配器类是适配器模式的核心，在对象适配器中，它通过继承Target并关联一个Adaptee对象使二者产生联系。

       ● **Adaptee（适配者类）**：适配者即被适配的角色，它定义了一个已经存在的接口，这个接口需要适配，适配者类一般是一个具体类，包含了客户希望使用的业务方法，在某些情况下可能没有适配者类的源代码。

根据对象适配器模式结构图，在对象适配器中，客户端需要调用request()方法，而适配者类Adaptee没有该方法，但是它所提供的specificRequest()方法却**是客户端所需要的**。为了使客户端能够使用适配者类，需要提供一个包装类Adapter，即适配器类。这个包装类包装了一个**适配者的实例**，从而将客户端与适配者衔接起来，在适配器的request()方法中调用适配者的specificRequest()方法。因为适配器类与适配者类是关联关系（也可称之为委派关系），所以这种适配器模式称为**对象适配器模式**。典型的对象适配器代码如下所示：



例子：软件公司开发人员决定使用适配器模式来重用算法库中的算法，其基本结构如图所示。

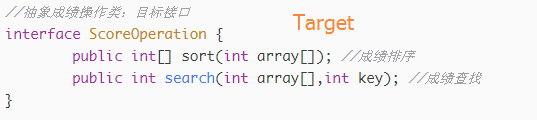


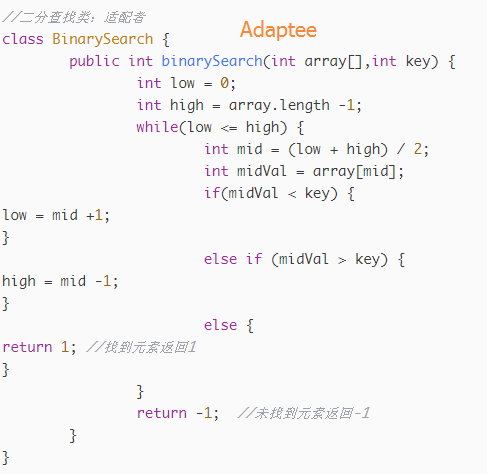
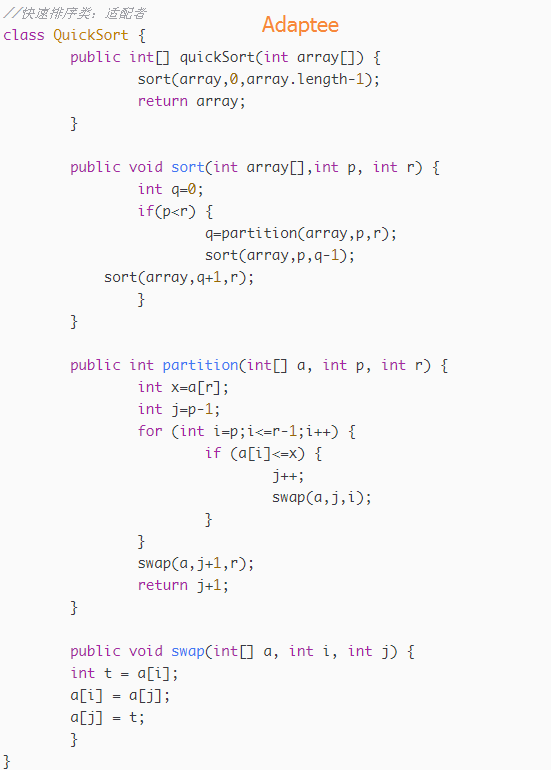
抽象目标（Target）： ScoreOperation接口

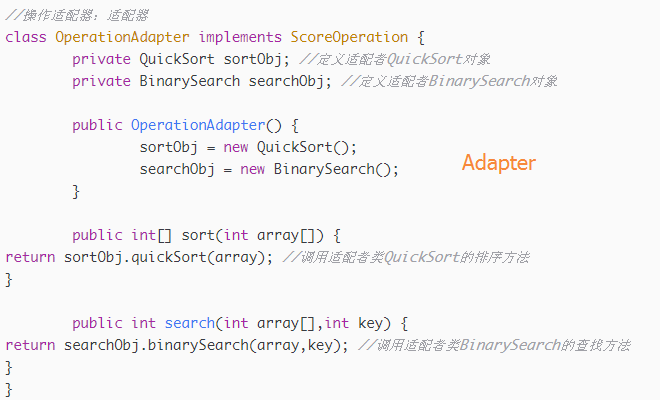
适配者（Adaptee）： QuickSort、BinarySearch

适配器（Adapter）： OperationAdapter

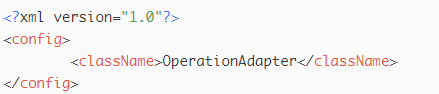
代码实现：







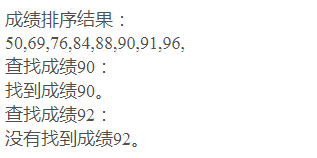
配置文件config.xml中存储了适配器类的类名



客户端测试代码：



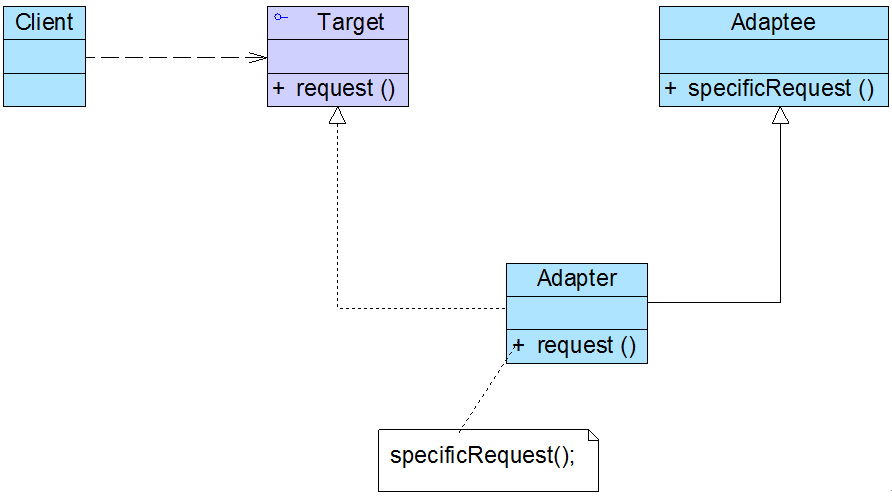
编译并运行程序，输出结果如下：



在本实例中使用了对象适配器模式，同时引入了配置文件，将适配器类的类名存储在配置文件中。如果需要使用其他排序算法类和查找算法类，可以增加一个新的适配器类，使用新的适配器来适配新的算法，原有代码无须修改。通过引入配置文件和反射机制，可以在不修改客户端代码的情况下使用新的适配器，无须修改源代码，符合“开闭原则”。

类适配器

除了对象适配器模式之外，适配器模式还有一种形式，那就是类适配器模式，类适配器模式和对象适配器模式**最大的区别**在于适配器和适配者之间的关系不同，对象适配器模式中适配器和适配者之间是关联关系，而类适配器模式中适配器和适配者是继承关系，类适配器模式结构如图所示：



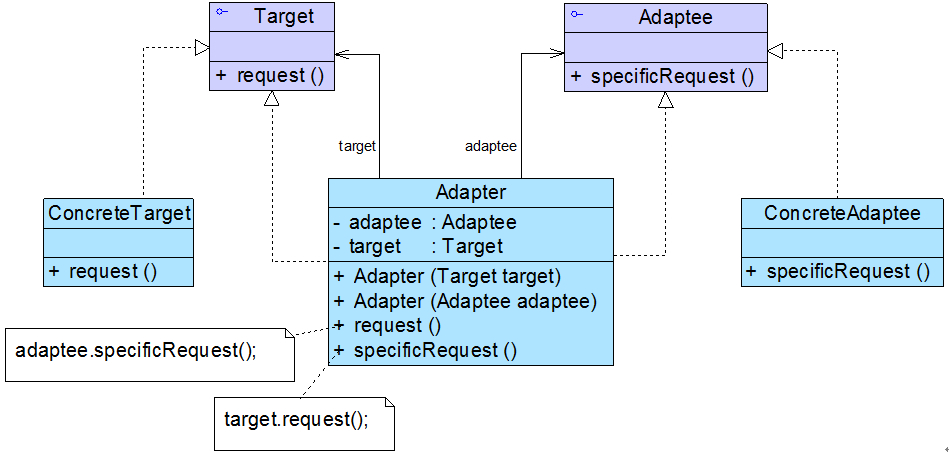
根据类适配器模式结构图，适配器类实现了抽象目标类接口Target，并继承了适配者类，在适配器类的request()方法中调用所继承的适配者类的specificRequest()方法，实现了适配。典型的类适配器代码如下所示：



由于Java、C#等语言不支持多重类继承，因此类适配器的使用受到很多限制，例如如果目标抽象类Target不是接口，而是一个类，就无法使用类适配器；此外，如果适配者Adaptee为最终(Final)类，也无法被类适配器（Adapter）使用。在Java等面向对象编程语言中，大部分情况下我们使用的是对象适配器，类适配器较少使用。

双向适配器

在对象适配器的使用过程中，如果在**适配器中**同时包含对**目标类**和**适配者类**的引用，适配者可以通过它调用目标类中的方法，目标类也可以通过它调用适配者类中的方法，那么该适配器就是一个双向适配器，其结构示意图如图所示：



双向适配器的实现较为复杂，在实际开发中，我们很少使用双向适配器。其典型代码如下所示：

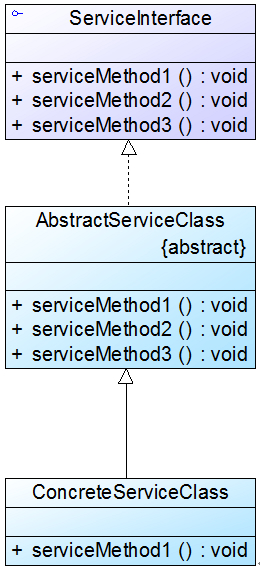


缺省适配器

缺省适配器模式是适配器模式的一种变体，其应用也较为广泛。缺省适配器模式的定义如下：

**缺省适配器模式(Default Adapter Pattern)**：当不需要实现一个接口所提供的所有方法时，可先设计一个抽象类实现该接口，并为接口中每个方法提供一个默认实现（空方法），那么该抽象类的子类可以选择性地覆盖父类的某些方法来实现需求，它适用于不想使用一个接口中的所有方法的情况，又称为单接口适配器模式。

缺省适配器模式结构如图9-7所示：



在缺省适配器模式中，包含如下三个角色：

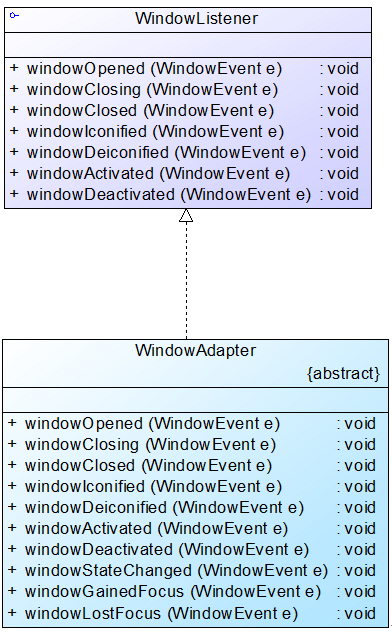
      ● ServiceInterface（适配者接口）：它是一个接口，通常在该接口中声明了大量的方法。

      ● AbstractServiceClass（缺省适配器类）：它是缺省适配器模式的核心类，使用空方法的形式实现了在ServiceInterface接口中声明的方法。通常将它定义为抽象类，因为对它进行实例化没有任何意义。

      ● ConcreteServiceClass（具体业务类）：它是**缺省适配器类的子类**，在没有引入适配器之前，它需要实现适配者接口，因此需要实现在适配者接口中定义的所有方法，而对于一些无须使用的方法也不得不提供空实现。在有了缺省适配器之后，可以直接继承该适配器类，根据需要有选择性地覆盖在适配器类中定义的方法。

在JDK类库的事件处理包java.awt.event中广泛使用了缺省适配器模式，如WindowAdapter、KeyAdapter、MouseAdapter等。

下面我们以处理窗口事件为例来进行说明：在Java语言中，一般我们可以使用两种方式来实现窗口事件处理类，一种是通过实现WindowListener接口，另一种是通过继承WindowAdapter适配器类。如果是使用第一种方式，直接实现WindowListener接口，事件处理类需要实现在该接口中定义的七个方法，而对于大部分需求可能只需要实现一两个方法，其他方法都无须实现，但由于语言特性我们不得不为其他方法也提供一个简单的实现（通常是空实现），这给使用带来了麻烦。而使用缺省适配器模式就可以很好地解决这一问题，在JDK中提供了一个适配器类WindowAdapter来实现WindowListener接口，该适配器类为接口中的每一个方法都提供了一个空实现，此时事件处理类可以继承WindowAdapter类，而无须再为接口中的每个方法都提供实现。如图所示：



适配器模式总结

适配器模式将现有接口转化为客户类所期望的接口，实现了对现有类的复用，它是一种使用频率非常高的设计模式，在软件开发中得以广泛应用，在Spring等开源框架、驱动程序设计（如JDBC中的数据库驱动程序）中也使用了适配器模式。

1. 主要优点

无论是对象适配器模式还是类适配器模式都具有如下优点：

(1) 将目标类和适配者类解耦，通过引入一个适配器类来重用现有的适配者类，无须修改原有结构。

   (2) 增加了类的透明性和复用性，将具体的业务实现过程封装在适配者类中，对于客户端类而言是透明的，而且提高了适配者的复用性，同一个适配者类可以在多个不同的系统中复用。

  (3) 灵活性和扩展性都非常好，通过使用配置文件，可以很方便地更换适配器，也可以在不修改原有代码的基础上增加新的适配器类，完全符合“开闭原则”。

 具体来说，**类适配器模式还有如下优点**：

     由于适配器类是适配者类的子类，因此可以在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。

**对象适配器模式还有如下优点**：

   (1) 一个对象适配器可以把多个不同的适配者适配到同一个目标；

   (2) 可以适配一个适配者的子类，由于适配器和适配者之间是关联关系，根据“里氏代换原则”，适配者的子类也可通过该适配器进行适配。

2. 主要缺点

类适配器模式的缺点如下：

      (1) 对于Java、C#等不支持多重类继承的语言，一次最多只能适配一个适配者类，不能同时适配多个适配者；

      (2) 适配者类不能为最终类，如在Java中不能为final类，C#中不能为sealed类；

      (3) 在Java、C#等语言中，类适配器模式中的目标抽象类只能为接口，不能为类，其使用有一定的局限性。

对象适配器模式的缺点如下：

      与类适配器模式相比，要在适配器中置换适配者类的某些方法比较麻烦。如果一定要置换掉适配者类的一个或多个方法，可以先做一个适配者类的子类，将适配者类的方法置换掉，然后再把适配者类的子类当做真正的适配者进行适配，实现过程较为复杂。

3. 适用场景

在以下情况下可以考虑使用适配器模式：

(1) 系统需要使用一些现有的类，而这些类的接口（如方法名）不符合系统的需要，甚至没有这些类的源代码。

(2) 想创建一个可以重复使用的类，用于与一些彼此之间没有太大关联的一些类，包括一些可能在将来引进的类一起工作。

[适配器模式](https://blog.csdn.net/LoveLion/article/details/17517213)

[不兼容结构的协调——适配器模式（一）：没有源码的算法库，适配器模式概述](https://blog.csdn.net/lovelion/article/details/8624325)

[不兼容结构的协调——适配器模式（二）：没有源码的算法库的适配器模式解决方案](https://blog.csdn.net/lovelion/article/details/8624412)

[不兼容结构的协调——适配器模式（三）：类适配器，双向适配器](https://blog.csdn.net/lovelion/article/details/8624428)

[不兼容结构的协调——适配器模式（四）：缺省适配器，适配器模式总结](https://blog.csdn.net/lovelion/article/details/8624633)