# 适配器模式简介

## 概述

1. 适配器模式的定义为：将一个类的接口转换成客户希望的另一个接口，适配器模式让那些接口不兼容的类可以一起工作（这里的接口指的是方法）。
2. 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（这里的接口指的是方法）的调用，也就是说，当客户类调用适配器的方法时在适配器的内部将调用适配者类的方法。
3. 适配器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接口。（这里的接口指的是方法）。
4. 适配器模式的别名为包装器模式，它既可以作为类结构型模式，也可以作为对象结构型模式。

## 结构

适配器模式包括类适配器和对象适配器。在类适配器中，适配器与适配者之间是继承或实现关系；在对象适配器模式中，适配器与适配者之间是关联关系。对象类型的适配器模式的结构如下：



1. Target（目标抽象类）：目标抽象类定义客户端所需的接口，可以是一个抽象类、接口或者是一个具体类。在类适配器中，由于Java语言不支持多重继承，它只能是接口。
2. Adaptee（适配者类）：适配者类即被适配的角色，一般是一个具体类，包含了客户端希望使用的业务方法，在某些情况下甚至没有适配者类的源码，而只知道它内部中业务方法的功能。
3. Adapter（适配器类）：作为一个转换器，对Target和Adaptee进行适配。适配器Adapter是适配器模式的核心，在类适配器中，它通过实现Target接口并继承Adaptee类来使两者产生联系；在对象适配器中，它通过继承Target并关联一个Adaptee对象使两者产生联系。

在实际开发中，对象适配器更加灵活，使用频率更高。

# 实现

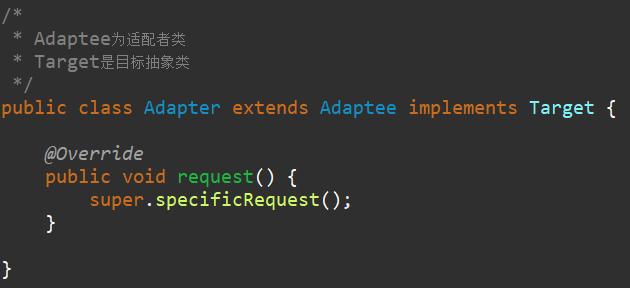
由于适配器模式包括类适配器模式和对象适配器模式两种形式，下面分别介绍这两种适配器模式的实现机制。

## 类适配器



根据上图所示的类适配器模式结构图，在类适配器中适配者类Adaptee没有request（）方法，而客户端期待这个方法，但在适配者类中实现了specificRequest（）方法，该方法正是客户端所需要的。为了使客户端能够通过调用request（）方法的形式来使用适配者类Adaptee中的specificRequest（）方法，提供一个中间类，即适配器类Adapter，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specificRequest（）方法，达到适配的目的。因为适配器类和与适配者类是继承关系，所以这种适配器模式称为类适配器模式。

典型的类适配器代码如下：

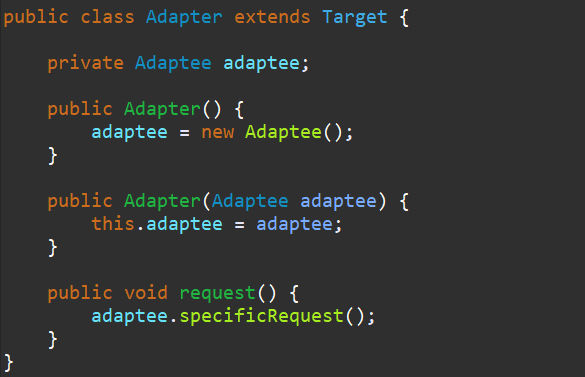


## 对象适配器



根据上图所示的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，需要提供一个中间类Adapter，即适配器类。这个适配器类中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：



# 缺省适配器

缺省适配器模式是适配器模式的一种变体，应用较为广泛。

## 概念

缺省适配器模式的定义如下：当不想实现一个接口中提供的所有抽象方法时，可以先定义一个抽象类实现该接口，在抽象类中覆写接口中所有的抽象方法，并且为这些方法只提供默认实现（空方法），然后该抽象类的子类可以选择性地覆写父类中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，又称为单单接口适配器模式。其结构图如下：



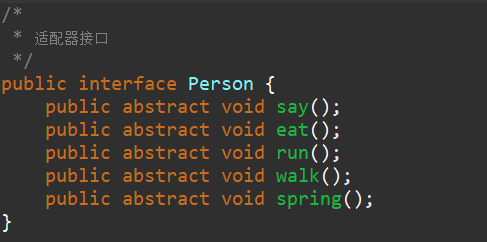
## 结构

在缺省适配器中包含3个角色。

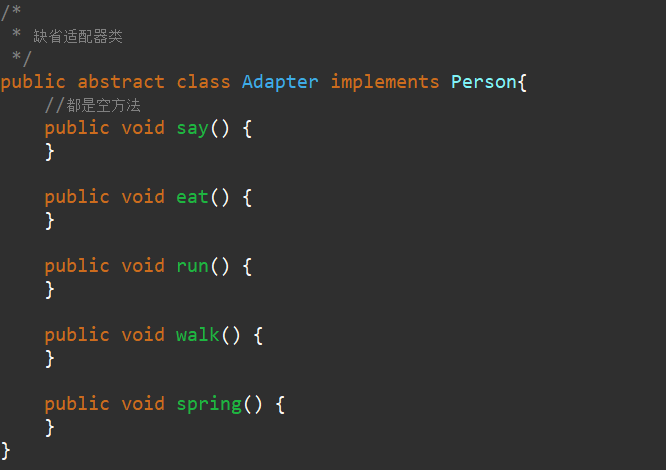
1. ServiceInterface（适配者接口）：它是一个接口，通常在接口中定义了大量的抽象方法。
2. AbstractServiceClass（缺省适配器类）：它是缺省适配器模式的核心类，使用空方法的形式实现了在ServiceInterface接口中声明的方法。通常将它定义为抽象类，因为对它进行实例化没有任何意义。
3. ConcreteServiceClass（具体业务类）：它是缺省适配器类的子类，在没有引入缺省适配器类之前它需要实现适配者接口中定义的所有抽象方法，而对于一些无须使用的方法不得不提供空实现；而有了缺省适配器类之后可以直接继承它，根据需要有选择性地覆写在缺省适配器类中定义的方法。

## 实现

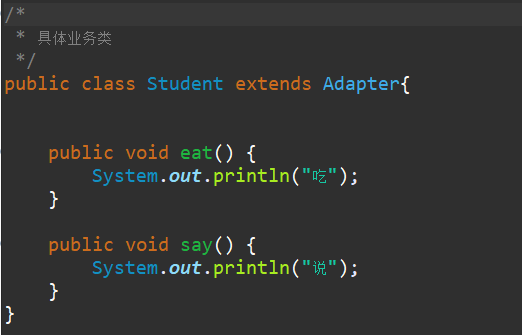
1）适配器接口Person，定义了大量的抽象方法



2）缺省适配器类



1. 具体业务类，继承缺省适配器类，这样一来就可以有选择性的覆写自己想要的方法了，而不需要实现接口然后实现接口中的全部抽象方法。



# 双向适配器

## 概念

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，那么这个适配器就是一个双向适配器。

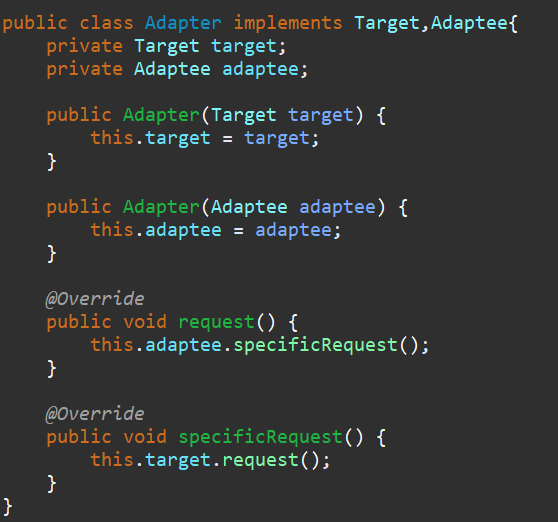
## 结构图

其结构图如下：



## 部分实现

典型代码如下：



在客户端类中使用：



# 优缺点及适用环境

适配器模式将现由接口转化为客户端所期望的接口，实现了对现有类的复用，是一种使用频率非常高的设计模式。

## 优点

无论是类适配器模式还是对象适配器模式都具有以下优点：

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，无须修改适配者类的原有结构。
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。

类适配器模式还有以下优点：

1. 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以下优点：

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，由于适配者和适配者之间是关联关系，适配者的子类也可以同该适配器进行搭配。

## 缺点

类适配器模式的缺点主要如下：

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，不同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。
3. 在Java、C#等语言中，类适配器模式中的目标抽象类只能是接口，不能是类，拥有一定的局限性。

对象适配器模式的缺点主要如下：

与类适配器模式相比，在对象适配器模式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成真正的适配者进行适配，实现过程较为复杂。

## 适用环境

在以下情况下可以考虑适用适配器模式：

1. 系统需要使用一些现有的类，而这些类中的方法不符合系统的需要，甚至没有这些类的源码。
2. 想创建一个可以重复使用的类，用于和一些彼此之间没有太大关联的类（包括一些可能在将来引进的类）一起工作。
3. 不想修改原来的适配者类。