# **[Java设计模式之《适配器模式》及应用场景](https://www.cnblogs.com/V1haoge/p/6479118.html)**

适配器就是一种适配中间件，它存在于不匹配的二者之间，用于连接二者，将不匹配变得匹配，简单点理解就是平常所见的转接头，转换器之类的存在。

　　适配器模式有两种：类适配器、对象适配器、接口适配器

　　前二者在实现上有些许区别，作用一样，第三个接口适配器差别较大。

1、类适配器模式：

　　原理：通过继承来实现适配器功能。

　　当我们要访问的接口A中没有我们想要的方法 ，却在另一个接口B中发现了合适的方法，我们又不能改变访问接口A，在这种情况下，我们可以定义一个适配器p来进行中转，这个适配器p要实现我们访问的接口A，这样我们就能继续访问当前接口A中的方法（虽然它目前不是我们的菜），然后再继承接口B的实现类BB，这样我们可以在适配器P中访问接口B的方法了，这时我们在适配器P中的接口A方法中直接引用BB中的合适方法，这样就完成了一个简单的类适配器。

　　详见下方实例：我们以ps2与usb的转接为例

参考第一个案例:

实例讲解：

　　我手中有个ps2插头的设备，但是主机上只有usb插头的插口，怎么办呢？弄个转换器，将ps2插头转换成为USB插头就可以使用了。

　　接口Ps2：描述ps2接口格式

　　接口Usb：描述USB接口格式

　　类Usber：是接口Usb的实现类，是具体的USB接口格式

　　Adapter：用于将ps2接口格式转换成为USB接口格式

2、对象适配器模式

　　原理：通过组合来实现适配器功能。

　　当我们要访问的接口A中没有我们想要的方法 ，却在另一个接口B中发现了合适的方法，我们又不能改变访问接口A，在这种情况下，我们可以定义一个适配器p来进行中转，这个适配器p要实现我们访问的接口A，这样我们就能继续访问当前接口A中的方法（虽然它目前不是我们的菜），然后在适配器P中定义私有变量C（对象）（B接口指向变量名），再定义一个带参数的构造器用来为对象C赋值，再在A接口的方法实现中使用对象C调用其来源于B接口的方法。

　　详见下方实例：我们仍然以ps2与usb的转接为例

参考第二个案例

实例讲解：

　　我手中有个ps2插头的设备，但是主机上只有usb插头的插口，怎么办呢？弄个转换器，将ps2插头转换成为USB插头就可以使用了。

　　接口Ps2：描述ps2接口格式

　　接口Usb：描述USB接口格式

　　类Usber：是接口Usb的实现类，是具体的USB接口格式

　　Adapter：用于将ps2接口格式转换成为USB接口格式

3、接口适配器模式

　　原理：通过抽象类来实现适配，这种适配稍别于上面所述的适配。

　　当存在这样一个接口，其中定义了N多的方法，而我们现在却只想使用其中的一个到几个方法，如果我们直接实现接口，那么我们要对所有的方法进行实现，哪怕我们仅仅是对不需要的方法进行置空（只写一对大括号，不做具体方法实现）也会导致这个类变得臃肿，调用也不方便，这时我们可以使用一个抽象类作为中间件，即适配器，用这个抽象类实现接口，而在抽象类中所有的方法都进行置空，那么我们在创建抽象类的继承类，而且重写我们需要使用的那几个方法即可。

参考第三个案例

4、适配器模式应用场景

类适配器与对象适配器的使用场景一致，仅仅是实现手段稍有区别，二者主要用于如下场景：

　　（1）想要使用一个已经存在的类，但是它却不符合现有的接口规范，导致无法直接去访问，这时创建一个适配器就能间接去访问这个类中的方法。

　　（2）我们有一个类，想将其设计为可重用的类（可被多处访问），我们可以创建适配器来将这个类来适配其他没有提供合适接口的类。

　　以上两个场景其实就是从两个角度来描述一类问题，那就是要访问的方法不在合适的接口里，一个从接口出发（被访问），一个从访问出发（主动访问）。

接口适配器使用场景：

　　（1）想要使用接口中的某个或某些方法，但是接口中有太多方法，我们要使用时必须实现接口并实现其中的所有方法，可以使用抽象类来实现接口，并不对方法进行实现（仅置空），然后我们再继承这个抽象类来通过重写想用的方法的方式来实现。这个抽象类就是适配器。