适配器模式

2020年2月6日　　17:22

1. 概念

把一个类的接口变换成客户端所期待的另一种接口，从而使原本接口不匹配而无法一起工作的两个类能够在一起工作。

- 类适配器模式
- 对象适配器模式

解决问题：原本由于接口不兼容而不能一起工作的那些类可以在一起工作

2. UML结构图

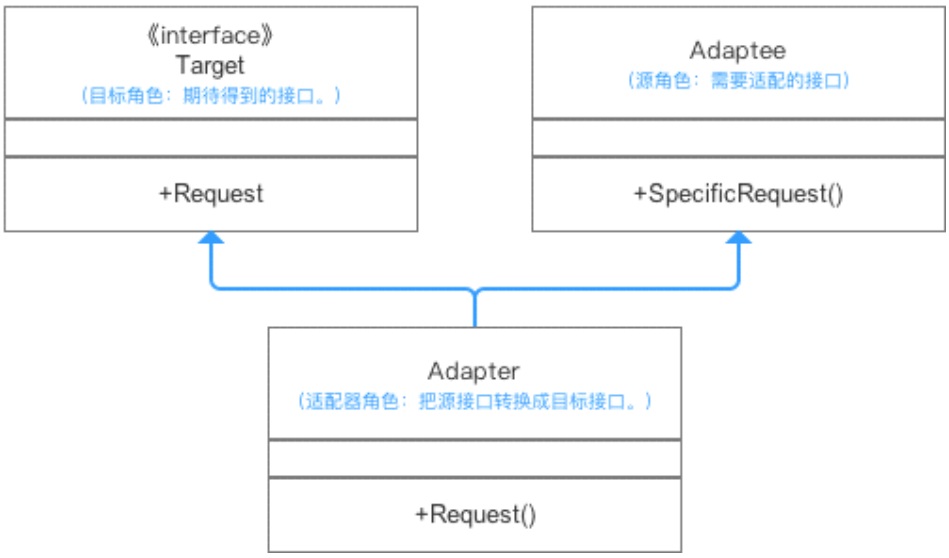

图2.1 类适配器模式结构图

在上图中可以看出：

- 冲突：Target期待调用Request方法，而Adaptee并没有（这就是所谓的不兼容了）。
- 解决方案：为使Target能够使用Adaptee类里的SpecificRequest方法，故提供一个中间环节Adapter类**（继承Adaptee & 实现Target接口 -** Adapter与Adaptee是继承关系，这决定了这个适配器模式是类**）**，把Adaptee的API与Target的API衔接起来（适配）。

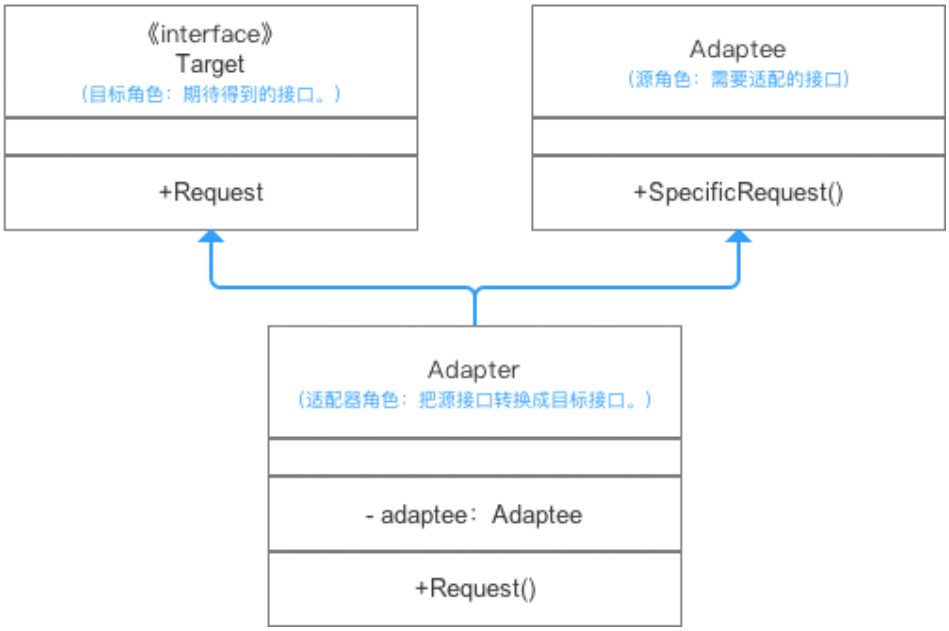

图2.2 对象适配器模式结构图

在上图中可以看出：

- 冲突：Target期待调用Request方法，而Adaptee并没有（这就是所谓的不兼容了）。
- 解决方案：为使Target能够使用Adaptee类里的SpecificRequest方法，故提供一个中间环节Adapter类（**包装了一个Adaptee的实例** - Adapter与Adaptee是委派关系，这决定了适配器模式是对象的），把Adaptee的API与Target的API衔接起来（适配）。

3. 适配者模式示例

家庭电压为220v交流电，为了给手机充电，需要使用适配器将其转为5v直流电才能进行。

1. Target接口 - 希望得到的接口

```
public interface DC5 {
    int output5v();
}
```

2. Adaptee类 - 被适配类

```
public class AC220 {
    public int outputAC220V(){
        int output = 200;
        System.out.println("输出220v交流电");
        return output;
    }
}
```

3. Adapter类 - 适配器

```
public class PowerAdapter implements DC5 {
    private AC220 ac220 = new AC220(); // 对象适配器模式
    @Override
    public int output5v() {
        int adapterInput = ac220.outputAC220V();

        // 变压器
        int adapterOutput = adapterInput/44;
        System.out.println("使用PowerAdapter输入VC:" + adapterInput + "V" + "输出DC:" +
adapterOutput + "V");

        return adapterOutput;
    }
}
```

4. 测试类

```
public class Test {
    public static void main(String[] args) {
        DC5 dc5 = new PowerAdapter();
        dc5.output5v();
```

```
    }
}
```

5. 结果

```
输出220v交流电
使用PowerAdapter输入VC:200V输出DC:4V
```

4. 优缺点

1. 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。
- 透明、简单：客户端可以调用同一接口，因而对客户端来说是透明的。这样做更简单 & 更直接
- 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。
- 解耦性：将目标类和适配者类解耦，通过引入一个适配器类重用现有的适配者类，而无需修改原有代码
- 符合开放-关闭原则：同一个适配器可以把被适配类和它的子类都适配到目标接口；可以为不同的目标接口实现不同的适配器，而不需要修改被适配类

2. 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

5. 应用场景

系统需要复用现有类，而该类的接口不符合系统的需求，可以使用适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作

1. 类和对象适配器模式的使用**场景**

（1）灵活使用时：选择对象的适配器模式。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。

（2）需要同时配源类和其子类：选择对象的适配器。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。

（3）需要重新定义Adaptee的部分行为：选择类适配器。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。

（4）仅仅希望使用方便时：选择类适配器。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。

6. 参考资料

https://www.jianshu.com/p/9d0575311214