概述

> 适配器模式将一个类的接口，转换成客户期望的另一个接口。适配器让原本接口不兼容的类可以合作无间

下面讲一件真事：本人去年年底公司年会抽奖抽中了一台粪叉（iphone x），由于没有现货，大概过了一个月左右，抽中的粪叉才送到我手中。拿到手一看，咦，是港版的。略微有些失望，因为港版的比较便宜。不过这台粪叉打算留给女友使用，不打算卖，港版就港版吧。

第二天，打算给粪叉充电的时候突然发现充电头是这样的：

很明显港版充电头的三个头头很粗，而大陆的插座的孔没有那么大。不过不怕，通过万能的某宝，买了个转换器：

将充电器插在转换器上，再将转换器插在插座上，问题迎刃而解。

从上面的生活例子中，我们遇到的问题是港版充电头和插座不兼容，而解决方法是买一个转换头，这是典型的适配器模式，转换头就是我们所说的适配器。

我们也可以拥有多种适配器，每一种适配器的功能都可不同，例如有的适配器是将三头的充电头转换成两头，有的将家用标准电压220v转换成适合你电器的电压。

用过笔记本电脑的肯定知道笔记本的电源线长这样：

而且它有一个名字叫**电源适配器**，以前我不懂它为什么叫“适配器”，也不懂为什么要弄成这么大块头，携带不是很方便。

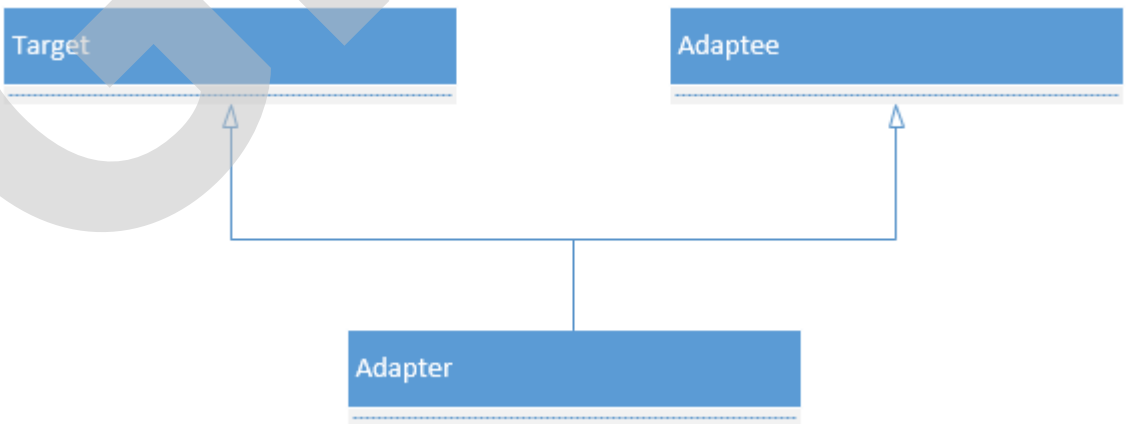
但如果你知道适配器模式，那么对“适配器”这三个字一定很敏感，很容易就能想到电源适配器的作用是解决家用电和电脑之间的兼容问题。电源适配器中包含了变压器，电容等组件，可以将交流电转换为电脑所需的直流电。

适配器模式在生活中应用极其广泛，那么在代码中它是什么样的呢？

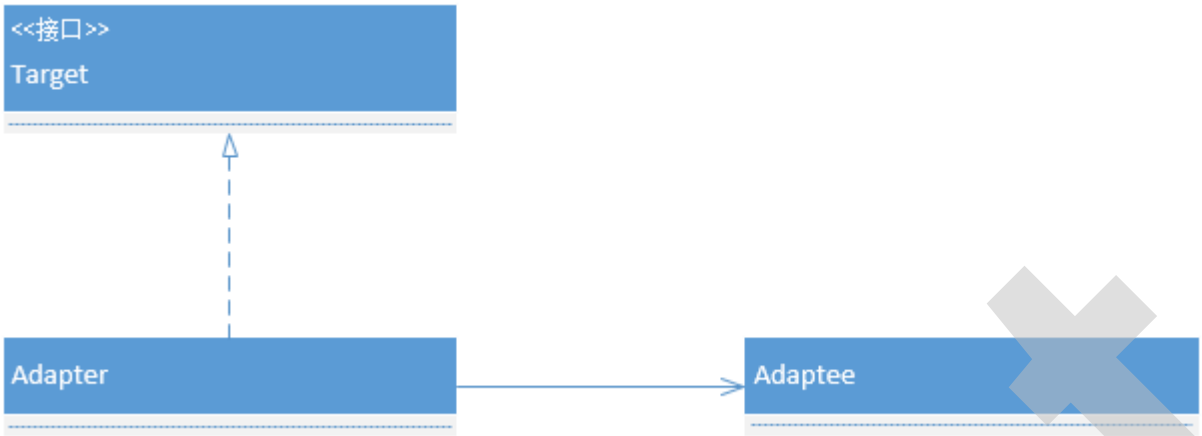
适配器主要分为两种：

- 类适配器
- 对象适配器

类适配器：



对象适配器：



上面类图中，Adapter为适配器，相当于前面插头例子中的转换器，Adaptee是要被适配的对象，相当于港版插头。Target是目标接口，相当于插座。

类适配器和对象适配器最大的区别就在于：类适配器中的Adapter是多继承于Target和Adaptee。由于JAVA不支持多继承，所以比较常用的对象适配器。

而在对象适配器中，Adapter和Adaptee是依赖关系，也可以说是组合关系。同时Target是接口（其实也可以是类）

说到这里，可能部分人会有些懵，搞不懂为什么要分这两种，没关系，对于适配器模式，最重要的是明白适配器在其中起到的作用是：作为一个中间者协调不兼容的双方，使其兼容。无论target是类还是接口，Adapter和Adaptee之间是继承关系还是组合关系，都无所谓，只是形式上的不同罢了。

大家对于设计模式的理解也不要仅仅局限在代码上，Target和Adaptee甚至可以是两个独立的系统。设计模式是解决方案，重要的是编程思想，而不是固定的模板。这一点很重要。

因此哪怕你不知道适配器模式分为"类适配器"和"对象适配器"也没关系，只要你理解适配器模式的核心思想就够了。概念和定义都是浮云，实战才是王道。

使用场景

那么在实际代码中如何使用适配器模式呢？

首先大家要明白一点，这一点非常重要：

适配器适合用于解决新旧系统（或新旧接口）之间的兼容问题，而不建议在一开始就直接使用

前面提到港版粪叉充电头不适配的问题，假设粪叉公司提供了大陆可用的插头和港用插头，让你二选一，你选哪个？

那还用说，逗比才选港用插头，有大陆通用的插头可以直接用，谁还乐意去额外整一个转换器？浪费钱。

同样的道理，如果从一开始接口和接口之间就没有兼容问题，请不要多此一举使用适配器模式。代码越简单越好，不要为了设计模式而设计模式。

代码实现

隔壁老王最近接了个音乐播放器的项目，要求支持mp3，wma等格式，于是老王定义了接口：

```
public interface MusicPlayer {
    /**
     * 播放音乐
     * @param type 音乐格式
     * @param filename 文件
     */
    public void play(String type, String filename);
}
```

老王兴致勃勃的要开始开发了，后来老王心想那么多种格式要支持，自己去一一实现太麻烦了，干脆直接找现成的库来用用。

于是他找到了：

```
public class ExistPlayer {

    public void playMp3(String filename){
        System.out.println("play mp3 : "+filename);
    }

    public void playWma(String filename){
        System.out.println("play wma : "+filename);
    }

}
```

老王发现这个库的接口跟老王自己定义的不一样，而且该库是个jar包，不能改它源码，就算能改也不能这么做，因为其他地方可能有调用此处代码，改了有可能会导致系统崩溃。

咋办？写个适配器呗。

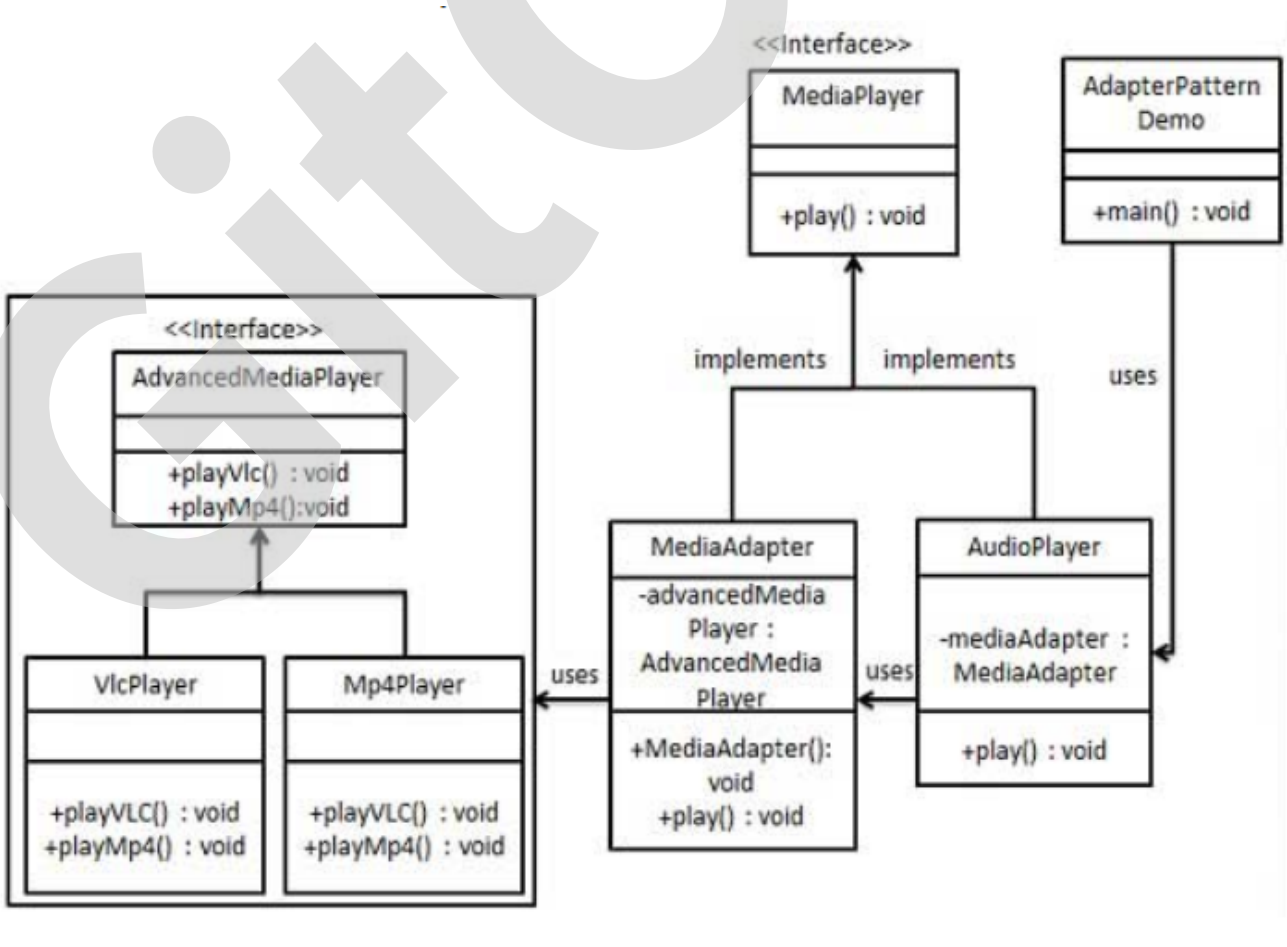
```
public class PlayerAdapter implements MusicPlayer
{
    //在适配器中使用旧接口
    private ExistPlayer player;

    public PlayerAdapter(){
        player = new ExistPlayer();
    }

    @Override
    public void play(String type, String filename) {
        if(type == "mp3"){
            player.playMp3(filename);
        }else if(type == "wma"){
            player.playWma(filename);
        }
    }
}
```

客户端调用：

```
MusicPlayer player = new PlayerAdapter();
player.play("mp3","xxx.mp3");
player.play("mp4","xxx.mp4");
```

音乐播放器代码类图：

如果你在其他教程里看到适配器模式的类图，发现和笔者发的类图不一样，不要慌，你细细比较会发现其实核心内容是一致的。例如下图是来自网上某教程：



为了让大家更容易理解，笔者举的例子都是最简单的。因此请大家自己细细品味上面类图领悟其精髓，这里不再多说了。

也许有人会问，既然不能改库的代码，那可以改老王自己定义的接口啊，改成一样的不就好了。说得对，如果能这么容易就解决接口兼容问题，那么笔者绝对建议你这么干。

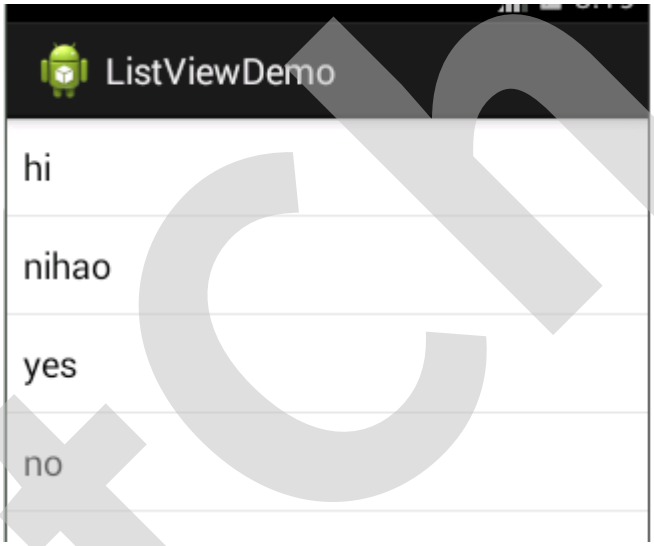
不过有时候条件不允许我们这么做，比如老王定义的这个接口已经在其他地方被使用了，不能牵一发而动全身。

再比如这个库只提供了播放mp3和wma文件，而我们还得继续找支持其他格式的库，并且其他库的接口又不一样了。因此我们必须要自己统一定义一套接口，通过适配器去兼容各种现成的库接口。

实际案例

安卓中的ListView列表

手机app中很常见的布局就是列表，如下：



使用代码也很简单：

```
ListView lv=(ListView) findViewById(R.id.lv);

//简单设置测试字符数组
String []data ={"hi","nihao","yes","no"};

ArrayAdapter<String> adapter=new ArrayAdapter<>(this,android.R.layout.simple_list_item_1,data);

lv.setAdapter(adapter);
```

不懂安卓也没关系，你只要看懂上面代码中几个关键变量：

1. lv：列表
2. data：列表的数据
3. adapter：适配器

看到Adapter，不用怀疑，必然是使用了适配器模式，一般情况下没人会无聊到不是适配器模式却故意用Adapter字样来误导别人。

那么安卓的列表为什么要使用适配器？它解决了什么问题？

listview是安卓的列表组件，并且列表的每一项可以是简单地显示文字，也可以是很复杂的布局。同时列表的数据结构也有多种，有的是简单数组（如上述例子），有的是List，List的元素也千变万化，列表组件根本不知道开发者会传什么样的数据结构进来，也不知道该如何将数据显示到界面上，因此就需要使用适配器来解决列表组件和数据之间的兼容问题。

在安卓开发中，谷歌大神们默认为我们提供了多种适配器，但我们也可以自定义适配器，例如当我们的列表数据是图文并茂的：

```
//本段代码来自网络
public class MyAdapter extends BaseAdapter {

    private List<Student> stuList;
    private LayoutInflater inflater;
    public MyAdapter() {}

    public MyAdapter(List<Student> stuList,Context context) {
        this.stuList = stuList;
        this.inflater=LayoutInflater.from(context);
    }

    @Override
    public int getCount() {
        return stuList==null?0:stuList.size();
    }

    @Override
    public Student getItem(int position) {
        return stuList.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }
    //关键代码
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        //加载布局为一个视图
        View view=inflater.inflate(R.layout.layout_student_item,null);
        Student student=getItem(position);
        //在view视图中查找id为image_photo的控件
        ImageView image_photo= (ImageView) view.findViewById(R.id.image_photo);
        TextView tv_name= (TextView) view.findViewById(R.id.name);
        TextView tv_age= (TextView) view.findViewById(R.id.age);
        image_photo.setImageResource(student.getPhoto());
        tv_name.setText(student.getName());
        tv_age.setText(String.valueOf(student.getAge()));
        return view;
    }
```

该适配器中关键代码是`getView()`方法，在这里我们做了数据和列表组件之间的转换操作。（具体代码细节不必深入）

总结

适配器模式适用于解决不同接口或不同系统间的兼容问题，想要修改旧系统的接口时应优先考虑使用适配器模式

优点

可以让两个不相干的类一起运行，无需修改旧代码，灵活性好

缺点

适配器模式用多了会使整个系统变得很复杂，因为你调用的是适配器，却不知道适配器内部做了多少转换操作，所以在能用别的办法解决问题时，尽量少用适配器模式。当然也没必要畏首畏尾不敢用。