第 17 章　适配器模式

适配器模式的作用是解决两个软件实体间的接口不兼容的问题。使用适配器模式之后，原本由于接口不兼容而不能工作的两个软件实体可以一起工作。

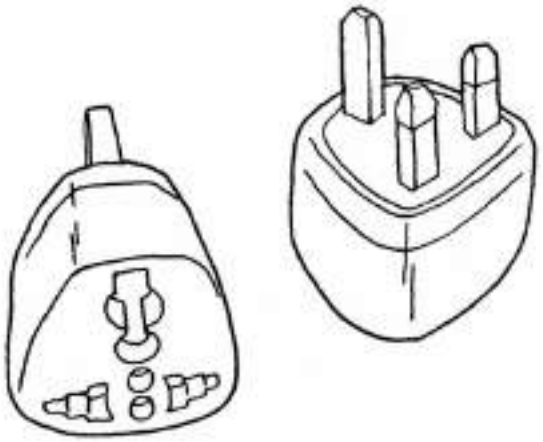
适配器的别名是包装器（wrapper），这是一个相对简单的模式。在程序开发中有许多这样的场景：当我们试图调用模块或者对象的某个接口时，却发现这个接口的格式并不符合目前的需求。这时候有两种解决办法，第一种是修改原来的接口实现，但如果原来的模块很复杂，或者我们拿到的模块是一段别人编写的经过压缩的代码，修改原接口就显得不太现实了。第二种办法是创建一个适配器，将原接口转换为客户希望的另一个接口，客户只需要和适配器打交道。

17.1　现实中的适配器

适配器在现实生活的应用非常广泛，接下来我们来看几个现实生活中的适配器模式。

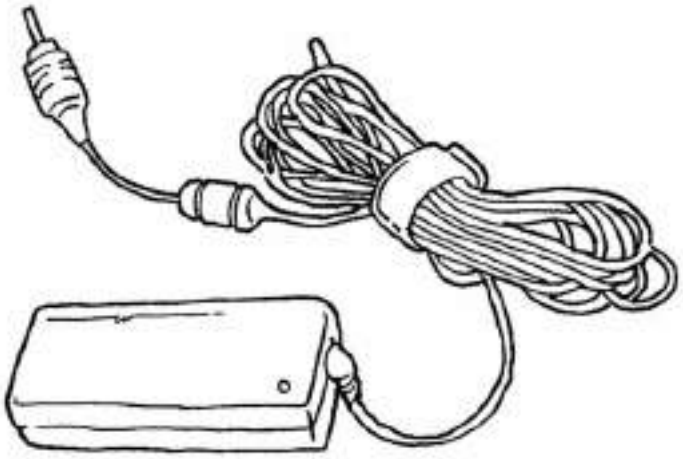
1. 港式插头转换器

港式的电器插头比大陆的电器插头体积要大一些。如果从香港买了一个Mac book，我们会发现充电器无法插在家里的插座上，为此而改造家里的插座显然不方便，所以我们需要一个适配器：



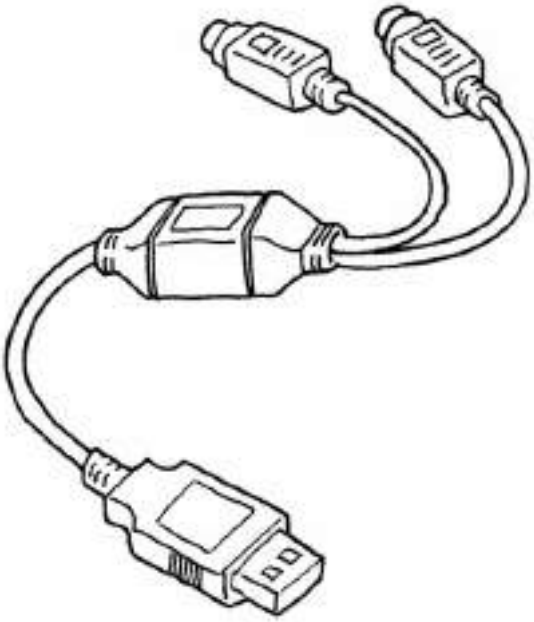
2. 电源适配器

Mac book电池支持的电压是20V，我们日常生活中的交流电压一般是220V。除了我们了解的220V交流电压，日本和韩国的交流电压大多是100V，而英国和澳大利亚的是240V。笔记本电脑的电源适配器就承担了转换电压的作用，电源适配器使笔记本电脑在100V~240V的电压之内都能正常工作，这也是它为什么被称为电源“适配器”的原因。



3. USB转接口

在以前的电脑上，PS2接口是连接鼠标、键盘等其他外部设备的标准接口。但随着技术的发展，越来越多的电脑开始放弃了PS2接口，转而仅支持USB接口。所以那些过去生产出来的只拥有PS2接口的鼠标、键盘、游戏手柄等，需要一个USB转接口才能继续正常工作，这是PS2-USB适配器诞生的原因。



17.2 适配器模式的应用

如果现有的接口已经能够正常工作，那我们就永远不会用上适配器模式。适配器模式是一种“亡羊补牢”的模式，没有人会在程序的设计之初就使用它。因为没有人可以完全预料到未来的事情，也许现在好好工作的接口，未来的某天却不再适用于新系统，那么我们可以用适配器模式把旧接口包装成一个新的接口，使它继续保持生命力。比如在JSON格式流行之前，很多cgi返回的都是XML格式的数据，如果今天仍然想继续使用这些接口，显然我们可以创造一个XML-JSON的适配器。

下面这个实例可以帮助我们深刻了解适配器模式。

回忆1.3节中多态的例子，当我们向googleMap 和baiduMap 都发出“显示”请求时，googleMap 和baiduMap 分别以各自的方式在页面中展现了地图：

```
var googleMap = {
    show: function(){
        console.log( '开始渲染谷歌地图' );
    }
};

var baiduMap = {
    show: function(){
        console.log( '开始渲染百度地图' );
    }
};

var renderMap = function( map ){
    if ( map.show instanceof Function ){
        map.show();
    }
};

renderMap( googleMap );     // 输出：开始渲染谷歌地图
renderMap( baiduMap );     // 输出：开始渲染百度地图
```

这段程序得以顺利运行的关键是googleMap 和baiduMap 提供了一致的

show 方法，但第三方的接口方法并不在我们自己的控制范围之内，假如 baiduMap 提供的显示地图的方法不叫show 而叫display 呢?

baiduMap 这个对象来源于第三方，正常情况下我们都不应该去改动它。此时我们可以通过增加baiduMapAdapter 来解决问题：

```
var googleMap = {
     show: function(){
         console.log( '开始渲染谷歌地图' );
     }
 };

var baiduMap = {
     display: function(){
         console.log( '开始渲染百度地图' );
     }
 };

var baiduMapAdapter = {
     show: function(){
         return baiduMap.display();
     }
 };

renderMap( googleMap );          // 输出：开始渲染谷歌地图
renderMap( baiduMapAdapter );    // 输出：开始渲染百度地图
```

再来看看另外一个例子。假设我们正在编写一个渲染广东省地图的页面。目前从第三方资源里获得了广东省的所有城市以及它们所对应的ID，并且成功地渲染到页面中：

```
var getGuangdongCity = function(){
    var guangdongCity = [
         {
            name: 'shenzhen',
            id: 11,
        }, {
            name: 'guangzhou',
            id: 12,
        }
    ];
```

```
    return guangdongCity;
};

var render = function( fn ){
    console.log( '开始渲染广东省地图' );
    document.write( JSON.stringify( fn() ) );
};

render( getGuangdongCity );
```

利用这些数据，我们编写完成了整个页面，并且在线上稳定地运行了一段时间。但后来发现这些数据不太可靠，里面还缺少很多城市。于是我们又在网上找到了另外一些数据资源，这次的数据更加全面，但遗憾的是，数据结构和正运行在项目中的并不一致。新的数据结构如下：

```
var guangdongCity = {
    shenzhen: 11,
    guangzhou: 12,
    zhuhai: 13
};
```

除了大动干戈地改写渲染页面的前端代码之外，另外一种更轻便的解决方式就是新增一个数据格式转换的适配器：

```
var getGuangdongCity = function(){
    var guangdongCity = [
         {
            name: 'shenzhen',
            id: 11,
        }, {
            name: 'guangzhou',
            id: 12,
        }
    ];

    return guangdongCity;
};
```

```
var render = function( fn ){
    console.log( '开始渲染广东省地图' );
    document.write( JSON.stringify( fn() ) );
};

var addressAdapter = function( oldAddressfn ){

    var address = {},
        oldAddress = oldAddressfn();

    for ( var i = 0, c; c = oldAddress[ i++ ]; ){
        address[ c.name ] = c.id;
    }

     return function(){
         return address;
     }
};


render( addressAdapter( getGuangdongCity ) );
```

那么接下来需要做的，就是把代码中调用getGuangdongCity 的地方，用经过addressAdapter 适配器转换之后的新函数来代替。

17.3 小结

适配器模式是一对相对简单的模式。在本书提到的设计模式中，有一些模式跟适配器模式的结构非常相似，比如装饰者模式、代理模式和外观模式（参见第19章）。这几种模式都属于“包装模式”，都是由一个对象来包装另一个对象。区别它们的关键仍然是模式的意图。

- 适配器模式主要用来解决两个已有接口之间不匹配的问题，它不考虑这些接口是怎样实现的，也不考虑它们将来可能会如何演化。适配器模式不需要改变已有的接口，就能够使它们协同作用。

- 装饰者模式和代理模式也不会改变原有对象的接口，但装饰者模式的作用是为了给对象增加功能。装饰者模式常常形成一条长的装饰链，而适配器模式通常只包装一次。代理模式是为了控制对对象的访问，通常也只包装一次。

- 外观模式的作用倒是和适配器比较相似，有人把外观模式看成一组对象的适配器，但外观模式最显著的特点是定义了一个新的接口。