**介绍**

外观模式（Facade）为子系统中的一组接口提供了一个一致的界面，此模块定义了一个高层接口，这个接口值得这一子系统更加容易使用。

**正文**

外观模式不仅简化类中的接口，而且对接口与调用者也进行了解耦。外观模式经常被认为开发者必备，它可以将一些复杂操作封装起来，并创建一个简单的接口用于调用。

外观模式经常被用于JavaScript类库里，通过它封装一些接口用于兼容多浏览器，外观模式可以让我们间接调用子系统，从而避免因直接访问子系统而产生不必要的错误。

外观模式的优势是易于使用，而且本身也比较轻量级。但也有缺点 外观模式被开发者连续使用时会产生一定的性能问题，因为在每次调用时都要检测功能的可用性。

下面是一段未优化过的代码，我们使用了外观模式通过检测浏览器特性的方式来创建一个跨浏览器的使用方法。

var addMyEvent = function (el, ev, fn) {  
 if (el.addEventListener) {  
 el.addEventListener(ev, fn, false);  
 } else if (el.attachEvent) {  
 el.attachEvent('on' + ev, fn);  
 } else {  
 el['on' + ev] = fn;  
 }  
};

再来一个简单的例子，说白了就是用一个接口封装其它的接口：

var mobileEvent = {  
 // ...  
 stop: function (e) {  
 e.preventDefault();  
 e.stopPropagation();  
 }  
 // ...  
};

**总结**

那么何时使用外观模式呢？一般来说分三个阶段：

**首先**，在设计初期，应该要有意识地将不同的两个层分离，比如经典的三层结构，在数据访问层和业务逻辑层、业务逻辑层和表示层之间建立外观Facade。

**其次**，在开发阶段，子系统往往因为不断的重构演化而变得越来越复杂，增加外观Facade可以提供一个简单的接口，减少他们之间的依赖。

**第三**，在维护一个遗留的大型系统时，可能这个系统已经很难维护了，这时候使用外观Facade也是非常合适的，为系系统开发一个外观Facade类，为设计粗糙和高度复杂的遗留代码提供比较清晰的接口，让新系统和Facade对象交互，Facade与遗留代码交互所有的复杂工作。