**介绍**

适配器模式（Adapter）是将一个类（对象）的接口（方法或属性）转化成客户希望的另外一个接口（方法或属性），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（对象）可以一些工作。速成包装器（wrapper）。

**正文**

我们来举一个例子，鸭子（Dock）有飞（fly）和嘎嘎叫（quack）的行为，而火鸡虽然也有飞（fly）的行为，但是其叫声是咯咯的（gobble）。如果你非要火鸡也要实现嘎嘎叫（quack）这个动作，那我们可以复用鸭子的quack方法，但是具体的叫还应该是咯咯的，此时，我们就可以创建一个火鸡的适配器，以便让火鸡也支持quack方法，其内部还是要调用gobble。

OK，我们开始一步一步实现，首先要先定义鸭子和火鸡的抽象行为，也就是各自的方法函数：

//鸭子  
var Duck = function(){  
  
};  
Duck.prototype.fly = function(){  
throw new Error("该方法必须被重写!");  
};  
Duck.prototype.quack = function(){  
throw new Error("该方法必须被重写!");  
}  
  
//火鸡  
var Turkey = function(){  
  
};  
Turkey.prototype.fly = function(){  
 throw new Error(" 该方法必须被重写 !");  
};  
Turkey.prototype.gobble = function(){  
 throw new Error(" 该方法必须被重写 !");  
};

然后再定义具体的鸭子和火鸡的构造函数，分别为：

//鸭子  
var MallardDuck = function () {  
 Duck.apply(this);  
};  
MallardDuck.prototype = new Duck(); //原型是Duck  
MallardDuck.prototype.fly = function () {  
 console.log("可以飞翔很长的距离!");  
};  
MallardDuck.prototype.quack = function () {  
 console.log("嘎嘎！嘎嘎！");  
};  
  
//火鸡  
var WildTurkey = function () {  
 Turkey.apply(this);  
};  
WildTurkey.prototype = new Turkey(); //原型是Turkey  
WildTurkey.prototype.fly = function () {  
 console.log("飞翔的距离貌似有点短!");  
};  
WildTurkey.prototype.gobble = function () {  
 console.log("咯咯！咯咯！");  
};

为了让火鸡也支持quack方法，我们创建了一个新的火鸡适配器TurkeyAdapter：

var TurkeyAdapter = function(oTurkey){  
 Duck.apply(this);  
 this.oTurkey = oTurkey;  
};  
TurkeyAdapter.prototype = new Duck();  
TurkeyAdapter.prototype.quack = function(){  
 this.oTurkey.gobble();  
};  
TurkeyAdapter.prototype.fly = function(){  
 var nFly = 0;  
 var nLenFly = 5;  
 for(; nFly < nLenFly;){  
 this.oTurkey.fly();  
 nFly = nFly + 1;  
 }  
};

该构造函数接受一个火鸡的实例对象，然后使用Duck进行apply，其适配器原型是Duck，然后要重新修改其原型的quack方法，以便内部调用oTurkey.gobble()方法。其fly方法也做了一些改变，让火鸡连续飞5次（内部也是调用自身的oTurkey.fly()方法）。

调用方法，就很明了了，测试一下便可以知道结果了：

var oMallardDuck = new MallardDuck();  
var oWildTurkey = new WildTurkey();  
var oTurkeyAdapter = new TurkeyAdapter(oWildTurkey);  
  
//原有的鸭子行为  
oMallardDuck.fly();  
oMallardDuck.quack();  
  
//原有的火鸡行为  
oWildTurkey.fly();  
oWildTurkey.gobble();  
  
//适配器火鸡的行为（火鸡调用鸭子的方法名称）  
oTurkeyAdapter.fly();  
oTurkeyAdapter.quack();

**总结**

那合适使用适配器模式好呢？如果有以下情况出现时，建议使用：

1. 使用一个已经存在的对象，但其方法或属性接口不符合你的要求；
2. 你想创建一个可复用的对象，该对象可以与其它不相关的对象或不可见对象（即接口方法或属性不兼容的对象）协同工作；
3. 想使用已经存在的对象，但是不能对每一个都进行原型继承以匹配它的接口。对象适配器可以适配它的父对象接口方法或属性。

另外，适配器模式和其它几个模式可能容易让人迷惑，这里说一下大概的区别：

1. 适配器和桥接模式虽然类似，但桥接的出发点不同，桥接的目的是将接口部分和实现部分分离，从而对他们可以更为容易也相对独立的加以改变。而适配器则意味着改变一个已有对象的接口。
2. 装饰者模式增强了其它对象的功能而同时又不改变它的接口，因此它对应程序的透明性比适配器要好，其结果是装饰者支持递归组合，而纯粹使用适配器则是不可能的。
3. 代理模式在不改变它的接口的条件下，为另外一个对象定义了一个代理。

参考：https://github.com/tcorral/Design-Patterns-in-Javascript/blob/master/Adapter/index.html