**介绍**

中介者模式（Mediator），用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。

主要内容来自：http://www.addyosmani.com/resources/essentialjsdesignpatterns/book/#mediatorpatternjavascript

**正文**

软件开发中，中介者是一个行为设计模式，通过提供一个统一的接口让系统的不同部分进行通信。一般，如果系统有很多子模块需要直接沟通，都要创建一个中央控制点让其各模块通过该中央控制点进行交互。中介者模式可以让这些子模块不需要直接沟通，而达到进行解耦的目的。

打个比方，平时常见的机场交通控制系统，塔台就是中介者，它控制着飞机（子模块）的起飞和降落，因为所有的沟通都是从飞机向塔台汇报来完成的，而不是飞机之前相互沟通。中央控制系统就是该系统的关键，也就是软件设计中扮演的中介者角色。

我们先用伪代码来理解一下：

// 如下代码是伪代码，请不要过分在意代码  
// 这里app命名空间就相当于扮演中介者的角色  
var app = app || {};  
   
// 通过app中介者来进行Ajax请求  
app.sendRequest = function ( options ) {  
 return $.ajax($.extend({}, options);  
}  
   
// 请求URL以后，展示View  
app.populateView = function( url, view ){  
 $.when(app.sendRequest({url: url, method: 'GET'})  
 .then(function(){  
 //显示内容  
 });  
}  
   
// 清空内容  
app.resetView = function( view ){  
 view.html('');  
}

在JavaScript里，中介者非常常见，相当于观察者模式上的消息Bus，只不过不像观察者那样通过调用pub/sub的形式来实现，而是通过中介者统一来管理，让我们在观察者的基础上来给出一个例子：

var mediator = (function () {  
 // 订阅一个事件，并且提供一个事件触发以后的回调函数  
 var subscribe = function (channel, fn) {  
 if (!mediator.channels[channel]) mediator.channels[channel] = [];  
 mediator.channels[channel].push({ context: this, callback: fn });  
 return this;  
 },  
  
 // 广播事件  
 publish = function (channel) {  
 if (!mediator.channels[channel]) return false;  
 var args = Array.prototype.slice.call(arguments, 1);  
 for (var i = 0, l = mediator.channels[channel].length; i < l; i++) {  
 var subscription = mediator.channels[channel][i];  
 subscription.callback.apply(subscription.context, args);  
 }  
 return this;  
 };  
  
 return {  
 channels: {},  
 publish: publish,  
 subscribe: subscribe,  
 installTo: function (obj) {  
 obj.subscribe = subscribe;  
 obj.publish = publish;  
 }  
 };  
  
} ());

调用代码，相对就简单了：

(function (Mediator) {  
  
 function initialize() {  
  
 // 默认值  
 mediator.name = "dudu";  
  
 // 订阅一个事件nameChange  
 // 回调函数显示修改前后的信息  
 mediator.subscribe('nameChange', function (arg) {  
 console.log(this.name);  
 this.name = arg;  
 console.log(this.name);  
 });  
 }  
  
 function updateName() {  
 // 广播触发事件，参数为新数据  
 mediator.publish('nameChange', 'tom'); // dudu, tom  
 }  
  
 initialize(); // 初始化  
 updateName(); // 调用  
  
})(mediator);

**中介者和观察者**

到这里，大家可能迷糊了，中介者和观察者貌似差不多，有什么不同呢？其实是有点类似，但是我们来看看具体的描述：  
观察者模式，没有封装约束的单个对象，相反，观察者Observer和具体类Subject是一起配合来维护约束的，沟通是通过多个观察者和多个具体类来交互的：每个具体类通常包含多个观察者，而有时候具体类里的一个观察者也是另一个观察者的具体类。

而中介者模式所做的不是简单的分发，却是扮演着维护这些约束的职责。

**中介者和外观模式**

很多人可能也比较迷糊中介者和外观模式的区别，他们都是对现有各模块进行抽象，但有一些微妙的区别。

中介者所做的是在模块之间进行通信，是多向的，但外观模式只是为某一个模块或系统定义简单的接口而不添加额外的功能。系统中的其它模块和外观模式这个概念没有直接联系，可以认为是单向性。

**完整的例子**

再给出一个完整的例子：

<!doctype html>  
<html lang="en">  
<head>  
 <title>JavaScript Patterns</title>  
 <meta charset="utf-8">  
</head>  
<body>  
<div id="results"></div>  
 <script>  
 function Player(name) {  
 this.points = 0;  
 this.name = name;  
 }  
 Player.prototype.play = function () {  
 this.points += 1;  
 mediator.played();  
 };  
 var scoreboard = {  
  
 // 显示内容的容器  
 element: document.getElementById('results'),  
  
 // 更新分数显示  
 update: function (score) {  
 var i, msg = '';  
 for (i in score) {  
 if (score.hasOwnProperty(i)) {  
 msg += '<p><strong>' + i + '<\/strong>: ';  
 msg += score[i];  
 msg += '<\/p>';  
 }  
 }  
 this.element.innerHTML = msg;  
 }  
 };  
  
 var mediator = {  
  
 // 所有的player  
 players: {},  
  
 // 初始化  
 setup: function () {  
 var players = this.players;  
 players.home = new Player('Home');  
 players.guest = new Player('Guest');  
 },  
  
 // play以后，更新分数  
 played: function () {  
 var players = this.players,  
 score = {  
 Home: players.home.points,  
 Guest: players.guest.points  
 };  
  
 scoreboard.update(score);  
 },  
  
 // 处理用户按键交互  
 keypress: function (e) {  
 e = e || window.event; // IE  
 if (e.which === 49) { // 数字键 "1"  
 mediator.players.home.play();  
 return;  
 }  
 if (e.which === 48) { // 数字键 "0"  
 mediator.players.guest.play();  
 return;  
 }  
 }  
 };  
  
 // go!  
 mediator.setup();  
 window.onkeypress = mediator.keypress;  
  
 // 30秒以后结束  
 setTimeout(function () {  
 window.onkeypress = null;  
 console.log('Game over!');  
 }, 30000);  
 </script>  
</body>  
</html>

**总结**

中介者模式一般应用于一组对象已定义良好但是以复杂的方式进行通信的场合，一般情况下，中介者模式很容易在系统中使用，但也容易在系统里误用，当系统出现了多对多交互复杂的对象群时，先不要急于使用中介者模式，而是要思考一下是不是系统设计有问题。

另外，由于中介者模式把交互复杂性变成了中介者本身的复杂性，所以说中介者对象会比其它任何对象都复杂。