[设计模式之观察者模式](http://www.cnblogs.com/TomXu/archive/2012/03/02/2355128.html)

**介绍**

观察者模式又叫发布订阅模式（Publish/Subscribe），它定义了一种一对多的关系，让多个观察者对象同时监听某一个主题对象，这个主题对象的状态发生变化时就会通知所有的观察者对象，使得它们能够自动更新自己。

使用观察者模式的好处：

1. 支持简单的广播通信，自动通知所有已经订阅过的对象。
2. 页面载入后目标对象很容易与观察者存在一种动态关联，增加了灵活性。
3. 目标对象与观察者之间的抽象耦合关系能够单独扩展以及重用。

**正文（版本一）**

JS里对观察者模式的实现是通过回调来实现的，我们来先定义一个pubsub对象，其内部包含了3个方法：订阅、退订、发布。

var pubsub = {};  
(function (q) {  
  
 var topics = {}, // 回调函数存放的数组  
 subUid = -1;  
 // 发布方法  
 q.publish = function (topic, args) {  
  
 if (!topics[topic]) {  
 return false;  
 }  
  
 setTimeout(function () {  
 var subscribers = topics[topic],  
 len = subscribers ? subscribers.length : 0;  
  
 while (len--) {  
 subscribers[len].func(topic, args);  
 }  
 }, 0);  
  
 return true;  
  
 };  
 //订阅方法  
 q.subscribe = function (topic, func) {  
  
 if (!topics[topic]) {  
 topics[topic] = [];  
 }  
  
 var token = (++subUid).toString();  
 topics[topic].push({  
 token: token,  
 func: func  
 });  
 return token;  
 };  
 //退订方法  
 q.unsubscribe = function (token) {  
 for (var m in topics) {  
 if (topics[m]) {  
 for (var i = 0, j = topics[m].length; i < j; i++) {  
 if (topics[m][i].token === token) {  
 topics[m].splice(i, 1);  
 return token;  
 }  
 }  
 }  
 }  
 return false;  
 };  
} (pubsub));

使用方式如下：

//来，订阅一个  
pubsub.subscribe('example1', function (topics, data) {  
 console.log(topics + ": " + data);  
});

//发布通知  
pubsub.publish('example1', 'hello world!');  
pubsub.publish('example1', ['test', 'a', 'b', 'c']);  
pubsub.publish('example1', [{ 'color': 'blue' }, { 'text': 'hello'}]);

怎么样？用起来是不是很爽？但是这种方式有个问题，就是没办法退订订阅，要退订的话必须指定退订的名称，所以我们再来一个版本：

//将订阅赋值给一个变量，以便退订  
var testSubscription = pubsub.subscribe('example1', function (topics, data) {  
 console.log(topics + ": " + data);  
});  
  
//发布通知  
pubsub.publish('example1', 'hello world!');  
pubsub.publish('example1', ['test', 'a', 'b', 'c']);  
pubsub.publish('example1', [{ 'color': 'blue' }, { 'text': 'hello'}]);  
  
//退订  
setTimeout(function () {  
 pubsub.unsubscribe(testSubscription);  
}, 0);  
  
//再发布一次，验证一下是否还能够输出信息  
pubsub.publish('example1', 'hello again! (this will fail)');

**版本二**

我们也可以利用原型的特性实现一个观察者模式，代码如下：

function Observer() {  
 this.fns = [];  
}  
Observer.prototype = {  
 subscribe: function (fn) {  
 this.fns.push(fn);  
 },  
 unsubscribe: function (fn) {  
 this.fns = this.fns.filter(  
 function (el) {  
 if (el !== fn) {  
 return el;  
 }  
 }  
 );  
 },  
 update: function (o, thisObj) {  
 var scope = thisObj || window;  
 this.fns.forEach(  
 function (el) {  
 el.call(scope, o);  
 }  
 );  
 }  
};  
  
//测试  
var o = new Observer;  
var f1 = function (data) {  
 console.log('Robbin: ' + data + ', 赶紧干活了！');  
};  
  
var f2 = function (data) {  
 console.log('Randall: ' + data + ', 找他加点工资去！');  
};  
  
o.subscribe(f1);  
o.subscribe(f2);  
  
o.update("Tom回来了！")  
  
//退订f1  
o.unsubscribe(f1);  
//再来验证  
o.update("Tom回来了！");

如果提示找不到filter或者forEach函数，可能是因为你的浏览器还不够新，暂时不支持新标准的函数，你可以使用如下方式自己定义：

if (!Array.prototype.forEach) {  
 Array.prototype.forEach = function (fn, thisObj) {  
 var scope = thisObj || window;  
 for (var i = 0, j = this.length; i < j; ++i) {  
 fn.call(scope, this[i], i, this);  
 }  
 };  
}  
if (!Array.prototype.filter) {  
 Array.prototype.filter = function (fn, thisObj) {  
 var scope = thisObj || window;  
 var a = [];  
 for (var i = 0, j = this.length; i < j; ++i) {  
 if (!fn.call(scope, this[i], i, this)) {  
 continue;  
 }  
 a.push(this[i]);  
 }  
 return a;  
 };  
}

**版本三**

如果想让多个对象都具有观察者发布订阅的功能，我们可以定义一个通用的函数，然后将该函数的功能应用到需要观察者功能的对象上，代码如下：

//通用代码  
var observer = {  
 //订阅  
 addSubscriber: function (callback) {  
 this.subscribers[this.subscribers.length] = callback;  
 },  
 //退订  
 removeSubscriber: function (callback) {  
 for (var i = 0; i < this.subscribers.length; i++) {  
 if (this.subscribers[i] === callback) {  
 delete (this.subscribers[i]);  
 }  
 }  
 },  
 //发布  
 publish: function (what) {  
 for (var i = 0; i < this.subscribers.length; i++) {  
 if (typeof this.subscribers[i] === 'function') {  
 this.subscribers[i](what);  
 }  
 }  
 },  
 // 将对象o具有观察者功能  
 make: function (o) {   
 for (var i in this) {  
 o[i] = this[i];  
 o.subscribers = [];  
 }  
 }  
};

然后订阅2个对象blogger和user，使用observer.make方法将这2个对象具有观察者功能，代码如下：

var blogger = {  
 recommend: function (id) {  
 var msg = 'dudu 推荐了的帖子:' + id;  
 this.publish(msg);  
 }  
};  
  
var user = {  
 vote: function (id) {  
 var msg = '有人投票了!ID=' + id;  
 this.publish(msg);  
 }  
};  
  
observer.make(blogger);  
observer.make(user);

使用方法就比较简单了，订阅不同的回调函数，以便可以注册到不同的观察者对象里（也可以同时注册到多个观察者对象里）：

var tom = {  
 read: function (what) {  
 console.log('Tom看到了如下信息：' + what)  
 }  
};  
  
var mm = {  
 show: function (what) {  
 console.log('mm看到了如下信息：' + what)  
 }  
};  
// 订阅  
blogger.addSubscriber(tom.read);  
blogger.addSubscriber(mm.show);  
blogger.recommend(123); //调用发布  
  
//退订  
blogger.removeSubscriber(mm.show);  
blogger.recommend(456); //调用发布  
  
//另外一个对象的订阅  
user.addSubscriber(mm.show);  
user.vote(789); //调用发布

**jQuery版本**

根据jQuery1.7版新增的on/off功能，我们也可以定义jQuery版的观察者：

(function ($) {  
  
 var o = $({});  
  
 $.subscribe = function () {  
 o.on.apply(o, arguments);  
 };  
  
 $.unsubscribe = function () {  
 o.off.apply(o, arguments);  
 };  
  
 $.publish = function () {  
 o.trigger.apply(o, arguments);  
 };  
  
} (jQuery));

调用方法比上面3个版本都简单：

//回调函数  
function handle(e, a, b, c) {  
 // `e`是事件对象，不需要关注  
 console.log(a + b + c);  
};  
  
//订阅  
$.subscribe("/some/topic", handle);  
//发布  
$.publish("/some/topic", ["a", "b", "c"]); // 输出abc  
   
  
$.unsubscribe("/some/topic", handle); // 退订  
  
//订阅  
$.subscribe("/some/topic", function (e, a, b, c) {  
 console.log(a + b + c);  
});  
  
$.publish("/some/topic", ["a", "b", "c"]); // 输出abc  
  
//退订（退订使用的是/some/topic名称，而不是回调函数哦，和版本一的例子不一样  
$.unsubscribe("/some/topic");

可以看到，他的订阅和退订使用的是字符串名称，而不是回调函数名称，所以即便传入的是匿名函数，我们也是可以退订的。

**总结**

观察者的使用场合就是：当一个对象的改变需要同时改变其它对象，并且它不知道具体有多少对象需要改变的时候，就应该考虑使用观察者模式。

总的来说，观察者模式所做的工作就是在解耦，让耦合的双方都依赖于抽象，而不是依赖于具体。从而使得各自的变化都不会影响到另一边的变化。

参考地址：

https://github.com/shichuan/javascript-patterns/blob/master/design-patterns/observer.html

http://www.addyosmani.com/resources/essentialjsdesignpatterns/book/#observerpatternjavascript

https://gist.github.com/661855