# jquery控件编写说明

## jiraBox控件的完整示例代码：

;(**function**($){  
  
 $.**fn**.jiraBox = **function**(options, param){  
 **if**((**typeof** options) === **"string"**){  
 **var** method = $.**fn**.jiraBox.**methods**[options];  
 **if** (method){  
 **return** method(**this**, param);  
 } **else** {  
 alert(**"没有"** + options + **"方法"**);  
 }  
 }**else if**(  
 (**typeof** options) === **"object"** &&  
 Object.**prototype**.toString.call(options).toLowerCase() == **"[object object]"** &&  
 !options.**length** ){  
 **return this**.each(**function**(){  
 **var** state = $.data(**this**, **'jiraBox'**);  
 **if**(state){  
 $.extend(state.**options**, options);  
 }**else**{  
 $.data(**this**, **'jiraBox'**, {  
 **options**: $.extend({}, $.**fn**.jiraBox.**defaults**, options),  
 **methods**: $.extend({}, $.**fn**.jiraBox.**methods**)  
 });  
 state = $.data(**this**, **'jiraBox'**);  
 state.**options**.init($(**this**));  
 }  
 });  
 }**else**{  
 alert(**"参数错误"**);  
 }  
 **return this**;  
 };  
  
 $.**fn**.jiraBox.**methods** = {  
 options: **function**($elem){},  
 getValue: **function**($elem){},  
 setValue: **function**($elem, data){},  
 loadData: **function**($elem, data){},  
 edit: **function**($elem){},  
 submit: **function**($elem){},  
 cancel: **function**($elem){},  
 enable: **function**($elem){},  
 disable: **function**($elem){},  
 showError: **function**($elem, msg){},  
 hideError: **function**($elem){},  
 showLoading: **function**($elem){},  
 hideLoading: **function**($elem){}  
 };  
  
 $.**fn**.jiraBox.**defaults** = {  
 *//各种属性、方法* **originalData**: [],  
 **data**: [],  
 **allData**: [],  
 **url**: **""**,  
 init: **function**($elem){},  
 autoSubmit: **function**($elem){},  
 submit: **function**($elem){},  
 cancel: **function**($elem){},  
 edit: **function**($elem){},  
 showList: **function**($elem){},  
 restore: **function**($elem){},  
 scrollDownHandler: **function**($elem){},  
 scrollUpHandler: **function**($elem){},  
 **render**: {  
 renderJiraBox: **function**($elem){},  
 renderEdit: **function**($elem){},  
 reRenderCursor: **function**($elem){},  
 reRenderBoxUl: **function**($elem){},  
 reRenderSelectUl: **function**($elem){}  
 },  
 **eventListener**: {  
 aClick: **function**(e){},  
 bindEditEvent: **function**(e){},  
 submitByEvent: **function**(e){},  
 cancelByEvent: **function**(e){},  
 showListByEvent: **function**(e){},  
 selectByEvent: **function**(e){},  
 deleteSelectedByEvent: **function**(e){},  
 inputPropertyChangeListener: **function**(e){},  
 keydownListener: **function**(e){},  
 windowClickHandle: **function**(e){}  
 }  
 };  
  
})(**jQuery**);

以上就是jquery开发组件的基本模板。

## 详细步骤说明

### 1、准备架子

;(function($){

})(jQuery);

说明：（1）在jquery环境下封装自己的插件，首先为避免与其他库的冲突，需要在插件的后面传一个jQuery参数进去，对应的函数里面的参数写入$；

（2）为避免出现问题，需在插件的前后加入分号（分号的增加并不会影响程序的运行），例如：

var testFunction=function(){

}

(function(){ //前面没有分号，这个时候就会报错

})()

### 2、链式操作

通常我们都是希望通过jquery对象来初始化组件，如 $("#a").pluginName(param); ，所以我们可以向jquery添加新的方法，方法名叫pluginName，这样我们就可以这样写：

$.fn.pluginName = function(param){

...

return this;

}

然后，很多时候我们使用jquery都喜欢使用它的链式操作，如 $("#a").html("xx").css("xx","xx"); ，所以，我们要为自己编写的插件返回jquery对象 return this;

### 3、初始化操作

（1）我们的jquery对象可能长度会大于1，如使用类选择器 $(".class")，所以我们通常会对每一个选择的元素都执行相同的操作，再返回jquery对象。

return this.each(function(){

...

})

通常会在这里为每一个元素编写初始化的内容，这里的内容视插件而言。对于jiraBox，是模仿了easyui的方法，把有用的数据存进元素的data属性中，就能达到元素与数据绑定，看一看easyui的部分源码：

$.**fn**.combobox = **function**(options, param){  
 **if** (**typeof** options == **'string'**){  
 **var** method = $.**fn**.combobox.methods[options];  
 **if** (method){  
 **return** method(**this**, param);  
 } **else** {  
 **return this**.combo(options, param);  
 }  
 }  
   
 options = options || {};  
 **return this**.each(**function**(){  
 **var** state = $.data(**this**, **'combobox'**);  
 **if** (state){  
 $.extend(state.**options**, options);  
 } **else** {  
 state = $.data(**this**, **'combobox'**, {  
 **options**: $.extend({}, $.**fn**.combobox.**defaults**, $.**fn**.combobox.parseOptions(**this**), options),  
 **data**: []  
 });  
 }  
 *create*(**this**);  
 **if** (state.**options**.**data**){  
 *loadData*(**this**, state.**options**.**data**);  
 } **else** {  
 **var** data = $.**fn**.combobox.parseData(**this**);  
 **if** (data.**length**){  
 *loadData*(**this**, data);  
 }  
 }  
 *request*(**this**);  
 });  
};

它的create(this)及以下的内容都可以看作为init初始化内容，这里由开发者自己实现。

（2）从下面代码看到，初始化的时候会先判断它是否具有pluginName属性，有就直接合并新的options，没有就赋pluginName属性。

**var** state = $.data(**this**, **'pluginName'**);  
 **if** (state){  
 $.extend(state.**options**, options);  
 } **else** {  
 state = $.data(**this**, **'pluginName'**, {  
 **options**: $.extend({}, $.**fn**.combobox.**defaults**, options),  
 **data**: []  
 });  
 }

例如easyui中的combobox，我们会这样来使用：

$("#a").combobox({

valueField: "v",

textField: "t",

onBeforeLoad: function(){},

onLoadSuccess: function(){}

})

然后在初始化的时候，就会把属性和方法替换原来的（$.fn.combobox.defaults）属性和方法，这样控件就能执行用户的自定义函数。

控件内部可以这样来处理：

*//确定按钮事件*$jq.**find**(**".xxx-submit"**).on(**"click"**, state.**options**.onBeforeLoad);

然后如果用户在options中提供onBeforeLoad，在合并之后，这个监听事件就是执行用户的自定义事件。

注意：这样合并是只要用户有提供，都会合并，可能会产生某个方法并不希望用户改变但被用户修改了导致控件无法正常运行。如easyui的combobox文档，并没有提供finder属性，但在defaults中有一个finder的属性，如果用户不遵循文档或者拼写错误，$("#a").combobox({finder: "xx"})把属性修改了，就会导致控件报错并无法正常运行。

解决：可以把defaults分成两个，一个是一一对应文档的，用户可以自定义替换，即直接合并；另一个是不被替换的属性和方法对象。

（3）**关于合并的说明：**例如出现这样的一个现象，用户在options中设置onLoad，然后组件默认的onLoad事件是具有一系列的样式效果如显示加载中，这样如果用户自定义的函数直接替换了原来的onLoad，就会导致用户不知道怎么处理相关的样式，所以在设计上需要根据实际情况进行更改。下面是easyui的示例：

//先在defaults中定义空函数

onBeforeLoad: **function**(param){},  
 onLoadSuccess: **function**(data){},

...

easyui把提供自定义处理的函数都设置为空，不把控件的默认处理写在里面，然后在某个函数中根据逻辑进行调用：

*/\*\*  
 \* request remote data if the url property is setted.  
 \*/* **function** *request*(target, url, param, remainText){  
 **var** opts = $.data(target, **'combobox'**).**options**;  
 **if** (url){  
 opts.**url** = url;  
 }  
 param = $.extend({}, opts.**queryParams**, param||{});  
*// param = param || {};* **if** (opts.onBeforeLoad.call(target, param) == **false**) **return**;  
  
 opts.loader.call(target, param, **function**(data){  
 *loadData*(target, data, remainText);  
 }, **function**(){  
 opts.onLoadError.apply(**this**, arguments);  
 });  
 }

可以理解为不把request作为options中的属性，而是把request方法中用到的onBeforeLoad、onLoadError等作为options中的属性，这样保留控件原有的处理并执行默认的空方法或用户替换后的方法，用户可以在自定义函数中通过返回false来阻止控件的默认行为。

根据easyui的例子，我们可以把之前提到的 $jq.**find**(**".xxx-submit"**).on(**"click"**, state.**options**.onBeforeLoad); 进行修改:

$jq.**find**(**".xxx-submit"**).on(**"click"**, xxxFunc); //不调用options中的方法

function xxxFunc(){

.....

var state = xx.pluginName;

if(state.options.onBeforeLoad() == false){return;}

.....

}

easyui也有直接替换的情况，但它似乎是做了相应的分类，看下图：



loader、loadFilter是属于属性一栏，onBeforeLoad是属于事件一栏，都是函数类型，属性类的用户可替换，但替换后需要自己进行处理，事件类的是替换空函数。我们看它的源码对比：

....

loader: **function**(param, success, error){  
 **var** opts = $(**this**).combobox(**'options'**);  
 **if** (!opts.**url**) **return false**;  
 $.ajax({  
 **type**: opts.**method**,  
 **url**: opts.**url**,  
 **data**: param,  
 **dataType**: **'json'**,  
 success: **function**(data){  
 success(data);  
 },  
 error: **function**(){  
 error.**apply**(**this**, arguments);  
 }  
 });  
 },

.....

loadFilter: **function**(data){  
 **return** data;  
 },

....

onBeforeLoad: **function**(param){},

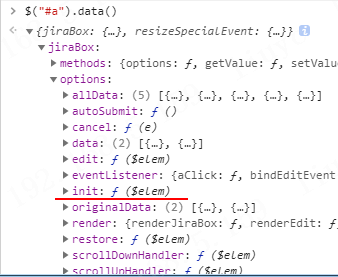
....

最后，我们回看一下上面提到的easyui的request函数，里面有一个opts.loader.call，即用户如果提供了自己的loader属性，函数体为空，就已经破坏了正常的request行为，需要用户自行实现，而事件类的原本就为空，就算用户提供了该函数，对request正常逻辑没有影响，是看作额外的处理。综上所述，用户提供的参数中函数类型的需要根据情况，是作为直接替换还是额外的处理，对于替换，就直接函数体替换，对于额外的处理，那原来的函数体就应该为空。

### 4、控件内部使用的函数

编写组件内部的函数。我们通常把默认的函数和参数都写在$.fn.pluginName.defaults中（即本文档中[jiraBox控件的完整示例代码](#_jiraBox控件的完整示例代码：)中的$.fn.jiraBox.methods），然后通过$.extend把defaults和用户参数options合并。由于我们在[第3点](#_3、初始化操作)把合并后的数据都保存在元素的data属性中，所以我们可以获取属性值然后调用相关的函数。即这样来调用init函数：

$.data(**this**, **'jiraBox'**, {  
 **options**: $.extend({}, $.**fn**.jiraBox.**defaults**, options),  
 **methods**: $.extend({}, $.**fn**.jiraBox.**methods**)  
});  
state = $.data(**this**, **'jiraBox'**);  
state.**options**.init($(**this**));



因为把所有的信息都存进了data属性中，所以我们的方法基本传jquery对象就足够了。

### 5、控件供用户使用的函数

为组件暴露一些供用户使用的函数。我们把方法都写在 $.fn.pluginName.methods 中，这也是模仿easyui的设计。easyui中的部分源码：

$.**fn**.combobox.**methods** = { *//统一存放，然后methods[方法名]查找* options: **function**(jq){ ... },  
 cloneFrom: **function**(jq, from) { ... },  
 getData: **function**(jq) { ... },  
 setValues: **function**(jq, values) { ... },  
 setValue: **function**(jq, value) { ... },  
 clear: **function**(jq) { ... },  
 reset: **function**(jq) { ... },  
 loadData: **function**(jq, data) { ... },  
 reload: **function**(jq, url) { ... },  
 select: **function**(jq, value) { ... },  
 unselect: **function**(jq, value) { ... },  
 scrollTo: **function**(jq, value{ ... }  
};

然后我们想调用其中的方法，先 $("#a").pluginName("options"); ，让它仍是执行pluginName函数，然后在函数中判断参数是否为字符串，并且能找到对应的key，就执行该方法：

$.**fn**.combobox = **function**(options, param){  
 **if** (**typeof** options == **'string'**){  
 **var** method = $.**fn**.combobox.methods[options];  
 **if** (method){  
 **return** method(**this**, param);  
 } **else** {  
 **return this**.combo(options, param);  
 }  
 }

...

}