**译自**[**jQuery Plugins / Authoring**](http://docs.jquery.com/Plugins/Authoring)

**创建插件**

看来 jQuery 你已经用得很爽了，想学习如何自己编写插件。非常好，这篇文档正适合你。用插件和方法来扩展 jQuery 非常强大，把最聪明的功能封装到插件中可以为你及团队节省大量开发时间。

**开始**

要编写一个 jQuery 插件，需要为 jQuery.fn 对象增加一个新的函数属性，属性名就是插件的名字

jQuery.fn.myPlugin = function() {

// 插件的具体内容放在这里

};

等一下！ 我熟悉并钟爱的 $ 哪儿去了？ 它还在，只是为了确保你的插件不与其它使用 $ 的库发生冲突，有一个最佳实践： 把 jQuery 传递给 IIFE（立即调用函数），并通过它映射成 $ ，这样就避免了在执行的作用域里被其它库所覆盖。

(function( $ ) {

$.fn.myPlugin = function() {

// 插件的具体内容放在这里

};

})( jQuery );

恩，这样好一些。 现在，在此闭包内我们可以随意用 $ 替换 jQuery。

**上下文**

现在，已经有了外壳，可以开始编写真正的插件代码了。但在这之前，关于上下文我有话要说。在插件函数的立即作用域中，关键字 this 指向调用插件的 jQuery 对象。这是个经常出错的地方，因为有些情况下 jQuery 接受一个回调函数，此时 this 指向原生的 DOM 元素。这常常导致开发者在 jQuery 函数中对 this 关键字多作一次无必要的包装。

(function( $ ){

$.fn.myPlugin = function() {

// 没有必要再作 $(this) ，因为"this"已经是 jQuery 对象了

// $(this) 与 $($('#element')) 是相同的

this.fadeIn('normal', function(){

// 在这里 this 关键字指向 DOM 元素

});

};

})( jQuery );

$('#element').myPlugin();

**基础**

现在理解了 jQuery 插件的上下文以后， 我们来写一个真正能做点儿事儿的插件。

(function( $ ){

$.fn.maxHeight = function() {

var max = 0;

this.each(function() {

max = Math.max( max, $(this).height() );

});

return max;

};

})( jQuery );

--

var tallest = $('div').maxHeight(); // 返回最高 div 的高度

这个简单的插件利用 [.height()](http://docs.jquery.com/action/edit/Manipulation/height) 来返回页面中最高 div 的高度

**保持 chainability**

前面的例子返回了页面上最高 div 的一个整数值，但很多时候插件只是以某种方式修改元素集合，并把它们传给调用链的下一个方法。 这正是 jQuery 设计的漂亮之处，也是它如此流行的原因之一。为保持插件的 chainability ，必须确保插件返回 this 关键字。

(function( $ ){

$.fn.lockDimensions = function( type ) {

return this.each(function() {

var $this = $(this);

if ( !type || type == 'width' ) {

$this.width( $this.width() );

}

if ( !type || type == 'height' ) {

$this.height( $this.height() );

}

});

};

})( jQuery );

--

$('div').lockDimensions('width').css('color', 'red');

插件在立即作用域中返回了 this 关键字，保持了 chainability ，所以 jQuery 集合可以被其它方法操作，例如 [.css()](http://docs.jquery.com/CSS/css)。因此，若插件无需真正的返回值，你应该一直在插件函数的立即作用域中返回 this 关键字。同样，如你所想，调用插件时的参数会被传递到插件函数的立即作用域中。在上例中，字符串 “width” 成了插件函数的 type 参数。

**默认设置和选项**

对于那些提供许多选项、更复杂、更可配置的插件，最佳实践是提供一个默认设置，它可在插件调用时（通过 [$.extend](http://docs.jquery.com/Utilities/jQuery.extend)）被扩展。这样调用插件时无需大量参数， 只要一个对象参数，内容为你希望不同于默认值的那部分设置。做法如下：

(function( $ ){

$.fn.tooltip = function( options ) {

// Create some defaults, extending them with any options that were provided

var settings = $.extend( {

'location' : 'top',

'background-color' : 'blue'

}, options);

return this.each(function() {

// Tooltip plugin code here

});

};

})( jQuery );

--

$('div').tooltip({

'location' : 'left'

});

在本例中，用给定选项调用 tooltip 插件后， 默认的 location 设置被覆盖为 "left"， 但 bacground-color 设置仍为默认值 "blue"。最终的设置对象看起来这样的：

{

'location' : 'left',

'background-color' : 'blue'

}

这是一个非常好的方式， 可以提供一个高度可配置的插件，又不必强制开发者定义所有选项。

**名称空间**

合理地为插件定义名称空间是插件开发中很重要的一部分。 正确的定义名称空间可以确保你的插件很难被其它插件或同一页面中的其它代码所覆盖。名称空间也可以让插件开发者的日子好过一些，因为它能帮你跟踪你的方法、事件和数据。

**插件方法**

单个插件**永远不要**在 jQuery.fn 对象中声明一个以上的名称空间

(function( $ ){

$.fn.tooltip = function( options ) {

// 这

};

$.fn.tooltipShow = function( ) {

// 不

};

$.fn.tooltipHide = function( ) {

// 好

};

$.fn.tooltipUpdate = function( content ) {

// !!!

};

})( jQuery );

这非常不好，因为它搞乱了 $.fn 名称空间。 要修正这个问题，你应该把所有插件方法收集到一个对象定义当中，并通过传递方法名称字符串调用。

(function( $ ){

var methods = {

init : function( options ) {

// 这

},

show : function( ) {

// 很

},

hide : function( ) {

// 好

},

update : function( content ) {

// !!!

}

};

$.fn.tooltip = function( method ) {

// Method calling logic

if ( methods[method] ) {

return methods[ method ].apply( this, Array.prototype.slice.call( arguments, 1 ));

} else if ( typeof method === 'object' || ! method ) {

return methods.init.apply( this, arguments );

} else {

$.error( 'Method ' + method + ' does not exist on jQuery.tooltip' );

}

};

})( jQuery );

// 调用 init 方法

$('div').tooltip();

// 调用 init 方法

$('div').tooltip({

foo : 'bar'

});

--

// 调用 hide 方法

$('div').tooltip('hide');

--

// 调用 update 方法

$('div').tooltip('update', 'This is the new tooltip content!');

这种插件架构使你可以在插件的父闭包中封装所有方法，调用时先传方法名称字符串，接下来再把你需要的其它参数传给该方法。这种封装和架构是 jQuery 插件社区的一个标准，已经被无数插件所使用，包括 [jQueryUI](http://jqueryui.com/) 中的插件和小部件。

**事件**

[bind](http://docs.jquery.com/Events/bind) 方法有个鲜为人知的特性：它支持为绑定事件定义名称空间。如果你的插件要绑定事件，最好为其定义名称空间。这样，回头想 [unbind](http://docs.jquery.com/Events/unbind) 的时候就不会影响到相同事件类型上的其它已绑定事件。要为事件定义名称空间，把 ".<namespace>" 附到要绑定的事件类型后面即可。

(function( $ ){

var methods = {

init : function( options ) {

return this.each(function(){

$(window).bind('resize.tooltip', methods.reposition);

});

},

destroy : function( ) {

return this.each(function(){

$(window).unbind('.tooltip');

})

},

reposition : function( ) {

// ...

},

show : function( ) {

// ...

},

hide : function( ) {

// ...

},

update : function( content ) {

// ...

}

};

$.fn.tooltip = function( method ) {

if ( methods[method] ) {

return methods[method].apply( this, Array.prototype.slice.call( arguments, 1 ));

} else if ( typeof method === 'object' || ! method ) {

return methods.init.apply( this, arguments );

} else {

$.error( 'Method ' + method + ' does not exist on jQuery.tooltip' );

}

};

})( jQuery );

--

$('#fun').tooltip();

// Some time later...

$('#fun').tooltip('destroy');

在本例中，当 tooltip 被 init 方法初始化的时候，它把 reposition 方法绑定到 window 对象的 resize 事件上，名称空间为 "tooltip"。 之后，如果开发者想要销毁对象，可以把插件的名称空间（即 "tooltip"）传给 unbind 方法，以便解除本插件对所有事件的绑定。这使得我们可以安全的地解除本插件的事件绑定，避免意外影响插件之外绑定的事件。

**数据**

插件开发中，你可能经常需要维护状态，或检查你的插件是否已在给定元素上做过初始化。jQuery [data](http://docs.jquery.com/action/edit/Utilities/data) 方法是针对每个元素跟踪变量的好办法。不过最好能用单一对象容纳所有变量并用单一名称空间访问此对象，而不是分别跟踪一堆不同名字的数据。

(function( $ ){

var methods = {

init : function( options ) {

return this.each(function(){

var $this = $(this),

data = $this.data('tooltip'),

tooltip = $('<div />', {

text : $this.attr('title')

});

// If the plugin hasn't been initialized yet

if ( ! data ) {

/\*

Do more setup stuff here

\*/

$(this).data('tooltip', {

target : $this,

tooltip : tooltip

});

}

});

},

destroy : function( ) {

return this.each(function(){

var $this = $(this),

data = $this.data('tooltip');

// Namespacing FTW

$(window).unbind('.tooltip');

data.tooltip.remove();

$this.removeData('tooltip');

})

},

reposition : function( ) { // ... },

show : function( ) { // ... },

hide : function( ) { // ... },

update : function( content ) { // ...}

};

$.fn.tooltip = function( method ) {

if ( methods[method] ) {

return methods[method].apply( this, Array.prototype.slice.call( arguments, 1 ));

} else if ( typeof method === 'object' || ! method ) {

return methods.init.apply( this, arguments );

} else {

$.error( 'Method ' + method + ' does not exist on jQuery.tooltip' );

}

};

})( jQuery );

data 方法可以帮你在插件的多次方法调用之间跟踪变量和状态。 把数据置于单一对象中，并为其定义名称空间有利于集中访问插件的所有属性，同时也减少了名称空间以便需要时删除。

**总结及最佳实践**

编写 jQuery 插件使库更加高效。把你最聪明、最有用的功能抽象成可重用代码，这将节省你的时间，进一步提高开发效率。下面是本文档的简要总结以及你开发下一个 jQuery 插件时的注意事项：

* 总是把插件包装在闭包中 { /\* plugin goes here \*/ })( jQuery );
* 不在插件函数的立即作用域中额外包装 this 关键字
* 总是让插件函数返回 this 关键字以保持 chainability ，除非插件有真正的返回值。
* 不要传给插件大量参数，应该传一个可以覆盖插件默认选项的设置对象。
* 在单个插件中，不要让一个以上的名称空间搞乱了 jQuery.fn 对象。
* 总是为方法、事件和数据定义名称空间。