深入理解jQuery插件开发

来源：jobbole 发布时间：2012-[illegible]239

如果你看到这篇文章，我确信你毫无疑问会认为jQuery是一个使用简便的库。jQuery可能使用起来很简单，但是它仍然有一些奇怪的地方，对它基本功能和概念不熟悉的人可能会难以掌握。但是不用担心，我下面已经把代码划分成小部分，做了一个简单的指导。那些语法看起来可能过于复杂，但是如果进入到它的思想和模式中，它是非常简单易懂的。

下面，我们有了一个插件的基本层次：

```
// Shawn Khameneh
// ExtraordinaryThoughts.com

(function($) {
    var privateFunction = function() {
        // 代码在这里运行
    }

    var methods = {
        init: function(options) {
            return this.each(function() {
                var $this = $(this);
                var settings = $this.<span id="33_nwp" s e="width: auto; height

                if(typeof(settings) == 'undefined') {

                    var defaults = {
                        propertyName: 'value',
                        onSomeEvent: function() {}
                    }

                    settings = $.extend({}, defaults, options);

                    $this.data('pluginName', settings);
                } else {
                    settings = $.extend({}, settings, options);
                }

                // 代码在这里运行

            });
        },
        destroy: function(options) {
```

猜你喜欢

阅读排行
跟着 8 张思维导图学习 Javascript
Web Uploader 功能齐全完美兼容
Nivo Slider - 世界上最棒的 jQue
AngularJS 、Backbone.js 和 En
早该知道的7个JavaScript技巧
50个jQuery 插件可将你的网站带
编写高性能JavaScript
2014年最酷的30个JavaScript库
AngularJS - 下一个大框架
为什么JavaScript是你应当学习的

推荐文档
JavaScript中的原型和继承
10个最好的 jQuery 视频插件
超实用的JavaScript技巧及最佳实
21个值得收藏的Javascript技巧
Web开发人员不能错过的 jQuery
用于构建交互式图表的最佳 jQuery
编写更好的jQuery代码的建议
15 个最佳的 jQuery 表格插件
对于Web开发很有用的jQuery效果
推荐给开发者的20款响应式jQuery

随机文档
强类型 JavaScript 的解决方案
深入浅出 JavaScript 中的 this
JavaScript开发者常忽略或误用的
15个构建交互式图表的最佳 jQuer
对AngularJS进行性能调优的7个建
让我们写快速的JavaScript，JS性
Hallo.js：一款所见即所得的Web编

```
        destroy: function(options) {
            return $(this).each(function() {
                var $this = $(this);

                $this.removeData('pluginName');
            });
        },
        val: function(options) {
            var someValue = this.eq(0).<span id="34_nwp" style="width: auto; hei

            return someValue;
        }
    };

    $.fn.pluginName = function() {
        var method = arguments[0];

        if(methods[method]) {
            method = methods[method];
            arguments = Array.prototype.slice.<span id="35_nwp" style="width: au
        } else if( typeof(method) == 'object' || !method ) {
            method = methods.init;
        } else {
            $.error( 'Method ' +  method + ' does not exist on jQuery.pluginName
            return this;
        }

        return method.apply(this, arguments);

    }

})(jQuery);
```

你可能会注意到，我所提到代码的结构和其他插件代码有很大的不同。根据你的使用和需求的不同，插件的开发方式也可能会呈现多样化。我的目的是澄清代码中的一些概念，足够让你找到适合自己的方法去理解和开发一个jQuery插件。

现在，来解剖我们的代码吧！

容器：一个即时执行函数

根本上来说，每个插件的代码是被包含在一个即时执行的函数当中，如下：

```
(function(arg1, arg2) {
   // 代码
})(arg1, arg2);
```

即时执行函数，顾名思义，是一个函数。让它与众不同的是，它被包含在一对小括号里面，这让所有的代码都在匿名函数的局部作用域中运行。这并不是说DOM（全局变量）在函数内是被屏蔽的，而是外部无法访问到函数内部的公共变量和对象命名空间。这是一个很好的开始，这样你声明你的变量和对象的时候，就不用担心着变量名和已经存在的代码有冲突。

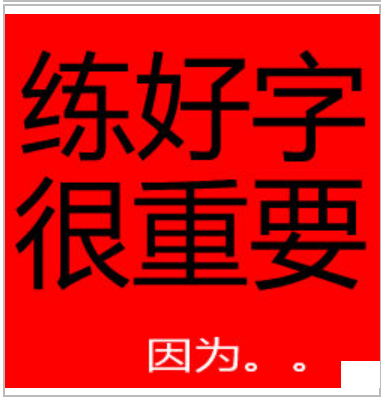
现在，因为函数内部所有的所有公共变量是无法访问的，这样要把jQuery本身作为一个内部的公共变量来使用就会成为问题。就像普通的函数一样，即时函数也根据引用传入对象参数。我们可以将jQuery对象传入函数，如下：

```
(function($) {

   // 局部作用域中使用$来引用jQuery
})(jQuery);
```

我们传入了一个把公共变量"jQuery"传入了一个即时执行的函数里面，在函数局部（容器）中我们可以通过"$"来引用它。也就是说，我们把容器当做一个函数来调用，而这个函数的参数就是jQuery。因为我们引用的"jQuery"作为公共变量传入，而不是它的简写"$"，这样我们就可以兼容Prototype库。如果你不用Prototype或者其它用"$"做简写的库的话，你不这样做也不会造成什么影响，但是知道这种用法仍是一件好

Angular.js VS. Ember.js：谁将
jQuery：在一个回调中处理多个请
现代JavaScript开发者的工具箱



事。

插件：一个函数

一个jQuery插件本质上是我们塞进jQuery命名空间中一个庞大的函数，当然，我们可以很轻易地用“jQuery.pluginName=function”，来达到我们的目的，但是如果我们这样做的话我们的插件的代码是处于没有被保护的暴露状态的。“jQuery.fn”是“jQuery.prototype”的简写，意味当我们通过jQuery命名空间去获取我们的插件的时候，它仅可写（不可修改）。它事实上可以为你干点什么事呢？它让你恰当地组织自己的代码，和理解如何保护你的代码不受运行时候不需要的修改。最好的说法就是，这是一个很好的实践！

通过一个插件，我们获得一个基本的jQuery函数：

```
(function($) {

    $.fn.pluginName = function(options) {

        // 代码在此处运行

        return this;
    }

})(jQuery);
```

上面的代码中的函数可以像其他的jQuery函数那样通过“$(‘#element’).pluginName()”来调用。注意，我是如何把“return this”语句加进去的；这小片的代码通过返回一个原来元素的集合（包含在this当中）的引用来产生链式调用的效果，而这些元素是被一个jQuery对象所包裹的。你也应该注意，“this”在这个特定的作用域中是一个jQuery对象，相当于“$(‘#element’)”。

根据返回的对象，我们可以总结出，在上面的代码中，使用“$(‘#element’).pluginName()”的效果和使用“$(‘#element’)”的效果是一样的。在你的即时执行函数作用域中，没必要用“$(this)”的方式来把this包裹到一个jQuery对象中，因为this本身已经是被包装好的jQuery对象。

多个元素：理解**Sizzle**

jQuery使用的选择器引擎叫Sizzle，Sizzle可以为你的函数提供多元素操作（例如对所有类名相同的元素）。这是jQuery几个优秀的特性之一，但这也是你在开发插件过程中需要考虑的事情。即使你不准备为你的插件提供多元素支持，但为这做准备仍然是一个很好的实践。

这里我添加了一小段代码，它让你的插件代码为多元素集合中每个元素单独地起作用：

```
function($) {

    // 向jQuery中被保护的“fn”<span id="26_nwp" style="width: auto; height: auto; 1
    $.fn.pluginName = function(options) {

        // 返回“this”（函数each（）的返回值也是this），以便进行链式调用。
        return this.each(function() {

            // 此处运行代码，可以通过“this”来获得每个单独的元素
            // 例如：  $(this).show();
            var $this = $(this);

        });

    }

})(jQuery);
```

在以上示例代码中，我并不是用 each（）在我的选择器中每个元素上运行代码。在那个被 each（）调用的函数的局部作用域中，你可以通过this来引用每个被单独处理的元素，也就是说你可以通过$(this)来引用它的jQuery对象。在局部作用域中，我用$this变量存储起jQuery对象，而不是每次调用函数的时候都使用$(this)，这会是个很好的实践。当然，这样做并不总是必要的；但我已经额外把它包含在我的代码中。还有要注意的是，我们将会对每个单独方法都使用 each（），这样到时我们就可以返回我们需要的值，而不是一个jQuery对象。

下面是一个例子，假如我们的插件支持一个 val 的方法，它可以返回我们需要的值：

```
$('#element').pluginName('val');
// 会返回我们需要的值，而不是一个jQuery对象
```

功能：公有方法和私有方法

一个基本的函数可能在某些情况下可以良好地工作，但是一个稍微复杂一点的插件就需要提供各种各样的方法和私有函数。你可能会使用不同的命名空间去为你的插件提供各种方法，但是最好不要让你的源代码因为多余的命名空间而变得混乱。

下面的代码定义了一个存储公有方法的JSON对象，以及展示了如何使用插件中的主函数中去判断哪些方法被调用，和如何在让方法作用到选择器每个元素上。

```
(function($) {

    // 在我们插件容器内，创造一个公共变量来<span id="16_nwp" style="width: auto; hei
    var privateFunction = function() {
        // code here
    }

    // 通过字面量创造一个对象，<span id="17_nwp" style="width: auto; height: auto;
    var methods = {
        // 在字面量对象中定义每个单独的方法
        init: function() {

            // 为了更好的灵活性，对来自主<span id="18_nwp" style="width: auto; heig
            return this.each(function() {
                // 为每个独立的元素创建一个jQuery对象
                var $this = $(this);

                // 执行代码
                // 例如： privateFunction();
            });
        },
        destroy: function() {
            // 对选择器每个元素都执行方法
            return this.each(function() {
                // 执行代码
            });
        }
    };

    $.fn.pluginName = function() {
        // 获取我们的方法，遗憾的是，如果我们用function(method){}来实现，这样会毁掉一
        var method = arguments[0];

        // 检验方法是否存在
        if(methods[method]) {

            // 如果方法存在，<span id="19_nwp" style="width: auto; height: auto; f
            // 注意：我这样做是为了等下更方便地使用each（）
            method = methods[method];

        // 如果方法不存在，检验对象是否为一个对象（JSON对象）或者method方法没有被传入
        } else if( typeof(method) == 'object' || !method ) {

            // 如果我们传入的是一个对象参数，或者根本没有参数，init方法会被调用
            method = methods.init;
        } else {

            // 如果方法不存在或者参数没传入，则报出错误。需要调用的方法没有被正确调用
            $.error( 'Method ' +  method + ' does not exist on jQuery.pluginName
            return this;
        }

        // 调用我们选中的方法
        // 再一次注意我们是如何将each（）从这里转移到每个单独的方法上的
        return method.<span id="20_nwp" style="width: auto; height: auto; float:

    }

})(jQuery);
```

注意我把 privateFunction 当做了一个函数内部的全局变量。考虑到所有的代码的运行都是在插件容器内进行的，所以这种做法是可以被接受的，因为它只在插件的作用域中可用。在插件中的主函数中，我检验了传入参数所指向的方法是否存在。如果方法不存在或者传入的是参数为对象， init 方法会被运行。最后，如果传入的参数不是一个对象而是一个不存在的方法，我们会报出一个错误信息。

同样要注意的是，我是如何在每个方法中都使用 this.each() 的。当我们在主函数中调用 method.call(this) 的时候，这里的 this 事实上就是一个jQuery对象，作为 this 传入每个方法中。所以在我们方法的即时作用域中，它已经是一个jQuery对象。只有在被 each（）所调用的函数中，我们才有必要将this包装在一个jQuery对象中。

下面是一些用法的例子：

```
/*
 注意这些例子可以在目前的插件代码中正确运行，并不是所有的插件都使用同样的代码结构
*/
// 为每个类名为 ".className" 的元素执行init方法
$('.className').pluginName();
$('.className').pluginName('init');
$('.className').pluginName('init', {}); // 向init方法传入“{}”对象作为函数参数
$('.className').pluginName({}); // 向init方法传入“{}”对象作为函数参数

// 为每个类名为 “.className” 的元素执行destroy方法
$('.className').pluginName('destroy');
$('.className').pluginName('destroy', {}); // 向destroy方法传入“{}”对象作为<span i

// 所有代码都可以正常运行
$('.className').pluginName('init', 'argument1', 'argument2'); // 把 "argument 1"
```

```
16
17  // 不正确的使用
18  $('.className').pluginName('nonexistantMethod');
19  $('.className').pluginName('nonexistantMethod', {});
20  $('.className').pluginName('argument 1'); // 会尝试调用 "argument 1" 方法
21  $('.className').pluginName('argument 1', 'argument 2'); // 会尝试调用 "argument 1
22  $('.className').pluginName('privateFunction'); // 'privateFunction' 不是一个方法
```

在上面的例子中多次出现了 {}，表示的是传入方法中的参数。在这小节中，上面代码可以可以正常运行，但是参数不会被传入方法中。继续阅读下一小节，你会知道如何向方法传入参数。 设置插件：传入参数许多插件都支持参数传入，如配置参数和回调函数。你可以通过传入JS键值对对象或者函数参数，为方法提供信息。如果你的方法支持多于一个或两个参数，那么没有比传入对象参数更恰当的方式。

```
1   (function($) {
2       var methods = {
3           init: function(options) {
4
5               // 在每个元素上执行方法
6               return this.each(function() {
7                   var $this = $(this);
8
9                   // 创建一个默认设置对象
10                  var defaults = {
11                      propertyName: 'value',
12                      onSomeEvent: function() {}
13                  }
14
15                  // 使用extend方法从options和defaults对象中构造出一个settings对象
16                  var settings = $.extend({}, defaults, options);
17
18                  // 执行代码
19
20              });
21          }
22      };
23
24      $.fn.pluginName = function() {
25          var method = arguments[0];
26
27          if(methods[method]) {
28              method = methods[method];
29
30              // 我们的方法是作为参数传入的，把它从参数列表中删除，因为调用方法时并不需要
31              arguments = Array.prototype.slice.<span id="10_nwp" style="width: au
32          } else if( typeof(method) == 'object' || !method ) {
33              method = methods.init;
34          } else {
35              $.error( 'Method ' +  method + ' does not exist on jQuery.pluginName
36              return this;
37          }
38
39          // 用apply方法来调用我们的方法并传入参数
40          return method.apply(this, arguments);
41
42      }
43
44  })(jQuery);
```

正如上面所示，一个“options”参数被添加到方法当中，和“arguments”也被添加到了主函数中。如果一个方法已经被声明，在参数传入方法之前，调用那个方法的参数会从参数列表中删除掉。我用了“apply（）”来代替了“call（）”，“apply（）”本质上是和“call（）”做着同样的工作的，但不同的是它允许参数的传入。这种结构也允许多个参数的传入，如果你愿意这样做，你也可以为你的方法修改参数列表，例如：“init:function(arg1, arg2){}”。

如果你是使用JS对象作为参数传入，你可能需要定义一个默认对象。一旦默认对象被声明，你可以使用“$.extend”来合并参数对象和默认对象中的值，以形成一个新的参数对象来使用（在我们的例子中就是“settings”）：

这里有一些例子，用来演示以上的逻辑：

```
1   var options = {
2       customParameter: 'Test 1',
3       propertyName: 'Test 2'
4   }
5
6   var defaults = {
7       propertyName: 'Test 3',
8       onSomeEvent: 'Test 4'
9   }
10
11  var settings = $.extend({}, defaults, options);
12  /*
13  settings == {
14      propertyName: 'Test 2',
15      onSomeEvent: 'Test 4',
```

```
    customParameter: 'Test 1'
}
*/
```

保存设置：添加持久性数据

有时你会想在你的插件中保存设置和信息，这时jQuery中的“data（）”函数就可以派上用场了。它在使用上是非常简单的，它会尝试获取和元素相关的数据，如果数据不存在，它就会创造相应的数据并添加到元素上。一旦你使用了“data（）”来为元素添加信息，请确认你已经记住，当不再需要数据的时候，用“removeDate（）”来删除相应的数据。

```
// Shawn Khameneh
// ExtraordinaryThoughts.com

(function($) {
    var privateFunction = function() {
        // 执行代码
    }

    var methods = {
        init: function(options) {

            // 在每个元素上执行方法
            return this.each(function() {
                var $this = $(this);

                // 尝试去获取settings，如果不存在，则返回“undefined”
                var settings = $this.data('pluginName');

                // 如果获取settings失败，则根据options和default创建它
                if(typeof(settings) == 'undefined') {

                    var defaults = {
                        propertyName: 'value',
                        onSomeEvent: function() {}
                    }

                    settings = $.extend({}, defaults, options);

                    // 保存我们新创建的settings
                    $this.<span id="2_nwp" style="width: auto; height: auto; flo
                } else {
                    / 如果我们获取了settings，则将它和options进行合并（这不是必须的，
                    settings = $.extend({}, settings, options);

                    // 如果你想每次都保存options，可以添加下面代码：
                    // $this.data('pluginName', settings);
                }

                // 执行代码

            });
        },
        destroy: function(options) {
            // 在每个元素中执行代码
            return $(this).each(function() {
                var $this = $(this);

                // 执行代码

                // 删除元素对应的数据
                $this.removeData('pluginName');
            });
        },
        val: function(options) {
            // 这里的代码通过.eq(0)来获取选择器中的第一个元素的，我们或获取它的HTML内
            var someValue = this.eq(0).<span id="3_nwp" style="width: auto; heigh

            // 返回值
            return someValue;
        }
    };

    $.fn.pluginName = function() {
        var method = arguments[0];

        if(methods[method]) {
            method = methods[method];
            arguments = Array.prototype.slice.<span id="4_nwp" style="width: auto
        } else if( typeof(method) == 'object' || !method ) {
            method = methods.init;
        } else {
            $.error( 'Method ' +  method + ' does not exist on jQuery.pluginName
            return this;
        }

        return method.apply(this, arguments);

    }

})(jQuery);
```

在上面的代码中，我检验了元素的数据是否存在。如果数据不存在，“options”和“default”会被合并，构建成一个新的settings，然后用“data（）”保存在元素中。

完成：请继续阅读学习！

下面是一些参考资料的链接：

- jQuery Documentation – Plugins/Authoring
- jQuery Documentation – Context, Call, and Apply
- jAlternative Method Using “$.fn.extend()”
- jAnother Guide Demonstrating Secondary Functions

最后但同样重要的一点是，我还强烈推荐《JavaScript模式 》这本书。我发现的一些最好的书是关于代码模式的，这本书会带领你开始开发更清晰的代码和插件。

相关文档：常用的Javascript设计模式

英文原文：Extraordinary Thougths ，编译：伯乐在线——戴嘉华

jquery | 插件

相关文档

- 最常见的 20 个 jQuery 面试问题及答
- 什么时候 AngularJS 会超越 jQuery
- jQuery 3.0 —— 下一代的 jQuery
- jQuery对象入门级介绍
- jQuery：在一个回调中处理多个请求
- jQuery在v1.13版本中不再支持IE6和IE
- 抛弃jQuery 深入原生的JavaScript
- 如何书写高质量的jQuery代码
- 8个对程序员来说有用的jQuery小贴士和
- 20 个强大的 Sublime Text 插件
- 值得Web开发人员学习的20个jQuery实例
- 学习jQuery的免费资源：电子书、视频
- 用于构建交互式图表的最佳 jQuery 图
- 编写更好的jQuery代码的建议

网友评论（共0条评论）

发表评论 / 共**0**条评论 理智评论文明上网，拒绝恶意谩骂

登录会员中心 发表评论