jQuery插件开发全解析

jQuery插件的开发包括两种：

一种是类级别的插件开发，即给jQuery添加新的全局函数，相当于给jQuery类本身添加方法。jQuery的全局函数就是属于jQuery命名空间的函数，另一种是对象级别的插件开发，即给jQuery对象添加方法。下面就两种函数的开发做详细的说明。

1、类级别的插件开发

类级别的插件开发最直接的理解就是给jQuery类添加类方法，可以理解为添加静态方法。典型的例子就是$.AJAX()这个函数，将函数定义于jQuery的命名空间中。关于类级别的插件开发可以采用如下几种形式进行扩展：

1.1 添加一个新的全局函数

添加一个全局函数，我们只需如下定义：

```
jQuery.foo = function() {
    alert('This is a test. This is only a test.');
};
```

调用的时候可以这样写： jQuery.foo(); 或 $.foo();

1.2 增加多个全局函数

添加多个全局函数，可采用如下定义：

```
jQuery.foo = function() {
    alert('This is a test. This is only a test.');
};
jQuery.bar = function(param) {
    alert('This function takes a parameter, which is "' + param + '".');
};
```

调用时和一个函数的一样的:jQuery.foo();jQuery.bar();或者$.foo();$.bar('bar');

1.3 使用jQuery.extend(object);

```
jQuery.extend({
    foo: function() {
        alert('This is a test. This is only a test.');
    },
    bar: function(param) {
        alert('This function takes a parameter, which is "' + param +'".');
    }
});
```

1.4 使用命名空间

虽然在jQuery命名空间中，我们禁止使用了大量的javaScript函数名和变量名。但是仍然不可避免某些函数或变量名将于其他jQuery插件冲突，因此我们习惯将一些方法封装到另一个自定义的命名空间。

```
jQuery.myPlugin = {
    foo:function() {
        alert('This is a test. This is only a test.');
    },
    bar:function(param) {
        alert('This function takes a parameter, which is "' + param + '".');
    }
};
```

采用命名空间的函数仍然是全局函数，调用时采用的方法：

```
$.myPlugin.foo();
$.myPlugin.bar('baz');
```

通过这个技巧（使用独立的插件名），我们可以避免命名空间内函数的冲突。

2、对象级别的插件开发

对象级别的插件开发需要如下的两种形式：、

形式1：

```
(function($){
    $.fn.extend({
        pluginName:function(opt,callback){
               // Our plugin implementation code goes here.
        }
    })
})(jQuery);
```

形式2：

```
(function($) {
    $.fn.pluginName = function() {
         // Our plugin implementation code goes here.
    };
})(jQuery);
```

上面定义了一个jQuery函数,形参是$，函数定义完成之后,把jQuery这个实参传递进去.立即调用执行。这样的好处是,我们在写jQuery插件时,也可以使用$这个别名,而不会与prototype引起冲突.

2.1 在JQuery名称空间下申明一个名字

这是一个单一插件的脚本。如果你的脚本中包含多个插件，或者互逆的插件（例如：$.fn.doSomething() 和 $.fn.undoSomething()），那么你需要声明多个函数名字。但是，通常当我们编写一个插件时，力求仅使用一个名字来包含它的所有内容。我们的示例插件命名为“highlight“

```
$.fn.hilight = function() {
  // Our plugin implementation code goes here.
};
```

我们的插件通过这样被调用：

```
$('#myDiv').hilight();
```

但是如果我们需要分解我们的实现代码为多个函数该怎么办？有很多原因：设计上的需要；这样做更容易或更易读的实现；而且这样更符合面向对象。 这真是一个麻烦事，把功能实现分解成多个函数而不增加多余的命名空间。出于认识到和利用函数是javascript中最基本的类对象，我们可以这样做。就像其他对象一样，函数可以被指定为属性。因此我们已经声明“hilight”为jQuery的属性对象，任何其他的属性或者函数我们需要暴露出来的，都可以在"hilight" 函数中被声明属性。稍后继续。

2.2 接受options参数以控制插件的行为

让我们为我们的插件添加功能指定前景色和背景色的功能。我们也许会让选项像一个options对象传递给插件函数。例如：

```
// plugin definition
$.fn.hilight = function(options) {
  var defaults = {
    foreground: 'red',
    background: 'yellow'
  };
  // Extend our default options with those provided.
  var opts = $.extend(defaults, options);
  // Our plugin implementation code goes here.
};
```

我们的插件可以这样被调用：

```
$('#myDiv').hilight({
```

```
  foreground: 'blue'
});
```

2.3 暴露插件的默认设置

我们应该对上面代码的一种改进是暴露插件的默认设置。这对于让插件的使用者更容易用较少的代码覆盖和修改插件。接下来我们开始利用函数对象。

```
// plugin definition
$.fn.hilight = function(options) {
  // Extend our default options with those provided.
  // Note that the first arg to extend is an empty object -
  // this is to keep from overriding our "defaults" object.
  var opts = $.extend({}, $.fn.hilight.defaults, options);
  // Our plugin implementation code goes here.
};
// plugin defaults - added as a property on our plugin function
$.fn.hilight.defaults = {
  foreground: 'red',
  background: 'yellow'
};
```

现在使用者可以包含像这样的一行在他们的脚本里：

```
//这个只需要调用一次，且不一定要在ready块中调用
$.fn.hilight.defaults.foreground = 'blue';
```

接下来我们可以像这样使用插件的方法，结果它设置蓝色的前景色：

```
$('#myDiv').hilight();
```

如你所见，我们允许使用者写一行代码在插件的默认前景色。而且使用者仍然在需要的时候可以有选择的覆盖这些新的默认值：

```
// 覆盖插件缺省的背景颜色
$.fn.hilight.defaults.foreground = 'blue';
// ...
// 使用一个新的缺省设置调用插件
$('.hilightDiv').hilight();
// ...
// 通过传递配置参数给插件方法来覆盖缺省设置
$('#green').hilight({
  foreground: 'green'
});
```

2.4 适当的暴露一些函数

这段将会一步一步对前面那段代码通过有意思的方法扩展你的插件（同时让其他人扩展你的插件）。例如，我们插件的实现里面可以定义一个名叫"format"的函数来格式化高亮文本。我们的插件现在看起来像这样，默认的format方法的实现部分在hiligth函数下面。

```
// plugin definition
$.fn.hilight = function(options) {
  // iterate and reformat each matched element
  return this.each(function() {
    var $this = $(this);
    // ...
```

```
    var markup = $this.html();
    // call our format function
    markup = $.fn.hilight.format(markup);
    $this.html(markup);
  });
};
// define our format function
$.fn.hilight.format = function(txt) {
return '<strong>' + txt + '</strong>';
};
```

我们很容易的支持options对象中的其他的属性通过允许一个回调函数来覆盖默认的设置。这是另外一个出色的方法来修改你的插件。这里展示的技巧是进一步有效的暴露format函数进而让他能被重新定义。通过这技巧，是其他人能够传递他们自己设置来覆盖你的插件，换句话说，这样其他人也能够为你的插件写插件。

考虑到这个篇文章中我们建立的无用的插件，你也许想知道究竟什么时候这些会有用。一个真实的例子是Cycle插件.这个Cycle插件是一个滑动显示插件，他能支持许多内部变换作用到滚动，滑动，渐变消失等。但是实际上，没有办法定义也许会应用到滑动变化上每种类型的效果。那是这种扩展性有用的地方。 Cycle插件对使用者暴露"transitions"对象，使他们添加自己变换定义。插件中定义就像这样：

```
$.fn.cycle.transitions = {
// ...
};
```

这个技巧使其他人能定义和传递变换设置到Cycle插件。

2.5 保持私有函数的私有性

这种技巧暴露你插件一部分来被覆盖是非常强大的。但是你需要仔细思考你实现中暴露的部分。一但被暴露，你需要在头脑中保持任何对于参数或者语义的改动也许会破坏向后的兼容性。一个通理是，如果你不能肯定是否暴露特定的函数，那么你也许不需要那样做。

那么我们怎么定义更多的函数而不搅乱命名空间也不暴露实现呢？这就是闭包的功能。为了演示，我们将会添加另外一个“debug”函数到我们的插件中。这个 debug函数将为输出被选中的元素格式到firebug控制台。为了创建一个闭包，我们将包装整个插件定义在一个函数中。

```
(function($) {
 // plugin definition
 $.fn.hilight = function(options) {
   debug(this);
   // ...
 };
 // private function for debugging
 function debug($obj) {
   if (window.console && window.console.log)
     window.console.log('hilight selection count: ' + $obj.size());
 };
//  ...
})(jQuery);
```

我们的“debug”方法不能从外部闭包进入,因此对于我们的实现是私有的。

2.6 支持Metadata插件

在你正在写的插件的基础上，添加对Metadata插件的支持能使他更强大。个人来说，我喜欢这个Metadata插件，因为它让你使用不多的"markup”覆盖插件的选项（这非常有用当创建例子时）。而且支持它非常简单。更新：注释中有一点优化建议。

```
$.fn.hilight = function(options) {
  // ...
```

```
// build main options before element iteration
var opts = $.extend({}, $.fn.hilight.defaults, options);
return this.each(function() {
  var $this = $(this);
  // build element specific options
  var o = $.meta ? $.extend({}, opts, $this.data()) : opts;
  //...
```

这些变动行做了一些事情：它是测试Metadata插件是否被安装
如果它被安装了，它能扩展我们的options对象通过抽取元数据
这行作为最后一个参数添加到JQuery.extend，那么它将会覆盖任何其它选项设置。现在我们能从"markup”处驱动行为,如果我们选择了“markup”：

```
<!--  markup  -->
<div class="hilight { background: 'red', foreground: 'white' }">
  Have a nice day!
</div>
<div class="hilight { foreground: 'orange' }">
  Have a nice day!
</div>
<div class="hilight { background: 'green' }">
  Have a nice day!
</div>
```

现在我们能高亮哪些div仅使用一行脚本：

```
$('.hilight').hilight();
```

2.7 整合

下面使我们的例子完成后的代码：

```
// 创建一个闭包
(function($) {
  // 插件的定义
  $.fn.hilight = function(options) {
    debug(this);
    // build main options before element iteration
    var opts = $.extend({}, $.fn.hilight.defaults, options);
    // iterate and reformat each matched element
    return this.each(function() {
      $this = $(this);
      // build element specific options
      var o = $.meta ? $.extend({}, opts, $this.data()) : opts;
      // update element styles
      $this.css({
        backgroundColor: o.background,
        color: o.foreground
      });
      var markup = $this.html();
      // call our format function
      markup = $.fn.hilight.format(markup);
      $this.html(markup);
    });
```

```
  };
  // 私有函数：debugging
  function debug($obj) {
    if (window.console && window.console.log)
      window.console.log('hilight selection count: ' + $obj.size());
  };
  // 定义暴露format函数
  $.fn.hilight.format = function(txt) {
    return '<strong>' + txt + '</strong>';
  };
  // 插件的defaults
  $.fn.hilight.defaults = {
    foreground: 'red',
    background: 'yellow'
  };
// 闭包结束
})(jQuery);
```

这段设计已经让我创建了强大符合规范的插件。我希望它能让你也能做到。

3、总结

jQuery为开发插件提拱了两个方法，分别是：

jQuery.fn.extend(object); 给jQuery对象添加方法。

jQuery.extend(object); 为扩展jQuery类本身.为类添加新的方法。

3.1 jQuery.fn.extend(object);

fn 是什么东西呢。查看jQuery代码，就不难发现。

```
jQuery.fn = jQuery.prototype = {
init: function( selector, context ) {//....
//......
};
```

原来 jQuery.fn = jQuery.prototype.对prototype肯定不会陌生啦。虽然 javascript 没有明确的类的概念，但是用类来理解它，会更方便。jQuery便是一个封装得非常好的类，比如我们用 语句 $("#btn1") 会生成一个 jQuery类的实例。

jQuery.fn.extend(object); 对jQuery.prototype进得扩展，就是为jQuery类添加“成员函数”。jQuery类的实例可以使用这个“成员函数”。

比如我们要开发一个插件，做一个特殊的编辑框，当它被点击时，便alert 当前编辑框里的内容。可以这么做：

```
$.fn.extend({
     alertWhileClick:function(){
         $(this).click(function(){
              alert($(this).val());
          });
      }
});
$("#input1").alertWhileClick(); //页面上为：<input id="input1" type="text"/>
```

$("#input1") 为一个jQuery实例，当它调用成员方法 alertWhileClick后，便实现了扩展，每次被点击时它会先弹出目前编辑里的内容。

3.2 jQuery.extend(object);

为jQuery类添加添加类方法，可以理解为添加静态方法。如：

```
$.extend({
```

```
    add:function(a,b){return a+b;}
});
```

便为 jQuery 添加一个为 add 的 “静态方法”，之后便可以在引入 jQuery 的地方，使用这个方法了，`$.add(3,4); //return 7`