jQuery 插件开发的三种方式

1、通过$.extend()来扩展 jQuery

2、通过$.fn 向 jQuery 添加新的方法

3、通过$.widget()应用 jQuery UI 的部件工厂方式创建

通常使用第二种方法来进行简单插件开发，说简单是相对于第三种方式。第三种方式是用来开发更高级 jQuery 部件的，该模式开发出来的部件带有很多 jQuery 内建的特性，比如插件的状态信息自动保存，各种关于插件的常用方法等。而第一种方式又太简单，仅仅是在 jQuery 命名空间或者理解成 jQuery 身上添加了一个静态方法而以。

一、入门

编写一个 jQuery 插件开始于给 jQuery.fn 加入新的功能属性，此处添加的对象属性的名称就是插件的名称，代码如下：

```
jQuery.fn.myPlugin = function() {
        //自己的插件代码
    };
```

为了避免和其他 JavaScript 库冲突，最好将 jQuery 传递给一个自我执行的封闭程序，jQuery 在此程序中映射为$符号，这样可以避免$号被其他库覆写。代码如下：

```
(function($) {
    $.fn.myPlugin = function() {
        //你自己的插件代码
    };
})(jQuery);
```

在这个封闭程序中，可以无限制的使用$符号来表示 jQuery 函数。

二、环境

在编写插件之前，须对插件所处的环境有个概念。在插件的范围里， this 关键字代表了这个插件将要执行的 jQuery 对象， 这里容易产生一个普遍的误区 ,因为在其他包含 callback 的 jQuery 函数中，this 关键字代表了原生的 DOM 元素。这常常会导致开发者误将 this 关键字无谓的包在 jQuery 中，代码如下：

```
(function($) {
    $.fn.myPlugin = function() {
        //此处没有必要将 this 包在$号中如$(this)，因为 this 已经是一个 jQuery 对象。
        //$(this)等同于  $($('#element'));
        this.fadeIn('normal', function() {
            //此处 callback 函数中 this 关键字代表一个 DOM 元素
        });
    };
})(jQuery);
$('#element').myPlugin();
```

三、维护 Chainability

很多时候，一个插件的意图仅仅是以某种方式修改收集的元素，并把它们传递给链中的下一个方法。 这是 jQuery 的设计之美，是 jQuery 如此受欢迎的原因之一。 因此，要保持一个插件的 chainability，必须确保插件返回 this 关键字。代码如下:

```
(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');
```

由于插件返回 this 关键字，它保持了 chainability，这样 jQuery 收集的元素可以继续被 jQuery 方法如.css 控制。 因此，如果插件不返回固有的价值，就应该总是在其作用范围内返回 this 关键字。

四、默认值和选项

对于比较复杂的和提供了许多选项可定制的的插件 ,最好有一个当插件被调用的时候可以被拓展的默认设置(通过使用$.extend)。 因此，相对于调用一个有大量参数的插件，可以调用一个对象参数，包含你想覆写的设置。代码如下：

```
(function ($) {
    $.fn.tooltip = function (options) {
```

```
            //创建一些默认值，拓展任何被提供的选项
            var settings = $.extend({
                'location': 'top',
                'background-color': 'blue'
            }, options);
            return this.each(function () {
                // Tooltip 插件代码
            });
        };
})(jQuery);


$('div').tooltip({
    'location': 'left'
});
```

在这个例子中 ,调用 tooltip 插件时覆写了默认设置中的 location 选项 ,background-color 选项保持默认值，所以最终被调用的设定值为：

```
{
    'location': 'left',
    'background-color': 'blue'
}
```

这是一个很灵活的方式 ,提供一个高度可配置的插件 ,而无需开发人员定义所有可用的选项。

五、命名空间

正确命名空间插件是插件开发的一个非常重要的一部分。 正确的命名空间，可以保证插件将有一个非常低的机会被其他插件或同一页上的其他代码覆盖。 命名空间也可以帮助开发人员更好地跟踪方法，事件和数据。

六、插件方法

在任何情况下，一个单独的插件不应该在 jQuery.fnjQuery.fn 对象里有多个命名空间。

```
(function($) {
    $.fn.tooltip = function(options) {
        // this
    };
    $.fn.tooltipShow = function() {
        // is
    };
    $.fn.tooltipHide = function() {
        // bad
    };
    $.fn.tooltipUpdate = function(content) {
        // !!!
    };
})(jQuery);
```

为了解决$.fn 使$.fn 命名空间混乱。需要收集对象文本中的所有插件的方法，通过传递该方法的字符串名称给插件以调用它们。

```
(function($) {
```

```
var methods = {
        init: function(options) {
            // this
        },
        show: function() {
            // is
        },
        hide: function() {
            // good
        },
        update: function(content) {
            // !!!
        }
};
$.fn.tooltip = function(method) {
        // 方法调用
        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!');
```

这种类型的插件架构允许封装所有的方法在父包中 ,通过传递该方法的字符串名称和额外的此方法需要的参数来调用它们。 这种方法的封装和架构类型是 jQuery 插件社区的标准，它被无数的插件在使用，包括 jQueryUI 中的插件和 widgets。

七、事件

一个鲜为人知 bind 方法的功能即允许绑定事件命名空间。 如果插件绑定一个事件，一个很好的做法是赋予此事件命名空间。 通过这种方式，当在解除绑定的时候不会干扰其他可能已经绑定的同一类型事件。开发人员可以通过追加命名空间到需要绑定的的事件通过 '.<namespace>'。代码如下：

```
(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();
//一段时间之后... ...
$('#fun').tooltip('destroy');
```

在这个例子中，当 tooltip 通过 init 方法初始化时，它将 reposition 方法绑定到 resize 事件并给 reposition 非那方法赋予命名空间通过追加.tooltip。 稍后， 当开发人员需要销毁 tooltip 的时候，我们可以同时解除其中 reposition 方法和 resize 事件的绑定，通过传递 reposition 的命名空间给插件。 这使我们能够安全地解除事件的绑定并不会影响到此插件之外的绑定。

八、数据

通常在插件开发的时候 ,开发人员可能需要记录或者检查插件是否已经被初始化给了一个元素。 使用 jQuery 的 data 方法是一个很好的基于元素的记录变量的途径。尽管如此，相对于记录大量的不同名字的分离的 data, 使用一个单独的对象保存所有变量，并通过一个单独的命名空间读取这个对象不失为一个更好的方法。代码如下:

```
(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);
```

将数据通过命名空间封装在一个对象中，可以更容易的从一个集中的位置读取所有插件的属性。

九、总结和最佳做法

编写 jQuery 插件允许开发人员做出库，将最有用的功能集成到可重用的代码，可以节省开发者的时间，使开发更高效。 开发 jQuery 插件时，要牢记：

1.始终包裹在一个封闭的插件：

```
 (function($) {
/* plugin goes here */
```

})(jQuery);

2.不要冗余包裹 this 关键字在插件的功能范围内

3.除非插件返回特定值，否则总是返回 this 关键字来维持 chainability 。

4.传递一个可拓展的默认对象参数而不是大量的参数给插件。

5.不要在一个插件中多次命名不同方法。

6.始终命名空间的方法，事件和数据。