第7章 jQuery插件开发

课程提要

- 认识插件
- 插件机制
- 插件中的this
- 插件案例

7.1 jQuery插件入门

jQuery插件是对jQuery框架的扩展。

一个顶级jQuery开发人员必须具备这样的技术：能够使用用户自定义方法扩展jQuery。

jQuery之所以盛行，很大程度上来源于开发社区驱动的插件生态系统。据说jQuery的扩展插件有上万种之多。我们学完这一章，也可以做出我们自己的酷炫插件了。

7.1.1 插件基础

在最基本层面，创建一个jQuery插件还是挺简单的。

虽然创建一个jQuery插件很"容易"，但是我们还是要了解jQuery插件创建的规范。遵循一些最佳实践可以保证你的jQuery插件具有优化的性能、更少的bug和更强的可扩展性，并为jQuery插件与代码运行jQuery环境之间提供高度的互操作性。

7.1.1.1 插件种类

jQuery的插件主要分3种类型：

1、封装对象方法的插件（原型扩展）

这种是大多数采用的方法。操作原型，对jQuery框架侵入性较小，例如：parent()方法、append()方法和addClass()方法等dom操作方法。

2、封装全局函数的插件（静态扩展）

常见如：$.trim()方法和$.now()方法，都是jQuery内部作为全局函数的插件附加到内核上去的。

3、选择器插件　　虽然jQuery的选择器功能十分强大，但有时用户的特殊需求，还是会扩展一些自己喜欢的选择器，例如用：color('red')来把选择器匹配的元素的文字样式一次性设置为红色。

7.1.1.2 命名规范

大家在使用jQuery插件时，都需要引入一个插件相关的jQuery库，名称是不是有一定规律?

通常情况下，jQuery插件采用下面的模式进行命名：

```
jQuery.pluginName.js
```

min版也采用与上面类似的命名规范，并添加一个min标记：

```
jQuery.pluginName.min.js
```

在命名jQuery插件时包含jQuery这一名称，原因也很简单——这是一个jQuery插件，需要依赖jQuery库。在命名上声明插件的（一个或多个）依赖，是为插件使用提供一个重要说明。

min版一般也是指源码压缩版。在min版的命名中，加上“.min”也是为了让两个版本可以共存，这一命名方式方便开发人员可以根据自己的需要，在full版本和min版本中随意切换。

7.1.1.3 插件中的冲突

在我们编写jQuery插件时，插件内容多多少少也会用到jQuery框架自身的特性。但是如果我们当前环境，不单单只有jQuery框架，还有其他的一些框架。更巧合的是其他框架的核心对象也使用的$....(那么我们框架实现起来就困难了)。

这时，我们要把$的控制权让渡给第一个实现它的那个库。jQuery官方API是这样说：

这有助于确保jQuery不会与其他库的$对象发生冲突。 在运行这个函数后，就只能使用jQuery变量访问jQuery对象。例如，在要用到$("div p")的地方，就必须换成jQuery("div p")。 '''注意:'''这个函数必须在你导入jQuery文件之后，并且在导入另一个导致冲突的库'''之前'''使用。当然也应当在其他冲突的库被使用之前，除非jQuery是最后一个导入的。

解绑jQuery对$的引用，代码如下：

```
jQuery.noConflict();

// 使用 jQuery
jQuery("div p").hide();

// 使用其他库的 $()
$("content").style.display = 'none';
```

当然，也可以创建一个新的别名用以在接下来的库中使用jQuery对象。

```
// 解绑后给jQuery对象，取一个新的别名
var j = jQuery.noConflict();

// 基于 jQuery 的代码
j("div p").hide();

// 基于其他库的 $() 代码
$("content").style.display = 'none';
```

7.1.1.4 插件中的闭包

关于闭包，是指在一个函数内部，调用函数以外的变量方式。

利用闭包的特性，既可以避免临时变量影响全局空间，又可以在插件内部继续使用$作为jQuery。一举两得！

常见jQuery插件都是以如下形式：

```
(function(){
    // 这里方式代码
})();
```

当然这里也可以传递参数进去，以供内部函数调用：

```
// 为了有更好的兼容性，开始前有一个分号
(function($){
// 这里放代码
alert($);

})(jQuery);
```

上面代码看似完美，但是却如抱火卧薪。来看下面一段代码：

```
var a = 1

(function($){
    // 这里还好使不，什么原因?
    alert($);
})(jQuery);
```

改进如下：

```
var a = 1
// 为了更好的兼容性，开始前有一个分号
;(function($){ // 此处将$作为函数的形参
    // 这里放代码，无论外部环境如何，$都作为jQuery对象的缩写别名
    alert($);
})(jQuery);  // 这里将jQuery对象作为参数传入了匿名函数
```

完整使用闭包的代码格式如下：

```
;(function($){
    // 这里编写插件代码，可以继续使用$作为jQuery的别名
    // 定义一个变量
    var obj;

    // 定义一个闭包函数
    var newPlugin = function(){
        // 基于闭包原理，在这里是可以使用上面的obj
        // 但是在匿名函数外，是无法获取obj的值
    }

    // 把新插件放入全局
    $.NewPlugin=newPlugin ;
```

```
})(jQuery);
```

以上代码有几个特点：

- 使用匿名函数，减少了全局变量污染
- 使用闭包，对内部信息进行了保护
- 在定义完毕之后，把新函数挂到jQuery，完成插件扩展

OK，说了这么多，在这大家只要记住一个基本语法就OK。如下：

```
;(function($){
    // 此处编写jQuery插件代码
})(jQuery)
```

7.1.2 jQuery插件机制

jQuery中主要提供了两种扩展方法：

第一种，对jQuery函数prototype属性的别名（jQuery.fn）进行扩展。

第二种，采用jQuery.extend()方法进行（静态）扩展。

7.1.2.1 使用jQuery.fn（原型扩展）

如果之前看到jQuery源码，应该看到了如下代码：

```
jQuery.fn = jQuery.prototype = { // 扩展代码}
```

前面我们了解过JavaScript中的原型继承，那么这里的jQuery继承同样是对原型的扩展。

HTML如下：

```
<!DOCTYPE HTML>
 <html>
   <head>
      <meta charset="utf-8" />
      <title>测试</title>
   </head>
   <body>
      <div id=”div1”>我一直都在这</div>
   <body>
 </html>
```

JavaScript如下：

```
// jQuery原型扩展
/*
 需求：扩展jQuery对象，支持统一设置宽度和高度的方法
     $.fn.扩展方法名=function(){}
*/
;(function(){
    //   $("#div1").height(150); 原生单独设置高度
    //   $("#div1").width(150); 原生单独设置宽度
    $.fn.setSize=function(height,width){
        $(this).height(height);
        $(this).width(width);
    };
})(jQuery);
```

在这里了解到的扩展只是一种单一扩展，如果我们要扩展多个，一个个写过于麻烦。jQuery插件机制中js，提供一种“批量”扩展，代码如下：

```
// jQuery原生方法
$(function(){
    //  console.debug($("#div1").html());// 取值
    //  $("#div1").html("我是来替换你的");// 赋值
});
$.fn.extend({
    // 扩展方法1
    setHtml:function(htmlText){
        this.html(htmlText);
    },
    // 扩展方法2
    getHtml:function(){
        return this.html();
    }

    // 扩展方法N .....
});
// 测试代码
$(function(){
    console.debug($("#div1").getHtml());
    $("#div1").setHtml("我是来替换你的！！");
});
```

7.1.2.2 使用jQuery.extend方法（静态扩展）

jQuery.extend方法是jQuery插件机制默认自带的方法，可以方便的对jQuery对象本身进行扩展。直接来看下面这段代码：

```
$.extend({
    log: function(message) {
        var now = new Date(),
            y = now.getFullYear(), // 获取年
            m = now.getMonth() + 1, // JavaScript中月份是从0开始的，稍作处理!
            d = now.getDate(), // 获取日期
```

```
            h = now.getHours(),
            min = now.getMinutes(),
            s = now.getSeconds();
        var time = y + '-' + m + '-' + d + ' ' + h + ':' + min + ':' + s;
        console.log(time + ' myLog: ' + message);
    }
});

$.log(‘你的插件已经初始化完毕！！’);
```

7.2 插件进阶

7.2.1 通用指南

7.2.1.1 用好this关键字

this通常用来指向执行当前上下文的对象。而在不同场景下，代表的意义又不一样。在jQuery开发中，this关键字通常引用的是当前正在操作的DOM对象；而在jQuery插件上下文中，this关键字引用的是当前jQuery实例本身。

如果入到如下自定义插件代码，如下：

```
$(this).html();
```

其实可以写成:

```
this.html();
```

唯一例外的就是当前jQuery集合中，遍历所有元素时。在$.each的循环体内，this引用的是当前这次循环时操作的dom对象本身。

```
;(function($){
    $.fn.bluesy= function(){
        // this引用的是jQuery对象本身
        return this.each(function(){
            // 循环体内，this关键字引用的是DOM元素
            $(this).css({
                "color":"#fff"
                ,"background-color":"lightblue"
                ,"text-shadow":"5px 5px 1px gray" // 水平，垂直，模糊，颜色
            })
        });
    };
})(jQuery);
```

7.2.1.2 总是保证$指向jQuery

这个大家应该非常熟悉：

```
(function($){
    // 无论全局$是什么意思
    // 这里的$总是表示jQuery
})(jQuery);
```

在这之上还有一种常见的增强，直接看代码：

```
(function($, window, document, undefine){
    // 无论全局$是什么意思
    // 这里的$总是表示jQuery
})(jQuery, window, document);
```

上面代码，第2和3参数，为window和document创建了一个局部引用，可以让这两个对象最精简，同时还可以缩短要查询的表，从而加载在函数体内加快对这些对象的访问速度。

最后一个参数，undefined看起来像是没有给他赋值，但是这要说道低版本的ECMAScript标准中，居然是可以对undefined重新赋值。如果环境中有人恶意的重新定义undefined的值，那么这里就是一个潜在bug的来源。

7.2.1.3 正确使用分号

在定义插件代码前加一个分号，是一个非常明智之举。

7.2.1.4 尽可能返回jQuery对象

jQuery设计的核心就是要让API可读性更好，使用更流畅。体现这一核心的关键点之一就是jQuery支持链式方法调用。

在jQuery编程实践中，支持链式调用的API通常广受好评。其目的就是要让方法的返回值指向jQuery对象中的this。

```
;(function($){
    $.fn.bluesy= function(){
        // this引用的是jQuery对象本身
        return this.css({
            "color":"#fff"
            ,"background-color":"lightblue"
            ,"text-shadow":"5px 5px 1px gray" // 水平，垂直，模糊，颜色
        });
    };
})(jQuery);
```

在如下场景，就会显得十分方便:

```
<!-- html -->
<p style="display: none;">学习前端开发，至此将改变您的人生</p>
<p style="display: none;">侯老师的微博</p>
// JavaScript
$(function(){
    $("a").bluesy().show();
});
```

值得注意的是，并不是所有方法都必须返回jQuery对象。如jQuery原生的html()方法，在不传值的情况下应该返回的是当前元素所有的html文本内容；又如jQuery原生的type方法，返回的是指定对象的类型字符串。想如上这些就没法返回对象了。

我们自己来做一个例子：

```
;(function($){
    $.fn.checkWeight=function(weight){
        if(weight < 120){
            return “你太瘦了！！”;
        }else if(weight >=120 && weight <=150){
            return “你很标准，继续保持！！”;
        }else if(weight > 150){
            return “该减肥咯！！”;
        }
    }
})(jQuery)
```

像这种方法，返回的就只是几个预算结果，我们不需要再对结果进行其他处理了。

7.2.1.5 遍历对象

在通常情况下，jQuery插件的设值操作，会影响查询到的所有元素。如果我们要自己来实现，就需要用到$.each来实现集合遍历了。如下所示：

```
(function($) {
    $.fn.randomInt = function() {
        return this.each(function() {
            // 虽然text方法可以一次性操作下面所有参数
            // 但是如果每一次的值都不一样，那么我们就得循环
            $(this).text((Math.random() * 1000).toFixed());
        });
    };
})(jQuery);
```

但有一些本来就很强大的方法，就没必要自己循环了，如前面：

```
;(function($){
    $.fn.bluesy= function(){
        return this.each(function(){
            $(this).css({ // 其实css方法本来就是批量设值的
                "color":"#fff"
                ,"background-color":"lightblue"
                ,"text-shadow":"5px 5px 1px gray" // 水平，垂直，模糊，颜色
            })
        });
    };
})(jQuery);
```

改成下面，效果一样：

```
;(function($){
    $.fn.randomNum= function(){
        // this引用的是jQuery对象本身
        return this.css({
            "color":"#fff"
            ,"background-color":"lightblue"
            ,"text-shadow":"5px 5px 1px gray" // 水平，垂直，模糊，颜色
        });
    };
})(jQuery);
```

7.2.2 最佳实践

7.2.2.1 一个插件一个空间名

现在的插件开发，成本越来越低。这也导致部分人们在命名上经常会有冲突的情况。所有逼得部分开发者，采用如下定义：

```
jQuery.fn.myPluginInit = function(){
    // init
};
jQuery.fn.myPluginShow = function(){
    // show
};
jQuery.fn.myPluginHide= function(){
    // hide
};
jQuery.fn.myPluginMove = function(){
    // move
};
```

实际上，我们只需要稍作调整，为插件创建单一入口，并采取一个优雅方式把方法暴露出去。

```
jQuery.fn.myPlugin = function(){
    // 单个命名空间
};
```

这是最佳实践的一个出发点，在此模式下，把所有方法以私有方法定在命名空间下，然后再结合闭包把方法查询表对外暴露出去。用第一参数，传入一个方法字符串；方法的第二个参数（开始），可以指定是这个方法所需的参数。在jQuery插件的开发社区中，这种类型的方法封装和架构是一种标准，无数插件都是用了这种模式，如jQuery EasyUI等都是。大致代码如下：

```
(function ($){
    var methods = {
        init : function(){
            // init
        };
        show : function(){
            // show
        };
```

```
    hide: function(){
        // hide
    };
    move : function(){
        // move
    };
};

 $.fn.myPlugin = function(methodName){
    if(methods[methodName]){
        return methods[methodName].apply(this,
                                        Array.prototype.slice.call(arguments, 1));
    }else if(typeof methodName=== 'object' || !methodName){
        return methods.init.apply(this, arguments );
    }else {
        $.error('方法'+methodName+'不存在!!');
    }

}
})(jQuery);
```

测试:

```
// 调用init
$("div").myPlugin();
// 调用show
$("div").myPlugin ("show");
// 调用move
$("div").myPlugin ("move","top:300");
```

7.2.2.2 为插件的事件定义命名空间

```
(function ($){
    var methods = {
        init : function(){
            $(window).on('resize.myPlugin ', methods.show )
        },
        destroy: function(){
            $(window).off('resize.myPlugin ')
        },
        show : function(){
            // show
        },
        hide: function(){
            // hide
        },
        move : function(){
            // move
        }
    },

        $.fn.myPlugin = function(methodName){
```

```
        }

})(jQuery);
```

7.2.2.3 接收一个配置参数

还记得我们前面定义的一个setSize方法，传入宽度和高度两个参数：

```
$.fn.setSize=function(height,width){
    $(this).height(height);
    $(this).width(width);
};
// 如果传入一个参数，比如只需要把宽度给为200
$(function(){
    $("div").setSize(null,200); //高度没值还要用null站位，是不是很烦
});
```

下面是改进版：

```
// 传入一个JSON配置对象
$.fn.setSize=function(option){
    $(this).height(option.height);
    $(this).width(option.width);
};
$("#div2").setSize({width:100}); // 这样是不是很方便
```

7.2.2.4 为插件设置公开的默认值

```
(function ($){
    var methods = {
        init : function(){
            $(window).on('resize.myPlugin ', methods.show )
        };
    };
    $.fn.myPlugin = function(methodName, options){
        // 避免覆盖全局默认
        var setting = $.extend({}, $.fn.myPlugin.defaults, options);
    }
    // 对全局暴露的默认参数（允许修改）
    $.fn.myPlugin.defaults = {
        width:500,
        height:400,
        color : 'blue'
    };
})(jQuery);
```

示例：

```
<script>
    ;(function($){
```

```
        var methods={
            init:function(opt){
                this.width(opt.width);
                this.height(opt.height);
                this.css('background',opt.color);
            },
            show:function(){
                this.html('今天星期五，我们在上课');
            },
            off:function(){
                console.log('关闭');
            }
        }
        $.fn.myplugin=function(method,option){
            var def={width:500,height:300,color:'#f00'}; //设置默认值
            var setting=$.extend({},def,option); //使用extend合成最终的设置
            methods[method].call(this,setting); //自动调用对应的方法
            return this; //返回jQuery对象自身，便于链式调用
        }
    })(jQuery);

    $('#box1').myplugin('init',{}).myplugin('show'); //链式调用
</script>
```

7.3 案例-数据表格插件

需求：制作一个数据表格插件。

功能：

1.完成表格数据的动态显示；

2.定义插件的默认参数；

3.实现表格的自定义事件。

7.3.1 完成表格数据的动态显示

1.制作一个表格模板，使用数据表格插件

1）创建html网页

2）编写表格代码，CSS样式

3）使用数据表格插件

插件名：showdata

引入jquery

引入插件文件 jquery.showdata.js

调用插件，并传参数 .showdata({url:'数据接口'});

2.开发插件

1）固定格式

2）获取数据

怎么去获取远程的数据？ iframe活动框架。

注意： iframe是通过异步加载数据的，必须等加载完以后才能获取到数据。

3）将数据拼接成HTML输出到页面上

7.3.2 定义插件的默认参数

以对象形式传参数，再在插件内部设置默认值。

```
$.extend 合并默认值对象和参数对象，形成设置对象
```

7.3.3 实现表格的自定义事件

1.使用插件时传入自定义事件

```
$('#userlist').showdata({
    url:"./data.json",
    click:function(){
        this.css('background','#f00');
    },
    mouseover:function(){
        console.log('鼠标划过事件');
    }
});
```

2.开发插件时接收到自定义事件并实现功能

a. 接收用户自定义事件并去除url属性

b. 遍历对象获取对象的键名（事件名称）和值（事件处理函数）

```
$.each(e,function(i,v){
    $(_that).on(i,'tr',v);
});
```

7.4 课程小结

- 认识插件
- 插件机制
- 插件中的this
- 插件案例

7.5 实训

练习上面所讲内容。