jQuery插件开发汇总_jquery_脚本之家

这篇文章主要为大家详细介绍了jQuery插件开发的相关资料,需要的朋友可以参考下

一、jQuery插件开发两个底层方法

jQuery.extend([deep], target [, object1] [, objectN])

将两个或更多对象的内容合并到第一个对象。

1、deep 如果是true，合并成为递归（又叫做深拷贝）

2、target 一个对象，如果附加的对象被传递给这个方法将那么它将接收新的属性，如果它是唯一的参数则将扩展jQuery的命名空间，这对于插件开发者希望向 jQuery 中添加新函数时是很有用的。

3、object1 一个对象，它包含额外的属性合并到第一个参数

4、包含额外的属性合并到第一个参数

当我们提供两个或多个对象给\(.extend()，对象的所有属性都添加到目标对象（target参数） 目标对象（第一个参数）将被修改，并且将通过\).extend()返回。然而，如果我们想保留原对象，我们可以通过传递一个空对象作为目标对象：

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

在默认情况下，通过$.extend()合并操作是不递归的;

```
var object1 = {apple: 0,banana: {weight: 52, price: 100},cherry: 97};
var object2 = {banana: {price: 200},durian: 100};
$.extend(object1, object2);
//{apple: 0, banana: {price:200}, cherry: 97, durian: 100}
$.extend(true, object1, object2);
//{apple: 0, banana: {weight: 52, price:200}, cherry: 97, durian: 100}
jQuery.fn.extend()
```

在jQuery源码中有jQuery.fn = jQuery.prototype = function(){……}即指向jQuery对象的原型链，对其它进行的扩展，作用在jQuery对象上面;

总结

1、jQuery.extend()能够创建全局函数或选择器，在实际开发中常使用jQuery.extend()方法作为插件方法传递系列选项结构的参数

2、jQuery.fn.extend()能够创建jQuery对象方法，一般用此方法来扩展jQuery的对象插件

二、jQuery插件开发通用框架

```
;(function($, window, document, undefined){
 //Plugin code here
})(jQuery, window, document);
```

使用分号是为了防止因前面的代码没有使用分号而导致插件函数不能正确解析

传入jQuery是为了确保在匿名函数中正确的使用jQuery对象，防止多库共存时$冲突

传入window、document并非必须，只不过为了更快的访问window和document
传入undefined是为了防止undefined变量被更改，确保undefined的准确性

三、jQuery插件开发的3种形式
1、类级别开发(封装全局函数的插件)
类级别写法：

```
//方式1
;(function($, window, document, undefined){
 $.pluginName = function(){
  //Plugin implementation code here
 };
})(jQuery, window, document);
//方式2 当全局函数较多时
;(function($, window, document, undefined){
 $.extend({
  pluginName = function(){
   //Plugin code here
  };
 })
})(jQuery, window, document);
```

调用方法：$.pluginName();

2、对象级别的插件开发
对象级别插件写法：

```
//方式1
;(function($, window, document, undefined){
 $.fn.pluginName = function(options) {
  return this.each(function() {
  //this关键字代表了这个插件将要执行的jQuery对象
  //return this.each()使得插件能够形成链式调用
   var defaults = {
    //pro : value
   };
   var settings = $.extend({}, defaults, options);
   // plugin implementationcode here
  });
 }
})(jQuery, window, document);
//方式2
;(function($, window, document, undefined){
 $.fn.extend({
```

```
  pluginName : function(){
   return this.each(function(){
    // plugin code here
   });
  };
 })
})(jQuery, window, document);
//方式3 这种类型的插件架构允许您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。
;(function($, window, document, undefined){
  // 在我们插件容器内，创造一个公共变量来构建一个私有方法
  var privateFunction = function() {
   // code here
  }
  // 通过字面量创造一个对象，存储我们需要的公有方法
  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，对来自主函数，并进入每个方法中的选择器其中的每个单独的元素都执行代码
    return this.each(function() {
     // 为每个独立的元素创建一个jQuery对象
     var $this = $(this);
     // 创建一个默认设置对象
     var defaults = {
      propertyName: 'value',
      onSomeEvent: function() {}
     }

     // 使用extend方法从options和defaults对象中构造出一个settings对象
     var settings = $.extend({}, defaults, options);
     // 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，这样会毁掉一切的
```

```
    var method = arguments[0];
    // 检验方法是否存在
    if(methods[method]) {
     // 如果方法存在，存储起来以便使用
     // 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.call(this);
  }
})(jQuery, window, document);
//方式4 面向对象的插件开发 将原型和构造函数组合使用，使得通过构造函数创建的每个实例都能继承相关属性与方法
;(function($, window, document, undefined){
 //定义Beautifier的构造函数
 var Beautifier = function(ele, opt) {
  this.$element = ele;
  this.defaults = {
  'color': 'red',
  'fontSize': '12px',
  'textDecoration':'none'
  };
  this.options = $.extend({}, this.defaults, opt);
 }
 //定义Beautifier的原型方法
 Beautifier.prototype = {
  beautify: function() {
  return this.$element.css({
   'color': this.options.color,
   'fontSize': this.options.fontSize,
   'textDecoration': this.options.textDecoration
  });
```

```
  }
 }
 //在插件中使用Beautifier对象
 $.fn.myPlugin = function(options) {
  //创建Beautifier的实体
  var beautifier = new Beautifier(this, options);
  //调用其方法
  return beautifier.beautify();
 }
})(jQuery, window, document);
```

调用方法：$.fn.pluginName();

3、通过$.widget()应用jQuery UI的部件工厂方式创建
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

四、编写JQuery插件需要注意的地方：
1、插件的推荐命名方法为：jquery.[插件名].js
2、所有的对象方法都应当附加到JQuery.fn对象上面，而所有的全局函数都应当附加到JQuery对象本身上。
3、可以通过this.each() 来遍历所有的元素
4、在jQuery开发中，this关键词通常引用的是当前正在操作的DOM元素，但在当前的jQuery插件上下文中，this关键词引用的是当前jQuery实例自身，唯一的例外是在当前jQuery集合中遍历所有元素时，$.each循环体内的this引用的是这一轮遍历所暴露的DOM元素
5、所有方法或函数插件，都应当以分号结尾，否则压缩的时候可能会出现问题。为了更加保险写，可以在插件头部添加一个分号（；），以免他们的不规范代码给插件带来 影响。
6、插件应该返回一个JQuery对象，以便保证插件的可链式操作。

以上就是jQuery插件开发的知识点汇总，希望对大家的学习有所帮助。

您可能感兴趣的文章:

- 1