jQuery.extend(object) ： 为jQuery类添加类方法，可以理解为添加静态方法

$拓展的方法是静态方法，可以使用$直接调用，其拓展的方式有两种，一般使用$.extend({});

```
1.    $.test = function(a,b) {
            return a+b;
  };

2 。$.extend({
          test:function(a,b) {
                        return a+b;
            }
    });
```

使用 ： $.test(4,4)

jQuery.fn.extend（object） ： 为jQuery类添加实例方法，new对象才能使用

jQuery.fn.extend(object) =

jQuery.prototype.extend(object)=jquery.fn.object

jQuery.fn.extend()的调用把方法扩展到了对象的prototype上，所以实例化一个jQuery对象的时候，它就具有了这些方法

而$.fn拓展的方法是实例方法，必须由“对象”$("")来调用，其拓展的方式同样有两种，一般使用$.fn.extend({ })。

```
    $.fn.test = function() {
              return $(this).val();
    };

  $.fn.extend({
            test:function() {
                  return $(this).val();
```

```
        }
  });
```

使用： $("#name").test()

$.fn是指jquery的命名空间，加上fn上的方法及属性，会对jquery实例每一个有效。

如扩展$.fn.abc(),即$.fn.abc()是对jquery扩展了一个abc方法,那么后面你的每一个jquery实例都可以引用这个方法了.

那么你可以这样子：$("#div").abc();

```
jQuery.fn = jQuery.prototype = {

};


(function( $ ){
$.fn.tooltip = function( options ) {
};
//等价于
var tooltip = {
function(options){
}
};
$.fn.extend(tooltip) = $.prototype.extend(tooltip) = $.fn.tooltip
})( jQuery );
```

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

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

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){returna+b;}
});
```

```
(function($){
2      $.fn.pluginName = function(){
3            // 插件代码写在这里
4       }
5 })(jQuery);
```

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

匿名函数&闭包的使用

闭包的作用：一个是前面提到的可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中

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

匿名函数：立即执行函数；相当于先申明一个函数，声明完后直接调用,第二个括号的意思是"立即调用"；

优点：防止污染命名空间 ，利用了闭包特性，因为有些对象我们并不经常使用并立即执行它，由于外部无法引用它内部的变量，因此在执行完后很快就会被释放，关键是这种机制不会污染全局对象

```
(function( $ ){
    .....
})(jQuery)
```

匿名函数&闭包

```
var testLambda = (function(){
    function testLambda(){
      alert("执行这个");
    }
    return {
        testLambda : testLambda,
```

```
        resetCounter: function(){
            alert("执行第二个");
    };
})();
```

调用方式 ： testLambda.testLambda();
testLambda.resetCounter();

```
function( $, window, document, undefined ){} ){
      //...code
}(jquery,window,document)
```

```
(function(name){
    alert(name);     //输出xiaoming
})("xiaoming");
```

jQuery插件开发方式主要有三种：

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

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。所以我们调用通过$.extend()添加的函数时直接通过$符号调用（$.myfunction()）而不需要选中DOM元素($('#example').myfunction())。请看下面的例子。

```
$.extend({
    sayHello: function(name) {
        console.log('Hello,' + (name ? name : 'Dude') + '!');
    }
})
$.sayHello(); //调用
$.sayHello('Wayou'); //带参调用
```

但这种方式无法利用jQuery强大的选择器带来的便利，要处理DOM元素以及将插件更好地运用于所选择的元素身上，还是需要使用第二种开发方式。你所见到或使用的插件也大多是通过此种方式开发。

```
$.fn.myPlugin = function(options) {
      var defaults = {
          'color': 'red',
          'fontSize': '12px'
       };
     var settings = $.extend({},defaults, options);//将一个空对象做为第一个参数
     //插件的具体功能实现
}
```

面向对象的插件开发，用自调用匿名函数包裹你的代码

优点：将需要的重要变量定义到对象的属性上，函数变成对象的方法，当我们需要的时候通过对象来获取，一来方便管理，二来不会影响外部命名空间，因为所有这些变量名还有方法名都是在对象内部

前面加上　；　防止与别人的代码相连时，别人忘记以；结尾，导致我们的代码没法编译

```
;(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);
```

2.缓存

再来看一个例子，设想我们有一个处理过程很耗时的函数对象，每次调用都会花费很长时间，

那么我们就需要将计算出来的值存储起来，当调用这个函数的时候，首先在缓存中查找，如果找不到，则进行计算

```
1.  var CachedSearchBox = (function(){
2.      var cache = {},
3.          count = [];
4.      return {
5.          attachSearchBox : function(dsid){
6.              if(dsid in cache){//如果结果在缓存中
7.                  return cache[dsid];//直接返回缓存中的对象
8.              }
9.              var fsb = new uikit.webctrl.SearchBox(dsid);//新建
10.             cache[dsid] = fsb;//更新缓存
11.             if(count.length > 100){//保正缓存的大小<=100
12.                 delete cache[count.shift()];
13.             }
14.             return fsb;
15.         },
16.
17.         clearSearchBox : function(dsid){
18.             if(dsid in cache){
19.                 cache[dsid].clearSelection();
20.             }
21.         }
22.     };
```

```
23. })();
```

4 闭包的另一个重要用途是实现面向对象中的对象，传统的对象语言都提供类的模板机制，

```
1. function Person(){
2.     var name = "default";
3.
4.     return {
5.         getName : function(){
6.             return name;
7.         },
8.         setName : function(newName){
9.             name = newName;
10.         }
11.     }
12. };
13.
14.
15. var john = Person();
16. print(john.getName());
```