jQuery 插件技术研究

------作者：夏俊　日期：2012-05-04

目录

jQuery 插件技术研究........1
1.1 研究 jQuery 插件技术的源头........1
1.2 对$.extend 的理解........9
1.3 Javascript 里的浅拷贝和深拷贝........12
1.3.1 浅拷贝........12
1.3.2 深拷贝........13
1.4 $.extend 用法详述........16
1.5 复制一个$.extend 方法........23
1.6 分析$.extend 源码........28
1.7 创建属于 jQuery 对象的插件........35
1.8 jQuery 框架里如何定义 jQuery 对象........37
1.9 jQuery 框架里定义 jQuery 对象的原理........43
1.10 再看看$.extend 和$.fn.extend 的原理........45

1.1 研究 jQuery 插件技术的源头

最近写了个网站，当时借鉴了很多相关网站前端技术，为了让客户的体验更加好，我在

网站前端加入了相当多的校验代码，因此代码显的特别臃肿。虽然开发过程中我将前端代码重构了三次，但是我还是对我原来写的代码不满意。五一假期我好好复习了下 javascript 的知识，这里试着总结下我对代码不满意的地方，大致有以下几点：

（1） 我一直都在琢磨 jQuery 源码的写法，觉得 jQuery 是我见过写的最棒的代码，因此代码里的写了大量的普通的 function，这个很不符合 jQuery 的风格，这点让我很不爽。

（2） 不同的页面其实有很多类似的操作，这些操作是可以抽取成为公共的方法，例如：不同页面里的文本框、下拉框、多选框和单选框，它们大多都会有 blur、focus 和 keyup 事件，如果我一个个绑定这些事件，真是一件很不爽的事情，而且最终的代码会让人感觉很没有档次，因此事件绑定应该要好好的封装下，这样代码的质量会更高，通用性更好。

（3） 页面里的 js 代码过多。我现在写前端页面总会考虑我这么开发会让我的页面运行的更快吗？提高页面的相应速度的方法很多，但是有几点只要你在开发时候处处留心很容易做到，例如：减小页面的大小，但是如果你的页面的 js 代码过多了，页面的大小会变大，或多或少会影响到页面在网络中传输的速度；好的 js 代码应该是尽力的把统一的方法写到外部的 js 文件，而且这个 js 文件数量要尽量少。而我的页面里 js 代码过多，这也是一个败笔。

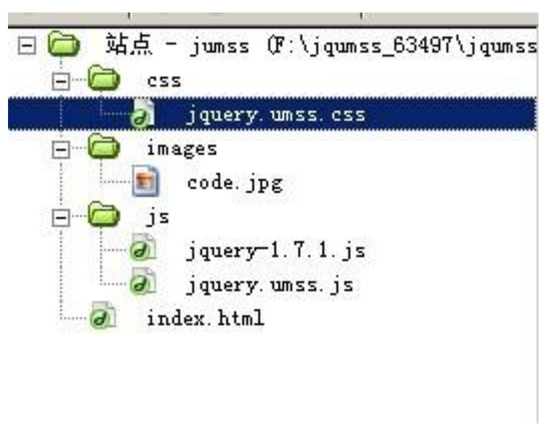
（4） 我一直都觉得用 json 的格式定义 javascript 变量是一个十分优雅的编程模式，但是这个技能我并不太熟练，因此当项目很赶的时候我不自觉的还是按原来的套路写 js 代码，这样的代码现在回头看看，真是有点原始（幼稚）。

（5） 既然是用 jQuery 框架做开发，那么代码应该尽量发挥出 jQuery 代码的特点，用最好的 jQuery 风格的代码去开发，才会让你的代码显得优质（也许 jQuery 代码的效率和原生态的 js 代码有性能的差别，不过我们平时做的项目里，jQuery 的效率的缺陷基本

都可以忽略不计），多去思考如何让你写的 jQuery 代码写的更好，是对你使用语言的尊重，这种尊重一定能让你进步的更快。

五一节我想到了用 jQuery 插件的模式重新封装我的代码，让我的代码和 jQuery 框架融为一体，此外尽力把公共的方法都抽取出来放到统一的 js 文件里，页面最好只留一定要写到页面里的代码。

下面就是我写的代码，首先还是先看看我的目录结构：



index.html 的代码如下：

```
<script type="text/javascript">
$(document).ready(function(){
    bindFormAttrEvt();
});

function bindFormAttrEvt(){
    var formattrs =
['#email','#mphone','#onepwd','#twopwd','#code','#codeimg','#b
tn'];
    var evttypes = 'focus blur keyup click';
    var evtMethods =
{'focus':evtFocusMethod,'blur':evtBlurMethod,'keyup':evtKeyUpM
ethod,'click':evtClickMethod};
    // 绑定事件
    $.bindEvtByTypeUmss(formattrs,evttypes,evtMethods);
    // 初始化验证码
    $.createUmssCode($('#codeimg'));
}

// 点击(click)事件方法
```

```
function evtClickMethod(evt){
    var objid = evt.target.id;
    switch(objid){
        case 'codeimg':
            $.createUmssCode($('#codeimg'));
        break;
        case 'btn':
            var flag = true;
            var checkarrs =
['email','mphone','onepwd','twopwd','code'];
            $.each(checkarrs,function(i,data){
                var evt = {'target':{'id':data}};
                if (flag){
                    flag = evtBlurMethod(evt);
                }
            });
            console.log(flag);
            if (flag){
                alert('数据提交成功！！！！！');
            }
        break;
        default:

        break;
    }
}

// 失去焦点（blur）事件方法
function evtBlurMethod(evt){
    var objid = evt.target.id;
    /*if (evt.target.id != null){
        objid = evt.target.id;
    }else{
        objid = evt;
    }*/

    switch(objid){
        case 'email':
            if ($.isEmailUmss($('#email').val()) == false){
                $('#emailmsg').text('电子邮件格式不正确!');
                return false;
            }else{
                $('#emailmsg').text('恭喜你！电子邮件格式正确!');
                return true;
```

```
            }
        break;
        case 'mphone':
            if ($.isMPhoneUmss($('#mphone').val()) == false){
                $('#mphonemsg').text('手机号码不正确!');
                return false;
            }else{
                $('#mphonemsg').text('恭喜你! 手机号码正确!');
                return true;
            }
        break;
        case 'onepwd':
            if ($.isPwdLengthMinUmss($('#onepwd').val())){
                $('#onepwdmsg').text('密码长度不能少于6个字符!');
                return false;
            }else if ($.isPwdLengthMaxUmss($('#onepwd').val())){
                $('#onepwdmsg').text('密码长度不能多于32个字符!');
                return false;
            }else if ($.pwdFormatCheckUmss($('#onepwd').val())){
                $('#onepwdmsg').text('密码由6~32位不连续的数字或英
文字母组成!');
                return false;
            }else{
                $('#onepwdmsg').text('恭喜你! 密码格式正确!');
                return true;
            }
        break;
        case 'twopwd':
            if ($('#onepwd').val() != $('#twopwd').val()){
                $('#twopwdmsg').text('两次密码不一致!');
                $('#twopwd').val('');
                return false;
            }else{
                $('#twopwdmsg').text('恭喜你! 密码一致的哈!');
                return true;
            }
        break;
        case 'code':
            if ($('#code').val() != $.umsscode){
                $('#codemsg').text('验证码输入不正确!');
                return false;
            }else{
                $('#codemsg').text('恭喜你! 验证码输入正确!');
                return true;
```

```
            }
        break;
        default:

        break;
    }
}

// 焦点（blur）事件方法
function evtFocusMethod(evt){
    var objid = evt.target.id;
    switch(objid){
        case 'email':
            $('#emailmsg').text('请输入电子邮件!');
        break;
        case 'mphone':
            $('#mphonemsg').text('请输入手机号码!');
        break;
        case 'onepwd':
            $('#onepwdmsg').text('请输入密码!');
        break;
        case 'twopwd':
            $('#twopwdmsg').text('请输入密码again!');
        break;
        case 'code':
            $('#code').text('请输入验证码!');
        break;
        default:

        break;
    }
}

// 键盘输入(keyup)事件
function evtKeyUpMethod(evt){
    var objid = evt.target.id;
    switch(objid){
        case 'email':
            $('#emailmsg').text('你输入的字符长度是:' +
$('#email').val().length);
        break;
        case 'mphone':
            $('#mphonemsg').text('你输入的字符长度是:' +
$('#mphone').val().length);
```

```
        break;
        case 'onepwd':
            $('#onepwdmsg').text('你输入的字符长度是:' +
$('#onepwd').val().length);
        break;
        case 'twopwd':
            $('#twopwdmsg').text('你输入的字符长度是:' +
$('#twopwd').val().length);
        break;
        case 'code':
            $('#codemsg').text('你输入的字符长度是:' +
$('#code').val().length);
        break;
        default:

        break;
    }
}
    </script>
```

jquery.umss.js 的代码：

```
//为了避免函数、对象以及变量和jQuery其他的函数、对象及变量冲突，所有方法都以Umss结束
;(function($){
    var emailregex =
/^([a-zA-Z0-9_-]{1,})((.[a-zA-Z0-9_-]{1,}){0,})@([a-zA-Z0-9_-]{1,})((.[a-zA-Z0-9_-]{1,}){1,})$/,
        mphoneregex = /^1[3|4|5|8][0-9]\d{4,8}$/,
        pwdregex = /^[a-zA-Z0-9]{6,32}$/;

    $.extend({
        'umsscode':'',
        'isEmailUmss':function(val){
            return emailregex.test(val);
        },
        'isMPhoneUmss':function(val){
            return mphoneregex.test(val);
        },
        'isPwdLengthMinUmss':function(val){
            return (val.length < 6)?true:false;
        },
        'isPwdLengthMaxUmss':function(val){
            return (val.length > 32)?true:false;
        },
```

```
        'pwdFormatCheckUmss':function(val){
            return !pwdregex.test(val);
        },
        'createUmssCode':function(jobj){
            $.umsscode = '';
            var codelen = 6;//验证码的长度
            var selectChar = new
Array(0,1,2,3,4,5,6,7,8,9,'A','B','C','D','E','F','G','H','I',
'J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y
','Z');//所有候选组成验证码的字符，当然也可以用中文的
            for (var i = 0;i < codelen;i++){
                var charindex = Math.floor(Math.random()*36);
                $.umsscode += selectChar[charindex];
            }
            jobj.val($.umsscode);
        },

'bindEvtByTypeUmss':function(formattrs,evttypes,evtMethods){
            $.each(formattrs,function(ind,obj){
                $(obj).bind(evttypes,function(evt){
                    var type = evt.type;
                    switch(type){
                        case 'blur':
                            evtMethods.blur(evt);
                        break;
                        case 'focus':
                            evtMethods.focus(evt);
                        break;
                        case 'keyup':
                            evtMethods.keyup(evt);
                        break;
                        case 'click':
                            evtMethods.click(evt);
                        break;
                        default:
                        break;
                    }
                });
            });
        }
    });
   })(jQuery)
```

jquery.umss.css 的代码如下：

```
#codeimg{
    background-image:url(../images/code.jpg);
    font-family:Arial;
    font-style:italic;
    color:Red;
    border:0;
    padding:2px 3px;
    letter-spacing:3px;
    font-weight:bolder;
    cursor:pointer;
}
.unchanged{
    border:0;
    width:80px;
}
form p {
    margin-left:300px;
}

form p input[type='button']{
    margin-left:150px;
    }
```

为了以这样的方式写出代码，我研究了下 jQuery 插件技术，发现 jQuery 插件技术还是非常有内涵的技术，下篇里我就会和大家好好聊下 jQuery 的插件技术。

(注意：代码最好在 firefox 或是 chrome 里运行，代码里还存在一点小错误，我在下篇里会指出并修复)

1.2 对$.extend 的理解

上面的代码里我编写 jQuery 插件使用到了$.extend 方法。这里要讲讲我以前对 jQuery 插件开发的误解，这种误解源自于我对 jQuery 插件开发理解的肤浅。

在我前一家公司，有位做前端的同事很喜欢把自己代码封装成 jQuery 插件，他曾经对我说：jQuery 插件技术是 jQuery 最让人激动人心的技术，关键就是使用 extend 方法，当时我阅读一些关于 jQuery 技术的资料，大多一开始都会提到 extend 方法的使用，可能自己学

习的时候不太仔细，认为 jQuery 插件技术就是使用 extend 封装好 javascript 代码，但我每次查看 jQuery 手册对 extend 的解释又很让我费解，手册上说来说去 extend 方法只不过用于复制对象的方法。

虽然上面我用 extend 成功写出了一个 jQuery 插件，对 extend 方法理解的疑惑任然没有破除，因此这里我要从文档的描述里的内容好好的研究下 extend 方法到底是咋回事。

jQuery 手册对 jQuery.extend 的解释：

jQuery.extend([deep], target, object1, [objectN]) 返回值:Object
用一个或多个其他对象来扩展一个对象，返回被扩展的对象。 如果不指定 target，则给 jQuery 命名空间本身进行扩展。这有助于插件作者为 jQuery 增加新方法。 如果第一个参数设置为 true，则 jQuery 返回一个深层次的副本，递归地复制找到的任何对象。否则的话，副本会与原对象共享结构。 未定义的属性将不会被复制，然而从对象的原型继承的属性将会被复制。

下面我逐句分析 jQuery.extend 方法的功能。

（1） **用一个或多个其他对象来扩展一个对象，返回被扩展的对象。**这句话很精辟，它概括了 extend 作用的精髓，extend 就是太上老君的炼丹炉，我们把各种不同的对象投进这个丹炉里就会产生一个融合这些对象所有功能的超级对象，这就是 extend 方法的作用，这个可以用数学公式形象的表述就是 A+B=AB。s

（2） **如果不指定 target，则给 jQuery 命名空间本身进行扩展。这有助于插件作者为 jQuery 增加新方法。**要理解这句话，就得分析下 extend 的参数了。在 jQuery1.7 的中文手册里把参数分为两个版本：

- 版本 V1.0：target（object），[object1（object）],[objectN（object）]，（圆括号里的内容是参数的类型），参数注释如下：

参数	
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将扩展 jQuery 的

	命名空间。
object1	待合并到第一个对象的对象。
objectN	待合并到第一个对象的对象。

- 版本 V1.4: [deep（object）],target（object）,object1（object）,[objectN（object）]，参数注释如下：

参数	
deep	如果设为 true，则递归合并。s
target	待修改对象。
object1	待合并到第一个对象的对象。
objectN	待合并到第一个对象的对象。

这句话似乎有点问题，如果不指定 target 应该如何理解了？是说 extend 方法里不传值吗？没有参数传入何来的给 jQuery 命名空间进行扩展啊。如果对比在版本 V1.0 里对参数的解释，如果 target 是唯一的参数那么这样的用法就是扩展 jQuery 的命名空间了，这个解释倒合理些，至少在前面我们写的 jQuery 插件里使用到这个用法。后面我会把 extend 的用法一一做测试，看看这句话到底是翻译错误了？还是我的理解上出现了问题。

（3） **如果第一个参数设置为 true，则 jQuery 返回一个深层次的副本，递归地复制找到的任何对象。否则的话，副本会与原对象共享结构。**从这句话应该我们越来越明白了 extend 方法的本质了，extend 就是一个 javascript 语言里的拷贝操作，在大多数包含对象概念的语言里，因为对象的名称存储的是对象的别名换种说法就是对象的引用及该对象的地址而不是对象本身，所以当对象进行拷贝操作时候就存在**浅拷贝和深拷贝**的问题。关于浅拷贝和深拷贝我在以前的博文里做过研究，如果还有那位童鞋不太命名二者的区别，可以参看下面的文章，文章链接如下：

java 笔记：关于复杂数据存储的问题--基础篇：数组以及浅拷贝与深拷贝的问题（上）

java 笔记：关于复杂数据存储的问题--基础篇：数组以及浅拷贝与深拷贝的问题

（下）

（4） **未定义的属性将不会被复制，然而从对象的原型继承的属性将会被复制。**第一句好理解没有定义过的对象属性当然不会被复制了，因为未定义就等于没有这个属性，后半句也好理解，extend 方法在做复制操作时候会把对象原型（prototype）继承到的属性也加以复制。

为了理解$.extend 方法我逐句的分析了 jQuery 手册里的解释，仔细回味下，extend 这个可以制作 jQuery 插件的方法原来就是一个做 javascript 对象拷贝操作的函数，一个对象拷贝复制函数就是插件技术的核心，这一下子还真的让人难以接受。

鉴于此，我打算在系统讲解 extend 方法前先好好看看在 javascript 语言里浅拷贝和深拷贝方法到底如何写成的，懂了这个或许会对我们正确理解 extend 的原理很有帮助。

1.3 Javascript 里的浅拷贝和深拷贝

Javascript 的赋值都是引用传递，就是说，在把一个对象赋值给另一个变量时候，那么新变量所指向的还是赋值对象原来的地址，并没有为新对象在堆区真正的产生一个新的对象，这个就是所谓的**浅拷贝**；**深拷贝**则是把原来拷贝的对象真正的复制成一个新对象，而新的变量是指向这个新对象的地址。

下面我们就来看看 javascript 里的两种拷贝的写法：

1.3.1 浅拷贝

代码如下：

```
// 浅拷贝测试
var scopyobj = shallowcopy({},orgval);
```

```
scopyobj.obj.content = 'New Object Value';//改变scopyobj里面引用对象的值
// 我们会发现scopyobj和orgval里的obj.content的值都发生了改变
console.log('scopyobj.obj.content:' +
scopyobj.obj.content);//scopyobj.obj.content:New Object Value
console.log('orgval.obj.content:' +
orgval.obj.content);//orgval.obj.content:New Object Value
// 我们操作数组，结果是一样的
scopyobj.arrs[1].Array02 = 'I am changed';
console.log('scopyobj.arrs[1].Array02:' +
scopyobj.arrs[1].Array02);//scopyobj.arrs[1].Array02:I am
changed
console.log('orgval.arrs[1].Array02:'                          +
orgval.arrs[1].Array02);//orgval.arrs[1].Array02:I am changed
```

上面的代码比较清晰了，这里我就不做过多的讲解。

1.3.2 深拷贝

深拷贝就比较复杂了，有个编程经验的朋友都知道常常被深拷贝纠结的数据类型其实就两大类：对象和数组，我们很难控制一个函数里传入的参数的数据类型，那么一个编写良好的数据类型判断函数就显的重要多了，下面就是 javascript 一种判断数据类型的方法，代码如下：

```
var whatType = Object.prototype.toString;
console.log('whatType:' +
whatType.call({'a':12}));//whatType:[object Object]
console.log('whatType:' +
whatType.call([1,2,3]));//whatType:[object Array]
console.log('whatType:' + whatType.call(1));//whatType:[object
Number]
console.log('whatType:' +
whatType.call('123'));//whatType:[object String]
console.log('whatType:' +
whatType.call(null));//whatType:[object Null]
console.log('whatType:' +
whatType.call(undefined));//whatType:[object Undefined]
console.log('whatType:' + whatType.call(function
(){}));//whatType:[object Function]
```

```
console.log('whatType:' +
whatType.call(false));//whatType:[object Boolean]
console.log('whatType:' + whatType.call(new
Date()));//whatType:[object Date]
    console.log('whatType:'                                    +
whatType.call(/^[a-zA-Z0-9]{6,32}$/));//whatType:[object
RegExp]
```

深拷贝会将对象内部的对象一一做复制操作，因此深拷贝的操作应该需要递归算法，这里我要再介绍一个函数：arguments.callee。callee 属性是 arguments 对象的一个成员，他表示对函数对象本身的引用，这有利于匿名函数的递归或确保函数的封装性，关于arguments.callee 的使用，大家看下面的代码：

```
var i = 0;
function calleeDemo(){
    var whatType = Object.prototype.toString;
    i++;
    if (i < 6){
        arguments.callee();
        console.log(arguments.callee);
        console.log(whatType.call(arguments.callee));//[object
Function]
    }
}
    calleeDemo();//打印 5 个 calleeDemo()
```

大家看到了 arguments.callee 的类型是 Function，而内容就是 calleeDemo()。

好了，打通了技术难点我们来看看深拷贝的代码应该如何书写了，代码如下：

```
// 深拷贝测试
var dorgval = {//测试数据
    num:1,
    str:'This is String',
    obj:{'content':'This is Object'},
    arrs:['Array NO 01',{'Array02':'This is Array NO 02'}]
},
xQuery = {
    'is':function(dobj,dtype){
        var toStr = Object.prototype.toString;
        return (dtype === 'null' && dtype === 'Null' && dtype ===
```

```
'NULL') || (dtype === 'Undefined' && dtype === 'undefined' && dtype
=== 'UNDEFINED') || toStr.call(dobj).slice(8,-1) == dtype;
    },
    'deepcopy':function(des,src){
        for (var index in src){
            var copy = src[index];
            if (des === copy){
                continue;//例如window.window === window，会陷入死循
环，父子相互引用的问题
            }
            if (xQuery.is(copy,'Object')){
                des[index] = arguments.callee(des[index] ||
{},copy);
            }else if (xQuery.is(copy,'Array')){
                des[index] = arguments.callee(des[index] ||
[],copy);
            }else{
                des[index] = copy;
            }
        }
        return des;
    }
};

var dcopyobj = xQuery.deepcopy({},dorgval);
dcopyobj.obj.content = 'Deep New Object Value';//改变dcopyobj里
面引用对象的值
// 测试
console.log('dcopyobj.obj.content:' +
dcopyobj.obj.content);//dcopyobj.obj.content:Deep New Object
Value
console.log('dorgval.obj.content:' +
dorgval.obj.content);//dorgval.obj.content:This is Object
// 测试
dcopyobj.arrs[1].Array02 = 'Deep I am changed';
console.log('dcopyobj.arrs[1].Array02:' +
dcopyobj.arrs[1].Array02);//dcopyobj.arrs[1].Array02:Deep I am
changed
console.log('dorgval.arrs[1].Array02:'                          +
dorgval.arrs[1].Array02);//dorgval.arrs[1].Array02:This is Array
NO 02
```

既然我们自己写出来了 javascript 的深拷贝和浅拷贝，那么我们再去研究 jQuery 里的深

浅拷贝操作一定会事半功倍的。

1.4 $.extend 用法详述

下面我借用 jQuery 手册里的实例代码来讲解$.extend 的用法。

（1） 测试 01：参数个数为 2，并且参数类型都是 object，代码如下：

```
<script type="text/javascript">
$(document).ready(function(xj){//为$定义一个别名xj，防止$冲突
    // 测试01:参数个数为2，并且参数都是object类型
    console.log('==================测试01 start');
    var settings = {'validate':false,'limit':5, 'name':"foo"},
        opts = {'validate':true,'name':'bar'};
    console.log(xj.extend(settings,opts));//Object
{ validate=true, limit=5, name="bar"}
    console.log(settings);//Object { validate=true, limit=5,
name="bar"}
    console.log(opts);//Object { validate=true, name="bar"}
    // 上面的复制操作是浅拷贝还是深拷贝
    settings = {'validate':false,'limit':5, 'name':"foo"},
        opts = {'validate':true,'name':'bar'};
    var resobj = xj.extend(settings,opts);
    resobj.name = 'sharp';
    console.log(resobj);//Object { validate=true, limit=5,
name="sharp"}
    console.log(settings);//Object { validate=true, limit=5,
name="sharp"}
    console.log(opts);//Object { validate=true, name="bar"}
    console.log('==================测试01 end');
});
</script>
```

有上面的结果我们似乎觉得extend默认是浅拷贝，默认下extend的复制到底是浅拷贝还是深拷贝，这个需要一个使用deep标记和不使用deep标记的比较过程，后面我将做这样的测试。

下面看我第一个测试实例。

（2） 测试 02:多参数，这里我使用 4 个参数，参数类型都是 object：

```
    // 测试02:多参数，这里我使用4个参数，参数类型都是object
```

```
    console.log('=================测试02 start');
    var empty = {},
        defaults = {'validate':false,'limit':5,'name':"foo"},
        secopts = {'validate':true,'name':"bar"},
        thirdopts = {'id':'JQ001','city':'shanghai'};
    var secsets = xj.extend(empty,defaults,secopts,thirdopts);
    console.log(empty);//Object { validate=true, limit=5, name="bar",id="JQ001",city="shanghai"}
    console.log(secsets);//Object { validate=true, limit=5, name="bar",id="JQ001",city="shanghai"}
    console.log(defaults);//Object { validate=false, limit=5, name="foo"}
    console.log(secopts);//Object { validate=true, name="bar"}
    console.log(thirdopts);//Object { id="JQ001", city="shanghai"}
        console.log('=================测试 02 end');
```

这个很简单没啥好说的了。

（3） 测试 03 :浅拷贝测试,参数为 3，第一个是是否深浅拷贝的标记，后面两个是对象,

代码如下：

```
    // 测试03  浅拷贝测试,参数为3，第一个是是否深浅拷贝的标记，后面两个是对象
    console.log('=================测试03 start');
    var shallowsets = {'validate':false,'limit':5, 'name':"foo"},
        shallowopts = {'validate':true,'name':'bar'};
    console.log(xj.extend(false,shallowsets,shallowopts));//Object { validate=true, limit=5, name="bar"}
    console.log(shallowsets);//Object { validate=false, limit=5, name="foo"}
    console.log(shallowopts);//Object { validate=true, name="bar"}
    shallowsets = {'validate':false,'limit':5, 'name':"foo"},
    shallowopts = {'validate':true,'name':'bar'};
    var shallowresobj = xj.extend(false,shallowsets,shallowopts);
    shallowresobj.name = 'ok';
    console.log(shallowresobj);//Object { validate=true, limit=5, name="ok"}
    console.log(shallowsets);//Object { validate=false, limit=5, name="foo"}
    console.log(shallowopts);//Object { validate=true, name="bar"}
```

```
var deepsets = {'validate':false,'limit':5, 'name':"foo"},
    deepopts = {'validate':true,'name':'bar'};
console.log(xj.extend(true,deepsets,deepopts));//Object
{ validate=true, limit=5, name="bar"}
console.log(deepsets);//Object { validate=true, limit=5,
name="bar"}
console.log(deepopts);//Object { validate=true, name="bar"}
deepsets = {'validate':false,'limit':5, 'name':"foo"},
deepopts = {'validate':true,'name':'bar'};
var deepresobj = xj.extend(true,deepsets,deepopts);
deepresobj.name = 'okdeep';
console.log(deepresobj);//Object { validate=true, limit=5,
name="okdeep"}
console.log(deepsets);//Object { validate=true, limit=5,
name="okdeep"}
console.log(deepopts);//Object { validate=true, name="bar"}
    console.log('==================测试 03 end');
```

上面的结果让我疑惑了，当我把 deep 参数设置为 false 时候，extend 的返回值和 target 的值不一致，extend 方法的返回值是最终拷贝的结果，而 target 还是原来的值，而且我去更改返回结果的值时候，target 没有被影响。当 deep 参数为 true 的结果和我们不设定 deep 的结果一样，那么我们可以这么理解了，默认下 extend 执行的是深拷贝操作，但是这个结论我还不想过早给出，后面我会分析 extend 方法的源码，研究完了源码我再给出自己的结论。

以上的例子都是用对象做参数，数组的结果和对象一样，所以这里不再写关于数组的测试代码，下面我会使用字符串以及数字类型做测试，看看 extend 返回的结果是咋样的。

（4） 测试 04:参数个数为 2，参数类型都是字符串

代码如下：

```
// 测试04:参数个数为2，参数类型都是字符串
console.log('==================测试04 start');
var strsets = 'strsets',stropts = 'opts';
var strobj = xj.extend(strsets,stropts);
```

```
console.log(strobj);//Object { 0="o", 1="p", 2="t", 3="s"}
console.log(strsets);//strsets
console.log(stropts);//opts
strsets = 'strsets',stropts = 'opts';
strobj = xj.extend(false,strsets,stropts);
console.log(strobj);//Object { 0="o", 1="p", 2="t", 3="s"}
console.log(strsets);//strsets
console.log(stropts);//opts
strobj = xj.extend(true,strsets,stropts);
console.log(strobj);//Object { 0="o", 1="p", 2="t", 3="s"}
console.log(strsets);//strsets
console.log(stropts);//opts
console.log('==================测试 04 end');
```

拷贝的都是字符串，使用 extend 真的没啥意义了，这个反过来也说明 extend 方法只是针对引用类型的数据做拷贝操作。

（5） 测试 05：参数个数为 2，target 是字符串，第二个是 object 类型，

代码如下：

```
// 测试05: 参数个数为2，target是字符串，第二个是object类型
console.log('==================测试05 start');
var targetstr = 'sharpxiajun',
   desobj08 = {'validate':false,'limit':5, 'name':"foo"};
console.log(xj.extend(targetstr,desobj08));//Object
{ validate=false, limit=5, name="foo"}
console.log(targetstr);//sharpxiajun
targetstr = 'sharpxiajun',
desobj08 = {'validate':false,'limit':5, 'name':"foo"};
console.log(xj.extend(false,targetstr,desobj08));//Object
{ 0="s", 1="h", 2="a", 更多...}
console.log(targetstr);//sharpxiajun
targetstr = 'sharpxiajun',
desobj08 = {'validate':false,'limit':5, 'name':"foo"};
console.log(xj.extend(true,targetstr,desobj08));//Object
{ validate=false, limit=5, name="foo"}
console.log(targetstr);//sharpxiajun
   console.log('==================测试 05 end');
```

这里要注意的是当 deep 设置为 false，extend 返回的结果不同，原因我现在说不清，看来从表面使用角度还是很难分析 extend 方法的原理，一定得从源码角度进行研究了。

最后我将完整的代码贴出来，便于大家测试使用：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>jQuery Copy Study</title>
</head>
<script type="text/javascript"
src="js/jquery-1.7.1.js"></script>
<body>
</body>
</html>
<script type="text/javascript">
$(document).ready(function(xj){//为$定义一个别名xj，防止$冲突
   // 测试01:参数个数为2，并且参数都是object类型
   console.log('==================测试01 start');
   var settings = {'validate':false,'limit':5, 'name':"foo"},
      opts = {'validate':true,'name':'bar'};
   console.log(xj.extend(settings,opts));//Object
{ validate=true, limit=5, name="bar"}
   console.log(settings);//Object { validate=true, limit=5,
name="bar"}
   console.log(opts);//Object { validate=true, name="bar"}
   // 上面的复制操作是浅拷贝还是深拷贝
   settings = {'validate':false,'limit':5, 'name':"foo"},
      opts = {'validate':true,'name':'bar'};
   var resobj = xj.extend(settings,opts);
   resobj.name = 'sharp';
   console.log(resobj);//Object { validate=true, limit=5,
name="sharp"}
   console.log(settings);//Object { validate=true, limit=5,
name="sharp"}
   console.log(opts);//Object { validate=true, name="bar"}
   console.log('==================测试01 end');

   // 测试02:多参数，这里我使用4个参数，参数类型都是object
   console.log('==================测试02 start');
   var empty = {},
      defaults = {'validate':false,'limit':5,'name':"foo"},
      secopts = {'validate':true,'name':"bar"},
      thirdopts = {'id':'JQ001','city':'shanghai'};
```

```
    var secsets = xj.extend(empty,defaults,secopts,thirdopts);
    console.log(empty);//Object { validate=true, limit=5,
name="bar",id="JQ001",city="shanghai"}
    console.log(secsets);//Object { validate=true, limit=5,
name="bar",id="JQ001",city="shanghai"}
    console.log(defaults);//Object { validate=false, limit=5,
name="foo"}
    console.log(secopts);//Object { validate=true, name="bar"}
    console.log(thirdopts);//Object { id="JQ001",
city="shanghai"}
    console.log('==================测试02 end');

    // 测试03  浅拷贝测试,参数为3，第一个是是否深浅拷贝的标记，后面两
个是对象
    console.log('==================测试03 start');
    var shallowsets = {'validate':false,'limit':5, 'name':"foo"},
        shallowopts = {'validate':true,'name':'bar'};
    console.log(xj.extend(false,shallowsets,shallowopts));//Obj
ect { validate=true, limit=5, name="bar"}
    console.log(shallowsets);//Object { validate=false, limit=5,
name="foo"}
    console.log(shallowopts);//Object { validate=true,
name="bar"}
    shallowsets = {'validate':false,'limit':5, 'name':"foo"},
    shallowopts = {'validate':true,'name':'bar'};
    var shallowresobj = xj.extend(false,shallowsets,shallowopts);
    shallowresobj.name = 'ok';
    console.log(shallowresobj);//Object { validate=true, limit=5,
name="ok"}
    console.log(shallowsets);//Object { validate=false, limit=5,
name="foo"}
    console.log(shallowopts);//Object { validate=true,
name="bar"}

    var deepsets = {'validate':false,'limit':5, 'name':"foo"},
        deepopts = {'validate':true,'name':'bar'};
    console.log(xj.extend(true,deepsets,deepopts));//Object
{ validate=true, limit=5, name="bar"}
    console.log(deepsets);//Object { validate=true, limit=5,
name="bar"}
    console.log(deepopts);//Object { validate=true, name="bar"}
    deepsets = {'validate':false,'limit':5, 'name':"foo"},
    deepopts = {'validate':true,'name':'bar'};
    var deepresobj = xj.extend(true,deepsets,deepopts);
```

```
    deepresobj.name = 'okdeep';
    console.log(deepresobj);//Object { validate=true, limit=5,
name="okdeep"}
    console.log(deepsets);//Object { validate=true, limit=5,
name="okdeep"}
    console.log(deepopts);//Object { validate=true, name="bar"}
    console.log('==================测试03 end');

    // 测试04:参数个数为2，参数类型都是字符串
    console.log('==================测试04 start');
    var strsets = 'strsets',stropts = 'opts';
    var strobj = xj.extend(strsets,stropts);
    console.log(strobj);//Object { 0="o", 1="p", 2="t", 3="s"}
    console.log(strsets);//strsets
    console.log(stropts);//opts
    strsets = 'strsets',stropts = 'opts';
    strobj = xj.extend(false,strsets,stropts);
    console.log(strobj);//Object { 0="o", 1="p", 2="t", 3="s"}
    console.log(strsets);//strsets
    console.log(stropts);//opts
    strobj = xj.extend(true,strsets,stropts);
    console.log(strobj);//Object { 0="o", 1="p", 2="t", 3="s"}
    console.log(strsets);//strsets
    console.log(stropts);//opts
    console.log('==================测试04 end');

    // 测试05：参数个数为2，target是字符串，第二个是object类型
    console.log('==================测试05 start');
    var targetstr = 'sharpxiajun',
        desobj08 = {'validate':false,'limit':5, 'name':"foo"};
    console.log(xj.extend(targetstr,desobj08));//Object
{ validate=false, limit=5, name="foo"}
    console.log(targetstr);//sharpxiajun
    targetstr = 'sharpxiajun',
    desobj08 = {'validate':false,'limit':5, 'name':"foo"};
    console.log(xj.extend(false,targetstr,desobj08));//Object
{ 0="s", 1="h", 2="a", 更多...}
    console.log(targetstr);//sharpxiajun
    targetstr = 'sharpxiajun',
    desobj08 = {'validate':false,'limit':5, 'name':"foo"};
    console.log(xj.extend(true,targetstr,desobj08));//Object
{ validate=false, limit=5, name="foo"}
    console.log(targetstr);//sharpxiajun
    console.log('==================测试05 end');
```

```
});
</script>
```

1.5 复制一个$.extend 方法

学会了 jQuery 插件技术，我们完全可以把 extend 方法从源代码里抠出来，自己为 jQuery 定义一个功能和 extend 一模一样的插件，复制一个$.extend 方法就是想运用一下编写 jQuery 的技术，这种运用也是非常有意义的，因为制作 jQuery 插件让我获得了一种研究 jQuery 源代码的方式，这种方式或许是我真正理解 jQuery 源代码的金钥匙所在。

下面是我插件的代码，文件名称是：jquery.xjcopy.js，代码如下：

```
;(function($){
    $.xjcopy = $.fn.xjcopy = function(){
        var options, name, src, copy, copyIsArray, clone,
            target = arguments[0] || {},
            i = 1,
            length = arguments.length,
            deep = false;

        // 如果第一个参数是布尔值，那么这是用户在设定是否要进行深浅拷贝
        if ( typeof target === "boolean" ) {
            deep = target;
            target = arguments[1] || {};
            // 如果第一个参数设置的深浅拷贝标记，那么i设为2，下一个参数才是我们要操作的数据
            i = 2;
        }

        // 如果传入的不是对象或者是函数，可能为字符串，那么把target = {}置为空对象
        if ( typeof target !== "object" && !$.isFunction(target) ) {
            target = {};
        }

        // 如果传入的参数只有一个，跳过下面的步骤
        if ( length === i ) {
            target = this;
            --i;
```

```
        }

        for ( ; i < length; i++ ) {
            // 只操作对象值非null/undefined的数据
            if ( (options = arguments[i]) != null ) {
                for ( name in options ) {
                    src = target[ name ];
                    copy = options[ name ];

                    // 避免死循环，这个和我写的深拷贝的代码类似
                    if ( target === copy ) {
                        continue;
                    }

                    // 通过递归的方式我们把对象和数组类型的数据合并起来
                    if ( deep && copy && ( $.isPlainObject(copy) ||
(copyIsArray = $.isArray(copy)) ) ) {
                        if ( copyIsArray ) {
                            copyIsArray = false;
                            clone = src && $.isArray(src) ? src : [];

                        } else {
                            clone = src && $.isPlainObject(src) ? src :
{};
                        }

                        // 不去改变原始对象，只是对原始对象做拷贝操作
                        target[ name ] = $.xjcopy( deep, clone, copy );

                    } else if ( copy !== undefined ) {
                        target[ name ] = copy;
                    }
                }
            }
        }
        // 返回结果
        return target;
    };
    })(jQuery)
```

测试页面 xjqcopy.html 代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>用jQuery插件的方式重新定义一个jQuery.extend以及
jQuery.fn.extend函数</title>
</head>
<script type="text/javascript"
src="js/jquery-1.7.1.js"></script>
<script type="text/javascript"
src="js/jquery.xjcopy.js"></script>
<body>
</body>
</html>
<script type="text/javascript">
$(document).ready(function(){
   // 测试01:参数个数为2，并且参数都是object类型
   console.log('==================测试01 start');
   var settings = {'validate':false,'limit':5, 'name':"foo"},
      opts = {'validate':true,'name':'bar'};
   console.log($.xjcopy(settings,opts));//Object
{ validate=true, limit=5, name="bar"}
   console.log(settings);//Object { validate=true, limit=5,
name="bar"}
   console.log(opts);//Object { validate=true, name="bar"}
   // 上面的复制操作是浅拷贝还是深拷贝
   settings = {'validate':false,'limit':5, 'name':"foo"},
      opts = {'validate':true,'name':'bar'};
   var resobj = $.xjcopy(settings,opts);
   resobj.name = 'sharp';
   console.log(resobj);//Object { validate=true, limit=5,
name="sharp"}
   console.log(settings);//Object { validate=true, limit=5,
name="sharp"}
   console.log(opts);//Object { validate=true, name="bar"}
   console.log('==================测试01 end');

   // 测试02:多参数，这里我使用4个参数，参数类型都是object
   console.log('==================测试02 start');
   var empty = {},
      defaults = {'validate':false,'limit':5,'name':"foo"},
      secopts = {'validate':true,'name':"bar"},
      thirdopts = {'id':'JQ001','city':'shanghai'};
   var secsets = $.xjcopy(empty,defaults,secopts,thirdopts);
   console.log(empty);//Object { validate=true, limit=5,
```

```
name="bar",id="JQ001",city="shanghai"}
   console.log(secsets);//Object { validate=true, limit=5,
name="bar",id="JQ001",city="shanghai"}
   console.log(defaults);//Object { validate=false, limit=5,
name="foo"}
   console.log(secopts);//Object { validate=true, name="bar"}
   console.log(thirdopts);//Object { id="JQ001",
city="shanghai"}
   console.log('==================测试02 end');

   // 测试03  浅拷贝测试,参数为3，第一个是是否深浅拷贝的标记，后面两
个是对象
   console.log('==================测试03 start');
   var shallowsets = {'validate':false,'limit':5, 'name':"foo"},
      shallowopts = {'validate':true,'name':'bar'};
   console.log($.xjcopy(false,shallowsets,shallowopts));//Obje
ct { validate=true, limit=5, name="bar"}
   console.log(shallowsets);//Object { validate=false, limit=5,
name="foo"}
   console.log(shallowopts);//Object { validate=true,
name="bar"}
   shallowsets = {'validate':false,'limit':5, 'name':"foo"},
   shallowopts = {'validate':true,'name':'bar'};
   var shallowresobj = $.xjcopy(false,shallowsets,shallowopts);
   shallowresobj.name = 'ok';
   console.log(shallowresobj);//Object { validate=true, limit=5,
name="ok"}
   console.log(shallowsets);//Object { validate=false, limit=5,
name="foo"}
   console.log(shallowopts);//Object { validate=true,
name="bar"}

   var deepsets = {'validate':false,'limit':5, 'name':"foo"},
      deepopts = {'validate':true,'name':'bar'};
   console.log($.xjcopy(true,deepsets,deepopts));//Object
{ validate=true, limit=5, name="bar"}
   console.log(deepsets);//Object { validate=true, limit=5,
name="bar"}
   console.log(deepopts);//Object { validate=true, name="bar"}
   deepsets = {'validate':false,'limit':5, 'name':"foo"},
   deepopts = {'validate':true,'name':'bar'};
   var deepresobj = $.xjcopy(true,deepsets,deepopts);
   deepresobj.name = 'okdeep';
   console.log(deepresobj);//Object { validate=true, limit=5,
```

```
name="okdeep"}
    console.log(deepsets);//Object { validate=true, limit=5, 
name="okdeep"}
    console.log(deepopts);//Object { validate=true, name="bar"}
    console.log('==================测试03 end');

    // 测试04:参数个数为2，参数类型都是字符串
    console.log('==================测试04 start');
    var strsets = 'strsets',stropts = 'opts';
    var strobj = $.xjcopy(strsets,stropts);
    console.log(strobj);//Object { 0="o", 1="p", 2="t", 3="s"}
    console.log(strsets);//strsets
    console.log(stropts);//opts
    strsets = 'strsets',stropts = 'opts';
    strobj = $.xjcopy(false,strsets,stropts);
    console.log(strobj);//Object { 0="o", 1="p", 2="t", 3="s"}
    console.log(strsets);//strsets
    console.log(stropts);//opts
    strobj = $.xjcopy(true,strsets,stropts);
    console.log(strobj);//Object { 0="o", 1="p", 2="t", 3="s"}
    console.log(strsets);//strsets
    console.log(stropts);//opts
    console.log('==================测试04 end');

    // 测试05: 参数个数为2，target是字符串，第二个是object类型
    console.log('==================测试05 start');
    var targetstr = 'sharpxiajun',
        desobj08 = {'validate':false,'limit':5, 'name':"foo"};
    console.log($.xjcopy(targetstr,desobj08));//Object 
{ validate=false, limit=5, name="foo"}
    console.log(targetstr);//sharpxiajun
    targetstr = 'sharpxiajun',
    desobj08 = {'validate':false,'limit':5, 'name':"foo"};
    console.log($.xjcopy(false,targetstr,desobj08));//Object 
{ 0="s", 1="h", 2="a", 更多...}
    console.log(targetstr);//sharpxiajun
    targetstr = 'sharpxiajun',
    desobj08 = {'validate':false,'limit':5, 'name':"foo"};
    console.log($.xjcopy(true,targetstr,desobj08));//Object 
{ validate=false, limit=5, name="foo"}
    console.log(targetstr);//sharpxiajun
    console.log('==================测试05 end');
});
</script>
```

大家可以看到使用我新封装的插件，和 extend 方法执行的结果一模一样。

1.6 分析$.extend 源码

在分析源码之前，我还要加一段 s 测试代码，代码如下：

```
<script type="text/javascript">
$(document).ready(function(){
    console.log('==================测试06 start');
    var targetobj =
{'id':'NO1111','name':'xiajun','age':23,'sex':'man','comment':{'test0
1':'t01','test02':'t02','test03':'t03'}},
        srcobj =
{'id':'NO1122','name':'sharp','comment':{'test01':'tt001','test02':'t
t002','test04':'t04'}};
    var resobj = $.xjcopy(targetobj,srcobj);
    console.log(resobj);//Object { id="NO1122", name="sharp", age=23,
sex="man",comment=Object { test01="tt001", test02="tt002",
test04="t04"}}
    console.log(targetobj);//Object { id="NO1122", name="sharp", age=23,
sex="man",comment=Object { test01="tt001", test02="tt002",
test04="t04"}}
    targetobj =
{'id':'NO1111','name':'xiajun','age':23,'sex':'man','comment':{'test0
1':'t01','test02':'t02','test03':'t03'}},
    srcobj =
{'id':'NO1122','name':'sharp','comment':{'test01':'tt001','test02':'t
t002','test04':'t04'}};
    resobj = $.xjcopy(false,targetobj,srcobj);
    console.log(resobj);//Object { id="NO1122", name="sharp", age=23,
sex="man",comment=Object { test01="tt001", test02="tt002",
test04="t04"}}
   console.log(targetobj);//Object { id="NO1111", name="xiajun", age=23,
sex="man",comment=Object { test01="t01", test02="t02", test03="t03"}}
    targetobj =
{'id':'NO1111','name':'xiajun','age':23,'sex':'man','comment':{'test0
1':'t01','test02':'t02','test03':'t03'}},
    srcobj =
{'id':'NO1122','name':'sharp','comment':{'test01':'tt001','test02':'t
t002','test04':'t04'}};
    resobj = $.xjcopy(true,targetobj,srcobj);
    console.log(resobj);//Object { id="NO1122", name="sharp", age=23,
```

```
sex="man",comment=Object { test01="tt001", test02="tt002",
test03="t03",test04="t04"}}
    console.log(targetobj);//Object { id="NO1122", name="sharp", age=23,
sex="man",comment=Object { test01="tt001", test02="tt002",
test03="t03",test04="t04"}}
    console.log('==================测试06 end');
});
</script>
```

我前面写的测试实例里没有大对象里包含小对象的情况，而深浅拷贝关键场景就是对象包含对象的特殊场景，因此这里把这种场景补上。从打印出来的结果我们看到当我们不设置 deep 属性时候，非 comment 属性的值是 targetobj 和 srcobj 合并的结果，而 comment 的返回值是 srcobj 的 comment 的值，同时 extend 方法的返回值和 target 的值是相同；当我们设定了 deep 属性的值，如果 deep 值为 false 时候，我们发现 extend 的返回值和不设定 deep 属性时候值是一样的，但是 targetobj 的值是不会改变的，有个朋友问我，当 deep 属性设置为 false，我们看到 targetobj 值没变，但是如果 srcobj 的属性个数超过了 targetobj 的个数，那么 srcobj 的多余属性会不会合并到 targetobj 呢？为了这个问题我再写了一个测试，代码如下：

```
    targetobj =
{'id':'NO1111','name':'xiajun','age':23,'comment':{'test01':'t01','te
st02':'t02','test03':'t03'}},
    srcobj =
{'id':'NO1122','name':'sharp','sex':'man','comment':{'test01':'tt001'
,'test02':'tt002','test04':'t04'}};
    resobj = $.xjcopy(false,targetobj,srcobj);
    console.log(resobj);//Object { id="NO1122", name="sharp", age=23,
sex="man",comment=Object { test01="tt001", test02="tt002",
test04="t04"}}
   console.log(targetobj);//Object { id="NO1111", name="xiajun", age=23,
comment=Object { test01="t01", test02="t02", test03="t03"}}
```

我们发现 targetobj 的值的确没变；如果我们把 deep 的值设置为 true，那么从结果我们看到 targetobj 和 extend 方法的返回值都是被合并后的结果。

呵呵，对于 extend 方法的使用现在比较清晰了吧，我们看测试结果会发现：

jQuery 里的 extend 方法存在着 bug，如果我们不设定一个参数 deep 的值最终结果当然不是 deep 为 true 所代表的深拷贝，但是它和 deep 属性设为 false 时候结果也有不同，目标对象 target 一个被改变一个没有任何变化，所以不设定 deep 属性的值也不能说是 deep 设为 false 的默认操作，这个应该算是 extend 方法的 bug 吧。

下面我们读读 extend 方法的源码，看看到底是什么样的原因产生了这样的结果。

源码的第一部分是为 extend 方法内部设定一些需要使用的局部变量，代码如下：

```
    // options是用来存储拷贝对象的源对象的临时变量，name是拷贝对象的属性值
    // src 用来存储拷贝对象目标的值，copy存储被拷贝对象的目标值比如我们示例代
码
    // 里的 src = targetobj['id'],copy = targetobj['id']
    // copyIsArray是个布尔值，用来存储对象是不是数组的标记
    // clone是作为合并的的临时对象（这个大家看深拷贝的代码慢慢体会了）
    // target这个是我们的extend参数的target
    // i是extend参数objectN在arguments里的索引值
    //length是指参数个数，deep深浅拷贝的标记，大家可以看到默认下deep是false
    var options, name, src, copy, copyIsArray, clone,
        target = arguments[0] || {},
        i = 1,
        length = arguments.length,
            deep = false;
```

讲解写在注释里的这里就不在累述了，我们接着读下面的代码：

```
    // 如果第一个参数是布尔值，那么这是用户在设定是否要进行深浅拷贝
    if ( typeof target === "boolean" ) {
        deep = target;//设定deep的值，这个好理解
        target = arguments[1] || {};//如果设定deep属性，那么target要重
新赋值
        // 如果第一个参数设置的深浅拷贝标记，那么i设为2，表明arguments的
objectN参数是从索引为2的值开始
        i = 2;
        }
```

注释比较清晰，这里也不啰嗦了，**不过这段代码是有问题，是有 bug 的。这段代码也是我们不设置第一个参数 deep 值和 deep 值设为 false 最后结果不一致所在，其实代码作者的原意是 deep 不被设置时候的结果和 deep 设置为 false 是一样的。但是如果参数为 false，**

typeof 判断类型不是 boolean 而是 object，大家看下面的测试代码就明白其中道理了：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>无标题文档</title>
</head>

<body>
</body>
</html>
<script type="text/javascript">
window.onload = function(){
    //alert(typeof false);//boolean
    //alert(typeof true);//boolean
    typeftn(false,{},{});
    typeftn(true,{},{});
}

function typeftn(deep,target,src){
    var bobj = arguments[0] || {};
    //console.log(typeof bobj);//如果是false，结果是object，如果是true结果
就是boolean
    alert(typeof bobj);
}
    </script>
```

先搁置一下这个 bug，我们接着读 extend 源码：

```
        // 如果传入的不是对象或者是函数，可能为字符串，那么把target = {}置为空对
象
        // extend最终返回值是target，我们看到target在方法里都是被设置为对象，所
以不管我们传什么样的
        // 的参数到extend方法里，最终结果都是object类型，这也说明如果objectN参
数是对象，extend同样会做合并操作
        // 如果objectN非object类型那么下面深浅拷贝操作也就没意义了
        if ( typeof target !== "object" && !$.isFunction(target) ) {
            target = {};
            }
```

下面的代码就是为什么 extend 可以编写插件的原因了，代码如下：

```
        // 如果传入的参数只有一个，跳过下面的步骤
        // 当参数只有一个时候，target被设置为this，这里就是关键所在，这里的解释我
```

```
要写在文档里
        if ( length === i ) {
            target = this;
            --i;
            }
```

当我们只传一个参数并且这个参数是 object 类型，那么 target 把设置为 this，所以当我们按 jQuery.extend 方式调用 extend 方法，那么 this 会指向 jQuery 对象，代码后面写--i，那么表明当我们只传一个参数时候，这个参数不是 target 参数而是变成了 objectN 参数，成为了被拷贝对象，最终参数的内容会被拷贝到 jQuery 对象内部，最终成为 jQuery 的全局对象的一个属性。

接下来的代码就是做深浅拷贝的代码，这个代码我在前面已经写过，这里也不多讲了，代码如下：

```
        // 下面的代码就是javascript里做深浅拷贝的代码,这个和我们前面自己写的深浅
拷贝的代码类似
        for ( ; i < length; i++ ) {
            // 只操作对象值非null/undefined的数据
            if ( (options = arguments[i]) != null ) {
                for ( name in options ) {
                    src = target[ name ];
                    copy = options[ name ];

                    // 避免死循环，这个和我写的深拷贝的代码类似
                    if ( target === copy ) {
                        continue;
                    }

                    // 通过递归的方式我们把对象和数组类型的数据合并起来
                    if ( deep && copy && ( $.isPlainObject(copy) ||
(copyIsArray = $.isArray(copy)) ) ) {
                        if ( copyIsArray ) {
                            copyIsArray = false;
                            clone = src && $.isArray(src) ? src : [];

                        } else {
                            clone = src && $.isPlainObject(src) ? src : {};
                        }
```

```
                    // 不去改变原始对象，只是对原始对象做拷贝操作
                    target[ name ] = $.xjcopy( deep, clone, copy );

                } else if ( copy !== undefined ) {
                    target[ name ] = copy;
                }
            }
        }
    }
    // 返回结果
        return target;
```

下面我将我写的和 extend 方法一模一样的的 xjcopy 方法改写下，修正 extend 方法里的 bug，代码如下：

```
;(function($){
    $.xjcopy = $.fn.xjcopy = function(){
        // options是用来存储拷贝对象的源对象的临时变量，name是拷贝对象的属性值
        // src 用来存储拷贝对象目标的值，copy存储被拷贝对象的目标值比如我们示例代
码
        // 里的 src = targetobj['id'],copy = targetobj['id']
        // copyIsArray是个布尔值，用来存储对象是不是数组的标记
        // clone是作为合并的的临时对象（这个大家看深拷贝的代码慢慢体会了）
        // target这个是我们的extend参数的target
        // i是extend参数objectN在arguments里的索引值
        //length是指参数个数，deep深浅拷贝的标记，大家可以看到默认下deep是false
        var options, name, src, copy, copyIsArray, clone,
            target = arguments[0] || {},
            i = 1,
            length = arguments.length,
            deep = false;

        // 如果第一个参数是布尔值，那么这是用户在设定是否要进行深浅拷贝
        if (length >2){
            if (deep === false || deep === true){
                deep = target;//设定deep的值，这个好理解
                target = arguments[1] || {};//如果设定deep属性，那么target
要重新赋值
                // 如果第一个参数设置的深浅拷贝标记，那么i设为2，表明arguments的
objectN参数是从索引为2的值开始
                i = 2;
            }
        }
```

```
        /*if ( typeof target === "boolean") {
            deep = target;//设定deep的值，这个好理解
            target = arguments[1] || {};//如果设定deep属性，那么target要重
新赋值
            // 如果第一个参数设置的深浅拷贝标记，那么i设为2，表明arguments的
objectN参数是从索引为2的值开始
            i = 2;
        }*/

        // 如果传入的不是对象或者是函数，可能为字符串，那么把target = {}置为空对
象
        // extend最终返回值是target，我们看到target在方法里都是被设置为对象，所
以不管我们传什么样的
        // 的参数到extend方法里，最终结果都是object类型，这也说明如果objectN参
数是对象，extend同样会做合并操作
        // 如果objectN非object类型那么下面深浅拷贝操作也就没意义了
        if ( typeof target !== "object" && !$.isFunction(target) ) {
            target = {};
        }

        // 如果传入的参数只有一个，跳过下面的步骤
        // 当参数只有一个时候，target被设置为this，这里就是关键所在，这里的解释我
要写在文档里
        if ( length === i ) {
            target = this;
            --i;
        }

        // 下面的代码就是javascript里做深浅拷贝的代码，这个和我们前面自己写的深浅
拷贝的代码类似
        for ( ; i < length; i++ ) {
            // 只操作对象值非null/undefined的数据
            if ( (options = arguments[i]) != null ) {
                for ( name in options ) {
                    src = target[ name ];
                    copy = options[ name ];

                    // 避免死循环，这个和我写的深拷贝的代码类似
                    if ( target === copy ) {
                        continue;
                    }

                    // 通过递归的方式我们把对象和数组类型的数据合并起来
                    if ( deep && copy && ( $.isPlainObject(copy) ||
```

```
(copyIsArray = $.isArray(copy)) ) ) {
                        if ( copyIsArray ) {
                            copyIsArray = false;
                            clone = src && $.isArray(src) ? src : [];

                        } else {
                            clone = src && $.isPlainObject(src) ? src : {};
                        }

                        // 不去改变原始对象，只是对原始对象做拷贝操作
                        target[ name ] = $.xjcopy( deep, clone, copy );

                    } else if ( copy !== undefined ) {
                        target[ name ] = copy;
                    }
                }
            }
        }
        // 返回结果
        return target;
    };
    })(jQuery)
```

关于 extend 的内容我这里讲完了。这里我要说明下，jQuery 里 extend 方法并不代表着 jQuery 插件技术，只能说 extend 是实现 jQuery 插件技术的一种手段。jQuery 的插件技术还有很多内容，其中就包括不使用 extend 实现插件的方式，关于插件的详细内容我会在以后的博客里写道。

最后我还想说说，对 jQuery 插件技术的深入理解可能是理解 jQuery 源码的一把很重要的钥匙，等我写完了对 jQuery 插件 s 技术的介绍，我就会继续我临摹 jQuery 的系列，好好分析下 jQuery 的源码。

1.7 创建属于 jQuery 对象的插件

前面我看到 jQuery 插件的方式：通过$.extend 方式可以定义属于 jQuery 本身的全局性的插件，为此我做了下面的测试，大家先看下面这段 js 代码：

```
;(function($){
   // 创建jQuery全局作用域的插件
   $.extend({
      'wholeftn':function(){
         console.log('你要用jQuery.wholeftn()方式调用，如果
jQuery(XX).wholeftn()就会报错');
      },
      'wholeattr':'全局jQuery属性'
   });
   // 创建jQuery对象的插件
   $.fn.extend({
      'partfrn':function(){
         console.log('你要用jQuery(XX).wholeftn()方式调用，如果
jQuery.wholeftn()就会报错');
      },
      'partattr':'局部jQuery作用域'
   });
   })(jQuery)
```

测试代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>jQuery.extend和jQuery.fn.extend区别</title>
</head>
<script type="text/javascript"
src="js/jquery-1.7.1.js"></script>
<script type="text/javascript"
src="js/jquery.extenddiff.js"></script>
<body>
</body>
</html>
<script type="text/javascript">
$(document).ready(function(){

   $.wholeftn();// 你要用jQuery.wholeftn()方式调用，如果
jQuery(XX).wholeftn()就会报错
   console.log($.wholeattr);// 全局jQuery属性

   $('body').partfrn();//你要用jQuery(XX).wholeftn()方式调用，如
果jQuery.wholeftn()就会报错
```

```
    console.log($('body').partattr);// 局部jQuery作用域

    // 错误测试
    console.log($('body').wholeattr);// undefined
    console.log($.partattr);// undefined
    //$('body').wholeftn();// $("body").wholeftn is not a function
    $.partftn();// $.partftn is not a function
});
</script>
```

我们发现$.extend 是创建 jQuery 对象全局的方法和属性，这很像 java 里的静态方法和静态变量，而用$.fn.extend 创建的是 jQuery(XX)对象的方法，二者是有区别的：区别在于一个是属于全局的一个是属于对象的。我们平时经常使用的$.ajax就是全局方法而$('div').html()就是属于 jQuery 对象的方法，我们仔细瞧瞧 jQuery 手册里，jQuery 几乎所有的属性和方法其实都可以按照属于 jQuery 全局和属于 jQuery 对象进行分类，因此我有个看法了：

想理解 jQuery 框架的原理，读懂它的源代码，插件技术是一个很好的切入点，整个 jQuery 框架大致就是分为三大部分：第一部分就是如何构建 jQuery 对象，第二部分是如何创建属于 jQuery 全局的属性和方法，第三部分就是如何创建属于 jQuery 对象的属性和方法了，而第一部分是 jQuery 框架的根本，后面两部分就是 jQuery 框架的延伸了，只要理解了第一部分，包括如何构建 jQuery 全局插件和 jQuery 对象插件我们就可以真正理解 jQuery 框架设计原理了。

下面我就会好好分析下 jQuery 框架是如何构建 jQuery 对象的。

1.8 jQuery 框架里如何定义 jQuery 对象

jQuery 对象的定义使用的是 javascript 最基础的技术：对象的创建和对象的继承技术，具体点就是创建一个 javascript 对象以及使用 javascript 里的 prototype 技术，大家先看下面的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>javascript 面向对象</title>
</head>

<body>
</body>
</html>
<script type="text/javascript">
// 定义一个Person类，也可以说是对象，javascript里面对象和类是混合在一起的
// 类或者说对象的定义又是构造函数和类的定义混合在一起的
function Person(name,age,job){
    this.name = name;
    this.age = age;
    this.job = job;
    this.sayName = function(){
        console.log(this.name);
    }
}

Person.prototype.commonType = 'Human';

Person.prototype.commonFtn  = function(){
    console.log('Name:' + this.name + ';Age:' + this.age + ';Job:' + this.job);
}

Person.staticType = 'Animal';

Person.staticFtn = function(){
    console.log('All are Mammal');
}

var per1 = new Person('张三',30,'man');
var per2 = new Person('Lucy',25,'woman');

per1.sayName();// 张三
per2.sayName();// Lucy

per1.commonFtn();// Name:张三;Age:30;Job:man
```

```
per2.commonFtn();// Name:Lucy;Age:25;Job:woman

console.log(per1.commonType);// Human
console.log(per2.commonType);// Human

console.log(Person.staticType);// Animal
Person.staticFtn();//All are Mammal

console.log(per1.staticType);// undefined
console.log(Person.commonType);// undefined
//per1.staticFtn();// per1.staticFtn is not a function
//Person.commonFtn();// Person.commonFtn is not a function

(new Person('李四',40,'man')).commonFtn();// Name:李四;Age:40;Job:man

</script>
```

代码里的 Person 换成 jQuery 是不是就和我们平时使用 jQuery.ajax,jQuery('div').html()很类似了，当然还是有点区别的，比如 jQuery 对象的定义都是如下模式：

Var Person = {}

这个方式和我们上面写的 function Person(){}是一样的，为了和 jQuery 保持一致我们可以代码修改成这样：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>javascript 面向对象</title>
</head>

<body>
</body>
</html>
<script type="text/javascript">
// 定义一个Person类，也可以说是对象，javascript里面对象和类是混合在一起的
// 类或者说对象的定义又是构造函数和类的定义混合在一起的
/*function Person(name,age,job){
   this.name = name;
```

```
    this.age = age;
    this.job = job;
    this.sayName = function(){
        console.log(this.name);
    }
}*/

var Person = function(name,age,job){
    this.name = name;
    this.age = age;
    this.job = job;
    this.sayName = function(){
        console.log(this.name);
    }
}

Person.prototype.commonType = 'Human';

Person.prototype.commonFtn  = function(){
    console.log('Name:' + this.name + ';Age:' + this.age + ';Job:'
+ this.job);
}

Person.staticType = 'Animal';

Person.staticFtn = function(){
    console.log('All are Mammal');
}

var per1 = new Person('张三',30,'man');
var per2 = new Person('Lucy',25,'woman');

per1.sayName();// 张三
per2.sayName();// Lucy

per1.commonFtn();// Name:张三;Age:30;Job:man
per2.commonFtn();// Name:Lucy;Age:25;Job:woman

console.log(per1.commonType);// Human
console.log(per2.commonType);// Human

console.log(Person.staticType);// Animal
Person.staticFtn();//All are Mammal
```

```
console.log(per1.staticType);// undefined
console.log(Person.commonType);// undefined
//per1.staticFtn();// per1.staticFtn is not a function
//Person.commonFtn();// Person.commonFtn is not a function

(new Person('李四',40,'man')).commonFtn();// Name:李四;Age:40;Job:man

</script>
```

这样就更加接近 jQuery 的写法，但是对于 jQuery 对象的调用还是有些区别的，至少我们使用 jQuery 对象时候没有在前面 new 一下，而是直接 jQuery（XXX）的，消除 new 这个关键字的方式其实很简单就是使用工厂模式，大家看下面的代码：

```
function createPerson(name,age,job){
    var o = new Object();
    o.name = name;
    o.age = age;
    o.job = job;
    o.sayName = function(){
        console.log(this.name);
    }
    return o;
}

    createPerson('Lily','16','woman').sayName();// Lily
```

大家可以看到，工厂模式消除了 new 的关键字。

我简单阅读过 jQuery1.0 版本的源码，构建 jQuery 对象的方式和这种简单的工厂模式很类似，但是之后的 jQuery 版本创建对象的方式就更加的有技巧了。下面我给出一个根据改进后的 jQuery 对象定义的方式定义的 xQuery 对象：

Xquery1.0.0.js 代码如下：

```
(function(window, undefined) {
    // 将经常使用的全局对象重新定义在自己写好的封闭作用域内是提高你代码速度以及质量的一致方式
    // 因为在javascript这种语言里局部变量的效率永远都是高于全局变量的
    var document = window.document,navigator = window.navigator,location = window.location;
```

```
    var xQuery = (function(){

        // 我们在定义xQuery里面再定义一个xQuery对象，这个技巧也是把构
建jQuery的代码封装到
        // 一个独立的封闭作用域里，这个作用域和插件技术里的xQuery区别
开来，从而提升了在定义
        // xQuery时候的代码效率
        var xQuery = function(selector,context){
            return new xQuery.fn.init();
        };

        // xQuery.fn是xQuery.prototype的别名
        xQuery.fn = xQuery.prototype = {
            init:function(){
                this.length = 0;
                this.test = function(){
                    return this.length;
                }
                return this;
            },
            xquery:'1.0.0',
            length:1,
            size:function(){
                return this.length;
            }
        };

        xQuery.fn.init.prototype = xQuery.fn; // 使用xQuery的原型
对象覆盖init的原型对象

        return xQuery;

    })();

    window.xQuery = window.$ = xQuery;
    })(window);
```

测试页面的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
```

```
charset=utf-8" />
<title>xQuery 测试</title>
</head>
<script type="text/javascript"
src="js/xquery-1.0.0.js"></script>
<body>
</body>
</html>
<script type="text/javascript">
console.log($().xquery);// 1.0.0
console.log($().test());// 0
console.log($().size());// 0
    </script>
```

呵呵，上面的代码是不是有点 jQuery 的味道啊，关于代码的解释我在注释里面写的比较详细了，大家又不理解的地方可以好好读下我写的注释。

这段代码就是构建 jQuery 对象的核心代码，不过理解这段代码还是很不容易的，讲也不太好讲，但是我还是要尝试把它讲明白。

1.9 jQuery 框架里定义 jQuery 对象的原理

我先看这段代码`new xQuery.fn.init();`，这就是使用了工厂模式返回jQuery对象，因此我们使用jQuery对象之前从不会new一下。但是jQuery里的new的对象很奇怪，他不是直接new xQuery(),直接new xQuery结果会如何了？这个不用写测试代码了，我们仔细看看这个就不行，因为这种写法会产生重复调用，结果会变成一个死循环直到报出内存溢出的错误。

大家要注意，使用`new xQuery.fn.init()`代码同时我们一定要加一句话`xQuery.fn.init.prototype = xQuery.fn;`，如果没有`xQuery.fn.init.prototype = xQuery.fn;`这段代码，代码也是有问题，我们注释掉代码，再执行下代码结果如下：

```
<script type="text/javascript">
```

```
console.log($().xquery);// undefined
console.log($().test());// 0
console.log($().size());// 没有结果
    </script>
```

我们会发现，xquery和size()都没有被定义，产生上面的结果的原因就是this指针，前面我的博客里讲到过javascript里面this指针的运用，如果有对javascript里this指针的使用不太明白的可以参见我的这篇博文，博文的地址是：

javascript 笔记：深入分析 javascript 里对象的创建（中）

javascript 里指针用法的核心是 :**this 在对象的方法中 ,this 总是指向调用该方法的对象。**

New xQuery.fn.init()里的this指针是指向init方法内部的元素而非是xQuery.fn下的元素，因此外部调用xquery和size()时候会报出没有定义的错误（**这个地方大家要细细体会，这个是关键所在**），为了让this指针指向xQuery.fn里的元素我们就得把xQuery.fn.init.prototype指向xQuery.fn，这样init对象的指针就能指向，xQuery.fn里的属性和方法了，jQuery里面用这样的技巧巧妙的改变了this指针的指向。最后我还是要总结下：

jQuery这样构造对象的原因有以下个理由：

1. **jQuery对象对外应该是可以直接被使用，而不需要使用new关键字来构建，因此jQuery对象的定义使用的是工厂模式；**
2. **但是普通的工厂模式又有自己的局限，我们希望return是构造对象本身，但是直接这么返回会导致重复调用的错误，因此jQuery只得重新定义一个对象作为返回对象避免这种重复调用的错误；**
3. **但是重新定义的返回对象又要包含jQuery本身所具有的属性和方法，换句话说我们要模拟return new jQuery(),但是又不能让代码产生重复调用的错误，因此最后使用原型技术改变了this指针的指向。**

1.10 再看看$.extend 和$.fn.extend 的原理

jQuery 源码里是这样的形式定义$.extend 和$.fn.extend 方法的：

jQuery.extend = jQuery.fn.extend = function() {

.....

});

如果我们只传一个对象参数到$.extend 和$.fn.extend 方法里，那么这个对象最后会拷贝到 this 指针里，$.extend 方法里的 this 指针指向的是 jQuery 对象本身，而$.fn.extend 里的 this 指针是指向 jQuery.fn 也就是 jQuery.prototype（jQuery 的原型对象），所以$.extend 可以扩展 jQuery 的属性和方法，而$.fn.extend 可以扩展 jQuery 对象的属性和方法。

其实换种写法构建 jQuery 插件和使用$.extend 和$.fn.extend 方法等价的。

下面我改改我前面写的插件代码，代码如下：

```
;(function($){
    // 创建jQuery全局作用域的插件
    $.wholeftn = function(){
        console.log('你要用jQuery.wholeftn()方式调用，如果
jQuery(XX).wholeftn()就会报错');
    };
    $.wholeattr = '全局jQuery属性';
    // 创建jQuery对象的插件
    $.fn.partfrn = function(){
        console.log('你要用jQuery(XX).wholeftn()方式调用，如果
jQuery.wholeftn()就会报错');
    },
    $.fn.partattr = '局部jQuery作用域';
    })(jQuery)
```

测试代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>jQuery插件的另一种写法</title>
</head>
```

```
<script type="text/javascript"
src="js/jquery-1.7.1.js"></script>
<script type="text/javascript"
src="js/jquery.newplug.js"></script>
<body>
</body>
</html>
<script type="text/javascript">
$(document).ready(function(){

    $.wholeftn();// 你要用jQuery.wholeftn()方式调用，如果
jQuery(XX).wholeftn()就会报错
    console.log($.wholeattr);// 全局jQuery属性

    $('body').partfrn();//你要用jQuery(XX).wholeftn()方式调用，如
果jQuery.wholeftn()就会报错
    console.log($('body').partattr);// 局部jQuery作用域

    // 错误测试
    console.log($('body').wholeattr);// undefined
    console.log($.partattr);// undefined
    //$('body').wholeftn();// $("body").wholeftn is not a function
    $.partftn();// $.partfrn is not a function
});

</script>
```

结果和使用$.extend 和$.fn.extend 一样的，但是这样的代码实在是丑陋，没有使用 extend 方法来的优雅。

好了，关于 jQuery 插件技术的分析到此结束，后面我还会写一篇文章，想聊聊编写插件的一些规范和技巧，这篇文章应该更加实用点。

只有当你真正理解某个技术的原理时候，你才能使用好这门技术，这就是我现在对我使用 jQuery 技术的要求，我会一直朝这个方向努力下去的。