

# JavaScript：零基础打造自己的类库

写作不易，转载请注明出处，谢谢。

文章类别：Javascript基础（面向初学者）

## 前言

在之前的章节中，我们已经不依赖jQuery，单纯地用JavaScript封装了很多方法，这个时候，你一定会想，这些经常使用的方法能不能单独整理成一个js文件呢？

当然可以，封装本来就是干这个用的。放在一个单独js文件里固然不错，其实我们也可以单独整一个js类库，一方面可以锻炼一下自己封装方法的能力，另一方面，也可以将自己学到的东西做一个整理。

出于这个目的，本文将介绍如何封装一个简单的js类库。（当然，只是开一个头，熟悉一下js基础而已。实际使用的话我感觉完全没有必要，因为jQuery已经很强大了，直接使用第三方的就可以）

## 1. 总体设计

所谓的js库，其实也就是一个js文件，我思前想后，决定取个名字叫“miniQuery”，是不是山寨的味道十足呢？哈哈，请不要在意这些小细节。

大概的设计如下：

1. 扩展方法的兼容（主要写一些兼容的扩展方法，比如forEach方法等）
2. 工具包定义（就是之前封装的utils.js，我们的miniQuery需要依赖这个工具包，为了方便，就干脆写在一个文件里面了。）
3. miniQuery定义

## 2. 扩展方法的兼容

```
// ----- 基本扩展，字符串,数组等-----//
function extend_base () {

    if(!String.prototype.format ){
        String.prototype.format = function() {
            var e = arguments;
            return this.replace(/\{(\d+)\}/g,function(t, n) {
                return typeof e[n] != "undefined" ? e[n] : t
            })
        };
    }

    if (!Array.prototype.forEach && typeof Array.prototype.forEach !== "function") {
        Array.prototype.forEach = function(callback, context) {
            // 遍历数组,在每一项上调用回调函数,这里使用原生方法验证数组.
            if (Object.prototype.toString.call(this) === "[object Array]") {
                var i,len;
                //遍历该数组所有的元素
                for (i = 0, len = this.length; i < len; i++) {
                    if (typeof callback === "function" && Object.prototype.hasOwnProperty.call(this, i)) {
                        if (callback.call(context, this[i], i, this) === false) {
                            break; // or return;
                        }
                    }
                }
            }
        };
    }

    if(!String.prototype.format ){
        Array.isArray = function(obj){
            return obj.constructor.toString().indexOf('Array') !== -1;
        }
    }

    //待补充 ...
}
```

我们定义一个extend\_base方法，里面主要对js内置对象的api做了一些兼容性补充，目前还不完善，只有寥寥几个方法。当然，如果你不考虑IE678的话，那么基本上不需要这一部分了。

定义完成后立即调用。

```
extend_base();
```

## 2. 工具包整合

```
// ----- 工具包-----//
var utils = {

    center : function(dom){
        dom.style.position = 'absolute';
        dom.style.top = '50%';
        dom.style.left = '50%';
        dom.style['margin-top'] = - dom.offsetHeight / 2 + 'px';
        dom.style['margin-left'] = - dom.offsetWidth / 2 + 'px';
    },

    /** dom相关 */
    isDom : ( typeof HTMLElement === 'object' ) ?
        function(obj){
            return obj instanceof HTMLElement;
        } :
        function(obj){
            return obj && typeof obj === 'object' && obj.nodeType === 1 && typeof obj.nodeName === 'string';
        },

    /** 数组相关 */
    isArray : function(obj){
        return obj.constructor.toString().indexOf('Array') !== -1;
    }

}
```

- center：控制dom元素相对于父盒子居中
- isDom：判断是否为dom元素
- isArray：判断是否为数组

## 3. miniQuery总体设计

终于到miniQuery了，在写代码之前，先简单说一下自执行函数。

可能你在很多书上，或者下载的源码里面，经常会看到这样的代码：

```
(function() {  
  
})();
```

这样子你或许觉得很奇怪，没事，我们一起来分析。

在js中，你如果把函数看作一个数据类型，和其他语言中的 Integer，Float，String等等一样，就会理解很多事情了。当然，其实在js中，函数本身就是一个对象，不然的话就不会出现call方法了。因为只有对象才可以调用方法嘛。不过，大部分情况下，你把函数理解为数据类型就可以了。

匿名函数：

```
function() {  
  
}
```

这是一个函数，因为没有函数名，所以是一个匿名函数。你定义了它，如果接下来你不想通过函数调用的方式来执行它，那么是不是可以直接给它打一个括号来执行呢？

像这样：

```
function() {  
  
}();
```

不过，因为js语法的关系，这样子是不能执行的，你需要用一对圆括号来包一下：

```
(  
    function() {  
        alert("你好！");  
    }()  
);
```



这样就可以了，下面是另一种写法：

```
(  
    function() {  
        alert("你好！");  
    }  
)();
```

这样也可以，这种写法会更多一点。它的意思就是说，我不关心你这个函数叫什么名字，反正你在被定义的时候就要给我执行，这就是所谓的自执行函数。

好，问题来了，怎么加参数呢？

以前我们习惯于这么写：

```
function say(str){  
    alert(str);  
}  
  
say("你好！");
```

依葫芦画瓢

```
(  
    function(str){  
        alert(str);  
    }  
)("你好！");
```

OK了。

是不是一样的意思呢？

没啥区别，以前怎么做，现在还怎么做，无非就是一个函数传参的事情罢了。

我们将圆括号的位置调整一下

```
( function(str){  
    alert(str);  
})("你好！");
```

这样差不多就是最终的版本了，我记得初学js的时候，看这种代码很吃力，好像在看外星语言一样，后来看多了也就习惯了。

自执行函数就是这么一回事，没什么大不了的。

有了上面的解释，以后如果你再遇到这种写法，就 so easy 啦。

所以，不要再恐惧了，它就是这么回事，没什么大不了的，我这么后知后觉的人都能写，你也可以。我花了半年的时间才看明白，我相信你现在只需要几分钟。我的意思是，如果你之前不知道这些话。

那么，什么时候用自执行函数呢？

当你觉得某个函数只需要执行一次，而且不需要在其他地方调用的时候，就用。

你可能会问了，我干嘛要这样写啊，反正就执行一次，我直接把实现代码写在外面不就行了？

原因很简单，因为那样的话，你定义的变量就会是全局的，而一般来说我们设计的原则是尽量不要使用全局变量。

而采用这种方式，我们就形成了一个匿名函数，函数的定义又会形成闭包，所以比较安全和简洁。

你可能还会觉得疑惑，我干嘛要这些写，如果我非要给函数取一个名字，然后马上调用呢？

额，其实我个人认为这也是没有问题的，但是你得费一番心思去给函数取名字，取 a,b,c,d 这样的名字肯定是不好的。那么，我私以为，还不如干脆就用匿名函数算了，省得麻烦。

如果这部分知识你以前就不知道，那么我建议你把这篇文章多看几遍，反正就是那么回事，没什么大不了的。我当初就是走了很多弯路，也没有人教我，只有靠自己在那瞎摸索和各种百度，当然，现在想想很简单了。

我们的miniQuery的定义就放在这个自执行函数里面，这样一来，只要有人调用了这个js文件，就能调用miniQuery函数了。

当然，你直接放在外面其实也没事，因为反正就一个方法，而且这个方法本来就是要暴露出去的。

这边为了说明自执行函数，就硬加进来了。

我们把miniQuery的定义丢进去。

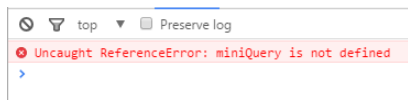
比如，像这样子的：

```
(function(){  
  
    var miniQuery = function(){  
        alert('Hello miniQuery!');  
    }  
  
})();
```

我们尝试在外面调用：

```
miniQuery();
```

很遗憾，调不到。



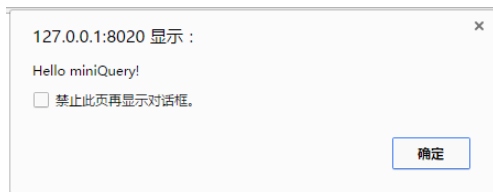
我们再回顾一下代码：

```
(function(){  
  
    var miniQuery = function(){  
        alert('Hello miniQuery!');  
    }  
  
})();  
  
miniQuery();
```

原来，miniQuery是存在于一个闭包中的，它可以访问到父级作用域的变量，但是反过来就不行，除非函数自己用 return 的方式将私有数据暴露出去。这些在之前的关于闭包的文章里面已经解释过了，这里不再赘述。

解决方法有很多，比如，最简单的，我们直接把var去掉，这样就会发生一次变量提升，miniQuery被升级为全局变量，挂在window对象上面。

```
(function(){  
  
    miniQuery = function(){  
        alert('Hello miniQuery!');  
    }  
  
})();  
  
miniQuery();
```

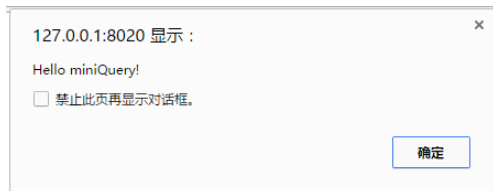


成了，简单明了，干干净净。

虽然我觉得很有道理，但是我看别人的代码，他们封装自己的js库的时候，几乎没有这样做的，因此我们也采用一种大众的做法。

即，我们把window作为参数传进去，然后手动将miniQuery挂上去。

```
(function(win){  
  
    var miniQuery = function(){  
        alert('Hello miniQuery!');  
    }  
  
    win.miniQuery = miniQuery;  
  
})(window);  
  
miniQuery();
```



是不是也可以呢？

如果你觉得每次写miniQuery太麻烦，那么我们可以给它换一个名字，比如 \$

```
(function(win){  
  
    var miniQuery = function(){  
        alert('Hello miniQuery!');  
    }  
  
    win.$ = miniQuery;  
  
})(window);
```

\$();

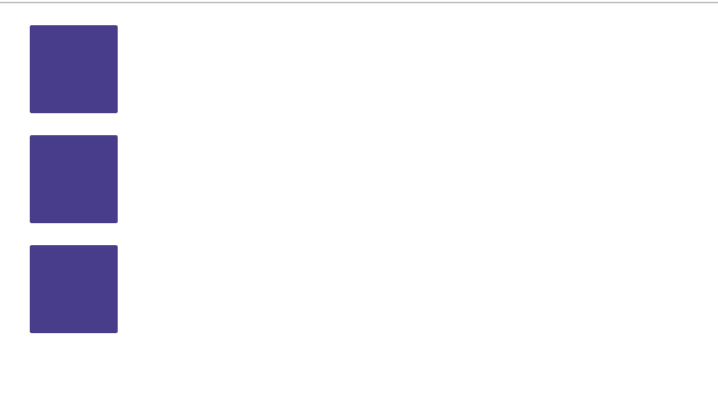
这样就差不多了。

4. miniQuery 包裹对象

我们先弄来一个测试用的网页：

```
.wrap {
  width:80px;
  height:80px;
  background:darkslateblue;
  margin:20px;
  border-radius: 2px;
}

<body>
  <div class='boxes'>
    <div id='box1' class='wrap'></div>
    <div id='box2' class='wrap'></div>
    <div id='box3' class='wrap'></div>
  </div>
</body>
```



举一个例子，现在我们要获取id为box1的盒子，并把它的背景色改为红色。

用js代码，我们会这样做：

```
var box2 = document.getElementById('box2');
box2.style.backgroundColor = 'red';
```

思路很清晰，分为简单的两步：

- 第一步：获取dom对象。
- 第二部：设置其背景色为红色。

同样的，我们的 miniQuery 也要这么做，首先得获取对象，然后进行操作。就好像你做饭，首先得有米面吧。所谓巧妇难为，无米之炊。

于是，我们有了下面的代码：

```
var miniQuery = function(selector){
  var miniQuery = document.getElementById(selector);
  console.log(miniQuery);
}
```

selector 代表选择器，它只是一个参数名字，参数列表的名称是可以自己定义的。你写 aaa ,bbb ,ccc 都没问题，只要你愿意的话。

我以前经常看别人写的代码，参数里面有callback，现在我知道是回调函数的意思。可是我以前不知道，然后就觉得很困惑，作为一个英语比日语还差的js玩家，我感到很那个啥。

其实无所谓，只是一个名字而已，你写什么都行，只要符合标识符的命名规范就成。

总有人觉得，看到参数里边写了context（上下文），callback（回调函数）这样的词汇，就觉得很困惑。

不要困惑啦，不要再惊恐啦，它就是一个名称罢了！

。。。

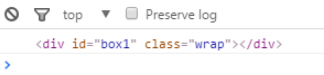
额，扯远了，继续回来。

我们在外面调用miniQuery~

window 上面挂的是 \$, 其实就是 miniQuery

```
$('box1');
```

运行结果



嗯，确实取到了呢。

接下里，我们给dom元素变更背景色为红色。

```
var miniQuery = function(selector){
  var miniQuery = document.getElementById(selector);
  miniQuery.style.backgroundColor = 'red';
}
```



效果确实出来了。

可是呢，如果用户过几天又来个需求，说我要把box1的宽度变为之前的两倍，你怎么办？

总不可能去修改源码吧！

这时候，我们就可以考虑能不能通过一个什么办法，我先用miniQuery把你传进来的东西包装成dom元素，保存起来返回给你，同时再给你返回一大堆方法，比如改变高度啊，添加背景色啊等等。那么，操作的就是之前保存的元素了。也就是你一开始希望操作的元素。

这是一个很好的想法，我们经过代码的重写，最终产生了这样的一个miniQuery函数：

```
var miniQuery = function(selector){
    var miniQuery = document.getElementById(selector);

    return {
        obj : miniQuery , //将dom元素保存起来，再返回给你

        // ----- css 相关 -----//
        backgroundColor : function(color){
            this.obj.style.backgroundColor = color;
        }
    }


}

win.$ = miniQuery;

})(window);
```

我们再调用一次，看看这回它给我们返回的是什么东东？

```
var $box = $('box1');
console.log($box);
```



可见，它给我们返回的是一个json对象，里面有 obj 变量和 backgroundColor 函数。这样的好处就是极大的扩展了我们的miniQuery，你给我一个选择器，我就包起来，然后不仅把它返回给你，而且还给你各种api方法！

于是我们就可以直接调用 backgroundColor 函数了。

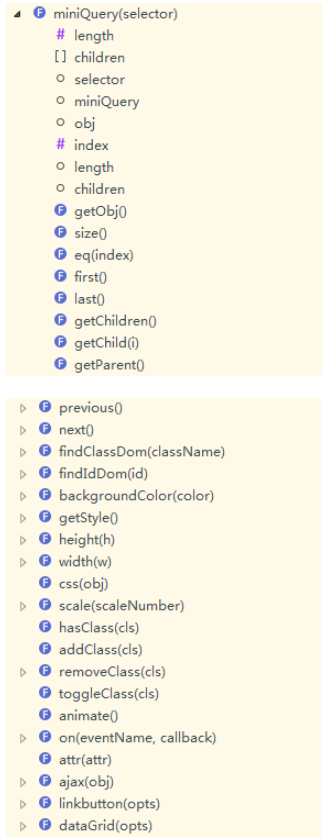
```
var $box = $('box1');
$box.backgroundColor('red');
```



成了。

我们现在返回的，不是一个单纯的dom元素，dom元素只是它的一部分。可以说，我们返回给用户的是一个miniQuery对象！

经过改进，我已经陆陆续续地给miniQuery添加了很多方法，大部分是模拟的jQuery：



顺便弄了两个小型的组件，一个是按钮，另一个是简单的数据列表。

按钮使用：

```
<link rel="stylesheet" type="text/css" href="css/mui.css"/>

<div class='box'></div>

var $box = $(''.box').eq(0);
$box.linkbutton();
```

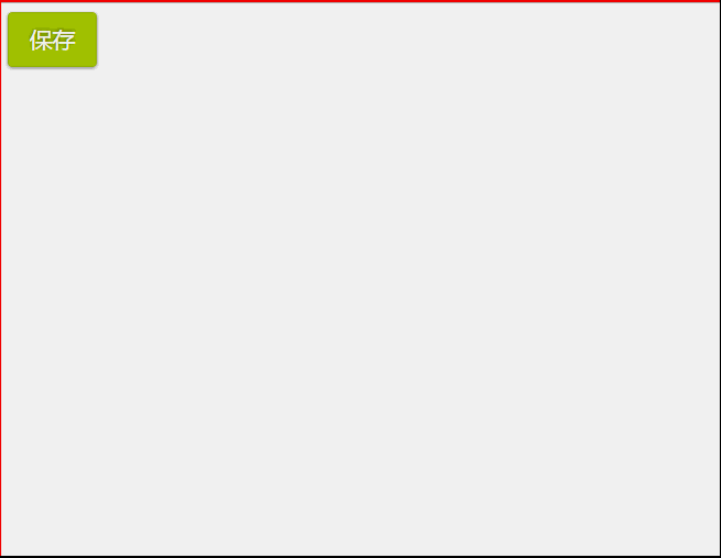


按钮的样式就出来了，然后我们来设置按钮的属性。

```
var $box = $(''.box').eq(0);
$box.linkbutton({
  text : '保存',
  click : function(){
    alert('保存成功! ');
  }
});
```



按钮的大小也自动变大了。



对应的css：

`mui.css`

```
.linkbutton {
padding: .4em .9em; /*em的好处就是随着父元素的字体大小而变化, 当该元素的字体变化时, 会自适应*/
border: 1px solid rgba(0,0,0,.1);
background-color: #ac0;
border-radius: .2em;
box-shadow: 0 1px 5px rgba(0,0,0,.5);
color: #fff;
text-shadow: 0 -.06em .24em rgba(0,0,0,.5); /*将阴影设置为半透明, 就无所谓底色了, 都能很好地适应*/
font-size: 130%;
line-height: 1.5; /*行高是字号的1.5倍*/
display:inline-block;
cursor:pointer;
font-family: "微软雅黑";
}
```

数据列表简单演示：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<link rel="stylesheet" type="text/css" href="css/mui.css"/>
<script type="text/javascript" src="js/miniQuery.js"></script>
</head>
<body>
<a id='btn0'></a>

<div id='grid0'></div>
</body>
<script>
$('#btn0').linkbutton({
text : '测试' ,
click : function(){
if(grid0.getSize() < 1){
alert('请选择一条数据! ');
return;
}
alert('您选择的是' + JSON.stringify(grid0.getSelected()));
}
});
var grid0 = mui.get('#grid0').dataGrid({
header : [
{name:'ID' , width:10 , type : 'checkColumn' } ,
{name:'标题' , type : 'column' , field : 'title' } ,
{name:'分类' , type : 'column' , field : 'type' } ,
{name:'作者' , type : 'column' , field : 'author' } ,
{name:'时间' , type : 'column' , field : 'time' } ,
] ,

});

grid0.load([
{title : '111' , type : 'A' , author : '张三' , time : '2015' } ,
{title : '222' , type : 'B' , author : '李四' , time : '2015' } ,
{title : '333' , type : 'C' , author : '王五' , time : '2015' } ,
{title : '444' , type : 'D' , author : '赵六' , time : '2015' } ,
]);

</script>
</html>
```

测试

ID	标题	分类	作者	时间
<input type="checkbox"/>	111	A	张三	2015
<input type="checkbox"/>	222	B	李四	2015
<input type="checkbox"/>	333	C	王五	2015
<input type="checkbox"/>	444	D	赵六	2015

当然，好多组件都还不够完善，我主要也是自己尝试一下，不过并不打算再拓展了。

自己做个小类库主要用于学习，以后还是用jQuery吧。

附录A

```
"use strict";

/**
 * miniQuery 和 工具类库
 * 版本 1.1 （修正了一部分Bug, 增加了一些方法）
 * 作者：剽悍一小兔
 */

// ----- 基本扩展，字符串,数组等-----//
function extend_base (){

    if(!String.prototype.format ){
        String.prototype.format = function() {
            var e = arguments;
            return this.replace(/\{(\d+)\}/g,function(t, n) {
                return typeof e[n] != "undefined" ? e[n] : t
            })
        };
    }

    if (!Array.prototype.forEach && typeof Array.prototype.forEach !== "function") {
        Array.prototype.forEach = function(callback, context) {
            // 遍历数组,在每一项上调用回调函数, 这里使用原生方法验证数组。
            if (Object.prototype.toString.call(this) === "[object Array]") {

```

```

        var i, len;
        //遍历该数组所有的元素
        for (i = 0, len = this.length; i < len; i++) {
            if (typeof callback === "function" && Object.prototype.hasOwnProperty.call(this, i)) {
                if (callback.call(context, this[i], i, this) === false) {
                    break; // or return;
                }
            }
        }
    }
};
}

if(!String.prototype.format ){
    Array.isArray = function(obj){
        return obj.constructor.toString().indexOf('Array') != -1;
    }
}

}

extend_base();

// ----- 工具包 -----//
var utils = {
    center : function(dom){
        dom.style.position = 'absolute';
        dom.style.top = '50%';
        dom.style.left = '50%';
        dom.style['margin-top'] = - dom.offsetHeight / 2 + 'px';
        dom.style['margin-left'] = - dom.offsetWidth / 2 + 'px';
    },

    /** dom相关 */
    isDom : ( typeof HTMLElement === 'object' ) ?
        function(obj){
            return obj instanceof HTMLElement;
        } :
        function(obj){
            return obj && typeof obj === 'object' && obj.nodeType === 1 && typeof obj.nodeName === 'string';
        },

    /** 数组相关 */
    isArray : function(obj){
        return obj.constructor.toString().indexOf('Array') != -1;
    }
}

// ----- miniQuery.js -----//
;(function(win){

    var miniQuery = function(selector){
        var miniQuery = null;
        var length = 0;
        var children = [];
        if(!selector) return;

        /** 1. 传入的是id */
        if(selector.toString().indexOf('#') != -1) {
            selector = selector.replace('#', '');
            miniQuery = document.getElementById(selector);
        }

        /** 2. 传入的是class */
        else if(selector.toString().indexOf('.') != -1){
            selector = selector.replace('.', '');
            miniQuery = document.getElementsByClassName(selector);
        }

        /** 3. 传入的是dom元素 */
        else if(utils.isDom(selector)){
            miniQuery = selector;
        }

        /** 4. 传入的是标签 */
        else if(typeof selector === 'string'){
            miniQuery = document.getElementsByTagName(selector);
            return miniQuery;
        }

        if(!miniQuery) return; //如果本类库包装不了, 就返回

        if(miniQuery.length){ //如果是一个类数组元素的话, 就获取他的长度
            length = miniQuery.length;
        }else{
            length = 1; //这种情况, 说明成功包裹了元素, 但是该元素还是存在的, 就将长度设定为1
        }

        children = miniQuery.children; //取得所有的孩子节点

        return {

            /** 属性区 */
            obj : miniQuery, //返回的dom元素
            index : 0, //默认的角度 (假如 miniquery 是一个类数组的话)
            length : length, //元素的个数 (假如 miniquery 是一个类数组的话)
            children : children, //所有孩子节点

            /** 方法区 */

            // ----- dom 相关 -----//

            /**获取dom对象本身, 返回纯粹的dom元素, 而非miniQuery元素*/
            getObj : function(){
                return this.obj;
            },

            /**获取元素的长度*/
            size : function(){
                return this.length;
            },
        }
    }

```



```

/** 假如 miniquery 是一个类数组的话, 用于返回其中一个元素 */
eq : function(index){
    if(length > 0) {
        return $(this.obj[index]); //eq返回的还是miniQuery对象
    }else{
        return null;
    }
} ,

/** 获得第一个匹配元素 */
first : function(){
    return $(this.obj[0]);
} ,

/** 获得最后一个匹配元素 */
last : function(){
    return $(this.obj[this.length - 1]);
} ,

/** 获得最后一个匹配元素 */
getChildren : function(){
    return this.obj.children;
} ,

/** 获得某一个孩子节点 */
getChild : function(i){
    return $(this.children[i]);
} ,

/** 获得父节点 */
getParent : function(){
    return $(this.obj.parentElement);
} ,

/** 获得上一个节点 */
previous : function(){
    var parent = this.getParent();
    var children = parent.children;
    for(var i = 0; i < children.length; i++){
        if(this.obj == children[i]){
            return $(children[i - 1]);
        }
    }
    return null;
} ,

/** 获得下一个节点 */
next : function(){
    var parent = this.getParent();
    var children = parent.children;
    for(var i = 0; i < children.length; i++){
        if(this.obj == children[i]) {
            return $(children[i + 1]);
        }
    }
    return null;
} ,

findClassDom : function(className){
    this.obj = this.obj.getElementsByClassName(className) ;
    return this ;
} ,

findIdDom : function(id){
    var $this = this;
    var children = this.getChildren();
    children = Array.prototype.slice.call(children); //obj 转 []
    children.forEach(function(item){
        //console.log(item.id);
        (id === item.id) && ($this = item) ;
    });
    return this ;
} ,

// ----- css 相关 -----//
/** 添加背景色 */
backgroundColor : function(color){
    this.obj.style.backgroundColor = color;
    return this;
} ,

/** 获取style */
getStyle : function(){
    var styleEle = null;
    if(window.getComputedStyle){
        styleEle = window.getComputedStyle(this.obj,null);
    }else{
        styleEle = ht.currentStyle;
    }
    return styleEle;
} ,

/** 设置或者拿到高度 */
height : function(h){
    if(!h) return this.getStyle().getPropertyValue('height');
    (typeof h == 'number') && (h = h + 'px');
    this.obj.style.height = h;
    return this;
} ,

/** 设置或者拿到宽度 */
width : function(w){
    if(!w) return this.getStyle().getPropertyValue('width');
    (typeof w == 'number') && (w = w + 'px');
    this.obj.style.width = w;
    return this;
} ,

/** 设置自定义样式 */
css : function(obj){
    if(!obj) return;
    for(var key in obj){
        //console.log(key + '=====' + obj[key]);
        this.obj.style[key] = typeof obj[key] == 'number' ? obj[key] + 'px' : obj[key];
    }
    return this;
} ,

/** 设置放大 倍数*/

```

```

scale : function(scaleNumber){
    this.css({
        scale : scaleNumber
    });
    return this;
} ,

hasClass : function(cls) {
    return this.obj.className.match(new RegExp('(\\s|^)' + cls + '(\\s|$)'));
} ,

addClass : function(cls){
    if (!this.hasClass(cls)) this.obj.className += " " + cls;
} ,

removeClass : function(cls) {
    if (this.hasClass(cls)) {
        //console.log(this.obj);
        var reg = new RegExp('(\\s|^)' + cls + '(\\s|$)');
        this.obj.className = this.obj.className.replace(reg, ' '); //修正bug, 之前右边少了一个this
    }
} ,

toggleClass : function(cls){
    if (this.hasClass(cls)){
        this.removeClass(cls);
    }else{
        this.addClass(cls);
    }
} ,

// ----- 动画 相关 -----//

//TODO
animate : function(){

} ,

// ----- 事件相关 -----//

on : function(eventName,callback){
    var $this = this;
    this.obj['on' + eventName] = function(){
        callback.call($this,$this.obj); //context指向$this, 参数传入dom对象
    };
    return this;
} ,

// ----- 属性相关 -----//

attr : function(attr){
    return this.obj.attributes[attr];
} ,

// ----- ajax相关 -----//

// ----- ui -----//

/** 按钮 */
linkbutton : function(opts){
    var opts = opts || {};
    /**添加基本样式*/
    this.addClass('linkbutton');
    this.on('mouseover', function(e){
        //console.log(e);
        this.css({
            backgroundColor: '#d4ef50'
        });
    }).on('mouseout',function(e){
        this.css({
            backgroundColor: '#ac0'
        });
    });

    opts.text && (this.obj.innerText = opts.text);
    opts.click && (this.on('click' , opts.click));
} ,

/** 数据列表 */
datagrid : function(opts){
    var $this = this;
    var opts = opts || {};
    var header = null; //表头
    var id = null; //grid的id, 唯一
    var tb_id = null;
    var tbody_id = null;
    var count = 0; //为了防止id重复
    var columns = []; //存放field
    var types = [];
    if(!this.obj.id) return;
    else id = this.obj.id;

    if(!opts.header) return;
    else header = opts.header;

    /**添加基本样式*/
    this.addClass('tableBox');

    //初始化表头
    function initHeader(){
        var time = new Date().getTime();
        tb_id = 'mui-table_' + time + '_' + count++;
        var html = "<table id='"+tb_id+"'><thead>" ;

        //拼接表头
        html += '<tr>' ;
        header.forEach(function(item){
            columns.push(item.field); //添加字段名
            types.push(item.type); //添加列类型
            var width = null;
            if(item.width) width = item.width + 'px'; //设置宽度
            if(width) width = "width='"+width+"' ";
            html += "<th "+width+">" + item.name + "</th>"
        });
    }

```

```

    });
    tbody_id = 'mui-table-tbody_' + time + '_' + count++;
    html += "</tr></thead><tbody id='"+tbody_id+"'></tbody>" ;
    html += '</table>' ;

    $this.obj.innerHTML = html;
}

//
initHeader();

return {
    tbody_id : tbody_id ,
    allData : null ,
    ids : [], //保存每一行的id
    index : 0, //作为行号和id
    //加载数据
    load : function(data){
        this.allData = data;
        var html = '';
        //console.log($('#' + tbody_id));
        var len = data.length; //总行数
        var columnSize = columns.length; //总列数
        //alert(len);
        for(var i = 0; i < len ; i++){
            this.ids.push('mui-dataGrid-tr_' + ( new Date().getTime() ) + '_' + this.index++ ) ;
            //console.log(this.ids[this.index - 1]);
            //console.log(this.ids[this.index - 1].substring(this.ids[this.index - 1].length - 1 )); //获取行号
            html += "<tr id='"+this.ids[this.index - 1]+"'>" ; /*之前在这里少了一个单引号，最终显示的数据只有全部的一半，现在已经更正*/
            //遍历列
            //console.log(types);
            for(var j = 0; j < columnSize ; j++){
                var columnName = columns[j];
                if(data[i][columnName]){
                    html += '<td>' + data[i][columnName] + '</td>';
                }else if(types[j] == 'checkColumn'){
                    html += '<td><input type="checkbox" value=""/></td>';
                }else {
                    html += '<td></td>';
                }
            }
            //列遍历完后，这一行才结束
            html += '</tr>'
        }

        //展示数据
        win.$('#' + this.tbody_id).obj.innerHTML = html;

        //给每一行添加事件
        this.ids.forEach(function(rowId){
            win.$('#' + rowId).on('click',function(){
                this.toggleClass('selected');
                if(this.hasClass('selected')){
                    this.obj.getElementsByTagName('input')[0].checked = true;
                }else {
                    this.obj.getElementsByTagName('input')[0].checked = false;
                }
            });
        });

    } ,

    //获取所有数据
    getData : function(){
        return this.allData;
    } ,

    //根据行号获取某一行
    getRow : function(rowIndex){
        return this.getData()[rowIndex];
    } ,

    //获取所有的行号
    getSize : function(){
        var len = 0;
        this.getSelected() && (len = this.getSelected().length) ;
        return len;
    } ,

    //返回选中的行，一条或者多条
    getSelected : function(){
        var rows = win.$('.selected').obj; //获取所有选中行
        var len = 0;
        len = rows.length;
        var arr = [];
        for(var i = 0; i < len; i++){
            //console.log(rows[i].id.substring(rows[i].id.length - 1));
            arr.push(this.getRow(rows[i].id.split('_')[2]) );
        }

        arr.length == 1 && ( arr = arr[0] );

        return arr;

        //this.ids[this.index - 1].substring(this.ids[this.index - 1].length - 1 )
    }

};

}

}

}

win.$ = miniQuery;

win.mui = {
    get : function(sel){
        return miniQuery(sel);
    }
}

```

```
    }  
  })(window);
```

## 附录B mui.css

```
.linkbutton {  
  padding: .4em .9em; /*em的好处就是随着父元素的字体大小而变化，当该元素的字体变化时，会自适应*/  
  border: 1px solid rgba(0,0,0,.1);  
  background-color: #ac0;  
  border-radius: .2em;  
  box-shadow: 0 1px 5px rgba(0,0,0,.5);  
  color: #fff;  
  text-shadow: 0 -.06em .24em rgba(0,0,0,.5); /*将阴影设置为半透明，就无所谓底色了，都能很好地适应*/  
  font-size: 130%;  
  line-height: 1.5; /*行高是字号的1.5倍*/  
  display:inline-block;  
  cursor:pointer;  
  font-family: "微软雅黑";  
}  
  
.tableBox{  
  width:1200px;  
  background:#f9f9f9;  
  margin:20px auto 0;  
  padding:6px;  
  position:relative;  
  font-family: 微软雅黑;  
}  
  
.tableBox table{  
  width:100%;  
  border:2px solid #fff;  
}  
  
.tableBox table tr {  
  border-collapse: separate;  
  border-spacing: 1px;  
}  
  
/*选中行*/  
.tableBox table tr.selected {  
  background:#cce4f3;  
}  
  
/*表头*/  
.tableBox th{  
  background: #eaeaea;  
  padding: 6px;  
  color: #666;  
  font-size: 14px;  
}  
  
.tableBox td {  
  font-size: 13px;  
  padding: 4px 10px;  
}
```

毕竟是自己DIY出来的，所以没仔细测试，肯定还有一些BUG。不过不管怎么说，都算是一次尝试吧，呵呵。