吴老师教学讲义

忽然抚尺一下，群响毕绝。撤屏视之，一人、一桌、一椅、一扇、一抚尺而已

吴　青

QQ:16910735

wuqing_bean@126.com

http://blog.sina.com/accpwulaoshi

jQuery 插件编写

jQuery 的扩展能力同它的核心一样强大。通过使用 jQuery 简洁的插件架构 ,我们能够把 jQery 的功能扩展得更加丰富.jQuery 的易扩展性，吸引了来自全球的开发者来共同编写 jQuery 的插件，最新最全的插件可以从 http://plugins.jquery.com/ 获取。

1. 插件种类

插件其实就是对现有的方法（或者叫函数）做一个封装，方便重用提高开发效率。jQeury 主要有 2 种类型：

- 实例对象方法插件

  开发能让所有的 jQuery 实例对象都可以调用的方法。也就是说，只要通过$()工厂获得的 jQuery 实例对象，都可以调用我们开发的方法。95%的插件都是这种类型

- 全局函数插件

  可以将独立的函数添加到 jQuery 命名空间中了。那么调用的时候就可以使用 $.函数名称() 或者 jQuery.函数名称（ ）来调用了。可以理解为静态方法。

2. 添加全局函数

我们可以将 jQuery 理解为类，$是这个类的别名。开发全局函数就是开发这个类的静态方法。如:$.get() ,$.post()。添加新的全局函数，实际上就是扩展 jQuery“类”本身，给 jQuery 命名空间上添砖加瓦.

加入新添加的全局函数的名称是 sayHello,功能是弹出一个 hello 的对话框.

2.1 增加一个全局函数

```
jQuery.sayHello=function(name){
    alert("你好,"+name);
}
```

调用:

```
$.sayHello('张三');

    或者

jQuery.sayHello("李四");
```

2.2 增加多个全局函数

使用 jQuery 提供的全局函数 extend 来一次加入多个函数

```
jQuery.extend({
        sayHello:function(name){
            alert("你好"+name);
        },
        sayBye:function(name){
            alert("再见"+name);
        }
    }
)
```

上面的代码是往 jQuery 命名空间中添加了两个函数 sayHello 和 syBye,调用同上

2.3 命名空间

前面的方式有可能与 jQuery 命名空间中的其它函数产生命名冲突的危险，也有可能与其它 jQuery 插件中定义的方法重名，所以我们可以考虑将我们定义的所有的函数封装到一个对象中去，相当于为我们的函数给了一个命名空间

代码如下:

```
jQuery.wq={
    sayHello:function(name){
        alert("你好"+name);
    },
    sayBye:function(name){
        alert("再见"+name);
    }
}
```

这种写法可以这样来理解: 首先我们为 jQuery 添加了一个 wq 属性，而这个属性本身是一个对象，后面我们使用 JSON 来定义了一个对象，这个对象有两个方法。所以调用就是如下形式:

调用:

```
$.wq.sayHello("王五");
jQuery.wq.sayBye("赵六");
```

注意:即使页面中包含了 jQuery 文件，也不应该认为简写形式"$"是始终有效的。因为"$" 只是一个别名而已,其它的库可以重新定义这个"$" .所以在定义插件的时候，最好使用 jQuery 来调用方法，或者重新定义"$"

3. 为 jQuery 实例对象创建新方法

对所有的 jQuery 实例对象中的方法进行扩展。

3.1 一次定义一个方法

```
jQuery.fn.sayHello=function(){
    alert("你好!");
}
```

调用:

```
$("div").sayHello();
```

jQuery.fn 是什么东西?查看源代码，我们发现 jQuery.fn=jQuery.prototype={//省略代码.....}

那 prototype 又是什么东西呢？我们可以吧 javaScript 中的类型 String，Date，Number，Boolean 当成类来对待，那么这些类（JavaScript 中本身没有类的概念，我们暂且用我们最熟悉的类来理解我们要陈述的内容。)中定义的方法或者属性是非常有限的，有时候就不够用。比如 String 中就没有类似于 java 中的 trim（ ） 去掉首尾空格的方法，那我们能不能在程序运行期间添加一个方法或者添加一个属性呢？

prototype 就是是在运行时期定义的一个属性和方法的集合，它对类的每一个对象都是有效的，而且不管这些对象是在 prototype 修改前还是修改后。

所以如果我们希望为 String 类中添加 trim()方法，我们可以使用如下代码: String.prototype.trim=function(){ … },这样每一个 String 类的实例对象都会有 trim 这个方法了。

同样道理，如果我们现在希望为所有的 jQuery 实例对象都添加一个方法，我们当然可以使用 jQuery.prototype.方法名=function(){ … }而 jQuery.prptotype=jQuery.fn ,所以我们通过 jQuery.fn 来对所所有的 jQuery 类的实例对象来扩展方法或者属性。

3.2 一次定义多个方法

前面一次定义一个方法太少，我们能不能一次多定义一些方法？jQuery 库提供了 jQuery.fn.extend 方法来一次定义多个方法

```
jQuery.fn.extend({
    sayHello:function(){
        alert("Hello");
    },
    sayBye:function(){
        alert("Bye Bye");
    }
});
```

可以看到 extend 方法的参数实际上就是一个 JSON 格式的对象。

调用的时候可以如下调用:

$(“div”).sayHello() 和 $(“div”).sayBye();

4. 插件方法内部的 this 关键字说明

上面的写法相当于每一个 jQuery 实例都可以使用，这跟全局函数没有什么区别。而我们定义的实例方法往往是需要在特定的环境中使用的。所以我们在编写插件方法的时候，应该考虑对象方法的环境。**“this”关键字在任何插件方法内部引用的都是当前的 jQuery 实例对象。**所以可以在 this 上调用任何 jQuery 方法。需要注意的是：我们使用的 jQuery 选择符可能会选取多个元素，哪么“当前的 jQuery 实例”有可能是一个元素，多个元素或者零个元素。我们必须考虑到这种情况。

如果我们使用选择器选中了多个元素，那就可以使用 each()方法来迭代每个元素，在 each 方法内部，**再使用 this，这个 this 指的就是每个 HTML DOM 元素的引用。**

5. 方法连缀

使用 jQuery 对象方法的时候，基本都能使用连缀的方式。那么我们使用插件的时候就必须为插件方法返回一个 jQuery 实例对象。

```
jQuery.fn.extend({
    sayHello:function(){
        alert("Hello");
        return this;
    },
    sayBye:function(){
        alert("Bye Bye");
        return this;
    }
});
```

6. 为插件方法定义默认值

通过使用 jQuery.extend()方法 ,可以方便提供随时可能被传入的参数覆盖的默认值，此时对方法的调用会大致保持相同.

注意:jQuery.extend 方法在 API 中的两个地方都有，一个在【核心】→【插件机制】中有，并且只有一个参数。一个在【工具】→【数组和对象操作】中。注意这里我们使用的是后者,形式为: jQuery.extend(target, object1)

它的作用是将后面对象中所有的属性和方法复制到前面对象中 ,即将 object1 中的属性和方法复制到 target 对象中。

所以我们在定义一个插件的时候，可以使用如下方式为插件方法指定默认值：

```
jQuery.fn.sayHello=function(properties){
    var defaults={
        name:"张三",
        age:20
    };
    jQuery.extend(defaults,properties);

    alert("第一个参数:"+defaults.name+"  第二个参数"+defaults.age);
    return this;
}
```

这样一来，我们就可以这样来调用了：

```
                $("div").sayHello({
                    name:"李四",
                    age:30
                });
或者:
                $("div").sayHello({
                    name:"王五"
                });
或者:
                $("div").sayHello({
                    age:25
                });
```

7. 插件开发技巧-闭包

我们在开发插件的过程中，是将代码写在了一个 js 文件中，那么这个 js 文件中有可能会定义很多的方法或者一些变量。那么这些方法或者变量就有可能与别的 js 文件中的变量或者方法冲突，那么如何将我们定义的 js 代码“暴露”一部分，隐藏一部分呢？也就是你虽然定义了，但是在其它地方访问不到，该暴露的暴露，不该暴露的就藏起来，要达到这个目的，就得用到“闭包 “

那何为“闭包 “？说简单点就是允许使用内部函数，即在函数中定义另外一个函数，而且内部函数可以访问在外部函数中声明的变量和其它内部函数。比如有如下定义:

```
//定义A函数
function A(){
    //定义B函数
    var B=function(){
        alert("这是B");
    }
    return B; //将B函数返回
}
var c=A();//此时C就是B函数
c();//其实就是调用 B 函数
```

可以看到，内部函数 B 在包含它的 A 函数之外执行了，这就形成了闭包。也就是 B 在外部函数 A 返回之后被执行了，而当 B 执行的时候它仍然可以访问 A 中定义的局部变量和其它内部函数。

利用闭包的特性，我们可以将我们需要暴露的方法暴露出来，比如 B，又可以隐藏一些局部变量和函数，比如在 A 中定义变量和函数，A 以外的其它函数是不能访问的，但是 B 是可以访问的。

其实上面的代码就是先执行 A 函数，得到结果，这个结果又是一个函数，然后再执行 B 函数即：

var c=A();

c();

既然我们的目的是想让 B 在 A 之外执行，那我们可以将代码做如下变通:

```
var C;
function A(){
    var B=function(){
        alert("这是B函数");
    }
    C=B;//将内部函数赋值给C
}
//让A函数执行，A执行之后就将B赋值给C了
A();
//现在执行C，实际上就是B的执行
C(); //弹出对话框 "这是 B 函数"
```

这与上面的效果是一样的。

能不能让 A 在定义之后马上就执行呢？我们可以定义一个匿名函数 放到一对括号中,然后再用一对小括号执行它即可:

```
var C;

(function(){
    var B=function(){
        alert("这是B函数");
    }
    C=B;//将内部函数赋值给C
})()

//现在执行C，实际上就是B的执行
C(); //弹出对话框 "这是 B 函数"
```

解释:

黄色背景的部分是匿名函数，放到了一对小括号中了（红色的小括号），后面的一对小括号（蓝色）表示要调用哪个匿名函数。网页加载这段代码的时候，函数就被调用，函数 B 就从内部跑到外部来了，然后就可以在外部调用了

能不能从外面传个参数进去交给函数 B 呢？改写代码如下:

```
(function($){
    //这里就可以使用$符号了.....

    //将B函数添加到JQuery的实例对象函数中
    $.fn.B=function(){
        alert("这是B函数");
    }
})(jQuery);
```

这段代码被浏览器加载就会被执行，jQuery 作为参数传递进去交给了 $符号，所以内部就能使用$符号了，在代码中，我们使用$.fn 为 jQuery 的实例对象添加了一个方法 B，页面上使用:

```
<script>
    $(document).ready(function(){
        $("h1").B();
    });
</script>
```

执行结果

所以常见的 jQuery 插件都是以下这种形式:

好处就在于代码内部定义的方法外部只有插件能够访问，这样就将一些代码隐藏起来了，该暴露的插件方法暴露在外部了。

```
(function($){

      //插件代码

})(jQuery)
```