

本系列文章导航

[从零开始学习 jQuery \(一\) 开天辟地入门篇](#)

[从零开始学习 jQuery \(二\) 万能的选择器](#)

[从零开始学习 jQuery \(三\) 管理 jQuery 包装集](#)

[从零开始学习 jQuery \(四\) 使用 jQuery 操作元素的属性与样式](#)

[从零开始学习 jQuery \(五\) 事件与事件对象](#)

[从零开始学习 jQuery \(六\) jQuery 中的 Ajax](#)

[从零开始学习 jQuery \(七\) jQuery 动画-让页面动起来!](#)

[从零开始学习 jQuery \(八\) 插播:jQuery 实施方案](#)

[从零开始学习 jQuery \(九\) jQuery 工具函数](#)

[从零开始学习 jQuery \(十\) jQueryUI 常用功能实战](#)

[从零开始学习 jQuery \(十一\) 实战表单验证与自动完成提示插件](#)

jQuery (一) 开天辟地入门篇

一.摘要

本系列文章将带您进入 jQuery 的精彩世界, 其中有很多作者具体的使用经验和解决方案, 即使你会使用 jQuery 也能在阅读中发现些许秘籍.

本篇文章是入门第一篇, 主要是简单介绍 jQuery, 通过简单示例指导大家如何编写 jQuery 代码以及搭建开发环境. 详细讲解了如何在 Visual Studio 中配合使用 jQuery.

转载请注明子秋出品!博客园首发!

二.前言

首 先道个歉! "从零开始学习 ASP.NET MVC"系列文章在即将介绍 Filter 时就没有更新了, 原因就是最近我一直在研究和学习 jQuery.看到本系列的名称和文章标题, 看过我的 MVC 系列文章的人会感到很熟悉. 不久要给公司的人做培训, 所以特意制作了本教程.

在写作的同时我参考了网上 jQuery 的系列教程文章, 在博客园和 Google 上并没有找到让我满意

的系列教程. 我喜欢将知识系统的,深入浅出的讲解.不喜欢写那种"学习笔记"式的文章. 同时本系列将很快全部写完(有工作压力就是有动力), 随后如果时间允许我会继续更新 MVC 系列文章.再一次对等待 MVC 文章的朋友们说声抱歉!

另外本系列文章的大部分知识点来源于图灵出版社的"jQuery 实战"一书. 推荐大家购买此书, 是 jQuery 书籍中的经典之作.

下面让我们开始 jQuery 之旅.

三.什么是 jQuery

jQuery 是一套 Javascript 脚本库. 在我的博客中可以找到"Javascript 轻量级脚本库"系列文章. Javascript 脚本库类似于.NET 的类库, 我们将一些工具方法或对象方法封装在类库中, 方便用户使用.

注意 jQuery 是脚本库, 而不是脚本框架. "库"不等于"框架", 比如"System 程序集"是类库, 而 "ASP.NET MVC"是框架. jQuery 并不能帮助我们解决脚本的引用管理和功能管理,这些都是脚本框架要做的事.

脚本库能够帮助我们完成编码逻辑,实现业务功能. 使用 jQuery 将极大的提高编写 javascript 代码的效率, 让写出来的代码更加优雅, 更加健壮. 同时网络上丰富的 jQuery 插件也让我们的工作变成了"有了 jQuery,天天喝茶水"--因为我们已经站在巨人的肩膀上了.

创建一个 ASP.NET MVC 项目时, 会发现已经自动引入了 jQuery 类库. jQuery 几乎是微软的御用脚本库了!完美的集成度和智能感知的支持,让.NET 和 jQuery 天衣无缝结合在一起!所以用.NET 就要选用 jQuery 而非 Dojo,ExtJS 等.

jQuery 有如下特点:

1.提供了强大的功能函数

使用这些功能函数, 能够帮助我们快速完成各种功能, 而且会让我们的代码异常简洁.

2.解决浏览器兼容性问题

javascript 脚本在不同浏览器的兼容性一直是 Web 开发人员的噩梦, 常常一个页面在 IE7,Firefox 下运行正常, 在 IE6下就出现莫名其妙的问题. 针对不同的浏览器编写不同的脚本是一件痛苦的事情. 有了 jQuery 我们将从这个噩梦中醒来, 比如在 jQuery 中的 Event 事件对象已经被格式化成为所有浏览器通用的, 从前要根据 event 获取事件触发者, 在 ie 下是 event.srcElements 而 ff等标准浏览器下下是 event.target. jQuery 则通过统一 event 对象,让我们可以在所有浏览器中使用 event.target 获取事件对象.

3.实现丰富的 UI

jQuery 可以实现比如渐变弹出, 图层移动等动画效果, 让我们获得更好的用户体验. 单以渐变效果为例, 从前我自己写了一个可以兼容 ie 和 ff的渐变动画, 使用大量 javascript 代码实现, 费心费力不说, 写完后没有太多帮助过一段时间就忘记了. 再开发类似的功能还要再次费心费力. 如今使用 jQuery 就可以帮助我们快速完成此类应用.

4.纠正错误的脚本知识

这一条是我提出的, 原因就是大部分开发人员对于 javascript 存在错误的认识. 比如在页面中编写加载时即执行的操作 DOM 的语句, 在 HTML 元素或者 document 对象上直接添加"onclick"属性, 不知道 onclick 其实是一个匿名函数等等. 拥有这些错误脚本知识的技术人员也能完成所

有的开发工作，但是这样的程序是不健壮的。比如"在页面中编写加载时即执行的操作 DOM 的语句"，当页面代码很小用户加载很快时没有问题，当页面加载稍慢时就会出现浏览器"终止操作"的错误.jQuery 提供了很多简便的方法帮助我们解决这些问题，一旦使用 jQuery 你就将纠正这些错误的知识--因为我们都是用标准的正确的 jQuery 脚本编写方法!

5.太多了! 等待我们一一去发现.

四.Hello World jQuery

按照惯例，我们来编写 jQuery 的 Hello World 程序，来迈出使用 jQuery 的第一步.

在本文最后可以下本章的完整源代码.

1.下载 jQuery 类库

jQuery 的项目下载放在了 Google Code 上，下载地址:

<http://code.google.com/p/jqueryjs/downloads/list>

上面的地址是总下载列表，里面有很多版本和类型的 jQuery 库，主要分为如下几类:

min: 压缩后的 jQuery 类库， 在正式环境上使用.如:[jquery-1.3.2.min.js](#)

vsdoc: 在 Visual Studio 中需要引入此版本的 jquery 类库才能启用智能感知.如: [jquery-1.3.2-vsdoc2.js](#)

release 包: 里面有没有压缩的 jquery 代码，以及文档和示例程序. 如:[jquery-1.3.2-release.zip](#)

2.编写程序

创建一个 HTML 页面，引入 jQuery 类库并且编写如下代码:

```
<!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>
```

```
<title>Hello World jQuery!</title>
```

```
<script type="text/javascript" src="scripts/jquery-1.3.2-
vsdoc2.js"></script>
```

```
</head>
```

```
<body>
```

```
    <div id="divMsg">Hello World!</div>
```

```
    <input id="btnShow" type="button" value="显示" />
```

```
    <input id="btnHide" type="button" value="隐藏" /><br />
```

```
    <input id="btnChange" type="button" value="修改内容为 Hello World, too!" />
```

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

```
        $("#btnShow").bind("click", function(event) { $("#divMsg").show();
    });
```

```
        $("#btnHide").bind("click", function(event) { $("#divMsg").hide();
    });
```

```
        $("#btnChange").bind("click", function(event)
    { $("#divMsg").html("Hello World, too!"); });
```

```
    </script>
```

```
</body>
```

```
</html>
```

效果如下:



页面上有三个按钮, 分别用来控制 Hello World 的显示,隐藏和修改其内容.

此示例使用了:

- (1)jQuery 的 Id 选择器: \$("#btnShow")
- (2) 事件绑定函数 bind()
- (3) 显示和隐藏函数. show()和 hide()
- (4) 修改元素内部 html 的函数 html()

在接下来的教程中我们将深入这些内容的学习.

五.启用 Visual Studio 对 jQuery 的智能感知

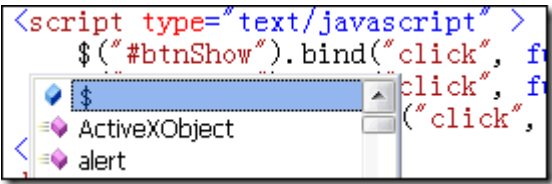
首先看一下 Visual Studio 带给我们的智能感知惊喜. 要让 Visual Studio 支持智能感知, 需要下列条件:

- 安装 VS2008 SP1
下载地址: <http://msdn.microsoft.com/en-us/vstudio/cc533448.aspx>
- 安装 VS 2008 Patch KB958502以支持"-vsdoc.js"Intellisense 文件.
该 补丁会导致 Visual Studio 在一个 JavaScript 库被引用时, 查找是否存在一个可选的"-vsdoc.js"文件, 如果存在的话, 就用它来驱动 JavaScript intellisense 引擎. 这些加了注释的"-vsdoc.js"文件可以包含对 JavaScript 方法提供了帮助文档的 XML 注释, 以及对无法自 动推断出的动态 JavaScript 签名的另外的代码 intellisense 提示. 你可以在"[这里](#)"了解该补丁的详情. 你可以在"[这里](#)"免费下载该补丁.

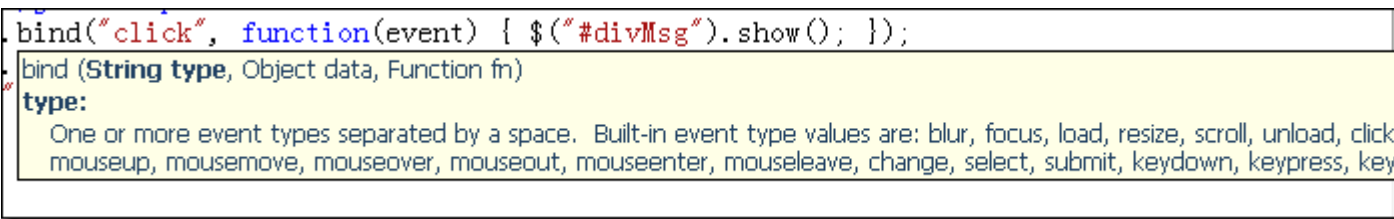
必须要引用 vsdoc 版本的 jquery 库

```
<script type="text/javascript" src="scripts/jquery-1.3.2-vsdoc2.js"></script>
```

在编写脚本的时候, 甚至刚刚输入"\$"的时候,VS 可以智能提示:



在使用方法时, 还会有更多的提示:



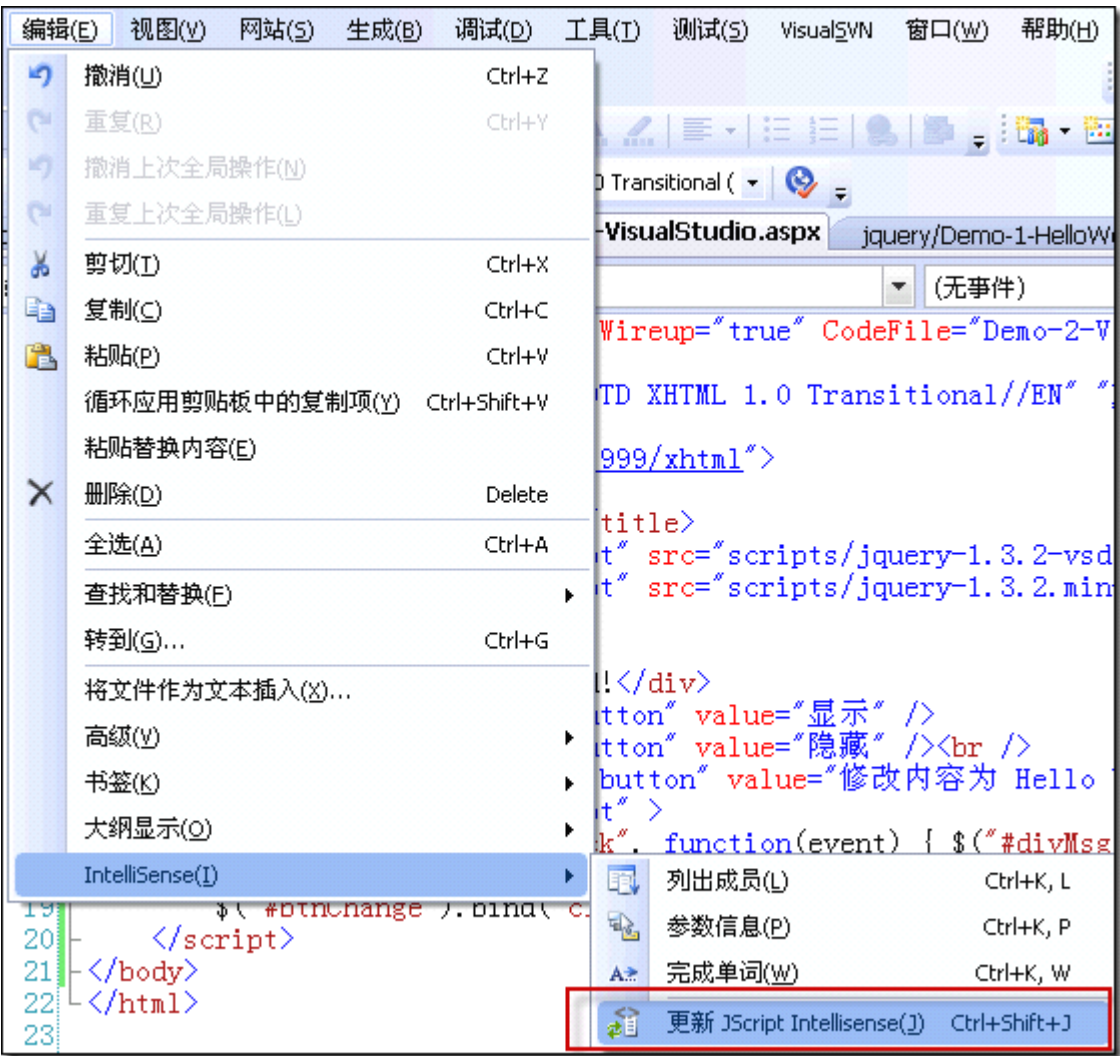
有了智能感知我们编写 javascript 变得和 C#一样快速,便捷,舒服.大部分情况可以一次编写成功而不用再为了一个大小写而查询 javascript 帮助文件.能够让 Visual Studio 对 jQuery 实现智能感

知的前提是要引入 vsdoc 版本的 jQuery 类库. 示例中我们引入了"jquery-1.3.2-vsdoc2.js"文件. 如果引用其他版本比如 min 版本的 jQuery 类库就无法启用智能提示.但是在正式环境下, 我们必须使用"min"版本的 jquery 库文件, 以1.3.2版本号为例,各个版本的大小如下:

jquery-1.3.2.js	118 KB
jquery-1.3.2.min.js	58 KB
jquery-1.3.2-vsdoc2.js	189 KB

其中第一个是未压缩的 jquery 库. 如果启用 gzip 压缩并且使用 min 版本的 jquery.js 可以在传输过程中压缩到19KB.

注意,如果我们更新了脚本, 可以通过"Ctrl+Shift+J"快捷方式更新 Visual Studio 的智能感知,或者单击 编辑->IntelliSense->更新 JScript Intellisense:



为了即能在 Visual Studio 中增加脚本提示, 又能在上线的时候使用 min 版本的脚本库, 我们一般是用如下方式引入 jQuery 库:

1. 控制编译结果

```
<script type="text/javascript" src="scripts/jquery-1.2.6.min.js"></script>

<%if (false)

{ %>
```

```
<script type="text/javascript" src="scripts/jquery-1.3.2-  
vsdoc2.js"></script>
```

```
<%} %>
```

这是网上推荐的方式. 编译后的页面上只有 min 版本的引用, 同时在开发时能够享受到智能感知. 但是注意这种方式引用的 **min** 类库只能是**1.2.6**或者之前的版本号. 最新的**1.3.2**的所有非 **vsdoc** 版本的 **jquery** 库引用后都会导致 **JScript Intellisense** 更新出错. 这是1.3.2版本的一个 bug, 期待后续版本中解决. 其实大家完全可以使用1.2.6版本的 min 库, 本教程涉及的 jquery 功能, 1.2.6版本基本都支持.

我们使用了 if(false)让编译后的页面不包含 vsdoc 版本 jquery 库的引用, 同样的思路还可以使用比如将脚本引用放入一个 Placeholder 并设置 visible=false 等.

2. 使用后端变量

为了能使用 1.3.2 版本的 min 库, 我们只能通过将脚本引用放在变量里, 通过页面输出的方式, 此种方式可以正常更新 JScript Intellisense. 但是可能有人和我一样不喜欢在前端使用变量:

```
<asp:Placeholder Visible="false" runat="server">  
  
    <script type="text/javascript" src="scripts/jquery-1.3.2-  
vsdoc2.js"></script>  
  
</asp:Placeholder>  
  
<% =jQueryScriptBlock %>
```

后台声明变量:

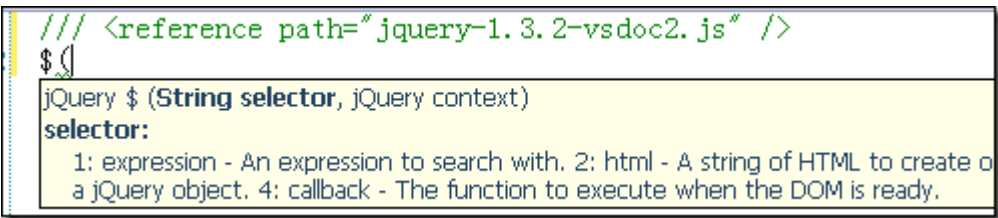
```
protected string jQueryScriptBlock = @"<script type=""text/javascript""  
src=""scripts/jquery-1.3.2.min.js""></script>";
```

六.在独立的.JS 文件中启用脚本智能感知

上面我们解决了在页面中智能感知的问题，其实在独立的.js 文件中我们同样可以启用脚本的智能感知，在 IntellisenseDemo.js 文件中,添加如下语句:

```
/// <reference path="jquery-1.3.2-vsdoc2.js" />
```

更新 JScript Intellisense, 会发现在脚本中也启用了智能提示:



注意,本文中讲解的脚本智能感知不仅适用于 **jQuery** 类库, 还适用于自己编写的 **javascript** 代码。

七.总结

本文简单介绍了 jQuery, 以及如何搭建脚本开发环境. 示例程序没有复杂的功能, 可能还无法让没有接触过 jQuery 的人认识到它的强大.但是仅凭借"多浏览器支持"这一特性, 就足以让我们学习并使用 jQuery, 因为如今想编写跨浏览器的脚本真的是一件困难的事情!

在后续文章中我们将深入学习 jQuery 选择器, 事件, 工具函数, 动画, 以及插件等.

本文代码下载:

<http://files.cnblogs.com/zhangziqu/Code-jQueryStudy-1.rar>

Tag 标签: [jQuery](#),[jQuery 教程](#)

jQuery (二) 万能的选择器

一.摘要

本章讲解 jQuery 最重要的选择器部分的知识. 有了 jQuery 的选择器我们几乎可以获取页面上任意的一个或一组对象, 可以明显减轻开发人员的工作量.

二.前言

编写任何 javascript 程序我们要首先获得对象, jQuery 选择器能彻底改变我们平时获取对象的方式, 可以获取几乎任何语意的对象, 比如"拥有 title 属性并且值中包含 test 的<a>元素", 完成这些工作只需要编写一个 jQuery 选择器字符串. 学习 jQuery 选择器是学习 jQuery 最重要的一步.

三.Dom 对象和 jQuery 包装集

无论是在写程序还是看 API 文档, 我们要时刻注意区分 Dom 对象和 jQuery 包装集.

1.Dom 对象

在传统的 javascript 开发中,我们都是首先获取 Dom 对象,比如:

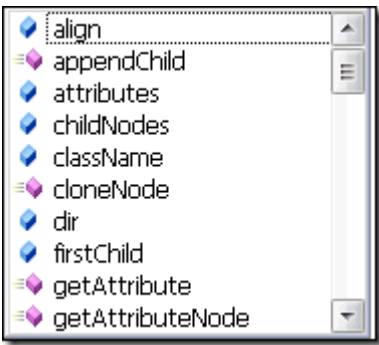
```
var div = document.getElementById("testDiv");
```

```
var divs = document.getElementsByTagName("div");
```

我们经常使用 document.getElementById 方法根据 id 获取单个 Dom 对象, 或者使用 document.getElementsByTagName 方法根据 HTML 标签名称获取 Dom 对象集合.

另外在事件函数中, 可以通过在方法函数中使用 this 引用事件触发对象(但是在多播事件函数中 IE6 存在问题), 或者使用 event 对象的 target(FF)或 srcElement(iIE6)获取到引发事件的 Dom 对象.

注意我们这里获取到的都是 Dom 对象, Dom 对象也有不同的类型比如 input, div, span 等. Dom 对象只有有限的属性和方法:

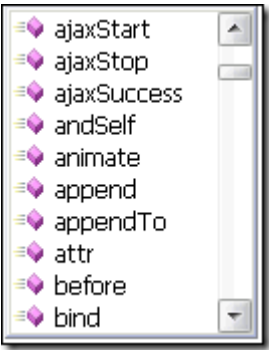


2.jQuery 包装集

jQuery 包装集可以说是 Dom 对象的扩充.在 jQuery 的世界中将所有的对象，无论是一个还是一组，都封装成一个 jQuery 包装集,比如获取包含一个元素的 jQuery 包装集:

```
var jQueryObject = $("#testDiv");
```

jQuery 包装集都是作为一个对象一起调用的. jQuery 包装集拥有丰富的属性和方法，这些都是 jQuery 特有的:



3.Dom 对象与 jQuery 对象的转换

(1) Dom 转 jQuery 包装集

如果要使用 jQuery 提供的函数，就要首先构造 jQuery 包装集. 我们可以使用本文即将介绍的 jQuery 选择器直接构造 jQuery 包装集,比如:

```
$("#testDiv");
```

上面语句构造的包装集只含有一个 id 是 testDiv 的元素.

或者我们已经获取了一个 Dom 元素,比如:

```
var div = document.getElementById("testDiv");
```

上面的代码中 div 是一个 Dom 元素，我们可以将 Dom 元素转换成 jQuery 包装集:

```
var domTojQueryObject = $(div);
```

小窍门:因为有了智能感知，所以我们可以通过智能感知的方法列表来判断一个对象啊是 **Dom** 对象还是 **jQuery** 包装集.

(2) jQuery 包装集转 Dom 对象

jQuery 包装集是一个集合，所以我们可以通过索引器访问其中的某一个元素:

```
var domObject = $("#testDiv")[0];
```

注意, 通过索引器返回的不再是 **jQuery** 包装集, 而是一个 **Dom** 对象!

jQuery 包装集的某些遍历方法,比如 each()中, 可以传递遍历函数, 在遍历函数中的 this 也是 Dom 元素,比如:

```
$("#testDiv").each(function() { alert(this) })
```

如果我们要使用 jQuery 的方法操作 Dom 对象,怎么办? 用上面介绍过的转换方法即可:

```
$("#testDiv").each(function() { $(this).html("修改内容") })
```

小结: 先让大家明确 Dom 对象和 jQuery 包装集的概念, 将极大的加快我们的学习速度. 我在学习 jQuery 的过程中就花了很长时间没有领悟到两者的具体差异, 因为书上并没有专门讲解两者的区别, 所以经常被"this 指针为何不能调用 jQuery 方法"等问题迷惑. 直到某一天豁然开朗, 发现只要能够区分这两者, 就能够在写程序时变得清清楚楚.

四. 什么是 jQuery 选择器

在 Dom 编程中我们只能使用有限的函数根据 id 或者 TagName 获取 Dom 对象.

在 jQuery 中则完全不同,jQuery 提供了异常强大的选择器用来帮助我们获取页面上的对象, 并且将对象以 **jQuery 包装集**的形式返回.

首先来看看什么是选择器:

```
//根据 ID 获取 jQuery 包装集
```

```
var jQueryObject = $("#testDiv");
```

上例中使用了 ID 选择器, 选取 id 为 testDiv 的 Dom 对象并将它放入 jQuery 包装集, 最后以 **jQuery 包装集**的形式返回.

"**\$**"符号在 jQuery 中代表对 jQuery 对象的引用,"jQuery"是核心对象, 其中包含下列方法:

jQuery(expression, context)

Returns: jQuery

这个函数接收一个 *CSS* 选择器的字符串, 然后用这个字符串去匹配一组元素。

This function accepts a string containing a CSS selector which is then used to match a set of elements.

jQuery(html, ownerDocument)

Returns: jQuery

根据 *HTML* 原始字符串动态创建 *Dom* 元素.

Create DOM elements on-the-fly from the provided String of raw HTML.

jQuery(elements)

Returns: jQuery

将一个或多个 *Dom* 对象封装 *jQuery* 函数功能(即封装为 *jQuery* 包装集)

Wrap jQuery functionality around a single or multiple DOM Element(s).

jQuery(callback)

Returns: jQuery

*\$(document).ready()*的简写方式

A shorthand for \$(document).[ready\(\)](#).

上面摘选自 jQuery 官方手册>Returns 的类型为 jQuery 即表示返回的是 jQuery 包装集.其中第一个方法有些问题,官方接口写的是 CSS 选择器,但是实际上这个方法不仅仅支持 CSS 选择器,而是所有 jQuery 支持的选择器,有些甚至是 jQuery 自定义的选择器(在 CSS 标准中不存在的选择器).为了能让大家理解的更清楚, 我将方法修改如下:

jQuery(selector, context)

Returns: jQuery 包装集

根据选择器选取匹配的对象, 以 *jQuery* 包装集的形式返回. *context* 可以是 *Dom* 对象集合或 *jQuery* 包装集, 传入则表示要从 *context* 中选择匹配的对象, 不传入则表示范围为文档对象(即页面全部对象).

上面这个方法就是我们选择器使用的核心方法.可以用"\$"代替 jQuery 让语法更简介, 比如下面两句话的效果相同:

```
//根据 ID 获取 jQuery 包装集
```

```
var jQueryObject = $("#testDiv");
```

```
//$是 jQuery 对象的引用:
```

```
var jQueryObject = jQuery("#testDiv");
```

接下来让我们系统的学习 jQuery 选择器.

五.jQuery 选择器全解

通俗的讲, Selector 选择器就是"一个表示特殊语意的字符串". 只要把选择器字符串传入上面的方法中就能够选择不同的 Dom 对象并且以 jQuery 包装集的形式返回.

但是如何将 jQuery 选择器分类让我犯难. 因为书上的分类和 jQuery 官方的分类截然不同. 最后我决定以实用为主, 暂时不去了解 CSS3选择器标准, 而按照 jQuery 官方的分类进行讲解.

jQuery 的选择器支持 CSS3选择器标准. 下面是 W3C 最新的 CSS3选择器标准:

<http://www.w3.org/TR/css3-selectors/>

标准中的选择器都可以在 jQuery 中使用.

jQuery 选择器按照功能主要分为"选择"和"过滤". 并且是配合使用的. 可以同时使用组合成一个选择器字符串. 主要的区别是"过滤"作用的选择器是指定条件从前面匹配的内容中筛选, "过滤"选择器也可以单独使用, 表示从全部"*"中筛选. 比如:

`$("#title")`

等同于:

`$("*:title")`

而"选择"功能的选择器则不会有默认的范围, 因为作用是"选择"而不是"过滤".

下面的选择器分类中, 带有"过滤器"的分类表示是"过滤"选择器, 否则就是"选择"功能的选择器.

jQuery 选择器分为如下几类:

[说明]

- 1.点击"名称"会跳转到此方法的 jQuery 官方说明文档.
- 2.可以在下节中的 jQuery 选择器实验室测试各种选择器

1. 基础选择器 Basics

名称	说明	举例
#id	根据元素 Id 选择	<code>\$("#divId")</code> 选择 ID 为 divId 的元素
element	根据元素的名称选择,	<code>\$("#a")</code> 选择所有<a>元素
.class	根据元素的 css 类选择	<code>\$(".bgRed")</code> 选择所用 CSS 类为 bgRed 的元素
*	选择所有元素	<code>\$("*")</code> 选择页面所有元素
selector1, selector2, selectorN	可以将几个选择器用","分隔开然后再拼成一个选择器字符串.会同时选中这几个选择器匹配的内容.	<code>\$("#divId, a, .bgRed")</code>

[学习建议]: 大家暂时记住基础选择器即可, 可以直接跳到下一节"jQuery 选择器实验室"进行动手练习, 以后再回来慢慢学习全部的选择器, 或者用到的时候再回来查询.

2.层次选择器 Hierarchy

名称	说明	举例
----	----	----

<u>ancestor descendant</u>	使用"form input"的形式选中form中的所有 input 元素.即 ancestor(祖先)为 from, descendant(子孙)为 input.	\$(".bgRed div") 选择 CSS 类为 bgRed 的元素中的所有<div>元素.
<u>parent > child</u>	选择 parent 的直接子节点 child. child 必须包含在 parent 中并且父类是 parent 元素.	\$(".myList>li") 选择 CSS 类为 myList 元素中的直接子节点对象.
<u>prev + next</u>	prev 和 next 是两个同级别的元素.选中在 prev 元素后面的 next 元素.	\$("#hibiscus+img") 选在 id 为 hibiscus 元素后面的 img 对象.
<u>prev ~ siblings</u>	选择 prev 后面的根据 siblings 过滤的元素 注:siblings 是过滤器	\$("#someDiv~[title]")选择 id 为 someDiv 的对象后面所有带有 title 属性的元素

3.基本过滤器 **Basic Filters**

名称	说明	举例
<u>:first</u>	匹配找到的第一个元素	查找表格的第一行:\$("tr:first")
<u>:last</u>	匹配找到的最后一个元素	查找表格的最后一行:\$("tr:last")
<u>:not(selector)</u>	去除所有与给定选择器匹配的元素	查找所有未选中的 input 元素 : \$("input:not(:checked)")
<u>:even</u>	匹配所有索引值为偶数的元素, 从 0 开始计数	查找表格的1、3、5...行:\$("tr:even")
<u>:odd</u>	匹配所有索引值为奇数的元素, 从 0 开始计数	查找表格的2、4、6行:\$("tr:odd")
<u>:eq(index)</u>	匹配一个给定索引值的元素 注:index 从 0 开始计数	查找第二行:\$("tr:eq(1)")
<u>:gt(index)</u>	匹配所有大于给定索引值的元素 注:index 从 0 开始计数	查找第二第三行, 即索引值是1和2, 也就是比0大:\$("tr:gt(0)")
<u>:lt(index)</u>	选择结果集中索引小于 N 的 elements 注:index 从 0 开始计数	查找第一第二行, 即索引值是0和1, 也就是比2小:\$("tr:lt(2)")
<u>:header</u>	选择所有 h1,h2,h3 一类的 header 标签.	给页面内所有标题加上背景色 : \$(".header").css("background", "#EEE");
<u>:animated</u>	匹配所有正在执行动画效果的元素	只有对不在执行动画效果的元素执行一个动画特效: \$("#run").click(function(){ \$("div:not(:animated)").animate({ left: "+=20" }, 1000); });

4. 内容过滤器 **Content Filters**

名称	说明	举例
:contains(text)	匹配包含给定文本的元素	查找所有包含 "John" 的 div 元素: \$("div:contains('John'))"
:empty	匹配所有不包含子元素或者文本的空元素	查找所有不包含子元素或者文本的空元素: \$("td:empty")
:has(selector)	匹配含有选择器所匹配的元素的元素	给所有包含 p 元素的 div 元素添加一个 text 类: \$("div:has(p)").addClass("test");
:parent	匹配含有子元素或者文本的元素	查找所有含有子元素或者文本的 td 元素: \$("td:parent")

5.可见性过滤器 **Visibility Filters**

名称	说明	举例
:hidden	匹配所有的不可见元素 注:在1.3.2版本中, hidden 匹配自身或者父类在文档中不占用空间的元素. 如果使用 CSS visibility 属性让其不显示但是占位, 则不输入 hidden.	查找所有不可见的 tr 元素: \$("tr:hidden")
:visible	匹配所有的可见元素	查找所有可见的 tr 元素: \$("tr:visible")

6.属性过滤器 **Attribute Filters**

名称	说明	举例
[attribute]	匹配包含给定属性的元素	查找所有含有 id 属性的 div 元素: \$("div[id]")
[attribute=value]	匹配给定的属性是某个特定值的元素	查找所有 name 属性是 newsletter 的 input 元素: \$("input[name='newsletter']").attr("checked", true);
[attribute!=value]	匹配给定的属性是不包含某个特定值的元素	查找所有 name 属性不是 newsletter 的 input 元素: \$("input[name!='newsletter']").attr("checked", true);
[attribute^=value]	匹配给定的属性是以某些值开始的元素	\$("input[name^='news'])
[attribute\$=value]	匹配给定的属性是以某些值结尾的元素	查找所有 name 以 'letter' 结尾的 input 元素: \$("input[name\$='letter'])
[attribute*=value]	匹配给定的属性是以包含某些值的元素	查找所有 name 包含 'man' 的 input 元素: \$("input[name*='man'])
[attributeFilter1][attribute	复合属性选择器, 需要	找到所有含有 id 属性, 并且它的

Filter2][attributeFilterN]	同时满足多个条件时使用。	name 属性是以 man 结尾的: \$("input[id][name\$='man']")
---	--------------	---

7.子元素过滤器 **Child Filters**

名称	说明	举例
:nth-child(index/even/odd/equation)	<p>匹配其父元素下的第N个子或奇偶元素</p> <p>'eq(index)' 只匹配一个元素，而这个将为每一个父元素匹配子元素。:nth-child 从1开始的，而:eq()是从0算起的！</p> <p>可以使用:</p> <p><code>nth-child(even)</code> <code>:nth-child(odd)</code> <code>:nth-child(3n)</code> <code>:nth-child(2)</code> <code>:nth-child(3n+1)</code> <code>:nth-child(3n+2)</code></p>	在每个 ul 查找第 2 个 li: \$("ul li:nth-child(2)")
:first-child	<p>匹配第一个子元素</p> <p>'first' 只匹配一个元素，而此选择符将为每个父元素匹配一个子元素</p>	在每个 ul 中查找第一个 li: \$("ul li:first-child")
:last-child	<p>匹配最后一个子元素</p> <p>'last'只匹配一个元素，而此选择符将为每个父元素匹配一个子元素</p>	在每个 ul 中查找最后一个 li: \$("ul li:last-child")
:only-child	<p>如果某个元素是父元素中唯一的子元素，那将会被匹配</p> <p>如果父元素中含有其他元素，那将不会被匹配。</p>	在 ul 中查找是唯一子元素的 li: \$("ul li:only-child")

8.表单选择器 **Forms**

名称	说明	解释
:input	匹配所有 input, textarea, select 和 button 元素	查找所有的 input 元素: \$("input")
:text	匹配所有的文本框	查找所有文本框: \$("text")
:password	匹配所有密码框	查找所有密码框: \$("password")
:radio	匹配所有单选按钮	查找所有单选按钮

<u>:checkbox</u>	匹配所有复选框	查找所有复选框: \$(":checkbox")
<u>:submit</u>	匹配所有提交按钮	查找所有提交按钮: \$(":submit")
<u>:image</u>	匹配所有图像域	匹配所有图像域: \$(":image")
<u>:reset</u>	匹配所有重置按钮	查找所有重置按钮: \$(":reset")
<u>:button</u>	匹配所有按钮	查找所有按钮: \$(":button")
<u>:file</u>	匹配所有文件域	查找所有文件域: \$(":file")


9.表单过滤器 **Form Filters**

名称	说明	解释
<u>:enabled</u>	匹配所有可用元素	查找所有可用的 input 元素: \$("input:enabled")
<u>:disabled</u>	匹配所有不可用元素	查找所有不可用的 input 元素: \$("input:disabled")
<u>:checked</u>	匹配所有选中的被选中元素(复选框、单选框等, 不包括 select 中的 option)	查找所有选中的复选框元素: \$("input:checked")
<u>:selected</u>	匹配所有选中的 option 元素	查找所有选中的选项元素: \$("select option:selected")

六 **jQuery 选择器实验室**

jQuery 选择器实验室使用的是"jQuery 实战"一书中的代码, 感觉对于学习选择器很有帮助.

我们的实验对象是一个拥有很多元素的页面:

Some images:

This is a <div> with an id of someDiv
Hello
Goodbye

- jQuery supports
 - CSS1
 - CSS2
 - CSS3
 - Basic XPath
- jQuery also supports
 - Custom selectors
 - Form selectors

Language	Type	Invented
Java	Static	1995
Ruby	Dynamic	1993
Smalltalk	Dynamic	1972
C++	Static	1983

Text:

Radio group: ☐ A ☐ B ☐ C

Checkboxes: ☐ 1 ☐ 2 ☐ 3 ☐ 4

Submit

在实验室页面的"Selector"输入框中输入 jQuery 选择器表达式， 所有匹配表达式的元素会显示红框：

Selector

Type a selector into the text field below and click the Apply button.

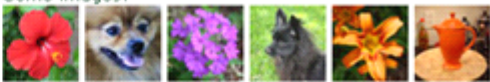
Selector:

Apply

jQuery statement:
\$(".myList").addClass("wrappedElement")

Matched elements: 1 match(es)
UL

DOM Sample

Some images:

This is a <div> with an id of someDiv
Hello
Goodbye

- jQuery supports
 - CSS1
 - CSS2
 - CSS3
 - Basic XPath
- jQuery also supports
 - Custom selectors
 - Form selectors

Language	Type	Invented
Java	Static	1995
Ruby	Dynamic	1993
Smalltalk	Dynamic	1972
C++	Static	1983

Text:

Radio group: ☒ A ☐ B ☐ C

Checkboxes: ☐ 1 ☐ 2 ☐ 3 ☐ 4

Submit

如上图所示， 在输入".myList"后点击"Apply"， 下面的输出框会显示运行结果， 右侧会将选中的元素用红框显示。

代码在本章最后可以下载。

七.API 文档

jQuery 官方 API: <http://docs.jquery.com/>

中文在线 API: <http://jquery.org.cn/visual/cn/index.xml>

中文 jQuery 手册下载: http://files.cnblogs.com/zhangziqu/jquery_api.rar

八.总结

本章节讲解的 jQuery 依然属于基础支持, 所以没有太多的应用实例. 虽然基础但是很难一次全部记住, jQuery 选择器可以说是最考验一个人 jQuery 功力的地方. 下一章我们讲解如何操作 jQuery 包装集以及动态创建新元素.

本章代码下载:

<http://files.cnblogs.com/zhangziqu/Code-jQueryStudy-2.rar>

Tag 标签: [jQuery](#), [jQuery 教程](#)

jQuery (三) 管理 jQuery 包装集

一.摘要

在使用 jQuery 选择器获取到 jQuery 包装集后，我们需要对其进行操作. 本章首先讲解如何动态的创建元素，接着学习如何管理 jQuery 包装集，比如添加,删除,切片等.

二.前言

本系列的2,3篇上面列举了太多的 API 相信大家看着眼晕. 不过这些基础还必须要讲，基础要扎实.其实对于这些列表大家可以跳过，等以后用到时再回头看或者查询官方的 API 说明.

本章内容很少，主要讲解动态创建元素和操作 jQuery 包装集的各个函数.

三.动态创建元素

1.错误的编程方法

我们经常使用 javascript 动态的创建元素，有很多程序员通过直接更改某一个容器的 HTML 内容.比如:

```
<!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>

    <title>动态创建对象</title>

</head>

<body>

<div id="testDiv">测试图层</div>

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

```
document.getElementById("testDiv").innerHTML = "<div style=\"border:solid 1px #FF0000\">动态创建的 div</div>";

</script>

</body>

</html>
```

上面的示例中我通过修改 testDiv 的内容,在页面上动态的添加了一个 div 元素. **但是请牢记,这是错误的做法!**

错误的原因:

(1) 在页面加载时改变了页面的结构. 在 IE6中如果网络变慢或者页面内容太大就会出现"终止操作"的错误. 也就是说"永远不要在页面加载时改变页面的 Dom 模型".

(2) 使用修改 HTML 内容添加元素, 不符合 Dom 标准. 在实际工作中也碰到过使用这种方法修改内容后, 某些浏览器中并不能立刻显示添加的元素, 因为不同浏览器的显示引擎是不同的. 但是如果我们使用 Dom 的 CreateElement 创建对象, 在所有的浏览器中几乎都可以. 但是在 jQuery 中如果传入的而是一个完整的 HTML 字符串, 内部也是使用 innerHTML. 所以也不是完全否定 innerHTML 函数的使用.

所以从现在开始请摒弃这种旧知识, 使用下面介绍的正确方法编程.

2.创建新的元素

下面介绍两种正确的创建元素的方式.

(1)使用 HTML DOM 创建元素

什么是 DOM?

通过 JavaScript, 您可以重构整个 HTML 文档. 您可以添加、移除、改变或重排页面上的项目。

要改变页面的某个东西, JavaScript 就需要对 HTML 文档中所有元素进行访问的入口。这个入口, 连同对 HTML 元素进行添加、移动、改变或移除的方法和属性, 都是通过文档对象模型来获得的 (DOM)。

在 1998 年, W3C 发布了第一级的 DOM 规范。这个规范允许访问和操作 HTML 页面中的每一个单独的元素。

所有的浏览器都执行了这个标准, 因此, DOM 的兼容性问题也几乎难觅踪影了。

DOM 可被 JavaScript 用来读取、改变 HTML、XHTML 以及 XML 文档。

DOM 被分为不同的部分 (核心、XML 及 HTML) 和级别 (DOM Level 1/2/3):

Core DOM

定义了一套标准的针对任何结构化文档的对象

[XML DOM](#)

定义了一套标准的针对 XML 文档的对象

HTML DOM

定义了一套标准的针对 HTML 文档的对象。

关于使用 HTML DOM 创建元素本文不做详细介绍，下面举一个简单的例子：

//使用 Dom 标准创建元素

```
var select = document.createElement("select");

select.options[0] = new Option("加载项1", "value1");

select.options[1] = new Option("加载项2", "value2");

select.size = "2";

var object = testDiv.appendChild(select);
```

通过使用 `document.createElement` 方法我们可以创建 Dom 元素，然后通过 `appendChild` 方法为添加到指定对象上。

(2) 使用 jQuery 函数创建元素

在 jQuery 中创建对象更加简单，比如创建一个 Div 元素：

```
$("<div style=\"border:solid 1px #FF0000\">动态创建的 div</div>")
```

我们主要使用 jQuery 核心类库中的一个方法：

[jQuery\(html, ownerDocument \)](#)

Returns: jQuery

根据 HTML 原始字符串动态创建 Dom 元素。

其中 html 参数是一个 HTML 字符串，在 jQuery1.3.2 中对此函数做了改进：

当 HTML 字符串是没有属性的元素是，内部使用 `document.createElement` 创建元素，比如：

//jQuery 内部使用 document.createElement 创建元素：

```
$("<div/>").css("border","solid 1px #FF0000").html(" 动 态 创 建 的  
div").appendTo(testDiv);
```

否则使用 `innerHTML` 方法创建元素:

//jQuery 内部使用 `innerHTML` 创建元素:

```
$("#<div style=\"border:solid 1px #FF0000\"> 动 态 创 建 的  
div</div>").appendTo(testDiv);
```

3.将元素添加到对象上

我们可以使用上面两种方式创建一个而元素,但是上面已经提到一定不要在页面加载时就改变页面的 DOM 结构,比如添加一个元素.正确的做法是在页面加载完毕后添加或删除元素.

传统上,使用 `window.onload` 完成上述目的:

//DOM 加载完毕后添加元素

//传统方法

```
window.onload = function() { testDiv.innerHTML = "<div  
style=\"border:solid 1px #FF0000\">动态创建的 div</div>"; }
```

虽然能够在 DOM 完整加载后,在添加新的元素,但是不幸的是浏览器执行 `window.onload` 函数不仅仅是在构建完 DOM 树之后,也是在所有图像和其他外部资源完整的加载并且在浏览器窗口显示完毕之后.所以如果某个图片或者其他资源加载很长时间,访问者就会看到一个不完整的页面,甚至在图片加载之前就执行了需要依赖动态添加的元素的脚本而导致脚本错误.

解决办法就是等 DOM 被解析后,在图像和外部资源加载之前执行我们的函数.在 jQuery 中让这一实现变得可行:

//jQuery 使用动态创建的`$(document).ready(function)`方法

```
$(document).ready(  
  
    function() { testDiv.innerHTML = "<div style=\"border:solid 1px  
#FF0000\">使用动态创建的$(document).ready(function)方法</div>"; }  
  
);
```

或者使用简便语法:

//jQuery 使用`$(function)`方法

```
$(  
  
    function() { testDiv.innerHTML += "<div style=\"border:solid 1px  
  
#FF0000\">使用$(function)方法</div>"; }  
  
);
```

使用\$()将我们的函数包装起来即可。而且可以在一个页面绑定多个函数，如果使用传统的window.onload 则只能调用一个函数。

所以请大家将修改 DOM 的函数使用此方法调用。另外还要注意 document.createElement 和 innerHTML 的区别。如果可以请尽量使用 document.createElement 和\$("<div/>")的形式创建对象。

四.管理 jQuery 包装集元素

既然学会了动态创建元素，接下来就会想要把这些元素放入我们的 jQuery 包装集中。

我们可以在 jQuery 包装集上调用下面这些函数，用来改变我们的原始 jQuery 包装集，并且大部分返回的都是过滤后的 jQuery 包装集。

jQuery 提供了一系列的函数用来管理包装集：

1.过滤 Filtering

名称	说明	举例
eq(index)	获取第 N 个元素	获取匹配的第二个元素： \$("p").eq(1)
filter(expr)	筛选出与指定表达式匹配的元素集合。	保留带有 select 类的元素： \$("p").filter(".selected")
filter(fn)	筛选出与指定函数返回值匹配的元素集合 这个函数内部将对每个对象计算一次（正如 '\$.each'）。如果调用的函数返回 false 则这个元素被删除，否则就会保留。	保留子元素中不含有 ol 的元素： \$("div").filter(function(index) { return \$("ol", this).size() == 0; });
is(expr) 注意：这个函数返回的不是 jQuery 包装集而是 Boolean 值	用一个表达式来检查当前选择的元素集合，如果其中至少有一个元素符合这个给定的表达式就返回 true。 如果没有元素符合，或者表达式无效，都返回 'false'。'filter' 内部实际也是在调用这个函数，所以	由于 input 元素的父元素是一个表单元素，所以返回 true： \$("input[type='checkbox']").parent().is("form")

	， filter()函数原有的规则在这里也适用。	
<u>map(callback)</u>	将一组元素转换成其他数组（不论是否是元素数组） 你可以用这个函数来建立一个列表，不论是值、属性还是 CSS 样式,或者其他特别形式。这都可以用'\$ <u>map()</u> '来方便的建立	把 form 中的每个 input 元素的值建立一个列表: \$(" <u>p</u> ").append(\$(" <u>input</u> ").map(function(){ return \$(this).val(); }).get().join(", "));
<u>not(expr)</u>	删除与指定表达式匹配的元素	从 p 元素中删除带有 select 的 ID 的元素: \$(" <u>p</u> ").not(\$("# <u>selected</u> ") <u>[0]</u>)
<u>slice(start, end)</u>	选取一个匹配的子集	选择第一个 p 元素: \$(" <u>p</u> ").slice(0, 1);

2.查找 Finding

名称	说明	举例
<u>add(expr)</u>	把与表达式匹配的元素添加到 jQuery 对象中。这个函数可以用于连接分别与两个表达式匹配的元素结果集。	动态生成一个元素并添加至匹配的元素中: \$(" <u>p</u> ").add("Again")
<u>children([expr])</u>	取得一个包含匹配的元素集合中每一个元素的所有子元素的元素集合。 可以通过可选的表达式来过滤所匹配的子元素。注意：parents()将查找所有祖辈元素，而 children()只考虑子元素而不考虑所有后代元素。	查找 DIV 中的每个子元素: \$(" <u>div</u> ").children()
<u>closest([expr])</u>	取得与表达式匹配的最新的父元素	为事件源最近的父类 li 对象更换样式: \$(document).bind("click", function (e) { \$(e.target).closest(" <u>li</u> ").toggleClass("highlight"); });
<u>contents()</u>	查找匹配元素内部所有的子节点（包括文本节点）。如果元素是一个 iframe，则查找文档内容	查找所有文本节点并加粗: \$(" <u>p</u> ").contents().not("[<u>nodeType</u> =1]").wrap("");
<u>find(expr)</u>	搜索所有与指定表达式匹配的元素。这个函数是找出正在处理的元素的后代元素的好方法。	从所有的段落开始，进一步搜索下面的 span 元素。与\$(" <u>p span</u> ")相同: \$(" <u>p</u> ").find("span")

	所有搜索都依靠jQuery表达式来完成。这个表达式可以使用 CSS1-3的选择器语法来写。	
next([expr])	<p>取得一个包含匹配的元素集合中每一个元素紧邻的后面同辈元素的元素集合。</p> <p>这个函数只返回后面那个紧邻的同辈元素，而不是后面所有的同辈元素（可以使用 nextAll）。可以用一个可选的表达式进行筛选。</p>	<p>找到每个段落的后面紧邻的同辈元素:</p> <p><code>\$("p").next()</code></p>
nextAll([expr])	<p>查找当前元素之后所有的同辈元素。</p> <p>可以用表达式过滤</p>	<p>给第一个 div 之后的所有元素加个类:</p> <p><code>\$("div:first").nextAll().addClass("after")</code>;</p>
offsetParent()	返回第一个有定位的父类 (比如 (relative 或 absolute)).	
parent([expr])	<p>取得一个包含着所有匹配元素的唯一父元素的元素集合。</p> <p>你可以使用可选的表达式来筛选。</p>	<p>查找每个段落的父元素:</p> <p><code>\$("p").parent()</code></p>
parents([expr])	取得一个包含着所有匹配元素的祖先元素的元素集合（不包含根元素）。可以通过一个可选的表达式进行筛选。	<p>找到每个 span 元素的所有祖先元素:</p> <p><code>\$("span").parents()</code></p>
prev([expr])	<p>取得一个包含匹配的元素集合中每一个元素紧邻的前一个同辈元素的元素集合。</p> <p>可以用一个可选的表达式进行筛选。只有紧邻的同辈元素会被匹配到，而不是前面所有的同辈元素。</p>	<p>找到每个段落紧邻的前一个同辈元素:</p> <p><code>\$("p").prev()</code></p>
prevAll([expr])	<p>查找当前元素之前所有的同辈元素</p> <p>可以用表达式过滤。</p>	<p>给最后一个之前的所有 div 加上一个类:</p> <p><code>\$("div:last").prevAll().addClass("before");</code></p>
siblings([expr])	取得一个包含匹配的元素集合中每一个元素的所有唯一同辈元素的元素集合。可以用可选的表达式进行筛选。	<p>找到每个 div 的所有同辈元素:</p> <p><code>\$("div").siblings()</code></p>

3.串联 Chaining

名称	说明	举例
<u>andSelf()</u>	<p>加入先前所选的加入当前元素中</p> <p>对于筛选或查找后的元素，要加入先前所选元素时将会很有用。</p>	<p>选取所有 div 以及内部的 p，并加上 border 类:</p> <pre>\$("#div").find("p").andSelf().addClass("border");</pre>
<u>end()</u>	<p>回到最近的一个"破坏性"操作之前。即，将匹配的元素列表变为前一次的状态。</p> <p>如 果之前没有破坏性操作，则返回一个空集。所谓的"破坏性"就是指任何改变所匹配的 jQuery 元素的操作。这包括在 Traversing 中任何返回一个 jQuery 对象的函数 -- 'add', 'andSelf', 'children', 'filter', 'find', 'map', 'next', 'nextAll', 'not', 'parent', 'parents', 'prev', 'prevAll', 'siblings' and 'slice'-- 再加上 Manipulation 中的 'clone'。</p>	<p>选取所有的 p 元素，查找并选取 span 子元素，然后再回过来选取 p 元素:</p> <pre>\$("#p").find("span").end()</pre>

五.常用函数举例

[待续]

六.总结

本篇文章内容较少，主要讲解如何动态创建元素以及管理 jQuery 包装集，接口文档列举了太多，实例部分还没来得及写。因为要睡觉明天还要上班，所以请各位见谅，等以后有空的时候补上！

jQuery (四) 使用 jQuery 操作元素的属性与样式

一.摘要

本篇文章讲解如何使用 jQuery 获取和操作元素的属性和 CSS 样式. 其中 DOM 属性和元素属性的区分值得大家学习.

二.前言

通过前面几章我们已经能够完全控制 jQuery 包装集了, 无论是通过选择器选取对象, 或者从包装集中删除,过滤元素. 本章将讲解如何使用 jQuery 获取和修改元素属性和样式.

三. 区分 DOM 属性和元素属性

一个 img 标签:

```

```

通常开发人员习惯将 id, src, alt 等叫做这个元素的"属性". 我将其称为"元素属性". 但是在解析成 DOM 对象时, 实际浏览器最后会将标签元素解析成"DOM 对象", 并且将元素的"元素属性"存储为"DOM 属性". 两者是有区别的.

虽然我们设置了元素的 src 是相对路径:images/image.1.jpg
但是在"DOM 属性"中都会转换成绝对路径:<http://localhost/images/image.1.jpg>.

甚至有些"元素属性"和"DOM 属性"的名称都不一样,比如上面的元素属性 class, 转换为 DOM 属性后对应 className.

牢记, 在 javascript 中我们可以直接获取或设置"DOM 属性":

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

$(function() {

    var img1 = document.getElementById("hibiscus");

    alert(img1.alt);

    img1.alt = "Change the alt element attribute";
```

```
        alert(img1.alt);

    })

</script>
```

所以如果要设置元素的 CSS 样式类, 要使用的是"DOM 属性"className"而不是"元素属性"class:

```
img1.className = "classB";
```

四. 操作"DOM 属性"

在 jQuery 中没有包装操作"DOM 属性"的函数, 因为使用 javascript 获取和设置"DOM 属性"都很简单. 在 jQuery 提供了 each()函数用于遍历 jQuery 包装集, 其中的 this 指针是一个 DOM 对象, 所以我们可以应用这一点配合原生 javascript 来操作元素的 DOM 属性:

```
$( "img" ).each( function( index ) {

    alert( "index:" + index + ", id:" + this.id + ", alt:" +

this.alt);

    this.alt = "changed";

    alert( "index:" + index + ", id:" + this.id + ", alt:" +

this.alt);

});
```

下面是 each 函数的说明:

each(callback) Returns: jQuery 包装集

对包装集中的每一个元素执行 callback 方法. 其中 callback 方法接受一个参数, 表示当前遍历的索引值,从0开始.

五. 操作"元素属性"

我们可以使用 javascript 中的 `getAttribute` 和 `setAttribute` 来操作元素的"元素属性".

在 jQuery 中给你提供了 `attr()`包装集函数, 能够同时操作包装集中所有元素的属性:

名称	说明	举例
<u>attr(name)</u>	取得第一个匹配元素的属性值。通过这个方法可以方便地从第一个匹配元素中获取一个属性的值。如果元素没有相应属性，则返回 <code>undefined</code> 。	返回文档中第一个图像的 <code>src</code> 属性值: <code>\$("img").attr("src");</code>
<u>attr(properties)</u>	将一个“名/值”形式的对象设置为所有匹配元素的属性。 这是一种在所有匹配元素中批量设置很多属性的最佳方式。 注意，如果你要设置对象的 <code>class</code> 属性，你必须使用 <code>'className'</code> 作为属性名。或者你可以直接使用 <code>.addClass(class)</code> 和 <code>.removeClass(class)</code> 。	为所有图像设置 <code>src</code> 和 <code>alt</code> 属性: <code>\$("img").attr({ src: "test.jpg", alt: "Test Image" });</code>
<u>attr(key, value)</u>	为所有匹配的元素设置一个属性值。	为所有图像设置 <code>src</code> 属性: <code>\$("img").attr("src","test.jpg");</code>
<u>attr(key, fn)</u>	为所有匹配的元素设置一个计算的属性值。 不提供值，而是提供一个函数，由这个函数计算的值作为属性值。	把 <code>src</code> 属性的值设置为 <code>title</code> 属性的值: <code>\$("img").attr("title", function() { return this.src });</code>
<u>removeAttr(name)</u>	从每一个匹配的元素中删除一个属性	将文档中图像的 <code>src</code> 属性删除: <code>\$("img").removeAttr("src");</code>

当使用 `id` 选择器时常常返回只有一个对象的 jQuery 包装集, 这个时候常使用 `attr(name)`函数获得它的元素属性:

```
function testAttr1(event) {  
  
    alert($("#hibiscus").attr("class"));  
  
}
```

注意 `attr(name)`函数只返回第一个匹配元素的特定元素属性值. 而 `attr(key, name)`会设置所有包装集中的元素属性:

```
//修改所有 img 元素的 alt 属性
```

`$("#img").attr("alt", "修改后的 alt 属性");`

而 [attr\(properties \)](#) 可以一次修改多个元素属性:

`$("#img").attr({title:"修改后的 title", alt: "同时修改 alt 属性"});`

另外虽然我们可以使用 [removeAttr\(name \)](#) 删除元素属性,但是对应的 DOM 属性是不会被删除的,只会影响 DOM 属性的值.

比如将一个 input 元素的 readonly 元素属性去掉,会导致对应的 DOM 属性变成 false(即 input 变成可编辑状态):

`$("#inputTest").removeAttr("readonly");`

六,修改 CSS 样式

修改元素的样式, 我们可以修改元素 CSS 类或者直接修改元素的样式.

一个元素可以应用多个 css 类, 但是不幸的是在 DOM 属性中是用一个以空格分割的字符串存储的, 而不是数组. 所以如果在原始 javascript 时代我们对元素添加或者删除多个属性时, 都要自己操作字符串.

jQuery 让这一切变得异常简单. 我们再也不用做那些无聊的工作了.

1. 修改 CSS 类

下表是修改 CSS 类相关的 jQuery 方法:

名称	说明	实例
addClass(classes)	为每个匹配的元素添加指定的类名。	为匹配的元素加上 'selected' 类: <code>\$("#p").addClass("selected");</code>
hasClass(class)	判断包装集中是否至少有一个元素应用了指定的 CSS 类	<code>\$("#p").hasClass("selected");</code>
removeClass([classes])	从所有匹配的元素中删除全部或者指定的类。	从匹配的元素中删除 'selected' 类: <code>\$("#p").removeClass("selected");</code>
toggleClass(class)	如果存在（不存在）就删除（添加）一个类。	为匹配的元素切换 'selected' 类: <code>\$("#p").toggleClass("selected");</code>
toggleClass(class, switch)	当 switch 是 true 时添加类, 当 switch 是 false 时删除类	每三次点击切换高亮样式: <code>var count = 0; \$("#p").click(function(){ \$(this).toggleClass("highlight", count++ % 3 == 0); });</code>

使用上面的方法，我们可以将元素的 CSS 类像集合一样修改，再也不必手工解析字符串。

注意 `addClass(class)` 和 `removeClass([classes])` 的参数可以一次传入多个 css 类，用空格分割,比如:

```
$( "#btnAdd" ).bind( "click", function( event ) { $( "p" ).addClass( "colorRed borderBlue" ); } );
```

removeClass 方法的参数可选，如果不传入参数则移除全部 CSS 类:

```
$( "p" ).removeClass();
```

2. 修改 CSS 样式

同样当我们想要修改元素的具体某一个CSS样式,即修改元素属性"style"时， jQuery 也提供了相应的方法:

名称	说明	实例
<code>css(name)</code>	访问第一个匹配元素的样式属性。	取得第一个段落的 color 样式属性的值: <code>\$("p").css("color");</code>
<code>css(properties)</code>	把一个“名/值对”对象设置为所有匹配元素的样式属性。 这是一种在所有匹配的元素上设置大量样式属性的最佳方式。	将所有段落的字体颜色设为红色并且背景为蓝色: <code>\$("p").css({ color: "#ff0011", background: "blue" });</code>
<code>css(name, value)</code>	在所有匹配的元素中，设置一个样式属性的值。 数字将自动转化为像素值	将所有段落字体设为红色: <code>\$("p").css("color", "red");</code>

七.获取常用属性

虽然我们可以通过获取属性,特性以及 CSS 样式来取得元素的几乎所有信息， 但是注意下面的实验:

```
<!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>
```

```
<title>获取对象宽度</title>
```

```
<script type="text/javascript" src="scripts/jquery-1.3.2-  
vsdoc2.js"></script>
```

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

```
$(function()
```

```
{
```

```
    alert("attr(\"width\"):\" + $("#testDiv").attr("width"));
```

```
//undefined
```

```
    alert("css(\"width\"):\" + $("#testDiv").css("width"));
```

```
//auto(ie6) 或 1264px(ff)
```

```
    alert("width():\" + $("#testDiv").width()); //正确的数值1264
```

```
    alert("style.width:\" + ($("#testDiv")[0].style.width ); //空值
```

```
})
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<div id="testDiv">
```

```
    测试文本</div>
```

</body>

</html>

我们希望获取测试图层的宽度， 使用 attr 方法获取"元素特性"为 undefined, 因为并没有为 div 添加 width. 而使用 css()方法虽然可以获取到 style 属性的值，但是在不同浏览器里返回的结果不同, IE6下返回 auto, 而 FF 下虽然返回了正确的数值但是后面带有"px". 所以 jQuery 提供了 width()方法, 此方法返回的是正确的不带 px 的数值.

针对上面的问题, jQuery 为常用的属性提供了获取和设置的方法，比如 width()用户获取元素的宽度，而 width(val)用来设置元素宽度.

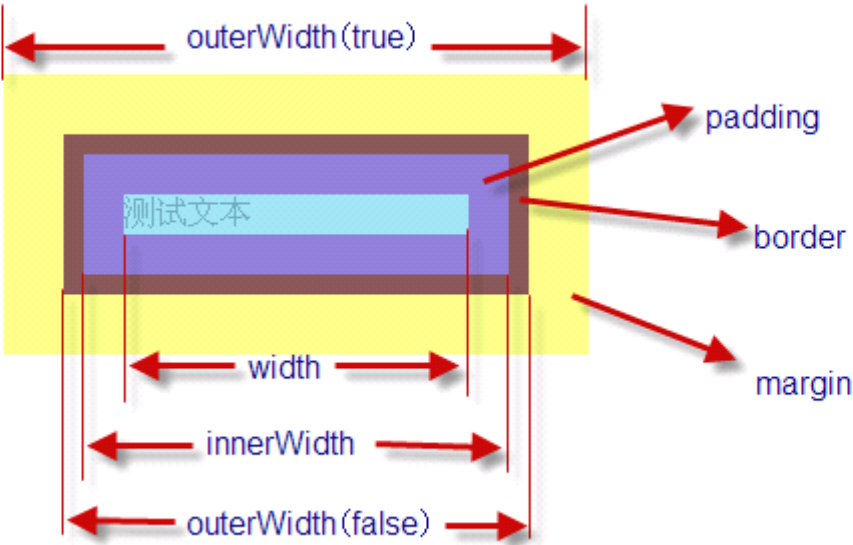
下面这些方法可以用来获取元素的常用属性值:

1.宽和高相关 Height and Width

名称	说明	举例
height()	取得第一个匹配元素当前计算的高度值（px）。	获取第一段的高: \$("p").height();
height(val)	为每个匹配的元素设置 CSS 高度 (hidth)属性的值。如果没有明确指定单位（如：em 或%），使用 px。	把所有段落的高设为 20: \$("p").height(20);
width()	取得第一个匹配元素当前计算的宽度值（px）。	获取第一段的宽: \$("p").width();
width(val)	为每个匹配的元素设置 CSS 宽度 (width)属性的值。 如果没有明确指定单位（如：em 或%），使用 px。	将所有段落的宽设为 20: \$("p").width(20);
innerHeight()	获取第一个匹配元素内部区域高度（包括补白、不包括边框）。 此方法对可见和隐藏元素均有效。	见最后示例
innerWidth()	获取第一个匹配元素内部区域宽度（包括补白、不包括边框）。 此方法对可见和隐藏元素均有效。	见最后示例
outerHeight([margin])	获取第一个匹配元素外部高度（默认包括补白和边框）。 此方法对可见和隐藏元素均有效。	见最后示例
outerWidth([margin])	获取第一个匹配元素外部宽度（默认包括补白和边框）。	见最后示例

	此方法对可见和隐藏元素均有效。	
--	-----------------	--

关于在获取长宽的函数中，要区别"inner", "outer"和 height/width 这三种函数的区别:



outerWidth 可以接受一个 bool 值参数表示是否计算 margin 值.

相信此图一目了然各个函数所索取的范围. 图片以 width 为例说明的, height 的各个函数同理.

2.位置相关 Positioning

另外在一些设计套弹出对象的脚本中,常常需要动态获取弹出坐标并且设置元素的位置.

但是很多的计算位置的方法存在着浏览器兼容性问题, jQuery 中为我们提供了位置相关的各个函数:

名称	说明	举例
offset()	获取匹配元素在当前窗口的相对偏移。 返回的对象包含两个整形属性: top 和 left。此方法只对可见元素有效。	获取第二段的偏移: var p = \$("p:last"); var offset = p.offset(); p.html("left: " + offset.left + ", top: " + offset.top);
position()	获取匹配元素相对父元素的偏移。 返回的对象包含两个整形属性: top 和 left。为精确计算结果,请在补白、边框和填充属性上使用像素单位。此方法只对可见元素有效。	获取第一段的偏移: var p = \$("p:first"); var position = p.position(); \$("p:last").html("left: " + position.left + ", top: " + position.top);
scrollTop()	获取匹配元素相对滚动条顶部的偏移。	获取第一段相对滚动条顶部的偏移: var p = \$("p:first"); \$("p:last").text("scrollTop:" + p.scroll

	此方法对可见和隐藏元素均有效。	Top());
<u>scrollTop(val)</u>	传递参数值时，设置垂直滚动条顶部偏移为该值。 此方法对可见和隐藏元素均有效。	设定垂直滚动条值: \$("div.demo").scrollTop(300);
<u>scrollLeft()</u>	获取匹配元素相对滚动条左侧的偏移。 此方法对可见和隐藏元素均有效。	获取第一段相对滚动条左侧的偏移: var p = \$("p:first"); \$("p:last").text("scrollLeft:" + p.scrollLeft());
<u>scrollLeft(val)</u>	传递参数值时，设置水平滚动条左侧偏移为该值。 此方法对可见和隐藏元素均有效。	设置相对滚动条左侧的偏移: \$("div.demo").scrollLeft(300);

八.总结

本篇文章主要讲解如何使用 jQuery 操作元素的属性和修改样式。请大家主要注意元素属性和 DOM 属性的区别。下一篇文章将讲解最重要的事件，介绍如何绑定事件，操作事件对象等。

Tag 标签: [jQuery](#),[jQuery 教程](#)

从零开始学习 jQuery (五) 事件与事件对象

一.摘要

事件是脚本编程的灵魂. 所以本章内容也是 jQuery 学习的重点. 本文将对 jQuery 中的事件处理以及事件对象进行详细的讲解.

二.前言

本 篇文章是至今为止本系列内容最多的一篇, 足以可见其重要性. 大家反映要多列举示例. 我会在时间允许的情况下尽量多列举示例. 真正的投入生产使用的实例暂时还无法加入到文章中, 但是可能最后我会列举一些作品供大家借鉴. 另外本人水平有限, 因为我不是 UI 设计师. 文章可能有错误的地方, 希望大家帮忙指出, 一起学习一起进步. 在技术的世界里我们是没有任何利益瓜葛. 希望大家都抱着彼此鼓励的心态, 对于回复中的激进评论我也都会考虑, 但是希望能够彼此尊重, 保护博客园这片程序员的净土!

三.事件与事件对象

曾经在我的 ["Javascript 公共脚本库系列\(二\): 添加事件多播委托的方法"](#) 和 ["Javascript 公共脚本库系列\(三\): 格式化事件对象/事件对象详解"](#) 两篇文章中, 曾讲解过 javascript 中的事件和事件对象.

首先看一下我们经常使用的添加事件的方式:

```
<!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>

<title>javascript 中的事件</title>

<script type="text/javascript" src="scripts/jquery-1.3.2-
vsdoc2.js"></script>
```

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

    $(function()

    {

        document.getElementById("testDiv2").onclick = showMsg;

    })

    function showMsg(event)

    {

        alert("!!!");

    }

</script>

</head>

<body>

    <div id="testDiv1" onclick="showMsg();">单击事件 1</div>

    <div id="testDiv2">单击事件 2</div>

</body>

</html>
```

我们最常使用为元素添加 onclick 元素属性的方式添加事件.

为 testDiv2 的添加 onclick 事件的方式是修改 Dom 属性.

在上一章中已经说明了什么是元素属性, 什么是 Dom 属性. 这两种方式的效果相同. 当单击 div 时会显示提示框.

请注意, 虽然效果相同, 但是并不等效.

```
document.getElementById("testDiv2").onclick = showMsg;
```

等效于:

```
<div id="testDiv1" onclick="alert('!!!');">单击事件 1</div>
```

注意两者的区别了吗? 我们常用的修改元素属性添加事件的方式, 实际上是建立了一个匿名函数:

```
document.getElementById("testDiv1").onclick = function(event)

{

    alert("!!!");

};
```

这个匿名函数的签名和我们手写的 showMsg 签名相同, 所以可以把 showMsg 直接赋值给 onclick.

这种方式的弊端是:

1. 只能为一个事件绑定一个事件处理函数. 使用 "=" 赋值会把前面为此时间绑定的所有事件处理函数冲掉.
2. 在事件函数(无论是匿名函数还是绑定的函数)中获取事件对象的方式在不同浏览器中要特殊处理:

IE 中, 事件对象是 window 对象的一个属性. 事件处理函数必须这样访问事件对象:

```
obj.onclick=function()

{

    var oEvent = window.event;

}
```

在 DOM 标准中,事件对象必须作为唯一参数传给事件处理函数:

```
obj.onclick=function()  
  
{  
  
    var oEvent = arguments[0];  
  
}
```

除了使用 argument[0]访问此参数, 我们也可以指定参数名称,上面的代码等同于:

```
obj.onclick=function(oEvent)  
  
{  
  
  
  
}
```

目前兼容 DOM 的浏览器有 Firefox,Safari,Opera,IE7等.

3. 添加多播委托的函数在不同浏览器中是不一样的.

下面是在"[Javascript 公共脚本库系列\(二\): 添加事件多播委托的方法](#)"文章中, 提供的兼容多浏览器添加多播委托的方法:

```
//统一的为对象添加多播事件委托的方法
```

```
/*
```

参数说明:

oTarget : 要添加事件的对象.比如"document".

sEventType : 事件类型.比如单击事件"click".

fnHandler : 发生事件时调用的方法. 比如一个静态函数"hideCalendar"

使用举例:

//单击页面的任何元素,只要没有取消冒泡,都可以关闭日历控件

```
var cf = document.getElementById("CalFrame");
```

```
if( cf != null && hideCalendar != null )
```

```
{
```

```
    ScriptHelper.addEventListener( document, "click", hideCalendar );
```

```
}
```

```
*/
```

```
scriptHelper.prototype.addEventListener = function(oTarget, sEventType,
fnHandler)
```

```
{
```

```
    if( oTarget.addEventListener )//for dom
```

```
{
```

```
    oTarget.addEventListener( sEventType, fnHandler, false )
```

```
}
```

```
    else if( oTarget.attachEvent )//for ie
```

```
{
```

```
    oTarget.attachEvent( "on" + sEventType, fnHandler);
```

```
}
```

}

所以我们首先应该摒弃<div onclick="..."></div>这种通过修改元素属性添加事件的方式. 尽量使用添加多播事件委托的方式为一个事件绑定多个事件处理函数, 比如为 document 对象的单击事件添加一个关闭弹出层的方法, 使用多播就不会影响 document 对象原有的事件处理函数.

四.jQuery 中的事件

有了 jQuery, 我们有了处理对象事件的一系列函数. 上面基础知识还是要懂, 但是再也不用自己去实现处理多播事件委托的函数了. 正所谓有了 jQuery, 天天喝茶水. 下面是在 jQuery 中最常使用的 bind()方法举例:

```
$("#testDiv4").bind("click", showMsg);
```

我们为 id 是 testDiv4的元素, 添加列 click 事件的事件处理函数 showMsg.

使用 jQuery 事件处理函数的好处:

1. 添加的是多播事件委托. 也就是为 **click** 事件又添加了一个方法, 不会覆盖对象的 **click** 事件原有的事件处理函数.

```
$("#testDiv4").bind("click", function(event) { alert("one"); });
```

```
$("#testDiv4").bind("click", function(event) { alert("two"); });
```

单击 testDiv4对象时, 依次提示"one"和"two".

2. 统一了事件名称.

添加多播事件委托时, ie 中是事件名称前面有"on". 但是使用 bind()函数我们不用区分 ie 和 dom , 因为内部 jQuery 已经帮我们统一了事件的名称.

3. 可以将对象行为全部用脚本控制.

让 HTML 代码部分只注意"显示"逻辑. 现在的趋势是将 HTML 的行为, 内容与样式切分干净. 其中用脚本控制元素行为, 用 HTML 标签控制元素内容, 用 CSS 控制元素样式. 使用 jQuery 事件处理函数可以避免在 HTML 标签上直接添加事件.

下面是基础的 jQuery 事件处理函数:

事件处理 **Event Handling:**

名称	说明	举例
bind(type, [data], fn)	为每一个匹配元素的特定事件（像 click）绑定一个事件处理器函数。	当每个段落被点击的时候，弹出其文本： \$("p").bind("click", function(){ alert(\$(this).text()); });

<u>one(type, [data], fn)</u>	<p>为每一个匹配元素的特定事件（像 click）绑定一个一次性的事件处理函数。</p>	<p>当所有段落被第一次点击的时候，显示所有其文本：</p> <pre>\$("p").one("click", function() { alert(\$(this).text()); });</pre>
<u>trigger(event, [data])</u>	<p>在每一个匹配的元素上触发某类事件。</p> <p>这个函数也会导致浏览器同名的默认行为的执行。比如，如果用 trigger() 触发一个 'submit'，则同样会导致浏览器提交表单。如果要阻止这种默认行为，应返回 false。</p> <p>你也可以触发由 bind() 注册的自定义事件</p>	<p>给一个事件传递参数：</p> <pre>\$("p").click(function (event, a, b) { // 一个普通的点击事件时，a 和 b 是 undefined 类型 // 如果用下面的语句触发，那么 a 指向 "foo", 而 b 指向 "bar" }).trigger("click", ["foo", "bar"]);</pre>
<u>triggerHandler(event, [data])</u>	<p>这个特别的方法将会触发指定的事件类型上所有绑定的处理函数。但不会执行浏览器默认动作。</p>	<p>如果你对一个 focus 事件执行了 .triggerHandler() ，浏览器默认动作将不会被触发，只会触发你绑定的动作：</p> <pre>\$("#old").click(function() { \$("input").trigger("focus"); }); \$("#new").click(function() { \$("input").triggerHandler("focus"); }); \$("input").focus(function() { \$("Focused!").appendTo("body").fadeOut(1000); });</pre>
<u>unbind(type, fn)</u>	<p>bind() 的反向操作，从每一个匹配的元素中删除绑定的事件。</p> <p>如果没有参数，则删除所有绑定的事件。</p> <p>你可以将你用 bind() 注册的自定义事件取消绑定。</p> <p>如果提供了事件类型作为参数，则只删除该类型的绑定事件。</p> <p>如果把在绑定时传递的处理函数作为第二个参数，则只有这个特定的事件处理函数会被删除。</p>	<p>把所有段落的所有事件取消绑定：</p> <pre>\$("p").unbind()</pre> <p>将段落的 click 事件取消绑定：</p> <pre>\$("p").unbind("click")</pre> <p>删除特定函数的绑定，将函数作为第二个参数传入：</p> <pre>var foo = function () { // 处理某个事件的代码 }; \$("p").bind("click", foo); // ... 当点击段落的时候会触发 foo \$("p").unbind("click", foo); // ... 再也不会被触发 foo</pre>

五.常用事件函数举例

1.[bind\(type, \[data\], fn \)](#) 函数举例

bind()是最常用的函数， 注意方法签名上 data 参数，可以在事件处理之前传递一些附加的数据：

```
function handler(event) {  
  
    alert(event.data.foo);  
  
}  
  
$("p").bind("click", {foo: "bar"}, handler)
```

注意 event 参数的使用. jQuery 中统一了事件对象，将事件对象作为事件处理函数的唯一参数传递.

data 参数我们也要通过 event.data 进行访问. 为何要提供 data 参数呢？

因为我们经常碰到这样的问题：希望在事件处理中根据事件源的某些数据进行特殊处理.

目前网上有两种存在争议的解决方法：

(1) 使用自定义元素属性存储数据.

比如：

```
<div id="testDiv5" customer="customer data 1">获取自定义数据-1</div>
```

在事件处理函数中获取数据：

```
$("#testDiv5").bind("click", function(event)  
  
{ alert($(event.target).attr("customer")); });
```

attr 函数是上一讲中的知识，用于获取元素的"元素属性"，而且可以获取自定义的元素属性. 单击 div 后将显示：



(2) 使用脚本将数据传递给事件处理函数:

```
<div id="testDiv6">获取自定义数据-2</div>
```

元素没有任何的自定义属性, 添加事件处理函数时将额外的数据传递:

```
$("#testDiv6").bind("click", { customer: "customer data 2" }, function(event)
{ alert(event.data.customer) });
```

点击 div 后的结果和方法1相同:



方法1便于存储和查找数据. 但是自定义属性通过不 W3C 验证.

方法2必须要自己想办法存储数据, 并且要制定规则查找指定元素的数据.

从"开发人员"的角度方法1要更加简单直观. 但是缺点比较严重. 所以如何取舍请大家自己决定.

one(type, [data], fn) 函数和 bind 一样, 但是只执行一次.

2. trigger(event, [data]) 和 triggerHandler(event, [data])

虽然为元素绑定了某些事件, 比如 click, 但是有时希望在程序中触发这些事件, 这两个函数可以实现此功能.

主要区别是 trigger 会出发浏览器默认的动作, 而 triggerHandler 不会出发.

通过下面的实例可以明确的区分这两个函数:

```
<!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>
```

<title>jQuery 事件处理:trigger 和 triggerHandler 示例</title>

```
<script type="text/javascript" src="scripts/jquery-1.3.2-  
vsdoc2.js"></script>
```

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

```
$(function()
```

```
{
```

```
$("#old").click(function()
```

```
{
```

```
$("#divResult").html("");
```

```
$("#input").trigger("focus");
```

```
});
```

```
$("#new").click(function()
```

```
{
```

```
$("#divResult").html("");
```

```
$("#input").triggerHandler("focus");
```

```
});
```

```
$("#input").focus(function()
```

```
{ $("#<span>Focused!</span>").appendTo("#divResult"); });
```

```
})
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<button id="old">
```

```
.trigger("focus")</button>
```

```
<button id="new">
```

```
.triggerHandler("focus")</button><br />
```

```
<br />
```

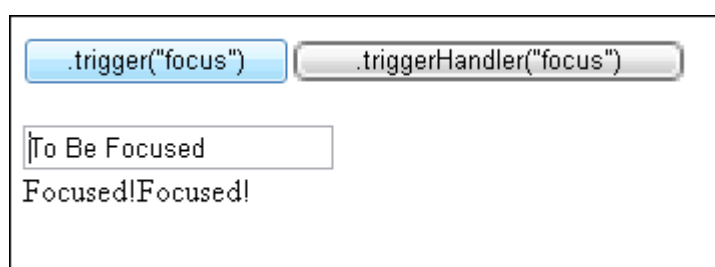
```
<input type="text" value="To Be Focused" />
```

```
<div id="divResult"></div>
```

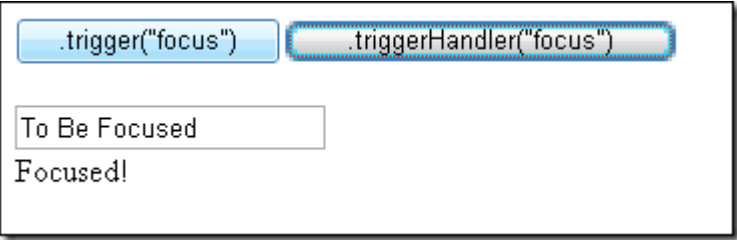
```
</body>
```

```
</html>
```

当单击".trigger"按钮时，会调用两次 Focused，并且 input 元素获得了焦点：



单击".triggerHandler"按钮时, 只调用一次,并且 input 元素没有获得焦点:



也就是说, trigger 函数出发了浏览器默认的获取焦点的行为,让 input 元素获得了焦点, 所以再次调用了 focus 事件处理函数.

triggerHandler 只调用为 focus 事件绑定的事件处理函数, 而不引发浏览器行为, 所以最后 input 元素没有获得焦点.

六.快捷事件 Event Helpers

BUG 提示:jquery-1.3.2-vsdoc2.js 这个最新的官方带智能提示的类库, 无法使用快捷事件, 比如 **click()**, **focus()**. 使用其他版本的类库则没有问题.

虽然我们可以使用事件处理函数完成对象事件的几乎所有操作, 但是 jQuery 提供了对常用事件的封装. 比如单击事件对应的两个方法 **click()**和 **click(fn)**分别用来触发单击事件和设置单击事件.

设置单击事件:

```
$("#testDiv").click(function(event) { alert("test div clicked ! "); });
```

等效于:

```
$("#testDiv").bind("click", function(event) { alert("test div clicked ! "); });
```

触发单击事件:

```
$("#testDiv").click();
```

等效于

```
$("#testDiv").trigger("click");
```

注意这里等效的是 trigger 而不是 triggerHandler.

此类方法在 jQuery 中英文叫做 Event Helpers, 我找不到很好的翻译方式, 所以按照功能称其为"快捷方法", 征集好的翻译名称!

下面是 jQuery 的快捷方法列表:

由于都是都是对应的事件, 所以不再写说明和举例了.

名称	说明	举例
blur()		
blur(fn)		
change()		
change(fn)		
click()		
click(fn)		
dblclick()		
dblclick(fn)		
error()		
error(fn)		
focus()		
focus(fn)		
keydown()		
keydown(fn)		
keypress()		
keypress(fn)		
keyup()		
keyup(fn)		
load(fn)		
mousedown(fn)		
mouseenter(fn)		
mouseleave(fn)		
mousemove(fn)		
mouseout(fn)		
mouseover(fn)		
mouseup(fn)		
resize(fn)		
scroll(fn)		
select()		
select(fn)		
submit()		
submit(fn)		
unload(fn)		

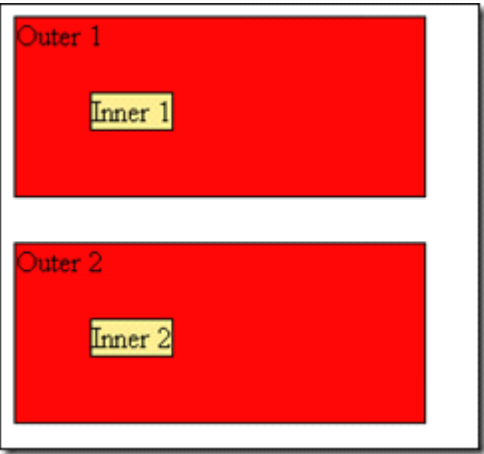
七. 交互帮助方法

除了基本的实践, jQuery 提供了两个和事件相关的帮助方法: [hover\(over, out \)](#) 和 [toggle\(fn, fn2,](#)

[fn3,fn4,...\)](#)

1. [hover\(over, out \)](#)

hover 函数主要解决在原始 javascript 中 mouseover 和 mouseout 函数存在的问题，看下面这个示例：



有两个 div(红色区域), 里面分别嵌套了一个 div(黄色区域). HTML 代码如下：

```
<div class="outer" id="outer1">

  Outer 1

  <div class="inner" id="inner1">Inner 1</div>

</div>

<div class="outer" id="outer2">

  Outer 2

  <div class="inner" id="inner2">Inner 2</div>

</div>

<div id="console"></div>
```

绑定如下事件：

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

  function report(event) {
```

```

        $('#console').append('<div>'+event.type+'</div>');

    }

    $(function(){

        $('#outer1')

            .bind('mouseover',report)

            .bind('mouseout',report);

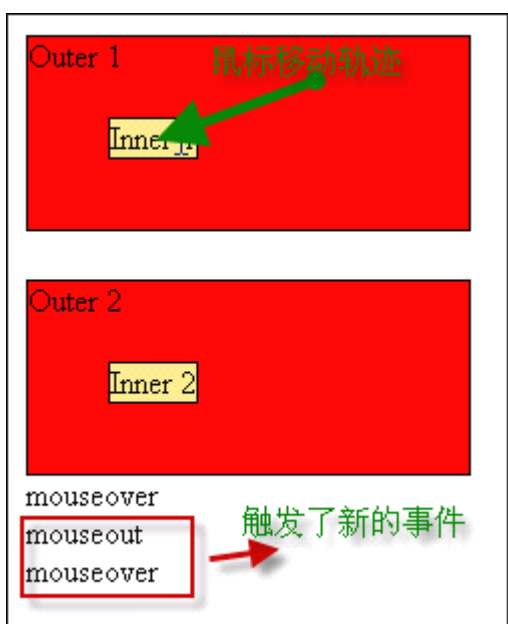
        $('#outer2').hover(report,report);

    });

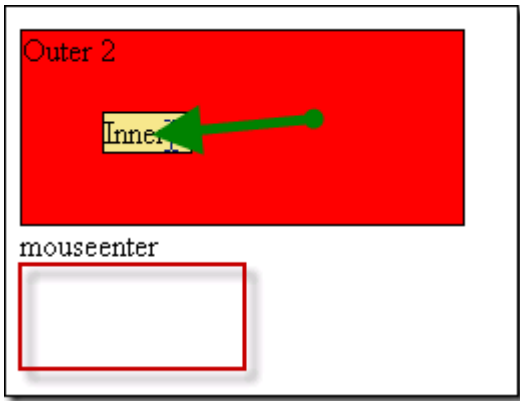
</script>

```

Outer1我们使用了 mouseover 和 mouseout 事件，当鼠标从 Outer1的红色区域移动到黄色区域时，会发现虽然都是在 outer1的内部移动，但是却触发了 mouseout 事件：



很多时候我们不希望出现上图的结果，而是希望只有鼠标在 Outer1内部移动时不触发事件，Outer2使用 Hover()函数实现了这个效果：



注意这里的事件名称进入叫做"mouseenter", 离开叫做"mouseleave", 而不再使用"mouseover"和"mouseleave"事件.

有经验的开发人员会立刻想到在制作弹出菜单时, 经常遇到这个问题: 为弹出菜单设置了 mouseout 事件自动关闭, 但是鼠标在弹出菜单内移动时常常莫名其妙触发 mouseout 事件让菜单关闭. hover()函数帮助我们很好的解决了这个问题.

2. [toggle\(fn, fn2, fn3,fn4,... \)](#)

toggle 函数可以为对象添加 click 事件绑定函数, 但是设置每次点击后依次的调用函数。

如果点击了一个匹配的元素，则触发指定的第一个函数，当再次点击同一元素时，则触发指定的第二个函数，如果有更多函数，则再次触发，直到最后一个。随后的每次点击都重复对这几个函数的轮番调用。

可以使用 unbind("click")来删除。

下面的示例演示如何使用 toggle 函数:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

    <title>toggle example</title>

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


    <script type="text/javascript" src="scripts/jquery-1.3.2-
vsdoc2.js"></script>
```

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

```
$(function()
```

```
{
```

```
$("#li").toggle(
```

```
function()
```

```
{
```

```
$(this).css({ "list-style-type": "disc", "color": "blue" });
```

```
},
```

```
function()
```

```
{
```

```
$(this).css({ "list-style-type": "square", "color": "red"
```

```
});
```

```
},
```

```
function()
```

```
{
```

```
$(this).css({ "list-style-type": "none", "color": "" });
```

```
}
```

```
);
```

```
})
```

```

</script>

</head>

<body>

    <ul>

        <li style="cursor:pointer">click me</li>

    </ul>

</body>

</html>
```

结果是每点击一次"click me"变换一次列表符号和文字颜色.

八.使用 jQuery 事件对象

使用事件自然少不了事件对象. 因为不同浏览器之间事件对象的获取, 以及事件对象的属性都有差异, 导致我们很难跨浏览器使用事件对象.

jQuery 中统一了事件对象, 当绑定事件处理函数时, 会将 jQuery 格式化后的事件对象作为唯一参数传入:

```
$("#testDiv").bind("click", function(event) {  });
```

关于 event 对象的详细说明, 可以参考jQuery 官方文档: <http://docs.jquery.com/Events/jQuery.Event>

jQuery 事件对象将不同浏览器的差异进行了合并, 比如可以在所有浏览器中通过 event.target 属性来获取事件的触发者(在 IE 中使用原生的事件对象, 需要访问 event.srcElement).

下面是 jQuery 事件对象可以在扩浏览器支持的属性:

属性名称	描述	举例
type	事件类型.如果使用一个事件处理函数来处理多个事	\$("#a").click(function(event) {

	件, 可以使用此属性获得事件类型,比如 click.	<pre>alert(event.type); });</pre>
target	获取事件触发者 DOM 对象	<pre>\$("#a[href=http://google.com]").click(function(event) { alert(event.target.href); });</pre>
data	事件调用时传入额外参数.	<pre>\$("#a").each(function(i) { \$(this).bind('click', {index:i}, function(e){ alert('my index is ' + e.data.index); }); });</pre>
relatedTarget	对于鼠标事件, 标示触发事件时离开或者进入的 DOM 元素	<pre>\$("#a").mouseout(function(event) { alert(event.relatedTarget); });</pre>
currentTarget	冒泡前的当前触发事件的 DOM 对象, 等同于 this.	<pre>\$("#p").click(function(event) { alert(event.currentTarget.nodeName); });</pre> <p>结果:P</p>
pageX/Y	鼠标事件中, 事件相对于页面原点的水平/垂直坐标.	<pre>\$("#a").click(function(event) { alert("Current mouse position: " + event.pageX + ", " + event.pageY); });</pre>
result	上一个事件处理函数返回的值	<pre>\$("#p").click(function(event) { return "hey" }); \$("#p").click(function(event) { alert(event.result); });</pre> <p>结果:"hey"</p>

timeStamp	事件发生时的时间戳.	<pre>var last; \$("p").click(function(event) { if(last) alert("time since last event " + event.timeStamp - last); last = event.timeStamp; });</pre>
------------------	------------	--

上面是jQuery 官方文档中提供的 event 对象的属性. 在"jQuery 实战"一书中还提供了下面的多浏览器支持的属性，时间关系我没有尝试每一个属性，大家可以帮忙验证是否在所有浏览器下可用:

属性名称	描述	举例
altKey	Alt 键是否被按下. 按下返回 true	
ctrlKey	ctrl 键是否被按下, 按下返回 true	
metaKey	Meta 键是否被按下, 按下返回 true. meta 键就是 PC 机器的 Ctrl 键, 或者 Mac 机器上面的 Command 键	
shiftKey	Shift 键是否被按下, 按下返回 true	
keyCode	对于 keyup 和 keydown 事件返回被按下的键. 不区分大小写, a 和 A 都返回65. 对于 keypress 事件请使用 which 属性, 因为 which 属性跨浏览时依然可靠.	
which	对于键盘事件, 返回触发事件的键的数字编码. 对于鼠标事件, 返回鼠标按键号(1左,2中,3右).	
screenX/Y	对于鼠标事件, 获取事件相对于屏幕原点的水平/垂直坐标	

事件对象除了拥有属性，还拥有事件。有一些是一定会用到的事件比如取消冒泡 stopPropagation() 等.下面是jQuery 事件对象的函数列表:

名称	说明	举例
preventDefault()	取消可能引起任何语意操作的事件. 比如 <a>元素的 href 链接加	<pre>\$("a").click(function(event){ event.preventDefault();</pre>

	载，表单提交以及 click 引起复选框的状态切换.	// do something });
isDefaultPrevented()	是否调用过 preventDefault() 方法	\$("#a").click(function(event){ alert(event.isDefaultPrevented()); event.preventDefault(); alert(event.isDefaultPrevented()); });
stopPropagation()	取消事件冒泡	\$("#p").click(function(event){ event.stopPropagation(); // do something });
isPropagationStopped()	是否调用过 stopPropagation() 方法	\$("#p").click(function(event){ alert(event.isPropagationStopped()); event.stopPropagation(); alert(event.isPropagationStopped()); });
stopImmediatePropagation()	取消执行其他的事件 处理函数并取消事件 冒泡. 如果同一个事件绑定 了多个事件处理函数, 在其中一个事件处理 函数中调用此方法后 将不会继续调用其他 的事件处理函数.	\$("#p").click(function(event){ event.stopImmediatePropagation(); }); \$("#p").click(function(event){ // This function won't be executed });
isImmediatePropagationStopped()	是否调用过 stopImmediatePropagation() 方法	\$("#p").click(function(event){ alert(event.isImmediatePropagation Stopped()); event.stopImmediatePropagation();

		<pre>alert(event.isImmediatePropagationStopped()); });</pre>
--	--	---

这些函数中 `stopPropagation()` 是我们最长用的也是一定会用到的函数。相当于操作原始 event 对象的 `event.cancelBubble=true` 来取消冒泡。

九. 总结

事件是 javascript 的灵魂，我花了很久写这篇文章,翻译jQuery 官方的 API 文档。列表中的很多例子直接从官网上摘抄的，有些列表中的方法我也没有用过，所以如果大家发现问题请及时通知我修改。

接下来的文章将分别讲解 Ajax 和动画效果。最后通过讲解两个我修改的 jQuery 插件来学习 jQuery 的插件开发。

Tag 标签: [jQuery,jQuery 教程](#)

从零开始学习 jQuery (六) AJAX 快餐

一.摘要

本系列文章将带您进入 jQuery 的精彩世界，其中有很多作者具体的使用经验和解决方案， 即使你会使用 jQuery 也能在阅读中发现些许秘籍.

本篇文章讲解如何使用 jQuery 方便快捷的实现 Ajax 功能.统一所有开发人员使用 Ajax 的方式.

二.前言

Ajax 让用户页面丰富起来，增强了用户体验. 使用 Ajax 是所有 Web 开发的必修课. 虽然 Ajax 技术并不复杂，但是实现方式还是会因为每个开发人员的而有所差异.jQuery 提供了一系列 Ajax 函数来帮助我们统一这种差异，并且让调用 Ajax 更加简单.

三.原始 Ajax 与 jQuery 中的 Ajax

首先通过实例，来看一下 jQuery 实现 Ajax 有多简单. 下面是一个使用原始 Ajax 的示例:

```
<!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>

<title>jQuery Ajax</title>

<script type="text/javascript">

$(function()

{

var xhr = new AjaxXmlHttpRequest();
```

```
$("#btnAjaxOld").click(function(event)

{

    var xhr = new AjaxXmlHttpRequest();

    xhr.onreadystatechange = function()

    {

        if (xhr.readyState == 4)

        {

            document.getElementById("divResult").innerHTML

= xhr.responseText;

        }

    }

    xhr.open("GET",

"data/AjaxGetCityInfo.aspx?resultType=html", true);

    xhr.send(null);

});

})
```

//跨浏览器获取 XMLHttpRequest 对象

```
function AjaxXmlHttpRequest()

{
```

```
var xmlHttp;

try

{

    // Firefox, Opera 8.0+, Safari

    xmlHttp = new XMLHttpRequest();

}

catch (e)

{

    // Internet Explorer

    try

    {

        xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");

    }

    catch (e)

    {

        try

        {
```

```
        xmlhttp = new
ActiveXObject("Microsoft.XMLHTTP");

    }

    catch (e)

    {

        alert("您的浏览器不支持 AJAX! ");

        return false;

    }

}

return xmlhttp;

}

</script>

</head>

<body>

    <button id="btnAjaxOld">原始 Ajax 调用</button><br />

    <br />

    <div id="divResult"></div>
```

```
</body>
```

```
</html>
```

上面的实例中, data/AjaxGetCityInfo.aspx?resultType=html 地址会返回一段 HTML 代码.

使用原始 Ajax, 我们需要做较多的事情, 比如创建 XMLHttpRequest 对象, 判断请求状态, 编写回调函数等.

而用 jQuery 的 Load 方法, 只需要一句话:

```
$("#divResult").load("data/AjaxGetCityInfo.aspx", { "resultType": "html" });
```

曾经我是一个原始 Ajax 的绝对拥护者, 甚至摒弃微软的 Asp.net Ajax, 因为我想要最高的代码灵活度. 使用原始 Ajax 让我感觉完成自己的工作更加轻松, 即使多写了一些代码. 但是当我去翻看别人的 Ajax 代码并且尝试修改的时候, 我改变了我的看法--我们的代码到处分布着创建 XMLHttpRequest 方法的函数, 或者某些 Ajax 程序逻辑性和结构性很差, 很难看懂.

我们可以将通用方法放到一个 js 文件中, 然后告诉大家"嘿伙伴们, 都来用这个 js 中的方法". 但是在某些时候有些新来的外包人员并不知道有这个 js 文件的存在. 而且其实这个通用的 js 就是一个公共的脚本类库, 我相信没有人会觉得自己开发一个类库会比 jQuery 更好!

所以我放弃了制造轮子的计划, 大家都使用 jQuery 编写 Ajax 相关的方法就可以解决各种差异性问题, 并且让工作更有效率.

现在只是用 jQuery 的 Ajax 函数, 我的页面变得简洁了:

```
<!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>
```

```
<title>jQuery Ajax</title>
```

```
<script type="text/javascript" src="scripts/jquery-1.3.2-  
vsdoc2.js"></script>
```

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

    $(function()

    {

        $("#btnAjaxJquery").click(function(event)

        {

            $("#divResult").load("data/AjaxGetCityInfo.aspx",

{ "resultType": "html" });

        });

    })

</script>

</head>

<body>

    <button id="btnAjaxJquery">使用 jQuery 的 load 方法</button>

    <br />

    <div id="divResult"></div>

</body>

</html>
```

四.jQuery Ajax 详解

jQuery 提供了几个用于发送 Ajax 请求的函数。其中最核心也是最复杂的是 [jQuery.ajax\(options\)](#),

所有的其他 Ajax 函数都是它的一个简化调用. 当我们想要完全控制 Ajax 时可以使用此结果, 否则还是使用简化方法如 `get`, `post`, `load` 等更加方便. 所以 [jQuery.ajax\(options\)](#) 方法放到最后一个介绍. 先来介绍最简单的 `load` 方法:

1. [load\(url, \[data\], \[callback\]\)](#)

Returns: jQuery 包装集

说明:

`load` 方法能够载入远程 HTML 文件代码并插入至 DOM 中。

默认使用 GET 方式, 如果传递了 `data` 参数则使用 Post 方式.

- 传递附加参数时自动转换为 POST 方式. jQuery 1.2 中, 可以指定选择符, 来筛选载入的 HTML 文档, DOM 中将仅插入筛选出的 HTML 代码. 语法形如 "`url #some > selector`", 默认的选择器是 "`body > *`".

讲解:

`load` 是最简单的 Ajax 函数, 但是使用具有局限性:

- 1 它主要用于直接返回 HTML 的 Ajax 接口
- 2 `load` 是一个 jQuery 包装集方法, 需要在 jQuery 包装集上调用, 并且会将返回的 HTML 加载到对象中, 即使设置了回调函数也还是会加载.

不过不可否认 `load` 接口设计巧妙并且使用简单. 下面通过示例来演示 Load 接口的使用:

```
<!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>
```

```
<title>jQuery Ajax - Load</title>
```

```
<script type="text/javascript" src="../../scripts/jquery-1.3.2-  
vsdoc2.js"></script>
```

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

```
$(function()

{

    $("#btnAjaxGet").click(function(event)

    {

        //发送 Get 请求

        $("#divResult").load("../data/AjaxGetMethod.aspx?param=btnAjaxGet_click" +
        "&timestamp=" + (new Date()).getTime());

    });

    $("#btnAjaxPost").click(function(event)

    {

        //发送 Post 请求

        $("#divResult").load("../data/AjaxGetMethod.aspx",
        { "param": "btnAjaxPost_click" });

    });

    $("#btnAjaxCallBack").click(function(event)

    {

        //发送 Post 请求, 返回后执行回调函数.
```

```

        $("#divResult").load("../data/AjaxGetMethod.aspx",
    {   "param":   "btnAjaxCallBack_click"   },   function(responseText,   textStatus,
XMLHttpRequest)

    {

        responseText   =   "   Add   in   the   CallBack   Function!

<br/>"   +   responseText

        $("#divResult").html(responseText);   //   或   者   :

$(this).html(responseText);

    });

});

$("#btnAjaxFiltHtml").click(function(event)

{

    //发送 Get 请求, 从结果中过滤掉 "鞍山" 这一项

    $("#divResult").load("../data/AjaxGetCityInfo.aspx?resultType=html"   +

"&timestamp="   +   (new   Date()).getTime()   +   "   ul>li:not(:contains('鞍山'))");

    });

})

</script>

```

```
</head>

<body>

    <button id="btnAjaxGet">使用 Load 执行 Get 请求</button><br />

    <button id="btnAjaxPost">使用 Load 执行 Post 请求</button><br />

    <button id="btnAjaxCallBack"> 使用 带有回调函数的 Load 方法
</button><br />

    <button id="btnAjaxFiltHtml"> 使用 selector 过滤返回的 HTML 内容
</button>

    <br />

    <div id="divResult"></div>

</body>

</html>
```

上面的示例演示了如何使用 Load 方法.

提示:我们要时刻注意浏览器缓存, 当使用 **GET** 方式时要添加时间戳参数 (**new Date().getTime()**) 来保证每次发送的 **URL** 不同, 可以避免浏览器缓存.

提示: 当在 **url** 参数后面添加了一个空格, 比如" "的时候, 会出现"无法识别符号"的错误, 请求还是能正常发送. 但是无法加载 **HTML** 到 **DOM**. 删除后问题解决.

2.jQuery.get(url, [data], [callback], [type])

Returns: XMLHttpRequest

说明:

通过远程 HTTP GET 请求载入信息。

这是一个简单的 GET 请求功能以取代复杂 \$.ajax 。请求成功时可调用回调函数。如果需要在出错时执行函数，请使用 \$.ajax。

讲解:

此函数发送 Get 请求, 参数可以直接在 url 中拼接, 比如:

```
$.get("../data/AjaxGetMethod.aspx?param=btnAjaxGet_click");
```

或者通过 data 参数传递:

```
$.get("../data/AjaxGetMethod.aspx", { "param": "btnAjaxGet2_click" });
```

两种方式效果相同, data 参数会自动添加到请求的 url 中

如果 url 中的某个参数, 又通过 data 参数传递, 不会自动合并相同名称的参数.

回调函数的签名如下:

```
function (data, textStatus) {  
  
    // data could be xmlDoc, jsonObj, html, text, etc...  
  
    this; // the options for this ajax request  
  
}
```

其中 data 是返回的数据, textStatus 表示状态码, 可能是如下值:

"timeout", "error", "notmodified", "success", "parsererror"

在回调函数中的 this 是获取 options 对象的引用. 有关 options 的各种说明, 请参见:

<http://docs.jquery.com/Ajax/jQuery.ajax#options>

type 参数是指 data 数据的类型, 可能是下面的值:

"xml", "html", "script", "json", "jsonp", "text".

默认为**"html"**.

[**jQuery.getJSON\(url, \[data\], \[callback\]\)**](#) 方法就相当于 jQuery.get(url, [data], [callback], **"json"**)

3. [jQuery.getJSON\(url, \[data\], \[callback\] \)](#)

Returns: XMLHttpRequest

相当于: `jQuery.get(url, [data],[callback], "json")`

说明:

通过 HTTP GET 请求载入 JSON 数据。

在 jQuery 1.2 中，您可以通过使用 [JSONP](#) 形式的回调函数来加载其他网域的 JSON 数据，如 "myurl?callback=?"。jQuery 将自动替换 ? 为正确的函数名，以执行回调函数。

注意：此行以后的代码将在这个回调函数执行前执行。

讲解:

getJSON 函数仅仅将 get 函数的 type 参数设置为"JSON"而已。在回调函数中获取的数据已经是按照 JSON 格式解析后的对象了：

```
$.getJSON("../data/AjaxGetCityInfo.aspx",    {    "resultType":    "json"    },  
  
function(data, textStatus)  
  
{  
  
    alert(data.length);  
  
    alert(data[0].CityName);  
  
});
```

服务器端返回的字符串如下：

```
[{"pkid":"0997","ProvinceId":"XJ","CityName":"阿克苏",  
"CityNameEn":"Akesu","PostCode":"843000","isHotCity":false},  
  
{"pkid":"0412","ProvinceId":"LN","CityName":"鞍山",  
"CityNameEn":"Anshan","PostCode":"114000","isHotCity":false}]
```

示例中我返回的饿是一个数组, 使用 `data.length` 可以获取数组的元素个数, `data[0]`访问第一个元素, `data[0].CityName` 访问第一个元素的 `CityName` 属性.

4.jQuery.getScript(url, [callback])

Returns: XMLHttpRequest

相当于: `jQuery.get(url, null, [callback], "script")`

说明:

通过 HTTP GET 请求载入并执行一个 JavaScript 文件。

jQuery 1.2 版本之前, `getScript` 只能调用同域 JS 文件。 1.2中, 您可以跨域调用 JavaScript 文件。注意: Safari 2 或更早的版本不能在全局作用域中同步执行脚本。如果通过 `getScript` 加入脚本, 请加入延时函数。

讲解:

以前我使用 `dojo` 类库时官方默认的文件不支持跨域最后导致我放弃使用 `dojo`(虽然在网上找到了可以跨域的版本, 但是感觉不够完美)。 所以我特别对这个函数的核心实现和使用做了研究。

首先了解此函数的 jQuery 内部实现, 仍然使用 `get` 函数, jQuery 所有的 Ajax 函数包括 `get` 最后都是用的是 `jQuery.ajax()`, `getScript` 将传入值为 `"script"` 的 `type` 参数, 最后在 Ajax 函数中对 `type` 为 `script` 的请求做了如下处理:

```
var head = document.getElementsByTagName("head")[0];
```

```
var script = document.createElement("script");
```

```
script.src = s.url;
```

上面的代码动态建立了一个 `script` 语句块, 并且将其加入到 `head` 中:

```
head.appendChild(script);
```

当脚本加载完毕后, 再从 `head` 中删除:

```
// Handle Script loading
```

```
if ( !jsonp ) {
```

```
    var done = false;
```

```
// Attach handlers for all browsers

script.onload = script.onreadystatechange = function(){

    if ( !done && (!this.readyState ||

        this.readyState == "loaded" || this.readyState

== "complete") ) {

        done = true;

        success();

        complete();

    }

    // Handle memory leak in IE

    script.onload = script.onreadystatechange = null;

    head.removeChild( script );

}

};

}
```

我主要测试了此函数的跨域访问和多浏览器支持.下面是结果:

	IE6	FireFox	注意事项
非跨域引用 js	通过	通过	回调函数中的 data 和 textStatus 均可用
跨域引用 js	通过	通过	回调函数中的 data 和 textStatus 均为 undefined

下面是我关键的测试语句, 也用来演示如何使用 getScript 函数:


```
$("#btnAjaxGetScript").click(function(event)

{

    $.getScript("../scripts/getScript.js", function(data, textStatus)

    {

        alert(data);

        alert(textStatus);

        alert(this.url);

    });

});
```

```
$("#btnAjaxGetScriptCross").click(function(event)

{

    $.getScript("http://resource.elong.com/getScript.js",

function(data, textStatus)

    {

        alert(data);

        alert(textStatus);

        alert(this.url);

    });

});
```

```
});
```

5. [jQuery.post\(url, \[data\], \[callback\], \[type\] \)](#)

Returns: XMLHttpRequest

说明:

通过远程 HTTP POST 请求载入信息。

这是一个简单的 POST 请求功能以取代复杂 \$.ajax 。请求成功时可调用回调函数。如果需要在出错时执行函数，请使用 \$.ajax。

讲解:

具体用法和 get 相同, 只是提交方式由"GET"改为"POST".

6. [jQuery.ajax\(options \)](#)

Returns: XMLHttpRequest

说明:

通过 HTTP 请求加载远程数据。

jQuery 底层 AJAX 实现。简单易用的高层实现见 \$.get, \$.post 等。

\$.ajax() 返回其创建的 XMLHttpRequest 对象。大多数情况下你无需直接操作该对象，但特殊情况下可用于手动终止请求。

\$.ajax() 只有一个参数：参数 key/value 对象，包含各配置及回调函数信息。详细参数选项见下。

注意: 如果你指定了 dataType 选项，请确保服务器返回正确的 MIME 信息，(如 xml 返回 "text/xml")。错误的 MIME 类型可能导致不可预知的错误。见 [Specifying the Data Type for AJAX Requests](#) 。

注意: 如果 dataType 设置为 "script", 那么所有的远程(不在同一域名下)的 POST 请求都将转化为 GET 请求。(因为将使用 DOM 的 script 标签来加载)

jQuery 1.2 中，您可以跨域加载 JSON 数据，使用时需将数据类型设置为 [JSONP](#)。使用 [JSONP](#) 形式调用函数时，如 "myurl?callback=?" jQuery 将自动替换 ? 为正确的函数名，以执行回调函数。数据类型设置为 "jsonp" 时，jQuery 将自动调用回调函数。

讲解:

这是 jQuery 中 Ajax 的核心函数，上面所有的发送 Ajax 请求的函数内部最后都会调用此函数。options 参数支持很多参数，使用这些参数可以完全控制 ajax 请求。在 Ajax 回调函数中的 this 对象也是 options 对象。

因为平时使用最多的还是简化了的 get 和 post 函数, 所以在此不对 options 参数做详细讲解了. options 参数文档请见:

<http://docs.jquery.com/Ajax/jQuery.ajax#options>

五.Ajax 相关函数.

jQuery 提供了一些相关函数能够辅助 Ajax 函数.

1. [jQuery.ajaxSetup\(options \)](#)

无返回值

说明:

设置全局 AJAX 默认 options 选项。

讲解:

有时我们的希望设置页面上所有 Ajax 属性的默认行为.那么就可以使用此函数设置 options 选项, 此后所有的 Ajax 请求的默认 options 将被更改.

options 是一个对象, 可以设置的属性请此连接: <http://docs.jquery.com/Ajax/jQuery.ajax#toptions>

比如在页面加载时, 我使用下面的代码设置 Ajax 的默认 option 选项:

```
$.ajaxSetup({

    url: "../data/AjaxGetMethod.aspx",

    data: { "param": "ziquu.zhang" },

    global: false,

    type: "POST",

    success: function(data, textStatus)

{ $("#divResult").html(data); }

});
```

上面的代码设置了一个 Ajax 请求需要的基本数据: 请求 url, 参数, 请求类型, 成功后的回调函数.

此后我们可以使用无参数的 get(), post()或者 ajax()方法发送 ajax 请求.完整的示例代码如下:

```
<!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>
```

```
<title>jQuery Ajax - Load</title>
```

```
<script type="text/javascript" src="../../scripts/jquery-1.3.2-  
vsdoc2.js"></script>
```

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

```
$(document).ready(function()
```

```
{
```

```
$.ajaxSetup({
```

```
url: "../../data/AjaxGetMethod.aspx",
```

```
data: { "param": "ziquu.zhang" },
```

```
global: false,
```

```
type: "POST",
```

```
success: function(data, textStatus)
```

```
{ $("#divResult").html(data); }
```

```
});
```

```
$("#btnAjax").click(function(event) { $.ajax(); });
```

```
$("#btnGet").click(function(event) { $.get(); });
```

```
$("#btnPost").click(function(event) { $.post(); });
```

```
$("#btnGet2").click(function(event)
{ $.get("../data/AjaxGetMethod.aspx",{ "param": "other" }); });

});
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<button id="btnAjax">不传递参数调用 ajax()方法</button><br />
```

```
<button id="btnGet">不传递参数调用 get()方法</button><br />
```

```
<button id="btnPost">不传递参数调用 post()方法</button><br />
```

```
<button id="btnGet2">传递参数调用 get()方法，使用全局的默认回调函数
```

```
</button><br />
```

```
<br />
```

```
<div id="divResult"></div>

</body>

</html>
```

注意当使用 `get()` 或者 `post()` 方法时，除了 `type` 参数将被重写为 "GET" 或者 "POST" 外，其他参数只要不传递都是使用默认的全局 `option`。如果传递了某一个选项，比如最后一个按钮传递了 `url` 和参数，则本次调用会以传递的选项为准。没有传递的选项比如回调函数还是会使用全局 `option` 设置值。

2.[serialize\(\)](#)

Returns: [String](#)

说明:

序列表格内容为字符串，用于 Ajax 请求。

序列化最常用在将表单数据发送到服务器端时。被序列化后的数据是标准格式，可以被几乎所有的而服务器端支持。

为了尽可能正常工作，要求被序列化的表单字段都有 `name` 属性，只有一个 `eid` 是无法工作的。

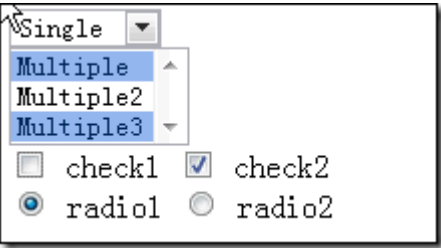
像这样写 `name` 属性:

```
<input id="email" name="email" type="text" />
```

讲解:

`serialize()` 函数将要发送给服务器的 `form` 中的表单对象拼接成一个字符串。便于我们使用 Ajax 发送时获取表单数据。这和一个 `Form` 按照 `Get` 方式提交时，自动将表单对象的名/值放到 `url` 上提交差不多。

比如这样一个表单:



生成的字符串为: `single=Single¶m=Multiple¶m=Multiple3&check=check2&radio=radio1`

提示: 代码见 [chapter6\7-serialize.htm](#)

3.[serializeArray\(\)](#)

Returns: [Array<Object>](#)

说明:

序列化表格元素 (类似 '.serialize()' 方法) 返回 JSON 数据结构数据。

注意, 此方法返回的是 JSON 对象而非 JSON 字符串。需要使用插件或者第三方库进行字符串化操作。

讲解:

看说明文档让我有所失望, 使用此函数获取到的是 JSON 对象, 但是 jQuery 中没有提供将 JSON 对象转化为 JSON 字符串的方法。

在 JSON 官网上没有找到合适的 JSON 编译器, 最后选用了 jquery.json 这个 jQuery 插件:

<http://code.google.com/p/jquery-json/>

使用起来异常简单:

```
var thing = {plugin: 'jquery-json', version: 1.3};

var encoded = $.toJSON(thing);           //'{"plugin": "jquery-json",
"version": 1.3}'

var name = $.evalJSON(encoded).plugin;    //"jquery-json"

var version = $.evalJSON(encoded).version; // 1.3
```

使用 [serializeArray\(\)](#) 再配合 **\$.toJSON** 方法, 我们可以很方便的获取表单对象的 JSON, 并且转换为 JSON 字符串:

```
$("#results").html( $.toJSON( $("form").serializeArray() ));
```

结果为:

```
[{"name": "single", "value": "Single"}, {"name": "param", "value":
"Multiple"}, {"name": "param", "value": "Multiple3"}, {"name": "check",
"value": "check2"}, {"name": "radio", "value": "radio1"}]
```

六.全局 Ajax 事件

在 [jQuery.ajaxSetup\(options \)](#) 中的 options 参数属性中, 有一个 global 属性:

global

类型:布尔值

默认值: true

说明:是否触发全局的 Ajax 事件.

这个属性用来设置是否触发全局的 Ajax 事件. 全局 Ajax 事件是一系列伴随 Ajax 请求发生的事件.主要有如下事件:

名称	说明
ajaxComplete(callback)	AJAX 请求完成时执行函数
ajaxError(callback)	AJAX 请求发生错误时执行函数
ajaxSend(callback)	AJAX 请求发送前执行函数
ajaxStart(callback)	AJAX 请求开始时执行函数
ajaxStop(callback)	AJAX 请求结束时执行函数
ajaxSuccess(callback)	AJAX 请求成功时执行函数

用一个示例讲解各个事件的触发顺序:

```
<!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>

    <title>jQuery Ajax - AjaxEvent</title>

    <script type="text/javascript" src="../../scripts/jquery-
1.3.2.min.js"></script>

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



```
$(document).ready(function()

{

    $("#btnAjax").bind("click", function(event)

    {

        $.get("../data/AjaxGetMethod.aspx");

    })

    $("#divResult").ajaxComplete(function(evt, request, settings)
{ $(this).append('<div>ajaxComplete</div>'); })

    $("#divResult").ajaxError(function(evt, request, settings)
{ $(this).append('<div>ajaxError</div>'); })

    $("#divResult").ajaxSend(function(evt, request, settings)
{ $(this).append('<div>ajaxSend</div>'); })

    $("#divResult").ajaxStart(function()
{ $(this).append('<div>ajaxStart</div>'); })

    $("#divResult").ajaxStop(function()
{ $(this).append('<div>ajaxStop</div>'); })

    $("#divResult").ajaxSuccess(function(evt, request, settings)
{ $(this).append('<div>ajaxSuccess</div>'); })
```

```
});
```

```
</script>
```

```
</head>
```

```
<body>
```

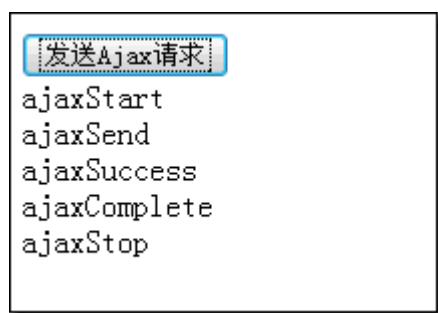
```
<br /><button id="btnAjax">发送 Ajax 请求</button><br/>
```

```
<div id="divResult"></div>
```

```
</body>
```

```
</html>
```

结果如图:



我们可以通过将默认 options 的 global 属性设置为 false 来取消全局 Ajax 事件的触发.

七.注意事项

如果在 Get 请求发送的 url 中有两个同名参数, 比如两个 param 参数:

http://localhost/AjaxGetMethod.aspx?param=Multiple¶m=Multiple3

使用服务器端方法获取 param 参数:

```
if (!String.IsNullOrEmpty(HttpContext.Current.Request["Param"]))
```

```
{  
  
    param = HttpContext.Current.Request["Param"];  
  
}
```

此时获取到得 param 是一个用","分隔多个值的字符串:

Multiple,Multiple3

八.总结

本文介绍如何使用 jquery 实现 Ajax 功能. 用于发送 Ajax 请求的相关函数如 load, get, getJSON 和 post 这些渐变 Ajax 方法, 对于核心的 ajax 方法没有过多介绍, 主要是通过配置复杂的参数实现完全控制 Ajax 请求. 另外讲解了 ajax 的辅助函数比如用于序列化表单对象为字符串的 serialize()方法, 用于将表单对象序列化为 JSON 对象的 serializeArray()方法. 这些在使用脚本获取数据实现与服务器端交互是很有用, JSON 格式的数据在处理大对象编程时将我们从混乱的属性字符串中解放出来.

jQuery 还提供录入全局 ajax 事件这一个特殊的事件, 并且可以在一个对象上设置这些事件, 在发送 Ajax 请求的各个生命周期上会调用这些事件, 可以通过修改默认的 options 对象的 global 属性打开或关闭全局事件.

目前本系列文章在加紧创作阶段. 所以代码和文章示例都没有来得及重新整理. 下面是本章的代码下载, 但是含有所有以前未整理的示例,请大家下载后看 chapter6文件夹, 里面是本章的所有示例:

<http://files.cnblogs.com/zhangziqu/Code-jQueryStudy.rar>

Tag 标签: [jQuery](#),[jQuery 教程](#)

从零开始学习 JQuery (七) JQuery 动画-让页面动起来!

一.摘要

本系列文章将带您进入 JQuery 的精彩世界, 其中有很多作者具体的使用经验和解决方案, 即使你会使用 JQuery 也能在阅读中发现些许秘籍.

开发人员一直痛疼做动画. 但是有了 JQuery 你会瞬间成为别人(那些不知道 JQuery 的人)眼里的动画高手! 本文将介绍 JQuery 的动画相关函数.原来做动画如此简单!

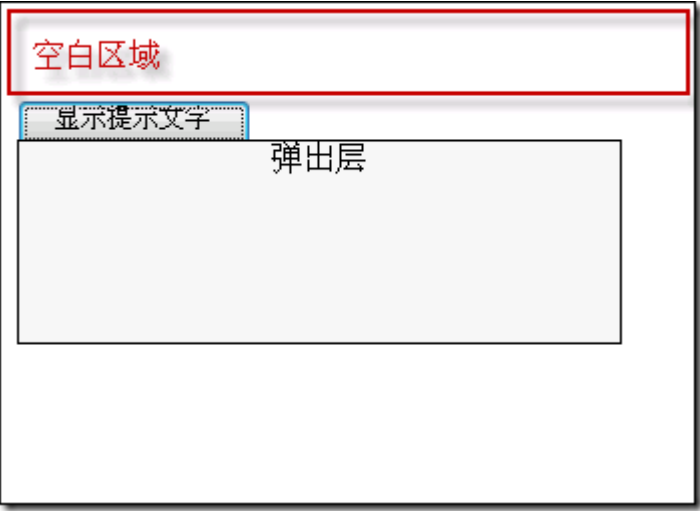
二.前言

本系列文章的实例都是针对某一个技术细节的, 因为我们要学习的是基础知识, 虽然总有人希望要复杂一些的应用示例, 但是我想还是让我们先把基础打牢, 有了扎实的基础凭借每个人的智慧一定能创造出更多更好的应用.

就在写这篇文章的前几天, 还有不止一个同事在为了"弹出层"效果而犯愁. 但是以后再面对这样的功能看过本篇文章的每一个人都可以开心的微笑了. JQuery, make work easy !

三.从实例开始

做 web 程序经常要使用弹出层, 比如单击文字或按钮显示一段提示文字等. 假设有如下需求:



- 单击图中的"显示提示文字"按钮, 在按钮的下方显示一个弹出层.
- 单击任何空白区域或者弹出层,弹出层消失.

用原始 javascript 我们也完全可以完成这个工作. 有以下几点注意事项:

1. 弹出层的位置需要动态计算. 因为触发弹出事件的对象可能出现在页面的任何位置, 比如截图中的位置.
2. 为 document 绑定单击是关闭弹出层的函数, 要使用多播委托, 否则可能冲掉其他人在 document 绑定的函数.
3. 为 document 绑定了关闭函数后, 需要在显示函数中取消事件冒泡, 否则弹出层将显示后立刻

关闭.

用 jQuery, 我们可以轻松地实现此实例:

```
<!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>
```

```
<title>jQuery - Start Animation</title>
```

```
<script type="text/javascript" src="../../scripts/jquery-1.3.2-  
vsdoc2.js"></script>
```

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

```
$(document).ready(function()
```

```
{
```

```
//动画速度
```

```
var speed = 500;
```

```
//绑定事件处理
```

```
$("#btnShow").click(function(event)
```

```
{
```

```
        //取消事件冒泡

        event.stopPropagation();

        //设置弹出层位置

        var offset = $(event.target).offset();

        $("#divPop").css({ top: offset.top + $(event.target).height()
+ "px", left: offset.left });

        //动画显示

        $("#divPop").show(speed);

    });

    //单击空白区域隐藏弹出层

    $(document).click(function(event) { $("#divPop").hide(speed) });

    //单击弹出层则自身隐藏

    $("#divPop").click(function(event) { $("#divPop").hide(speed)

});

});

</script>

</head>

<body>
```

```
<div>
```

```
<br /><br /><br />
```

```
<button id="btnShow">显示提示文字</button>
```

```
</div>
```

```
<!-- 弹出层 -->
```

```
<div id="divPop" style="background-color: #f0f0f0; border: solid 1px  
#000000; position: absolute; display:none;
```

```
width: 300px; height: 100px;">
```

```
<div style="text-align: center;">弹出层</div>
```

```
</div>
```

```
</body>
```

```
</html>
```

除了实现了基本的显示和隐藏功能, 现在显示和隐藏弹出层是渐变动画效果! jQuery 的动画函数如此简单, 第一次我在项目中使用时带给了我意外的惊喜. 曾经我一直为跨浏览器计算位置头痛, 但是通过 jQuery 的 `offset()` 函数和 `height()` 函数, 可以精确的计算弹出层的位置. 这些函数是封装好且跨浏览器的. 需要注意要在设置弹出层位置属性时, 加上 "px", 否则在 Firefox 下容易出现问題.

jQuery 的动画函数主要分为三类:

- 1 基本动画函数: 既有透明度渐变, 又有滑动效果. 是最常用的动画效果函数.
- 2 滑动动画函数: 仅使用滑动渐变效果.
- 3 淡入淡出动画函数: 仅使用透明度渐变效果.

这三类动画函数效果各不相同, 用法基本一致. 大家可以自己尝试.

另外也许上面的三类函数效果都不是我们想要的, 那么 jQuery 也提供了自定义动画函数, 将控制权放在我们手里让我们自己定义动画效果.

下面对三类内置动画函数和自定义动画函数分别讲解.

四. 基本动画函数

上例中使用的 show()和 hide()是我们使用最多的基本动画函数.

下面是 jQuery 的基本动画函数:

基本动画函数 **Basics**

名称	说明	举例
show()	<p>显示隐藏的匹配元素。</p> <p>这个就是 'show(speed, [callback])' 无动画的版本。如果选择的元素是可见的，这个方法将不会改变任何东西。无论这个元素是通过 hide()方法隐藏的还是在 CSS 里设置了 display:none;，这个方法都将有效。</p>	<p>显示所有段落:</p> <p><code>\$("p").show()</code></p>
show(speed, [callback])	<p>以优雅的动画显示所有匹配的元素，并在显示完成后可选地触发一个回调函数。</p> <p>可以根据指定的速度动态地改变每个匹配元素的高度、宽度和不透明度。在 jQuery 1.3 中，padding 和 margin 也会有动画，效果更流畅。</p>	<p>用缓慢的动画将隐藏的段落显示出来，历时600毫秒:</p> <p><code>\$("p").show(600);</code></p>
hide()	<p>隐藏显示的元素</p> <p>这个就是 'hide(speed, [callback])' 的无动画版。如果选择的元素是隐藏的，这个方法将不会改变任何东西。</p>	<p>隐藏所有段落:</p> <p><code>\$("p").hide()</code></p>
hide(speed, [callback])	<p>以优雅的动画隐藏所有匹配的元素，并在显示完成后可选地触发一个回调函数。</p> <p>可以根据指定的速度动态地改变每个匹配元素的高度、宽度和不透明度。在 jQuery 1.3 中，padding 和 margin 也会有动画，效果更流畅。</p>	<p>用600毫秒的时间将段落缓慢的隐藏:</p> <p><code>\$("p").hide("slow");</code></p>
toggle()	<p>切换元素的可见状态。</p>	<p>切换所有段落的可见状态:</p>

	如果元素是可见的，切换为隐藏的；如果元素是隐藏的，切换为可见的。	<code>\$("#p").toggle()</code>
<u>toggle(switch)</u>	<p>根据 switch 参数切换元素的可见状态（ture 为可见，false 为隐藏）。</p> <p>如果 switch 设为 true，则调用 show() 方法来显示匹配的元素，如果 switch 设为 false 则调用 hide() 来隐藏元素。</p>	<p>切换所有段落的可见状态：</p> <pre>var flip = 0; \$("#button").click(function () { \$("#p").toggle(flip++ % 2 == 0); });</pre>
<u>toggle(speed, [callback])</u>	<p>以优雅的动画切换所有匹配的元素，并在显示完成后可选地触发一个回调函数。</p> <p>可以根据指定的速度动态地改变每个匹配元素的高度、宽度和不透明度。在 jQuery 1.3 中，padding 和 margin 也会有动画，效果更流畅。</p>	<p>用200毫秒将段落迅速切换显示状态，之后弹出一个对话框：</p> <pre>\$("#p").toggle("fast",function(){ alert("Animation Done."); });</pre>

1. 使用基本动画函数

基本的动画函数主要分为 show, hide 和 toggle 三个。都提供了无参数的版本，表示不适用动画切换元素的显示状态：

```
$("#divPop").show();
```

```
$("#divPop").hide();
```

```
$("#divPop").toggle();
```

都提供了两个参数的重载，因为回调函数可以省略，所以可以像开篇实例中使用的，传入一个数值作为唯一参数，则会在参数规定的时间内用动画效果显示/隐藏元素：

```
$("#divPop").show(200);
```

```
$("#divPop").hide("fast");
```

```
$("#divPop").toggle("slow");
```

如果传递了200, 表示图层会在200毫秒内通过渐变的形式显示出来. speed 参数可以使用三种预定速度之一的字符串("slow", "normal", or "fast")或表示动画时长的毫秒数值(如: 1000).

三个函数都可以传入回调函数 callback,签名如下:

```
function callback() {  
  
    this; // dom element  
  
}
```

在回调函数中的 this 是执行此函数的 DOM 对象. 会在动画结束时执行.

2. 使用 toggle 函数

toggle 函数是功能更强大的函数, 可以切换元素的可见状态. 我们经常遇到需要使用 toggle 的情况. 比如希望一段文字第一次单击显示弹出层, 第二次单击隐藏弹出层.

我们将开篇实例稍作修改即可实现这个效果:

```
<!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>  
  
    <title>jQuery Animation - Toggle </title>  
  
  
    <script type="text/javascript" src="../../scripts/jquery-1.3.2-  
vsdoc2.js"></script>  
  
  
    <script type="text/javascript">
```

```
$(document).ready(function()

{

    //动画速度

    var speed = 500;


    //绑定事件处理

    $("#btnShow").click(function(event)

    {

        //取消事件冒泡

        event.stopPropagation();

        //设置弹出层位置

        var offset = $(event.target).offset();

        $("#divPop").css({ top: offset.top + $(event.target).height()
+ "px", left: offset.left });

        //切换弹出层的显示状态

        $("#divPop").toggle(speed);


    });

});
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<div>
```

```
<br /><br /><br />
```

```
<button id="btnShow">提示文字</button>
```

```
</div>
```

```
<!-- 弹出层 -->
```

```
<div id="divPop" style="background-color: #f0f0f0; border: solid 1px  
#000000; position: absolute; display:none;
```

```
width: 300px; height: 100px;">
```

```
<div style="text-align: center;">弹出层</div>
```

```
</div>
```

```
</body>
```

```
</html>
```

toggle()函数可以传递一个 boolean 值的参数, 比如: toggle(true)等同于 show(), toggle(fasle)等同于 hide()。

五. 滑动动画函数

基本动画函数的效果是一个综合了滑动和透明度渐变的函数, jQuery 还单独提供了只有滑动效果的相关函数.

滑动动画函数 **Sliding**

名称	说明	举例
<u>slideDown(speed, [callback])</u>	通过高度变化（向下增大）来动态地显示所有匹配的元素，在显示完成后可选地触发一个回调函数。 这个动画效果只调整元素的高度，可以使匹配的元素以“滑动”的方式显示出来。在 jQuery 1.3中，上下的 padding 和 margin 也会有动画，效果更流畅。	用 600毫秒缓慢的将段落滑下： <code>\$("p").slideDown("slow");</code>
<u>slideUp(speed, [callback])</u>	通过高度变化（向上减小）来动态地隐藏所有匹配的元素，在隐藏完成后可选地触发一个回调函数。	用 600毫秒缓慢的将段落滑上： <code>\$("p").slideUp("slow");</code>
<u>slideToggle(speed, [callback])</u>	通过高度变化来切换所有匹配元素的可见性，并在切换完成后可选地触发一个回调函数。	用 600毫秒缓慢的将段落滑上或滑下： <code>\$("p").slideToggle("slow");</code>

讲解

slideDown 就是 show 的滑动效果版本, slideUp 就是 hide 的滑动效果版本, slideToggle 就是 toggle 的滑动效果版本.

参数完全相同:

`$("#divPop").slideDown(200);`

`$("#divPop").slideUp("fast");`

`$("#divPop").slideToggle("slow");`

六.淡入淡出动画函数

淡出淡出函数只提供透明度渐变的效果.

淡入淡出函数 **Fading**

名称	说明	举例
<u>fadeIn(speed, [callback])</u>	通过不透明度的变化来实现	用 600毫秒缓慢的将段落淡

	<p>所有匹配元素的淡入效果，并在动画完成后可选地触发一个回调函数。</p> <p>这个动画只调整元素的不透明度，也就是说所有匹配的元素的高度和宽度不会发生变化。</p>	<p>入：</p> <pre>\$("#p").fadeIn("slow");</pre>
fadeOut(speed, [callback])	<p>通过不透明度的变化来实现所有匹配元素的淡出效果，并在动画完成后可选地触发一个回调函数。</p>	<p>用600毫秒缓慢的将段落淡出：</p> <pre>\$("#p").fadeOut("slow");</pre>
fadeTo(speed, opacity, [callback])	<p>把所有匹配元素的不透明度以渐进方式调整到指定的不透明度，并在动画完成后可选地触发一个回调函数。</p>	<p>用600毫秒缓慢的将段落的透明度调整到0.66，大约2/3的可见度：</p> <pre>\$("#p").fadeTo("slow", 0.66);\$("#p").fadeTo("slow", 0.66);</pre>

讲解

fadeIn 和 fadeOut 两个函数对应 show 和 hide, 用于将对象以透明度渐变的效果显示和隐藏：

```
$("#divPop").fadeIn(200);
```

```
$("#divPop").fadeOut("fast");
```

透明度渐变没有切换函数.

需要特别讲解的是 fadeTo 函数. 这个函数能让对象渐变到指定的透明度上. opacity 参数取值从0-1, 比如0.6表示透明度为60%.

和 fadeIn 与 fadeOut 不同的是, **fadeTo 函数只改变对象的透明度, 即使透明度为0对象仍然占位.** 而 fadeIn 和 fadeOut 最后一定会改变对象的 display 属性, fadeOut 后对象将从页面上消失(不占位), 但是 fadeTo 仅仅是让其透明(占位).

fadeTo 函数可以配合 fadeIn 使用. 比如默认的情况下, fadeIn 最后让对象完全显示：



但是如果之前使用过 fadeTo 设置弹出层的透明度, 则可以让其以半透明：



核心代码如下:

```
//设置弹出层的透明度

$("#divPop").fadeTo(0, 0.66);

//让弹出层透明显示

if ($("#divPop").css("display") == "none")

{

    $("#divPop").fadeIn(speed);

}

else

{

    $("#divPop").fadeOut(speed);

}
```

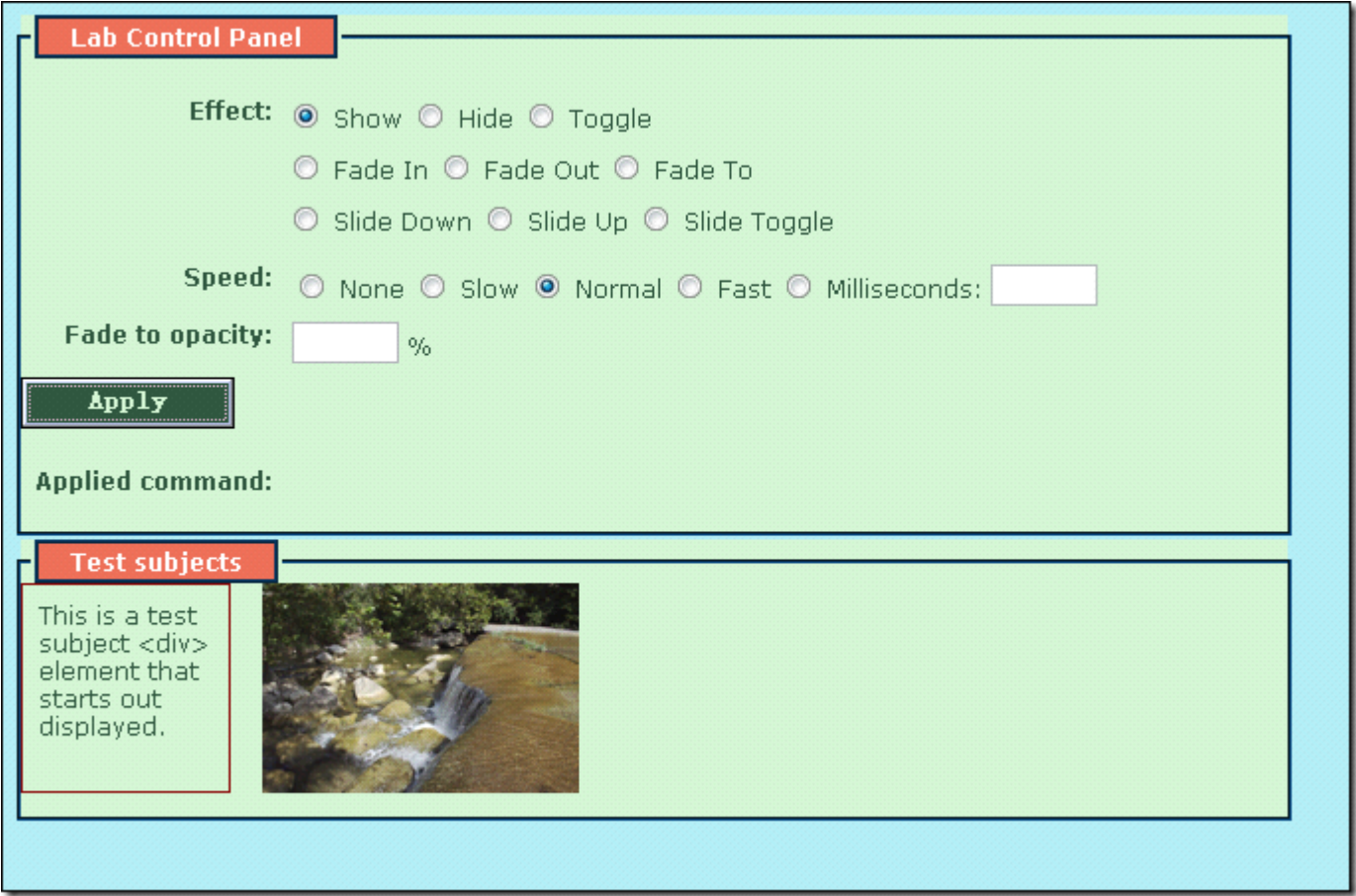
用 fadeTo 设置了弹出层透明度后, 在使用 fadeIn 会让对象显示并且渐变到 fadeTo 设置的透明度.

这里介绍的仅仅是两个函数的特性, 实际应用中并不一定要两者配合使用.

六. 动画实验室

动画实验室是"jQuery 实战"一书中的示例, 方便我们查看上面三种动画的效果. 对应源代码的

chapter7\lab.effects.html 文件.源代码在本文最后提供下载.实验室截图如下:



七.自定义动画函数

上面三个渐变动画函数已经基本满足了我们日常需求。 但是如果我们一定要创建自己的特殊的效果,jQuery 也为我们提供了相关函数.

自定义动画函数 Custom

名称	说明	举例
animate(params, [duration], [easing], [callback])	<p>用于创建自定义动画的函数。</p> <p>这个函数的关键在于指定动画形式及结果样式属性对象。这个对象中每个属性都表示一个可以变化的样式属性（如“height”、“top”或“opacity”）。注意：所有指定的属性必须用骆驼形式，比如用 marginLeft 代替 margin-left.</p> <p>而每个属性的值表示这个样式属性到多少时动画结束。如果是一个数值，样式属性就会从当前的值渐变到指定的值。如果使用的是“hide”、“show”或“toggle”这样的字符串值，则会为该属性调用默认的动</p>	<p>点击按钮后 div 元素的几个不同属性一同变化:</p> <p>// 在一个动画中同时应用三种类型的效果</p> <pre>\$("#go").click(function(){ \$("#block").animate({ width: "90%", height: "100%", fontSize: "10em", borderWidth: 10 }, 1000); });</pre>

	<p>画形式。</p> <p>在 jQuery 1.2 中，你可以使用 <code>em</code> 和 <code>%</code> 单位。另外，在 jQuery 1.2 中，你可以通过在属性值前面指定 <code>"+="</code> 或 <code>"-="</code> 来让元素做相对运动。</p> <p>jQuery 1.3中，如果 <code>duration</code> 设为0则直接完成动画。而在以前版本中则会执行默认动画。</p>	
<u>animate(params, options)</u>	<p>用于创建自定义动画的函数。</p> <p>这个函数的关键在于指定动画形式及结果样式属性对象。这个对象中每个属性都表示一个可以变化的样式属性（如“<code>height</code>”、“<code>top</code>”或“<code>opacity</code>”）。注意：所有指定的属性必须用骆驼形式，比如用 <code>marginLeft</code> 代替 <code>margin-left</code>。</p> <p>而每个属性的值表示这个样式属性到多少时动画结束。如果是一个数值，样式属性就会从当前的值渐变到指定的值。如果使用的是“<code>hide</code>”、“<code>show</code>”或“<code>toggle</code>”这样的字符串值，则会为该属性调用默认的动画形式。</p> <p>在 jQuery 1.2 中，你可以使用 <code>em</code> 和 <code>%</code> 单位。另外，在 jQuery 1.2 中，你可以通过在属性值前面指定 <code>"+="</code> 或 <code>"-="</code> 来让元素做相对运动。</p>	<p>第一个按钮按了之后展示了不在队列中的动画。在 <code>div</code> 扩展到90%的同时也在增加字体，一旦字体改变完毕后，边框的动画才开始：</p> <pre>\$("#go1").click(function(){ \$("#block1").animate({ width: "90%"}, { queue: false, duration: 5000 }) .animate({ fontSize: '10em' } , 1000) .animate({ borderWidth: 5 }, 1000); }); \$("#go2").click(function(){ \$("#block2").animate({ width: "90%"}, 1000) .animate({ fontSize: '10em' } , 1000) .animate({ borderWidth: 5 }, 1000); });</pre>
<u>stop([clearQueue], [gotoEnd])</u>	<p>停止所有在指定元素上正在运行的动画。</p> <p>如果队列中有等待执行的动画(并且 <code>clearQueue</code> 没有设为 <code>true</code>)，他们将被马上执行</p> <p><code>clearQueue(Boolean)</code>: 如果设置成 <code>true</code>，则清空队列。可以立即结束动画。</p> <p><code>gotoEnd (Boolean)</code>: 让当前正在执行的动画立即完成，并且重设 <code>show</code> 和 <code>hide</code> 的原始样式，调用回调函数等。</p>	<p>点击Go之后开始动画,点 Stop之后会在当前位置停下来:</p> <pre>// 开始动画 \$("#go").click(function(){ \$(".block").animate({left: '+200px'}, 5000); }); // 当点击按钮后停止动画 \$("#stop").click(function(){ \$(".block").stop();</pre>

		});
--	--	-----

参数说明

1.params(可选)

类型:Options

说明:一组包含作为动画属性和终值的样式属性和及其值的集合.

讲解:通过把元素的样式属性值, 从当前值逐渐调整到 params 设置的值而产生动画效果.

2.duration(可选)

类型:String,Number

说明:三种预定速度之一的字符串("slow", "normal", or "fast")或表示动画时长的毫秒数值(如: 1000)

讲解:动画效果持续的时间, 时间越长则变得越慢. 如果省略则不会产生动画.

3.easing(可选)

类型:String

说明:要使用的擦除效果的名称(需要插件支持).默认 jQuery 提供"linear" 和 "swing".

讲解:为了让元素逐渐达到 params 设置的最终效果, 我们需要有一个函数来实现渐变, 这类函数就叫做 easing 函数. 但是需要这里传递的只是 easing 函数名称, 使用前需要先将 easing 函数注册到 jQuery 上.

4.options 参数

类型:Options

说明:一组包含动画选项的值的集合。

讲解:所支持的属性如下:

- duration: 与上面的 duration 参数相同
- easing: 与上面的 easing 参数相同
- complete :类型为 Function, 在动画完成时执行的函数
- step: [Callback](#)
- queue (Boolean): (默认值: true) 设定为 false 将使此动画不进入动画队列 (jQuery 1.2中新增)

讲解

自定义动画属于高级应用, 在这里我暂时无法做详细的讲解.下面通过两个示例让大家简单了解如何使用自定义动画.

Bug 提示: 下面两个示例使用 vsdoc2智能提示版本的 jQuery 类库在 FireFox 下存在透明度无法渐变的问题. 请使用其他版本.

自定义坠落动画:

这个示例让一个图层从屏幕最上方掉落到最下方, 并且消失.

```
<!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>
```

```
<title>jQuery Animation - fadeTo </title>
```

```
<script type="text/javascript" src="../scripts/jquery-1.3.2.js"></script>
```

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

```
$(document).ready(function()
```

```
{
```

```
$("#divPop")
```

```
.animate(
```

```
{
```

```
"opacity": "hide",
```

```
"top": $(window).height() - $("#divPop").height() -
```

```
$("#divPop").position().top
```

```
},
```

600,

```
function() { $("#divPop").hide(); }
```

```
);
```

```
});
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<div id="divPop" style="background-color: #f0f0f0; border: solid 1px  
#000000;
```

```
width: 300px; height: 100px; position:absolute;">
```

```
<div style="text-align: center;">弹出层</div>
```

```
</div>
```

```
</body>
```

```
</html>
```

自定义消散动画:

这个示例让一个 div 越来越大最后消失:

```
<!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>
```

```
<title>jQuery Animation - fadeTo </title>
```

```
<script type="text/javascript" src="../../scripts/jquery-1.3.2.js"></script>
```

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

```
$(document).ready(function()
```

```
{
```

```
$("#divPop")
```

```
.animate(
```

```
{
```

```
"opacity": "hide",
```

```
"width": $(window).width()-100 ,
```

```
"height": $(window).height()-100
```

```
},
```

```
500
```

```
);
```

```
});
```

```
</script>

</head>

<body>

    <div id="divPop" style="background-color: #f0f0f0; border: solid 1px
#000000;

    width: 300px; height: 100px; position:absolute;">

    <div style="text-align: center;">弹出层</div>

</div>

</body>

</html>
```

八.全局控制属性

最后讲一下和动画相关的属性:

名称: [jQuery.fx.off](#)

返回值: [Boolean](#)

说明:

关闭页面上所有的动画。

讲解:

把这个属性设置为 **true** 可以立即关闭所有动画(所有效果会立即执行完毕)。有些情况下可能需要这样, 比如:

- * 你在配置比较低的电脑上使用 jQuery。
- * 你的一些用户由于动画效果而遇到了 [可访问性问题](#)

当把这个属性设成 `false` 之后，可以重新开启所有动画。

比如下面的代码会执行一个禁用的动画：

```
jQuery.fx.off = true;
```

```
$("#divPop").show(1000);
```

虽然使用了动画效果的 `show` 函数，但是因为关闭了所有动画，所以 `div` 会立刻显示出来而没有渐变效果。

九.总结

本文讲解了 jQuery 提供的三种动画函数：基本动画，滑动动画和淡入淡出动画。使用这三种动画已经基本可以满足我们的日常开发需求，让我们的页面动起来。简单举例讲解了自定义动画。对于想深入研究的人本文只能起到抛砖引玉的效果。

代码下载：

<http://files.cnblogs.com/zhangziqu/Code-jQueryStudy.rar>

Tag 标签: [jQuery](#), [jQuery 教程](#)

jQuery (八) 插播:jQuery 实施方案

一.摘要

本系列文章将带您进入 jQuery 的精彩世界, 其中有很多作者具体的使用经验和解决方案, 即使你会使用 jQuery 也能在阅读中发现些许秘籍.

本篇文章属于临时插播, 用于介绍我在本公司的 jQuery 实施方案.

二.前言

有了前几章扎实的基础知识我们已经可以在项目中投入使用 jQuery 了.再继续深入学习 jQuery 前插播一下我的 jQuery 实施方案.

每个公司的情况都不同.比如我们公司的页面文件都为用户控件, 物理路径和虚拟路径没有绝对的关系, 所以无法使用相对路径(否则生产环境中会找不到文件). 项目繁多, 同一个虚拟目录的不同文件夹对应不同项目工程等等.

此方案并不是通用的, 但是有些方法可以借鉴, 同时也是希望能和大家一起讨论帮忙指正.

三.类库文件管理方案

存放根路径: src\Assembly\resource.eLong.Web.Files\Resource\JSLib\jquery\

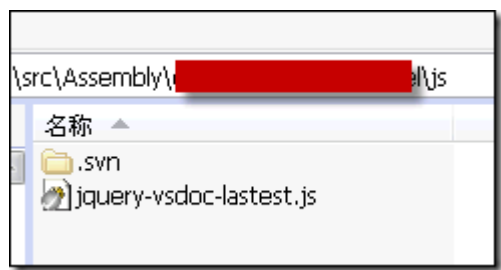
引用根路径: <http://resource.elong.com/JSLib/jquery/>

压缩版本引用路径: <http://resource.elong.com/JSLib/jquery/jquery-min-lastest.js>

根路径文件列表:

1.3.2	文件夹
plugin	文件夹
jquery.extend-lastest.js	0 KB JScript 脚本文件
jquery-lastest.js	118 KB JScript 脚本文件
jquery-min-lastest.js	58 KB JScript 脚本文件
jquery-vsdoc-lastest.js	189 KB JScript 脚本文件

在每一个 Web 工程项目下面建立 js 文件夹, 放置 jquery-vsdoc-lastest.js 文件:



说明：

首先将所有版本的 jQuery 类库放到静态服务器上，并且按照文件夹存放类库文件。但是会选出来一个最新版本作为引用并且放置在根目录。

根目录下面包含如下文件：

名称	内容	说明
文件夹1.3.2	按照版本号组织的 jQuery 类库。	对于1.3.2即以后的每个 jquery 版本，会按照文件夹存放类库文件。但是会用最新的稳定版本覆盖各 lastest 名称的 js 文件。
文件夹 plugin	存放插件的文件夹	将各种插件以文件夹的形式存放到此路径下
jquery.extend-lastest.js	elong 自己扩展的 jQuery 方法	未来我们将打造自己的 javascript 类库。其中有一些核心的类库放在此文件中。相当于我们自己的工具类库。通过扩展 jQuery 实现。 主要分为工具函数和包装集函数两类。
jquery-lastest.js	jQuery 未压缩类库最新版本	最新的稳定版本的 jQuery 原始类库。
jquery-min-lastest.js	jQuery 压缩类库最新版本	最新的稳定版本的 jQuery 压缩类库。
jquery-vsdoc-lastest.js	jQuery 智能提示类库最新版本	最新的稳定版本的 jQuery 智能提示类库。

为了在开发时实现智能感知，还需要将智能感知版本放置在每个 web 项目的 js 文件夹中。因为 Web 项目众多所以请以后第一个使用者建立此文件夹并放入文件。

四.类库引用方案

在所有的页面 head 中，最先引用 jQuery 的类库，使用绝对路径：

<http://resource.elong.com/JSLib/jquery/jquery-min-lastest.js>

然 后通过 if(fasle)引入智能提示版本的脚本块。路径使用"~"从根目录开始查找。我在各个频道的项目源代码中统一建立 js 文件夹并且放置 jquery-vsdoc-lastest.js 最新的智能感知版本类库。注意此文件不需要打包上传，仅用于开发时的智能提示。

这样可以确保编译后的页面只引入了压缩版本的 jQuery 类库。

示例代码:

```
<head runat="server">

    <title>jQuery 引用方案</title>

    <script                                type="text/javascript"
src="http://resource.elong.com/JSLib/jquery/jquery-min-lastest.js"></script>

    <% if (false){%>

        <script                                src="~/js/jquery-vsdoc-lastest.js"
type="text/javascript"></script>

    <% }%>

</head>
```

说明:

在我们的网站中, 静态文件存储在另外一个二级域名 resource.elong.com 下, 使用了 CDN. 为了保证测试环境和正式环境一致只能使用绝对路径引用 jQuery 库. 但是**使用绝对路径引用 jQuery 智能提示版本后不会出现脚本智能提示.**所以我们通过此特性直接引用绝对路径的压缩版本 jQuery 类库, 从而巧妙的解决了**1.3.2压缩版本引入后智能提示系统出错**的问题.

虽然动态页面可以通过 if(false) 取消引入智能提示版本类库, 但是在 HTML 页面上就无法使用服务器语句块.所以对于 **HTML** 页面需要在开发完毕程序发布前手工删除智能提示版本的引用.

五.开发使用方案

- jQuery 是脚本库而不是脚本框架, 无法限制使用者如何使用, 所以很容易让页面上的脚本变得混乱.
- 在没有找到何时的脚本管理框架前, 使用如下方式在页面上使用脚本:
1. 在页面底部添加<script>区域, 两个 function 分别放置 **"事件绑定"** 和 **"加载时执行"** 的语句. 即使在加载时执行的 javascript 也必须要保证 DOM 加载完毕后执行. 所以两个 function 都被嵌套在\$(Q)中保证在 DOM 加载完毕后调用.
 2. 应尽量避免在头部加载脚本. 必须在头部加载的可以在页面 head 中添加一个 script 区域.

3."自定义函数"要放在"事件绑定"和"加载时执行"语句块之上, 并且不需要包含在\$(Q)中.

下面是一个完整页面的示例代码:

```
<%@ Page Language="C#" %>

<!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 runat="server">

    <title>jQuery 引用方案</title>

    <script type="text/javascript"
src="http://resource.elong.com/JSLib/jquery/jquery-min-lastest.js"></script>

    <% if (false){%>

        <script src="~/js/jquery-vsdoc-lastest.js"
type="text/javascript"></script>

    <% }%>

    <script type="text/javascript">

        //必须放在头部加载的语句块. 尽量避免使用

    </script>

</head>

<body>

    <div id="divMsg">Hello World!</div>
```

```
<input id="btnShow" type="button" value="显示" />
```

```
<input id="btnHide" type="button" value="隐藏" /><br />
```

```
<input id="btnChange" type="button" value="修改内容为 Hello World, too!" />
```

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

```
//用户自定义方法
```

```
function demoMethod(event)
```

```
{
```

```
    $("#divMsg").hide(500);
```

```
}
```

```
//事件绑定
```

```
$(function()
```

```
{
```

```
    $("#btnShow").bind("click", function(event)
```

```
{ $("#divMsg").show(500); });
```

```
    $("#btnHide").bind("click", demoMethod);
```

```
    $("#btnChange").bind("click", function(event)
```

```
{ $("#divMsg").html("Hello World, too!"); });
```

```
});
```

```
//加载时执行的语句
```

```
$(function()
```

```
{
```

```
$("#btnShow").attr("value", "被修改后的显示按钮")
```

```
});
```

```
</script>
```

```
</body>
```

```
</html>
```

六.总结

在确认了没有公布任何保密信息后，我发表了本文。没有太多技术含量，主要是针对所在公司推广 jQuery 的具体实施方法。

另外我一直想找一个成型的脚本框架用来组织管理各种 js 类库和 js 文件。这都需要在以后的工作中探索。

Tag 标签: [jQuery](#),[jQuery 教程](#)

jQuery (九) jQuery 工具函数

一.摘要

本系列文章将带您进入 jQuery 的精彩世界, 其中有很多作者具体的使用经验和解决方案, 即使你会使用 jQuery 也能在阅读中发现些许秘籍.

我们经常要使用脚本处理各种业务逻辑, 最常见的就是数组和对象的操作. jQuery 工具函数为我们操作对象和数组提供了便利条件.

二.前言

大部分人仅仅使用 jQuery 的选择器选择对象, 或者实现页面动画效果. 在处理业务逻辑时常常自己编写很多算法. 本文提醒各位 jQuery 也能提高我们操作对象和数组的效率. 并且可以将一些常用算法扩充到 jQuery 工具函数中, 实现脚本函数的复用.

三.什么是工具函数

工具函数是指在 jQuery 对象(即变量"\$")上定义的函数. 这些函数都是工具类函数.比如 C#中最常用的 trim()函数:

```
$.trim(" text ");
```

在原始 javascript 中并没有提供同时去除前后空格的 trim 函数. 所以这一类常用的工具函数统称为 "Utilities" 函数.对应 jQuery 官方文档:

<http://docs.jquery.com/Utilities>

"\$"其实是"window"对象的属性, 所以下面几句话是等价的:

```
$.trim(" text ");
```

```
window.$.trim(" text ");
```

```
window.jQuery(" text ");
```

```
jQuery.trim(" text ");
```

四.工具函数分类

工具函数主要分为下面几类:

- 浏览器及特性检测
- 数组和对象操作
- 测试操作
- 字符串操作
- **Url** 操作

区别于前几章的讲解方式, 本文不在列举函数列表. 大家在应用中, 比如遇到想操作一个字符串, 可以首先从在"**API 文档/Utilities/字符串操作**"中查找是否已经提供了快捷的工具函数. 如果没有再考虑自己开发.

下面使用实例具体的每个分类下常用的工具函数.

五.浏览器及特性检测

jQuery 的优秀就在于其跨浏览器的特性, 通常我们不用再针对不同浏览器书写不同的代码. 但是如果是 jQuery 开发人员或者插件开发人员就要自行处理浏览器差异, 以便为用户提供跨浏览器的特性.

jQuery 提供了下列属性用于获取浏览器特性:

jQuery.support	1.3版本新增
jQuery.browser	已废除
jQuery.browser.version	已废除
jQuery.boxModel	已废除

在1.3版本中已经废除了三个属性, 这里不再讲解. 让我们将注意力放在 **jQuery.support** 函数上.

jQuery.support

返回值: [Object](#)

说明:

jQuery 1.3 新增。一组用于展示不同浏览器各自特性和 bug 的属性集合。

jQuery 提供了一系列属性, 你也可以自由增加你自己的属性。其中许多属性是很低级的, 所以很难说他们能否在日新月异的发展中一直保持有效, 但这这些主要用于插件和内核开发者。

所有这些支持的属性值都通过特性检测来实现, 而不是用任何浏览器检测。以下有一些非常棒的资源用于解释这些特性检测是如何工作的:

- <http://peter.michaux.ca/articles/feature-detection-state-of-the-art-browser-scripting>
- <http://yura.thinkweb2.com/cft/>
- http://www.jibbering.com/faq/faq_notes/not_browser_detect.html

jQuery.support 主要包括以下测试:

boxModel: 如果这个页面和浏览器是以 W3C CSS 盒式模型来渲染的，则等于 true。通常在 IE 6 和 IE 7 的怪癖模式中这个值是 false。在 document 准备就绪前，这个值是 null。

cssFloat: 如果用 cssFloat 来访问 CSS 的 float 的值，则返回 true。目前在 IE 中会返回 false,他用 styleFloat 代替。

hrefNormalized: 如果浏览器从 getAttribute("href")返回的是原封不动的结果，则返回 true。在 IE 中会返回 false，因为他的 URLs 已经常规化了。

htmlSerialize: 如果浏览器通过 innerHTML 插入链接元素的时候会序列化这些链接，则返回 true，目前 IE 中返回 false。

leadingWhitespace: 如果在使用 innerHTML 的时候浏览器会保持前导空白字符，则返回 true，目前在 IE 6-8 中返回 false。

noCloneEvent 如果浏览器在克隆元素的时候不会连同事件处理函数一起复制，则返回 true，目前在 IE 中返回 false。

objectAll: 如果在某个元素对象上执行 getElementsByTagName("*")会返回所有子孙元素，则为 true，目前在 IE 7 中为 false。

opacity: 如果浏览器能适当解释透明度样式属性，则返回 true，目前在 IE 中返回 false，因为他用 alpha 滤镜代替。

scriptEval: 使用 appendChild/createTextNode 方法插入脚本代码时，浏览器是否执行脚本，目前在 IE 中返回 false，IE 使用 .text 方法插入脚本代码以执行。

style: 如果 getAttribute("style")返回元素的行内样式，则为 true。目前 IE 中为 false，因为他用 cssText 代替。

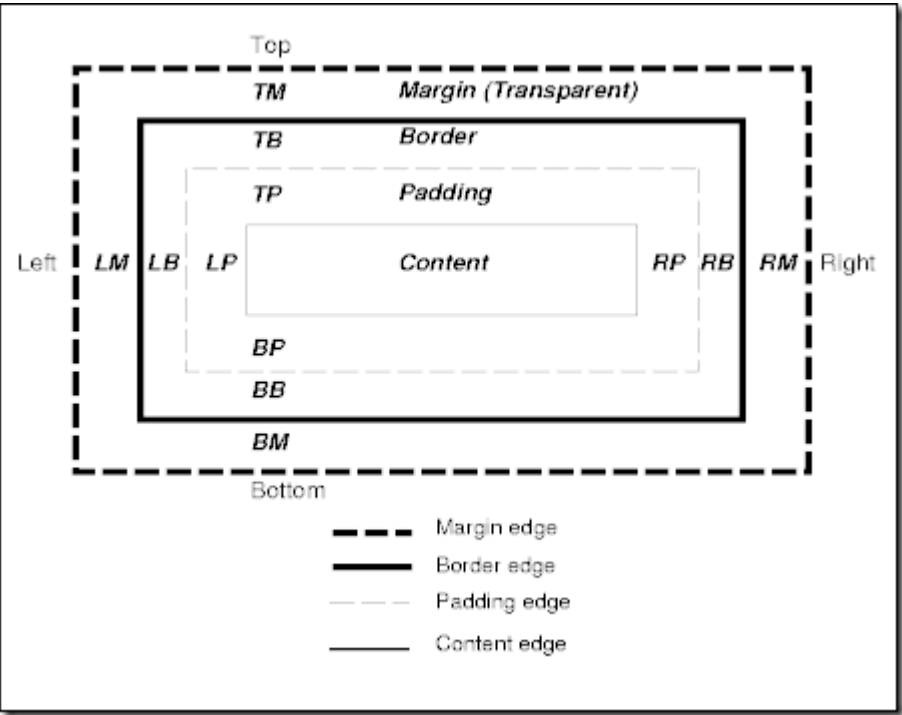
tbody: 如果浏览器允许 table 元素不包含 tbody 元素，则返回 true。目前在 IE 中会返回 false，他会自动插入缺失的 tbody。

讲解:

针对上面众多的浏览器特性属性， 本文只讲解两个特性.

1.盒式模型 boxModel

下图是 W3C 标准中的盒式模型图:



假设如下元素:

```
<style type="text/css">

.boxModel

{

    width:200px;

    height:50px;

    padding:10px;

    border:solid 5px #FF0000;

    background-color:#acacac;

}

</style>

<div id="divBox" class="boxModel">
```

显示效果如图:



在 CSS 中设定元素宽度为200px, 下面以此元素为例讲解盒式模式.

W3C 盒式模型:

元素的宽度和高度为盒式模型图中的 Context 部分, 不包括 padding, border 和 margin 部分.

目前除了 IE 所有的浏览器都**仅支持 W3C 盒式模型**. 在 W3C 盒式模型中, 示例中包含红框在内的区域内容宽度为 $200+2*10+2*5=230\text{px}$, 高度为 $50+2*10+2*5=80\text{px}$.

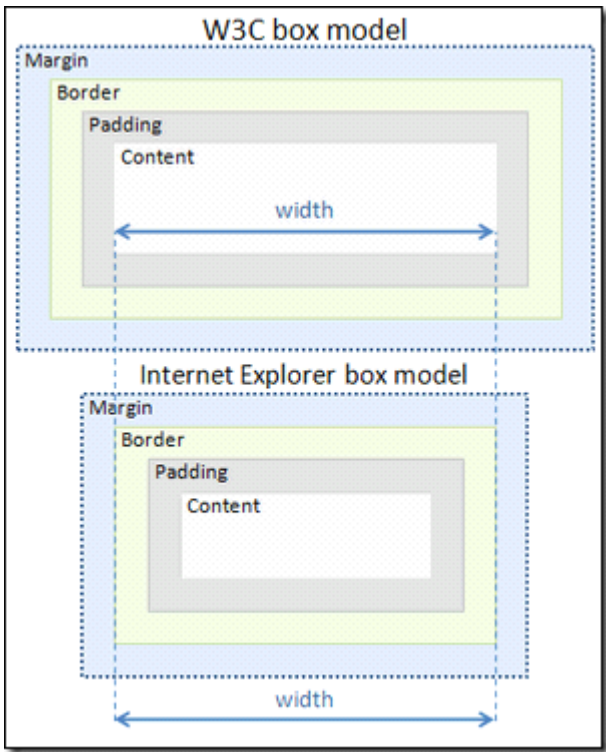
IE 盒式模型:

设置的宽度包括 padding,border. 实际内容宽度 $\text{content Width} = \text{width} - \text{padding} - \text{border}$

在 IE5.5及更早的版本中, 使用了此模型. 在更高的 IE 版本上如果由于某些原因让浏览器运行在怪异模式下则也会使用此盒式模式.所以需要在页面上声明正确的 DOCTYPE. 有关 DOCTYPE 请参考此文:

<http://www.cnblogs.com/zhangziquu/archive/2009/01/15/doctype.html>

下面是两种盒式模式的对比:



我们可以使用 `jQuery.support.boxModel` 属性来获取浏览器是否使用了 W3C 盒式模型. true 表示使用 W3C boxModel.

2.浮动样式

通过 javascript 脚本设置元素的 float 样式时, IE 和 FireFox 存在不同, IE 使用 `style.styleFloat`, FireFox 使用 `style.cssFloat`:

```
div.style.styleFloat = "left"; //IE
```

```
div.stlye.cssFloat = "left"; //FF
```

jQuery.support.cssFloat属性返回 true 则表示可以使用 cssFloat 来设置 float 样式. IE 中返回 false;

注意, 我们可以通过 CSS()方法设置 float 样式, jQuery 内部会自动帮我们判断是使用 styleFloat 还是 cssFloat:

```
$("#divResult").css("float","left"); //兼容 IE 和 FF
```

六.数组和对象操作

实现 UI 我们常常操作 DOM 对象或者 jQuery 包装集, 但是实现算法或者业务逻辑时往往操作的是数组和对象.

下面讲解最常用的数组和对象相关的工具函数.

1.迭代

jQuery.each(object, callback)

返回值:Object

说明:

通用例遍方法, 可用于例遍对象和数组。

不同于例遍 jQuery 对象的 `$.each()` 方法, 此方法可用于例遍任何对象。回调函数拥有两个参数: 第一个为对象的成员或数组的索引, 第二个为对应变量或内容。如果需要退出 each 循环可使回调函数返回 false, 其它返回值将被忽略。

讲解:

对于 jQuery 包装集我们可以使用 each(callback)方法迭代包装集中的每一个元素. callback 是一个会函数, 接受一个参数表示当前访问对象的索引.

```
$("#img").each(function(i){
```

```
    this.src = "test" + i + ".jpg";
```

```
});
```

对于数组我们可以使用 [jQuery.each\(object, callback \)](#) 来遍历, 这等同于使用 for 循环.

注意传入的第一个参数可以是数组或者对象.如果数组,则遍历数组中的每一个对象. **第一个参数表示索引,第二个参数表示值, this 表示当前遍历的元素**, 可以通过返回 false 终止迭代, 比如下面的示例遍历到第二个元素后会终止:

```
$.each(["a", "b", "c"], function(i, n)

{

    alert("Item #" + i + ": " + n); //可以获取到 i 值

    if (i >= 1)

    {

        return false;

    }

});
```

```
$("#iterateArray").click(function(event)

{

    var array = $.each(["a", "b", "c"], function(i, n)

    {

        alert("Item #" + i + ": " + n ); //第一个参数 i 表示索引, this 表示当前遍历的对象

        if (i >= 1)

        {
```

```

        return false;

    }

});

});

```

如果传递的是对象，则遍历对象的每一个属性，即使函数返回 false 也依然会遍历完所有的属性，第一个参数表示属性 **key**(属性名称,是 **object** 类型),第二个参数表示值,,**this** 表示当前属性的值:

```

$("#iterateObject").click(function(event)

{

    $.each({ name: "ziqu.zhang", sex: "male", status: "single"

}, function(i, n)

{

    alert("Item #" + i.toString() + ": " + n ); //第一

```

个参数 i 表示属性的 **key(object)**, this 表示属性值

```

    if (i >= 1)

    {

        return false;

    }

});

});

```

each 将是我们最常使用的函数，特别注意 each 虽然迭代每一个元素或属性，但是在迭代函数中并不会改变当前元素的值，也就是无法改变返回后的对象.如果需要改变数组中的每一个元素并

且将结果返回, 因使用 [jQuery.map\(array, callback \)](#) 函数.

2.筛选

[jQuery.grep\(array, callback, \[invert\] \)](#)

返回值: **Array**

说明:

使用过滤函数过滤数组元素。

此函数至少传递两个参数: 待过滤数组和过滤函数。过滤函数必须返回 `true` 以保留元素或 `false` 以删除元素。

讲解:

默认 `invert` 为 `false`, 即过滤函数返回 `true` 为保留元素. 如果设置 `invert` 为 `true`, 则过滤函数返回 `true` 为删除元素.

下面的示例演示如何过滤数组中索引小于 0 的元素:

```
$.grep( [0,1,2], function(n,i){  
  
    return n > 0;  
  
});
```

返回的结果是[1,2]

3.转换

[jQuery.map\(array, callback \)](#)

返回值: **Array**

说明:

将一个数组中的元素转换到另一个数组中。

作为参数的转换函数会为每个数组元素调用, 而且会给这个转换函数传递一个表示被转换的元素作为参数。转换函数可以返回转换后的值、`null` (删除数组中的项目) 或一个包含值的数组, 并扩展至原始数组中。

讲解:

1.3.2版本中此函数和 `each` 函数已经几乎相同(以前稍有不同), 现在唯一的区别就是回调函数可以改变当前元素.返回 `null` 则删除当前元素.

下面是几个例子:

```
var arr = [ "a", "b", "c", "d", "e" ]

$("div").text(arr.join(", "));

arr = jQuery.map(arr, function(n, i){

    return (n.toUpperCase() + i);

});

$("p").text(arr.join(", "));

arr = jQuery.map(arr, function (a) { return a + a; });

$("span").text(arr.join(", "));
```

4.合并

合并对象是我们常常编写的功能，通常使用臃肿的 for 循环来进行。jQuery 为我们提供了很多功能的合并函数：

名称	说明	举例
jQuery.extend([deep], target, object1, [objectN])	<p>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。</p> <p>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。</p> <p>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。</p> <p>为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。</p>	<p>合并 settings 和 options，修改并返回 settings：</p> <pre>var settings = { validate: false, limit: 5, name: "foo" }; var options = { validate: true, name: "bar" }; jQuery.extend(settings, options);</pre> <p>结果：</p> <pre>settings == { validate: true, limit: 5, name: "bar" }</pre>

<u>jQuery.makeArray(obj)</u>	<p>将类数组对象转换为数组对象。</p> <p>类数组对象有 length 属性，其成员索引为 0 至 length - 1。实际中此函数在 jQuery 中将自动使用而无需特意转换。</p>	<p>将 DOM 对象集合转换为数组:</p> <pre>var arr = jQuery.makeArray(document.g etElementsByTagName("div"));</pre>
<u>jQuery.inArray(value, array)</u>	<p>确定第一个参数在数组中的位置，从0开始计数(如果没有找到则返回 -1)。</p>	<p>查看对应元素的位置:</p> <pre>var arr = [4, "Pete", 8, "John"]; jQuery.inArray("John", arr); //3 jQuery.inArray(4, arr); //0 jQuery.inArray("David", arr); //-1</pre>
<u>jQuery.merge(first, second)</u>	<p>合并两个数组</p> <p>返回的结果会修改第一个数组的内容——第一个数组的元素后面跟着第二个数组的元素。要去除重复项，请使用 \$.unique()</p>	<p>合并两个数组到第一个数组上:</p> <pre>\$.merge([0,1,2], [2,3,4])</pre> <p>结果:</p> <pre>[0,1,2,2,3,4]</pre>
<u>jQuery.unique(array)</u>	<p>删除数组中重复元素。只处理删除 DOM 元素数组，而不能处理字符串或者数字数组。</p>	<p>删除重复 div 标签:</p> <pre>\$.unique(document.getElement sByTagName("div"));</pre> <pre>[<div>, <div>, ...]</pre>

讲解:

上面的函数看着有些混乱。 看看我们以后会常用的.

首先是 [jQuery.merge\(first, second \)](#). 将两个数组合并. 下面这个示例说明如何使用此函数:

```
<html xmlns="http://www.w3.org/1999/xhtml">

<head>

  <title>jQuery Utilities - jQuery.merge</title>

  <script src="../../scripts/jquery-1.3.2-vsdoc2.js"
type="text/javascript"></script>
```



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

    $(function()

    {

        $("#go").click(function(event)

        {

            $("#divResult").html("");

            var first = [1, 3, 5];

            $("#divResult").append("<span>first:[" + first.join(",") +
"]</span>").append("<br/>");

            var second = [2, 4, 6];

            $("#divResult").append("<span>second:[" + second.join(",")
+ "]"</span>").append("<br/>");

            var result = $.merge(first, second);

            $("#divResult").append("<span>result:[" + result.join(",")
+ "]"</span>").append("<br/>");

            $("#divResult").append("<span>first   after   merged:[" +
first.join(",") + "]"</span><br/>");

            $("#divResult").append("<span>second   after   merged:[" +
second.join(",") + "]"</span><br/>");
```

```
});
```

```
});
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<button id="go">
```

```
合并数组</button>
```

```
<br />
```

```
<div id="divResult">
```

```
</div>
```

```
</body>
```

```
</html>
```

结果如图:

```
合并数组
first:[1,3,5]
second:[2,4,6]
result:[1,3,5,2,4,6]
first after merged:[1,3,5,2,4,6]
second after merged:[2,4,6]
```

另外不能因为有了 **jQuery** 就忘记我们的原始 **javascript**. 比 **merge** 更常用的其实是 **join** 和 **split** 函数.

`merge` 函数会改变第一个合并的数组, 如果是我设计我就不会这么做. 因为返回值已经是合并后的数组了. 如此设计让函数产生歧义.

列表中的那么多函数不再一一讲解. 先用先查. 除了 [jQuery.extend](#) 这个不得不提的函数. 下面单提一个小结讲解.

5. jQuery.extend

在开发插件的时候最常用此函数函数来处理 options.

下面是 fancybox 插件获取 options 的代码:

```
settings = $.extend({}, $.fn.fancybox.defaults, settings);
```

上面的代码 `target` 是一个空对象, 将默认设置 `defaults` 作为第一个对象, 将用户传入的设置 `setting` 合并到 `default` 上, `setting` 上有的属性以 `setting` 为准. `setting` 没有传入的属性则使用 `default` 的默认值. 然后将合并的结果复制给 `target` 并作为函数返回值返回.

看一个完整的示例:

```
var empty = {}
```

```
var defaults = { validate: false, limit: 5, name: "foo" };
```

```
var options = { validate: true, name: "bar" };
```

```
var settings = jQuery.extend(empty, defaults, options);
```

结果:

```
settings == { validate: true, limit: 5, name: "bar" }
```

```
empty == { validate: true, limit: 5, name: "bar" }
```

`target` 参数要传递一个空对象是因为 `target` 的值最后将被改变. 比如:

```
var defaults = { validate: false, limit: 5, name: "foo" };
```

```
var options = { validate: true, name: "bar" };
```

```
var settings = jQuery.extend(defaults, options);
```

上面的代码将 defaults 作为 target 参数，虽然最后 settings 的结果一样，但是 **defaults** 的值被改变了！而插件中的默认值应该都是固定！所以使用时请注意 target 参数的用法。

下面是我的完整示例和结果：

```
<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>jQuery Utilities - jQuery.extend</title>

<script src="../../scripts/jquery-1.3.2-vsdoc2.js"
type="text/javascript"></script>

<script type="text/javascript">

$.toObjectString = function (obj)

{

    var result = "{";

    var counter = 0;

    $.each(obj, function(i, n)

    {

        if (counter > 0) { result += ","; }

        result += i.toString() + ":" + n.toString();

        counter++;

    });

});
```

```
result += "}";
```

```
return result;
```

```
}
```

```
$(function()
```

```
{
```

```
$("#go1").click(function(event)
```

```
{
```

```
$("#divResult").html("");
```

```
var empty = {}
```

```
var defaults = { validate: false, limit: 5, name: "foo" };
```

```
var options = { validate: true, name: "bar" };
```

```
$("#divResult").append("<span>empty:" +
```

```
$.toObjectString(empty) + "</span>").append("<br/>");
```

```
$("#divResult").append("<span>defaults:" +
```

```
$.toObjectString(defaults) + "</span>").append("<br/>");
```

```
$("#divResult").append("<span>options:" +
```

```
$.toObjectString(options) + "</span>").append("<br/>");
```

```

    var settings = jQuery.extend(empty, defaults, options);

    $("#divResult").append("<span>settings  after  extend:" +
$.toObjectString(settings) + "</span>").append("<br/>");

    $("#divResult").append("<span>defaults  after  extend:" +
$.toObjectString(defaults) + "</span>").append("<br/>");

    $("#divResult").append("<span>options  after  extend:" +
$.toObjectString(options) + "</span>").append("<br/>");

});

```

```

$("#go2").click(function(event)

{

    $("#divResult").html("");

    var defaults = { validate: false, limit: 5, name: "foo" };

    var options = { validate: true, name: "bar" };

    $("#divResult").append("<span>defaults:" +
$.toObjectString(defaults) + "</span>").append("<br/>");

```

```
$("#divResult").append("<span>options:" +  
$.toObjectString(options) + "</span>").append("<br/>");
```

```
var settings = jQuery.extend(defaults, options);
```

```
$("#divResult").append("<span>settings after extend:" +  
$.toObjectString(settings) + "</span>").append("<br/>");
```

```
$("#divResult").append("<span>defaults after extend:" +  
$.toObjectString(defaults) + "</span>").append("<br/>");
```

```
$("#divResult").append("<span>options after extend:" +  
$.toObjectString(options) + "</span>").append("<br/>");
```

```
});
```

```
});
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<button id="go1" style="height:40px;width:400px;">
```

```
jQuery.extend(empty, defaults, options)</button>
```

```
<button id="go2" style="height:40px;width:400px;">

    jQuery.extend(defaults, options)</button>

<br />

<div id="divResult">

</div>

</body>

</html>
```

结果:

jQuery.extend(empty, defaults, options)

jQuery.extend(defaults, options)

empty: {}
defaults: {validate:false, limit:5, name:foo}
options: {validate:true, name:bar}
settings after extend: {validate:true, limit:5, name:bar}
defaults after extend: {validate:false, limit:5, name:foo}
options after extend: {validate:true, name:bar}

jQuery.extend(empty, defaults, options)

jQuery.extend(defaults, options)

defaults: {validate:false, limit:5, name:foo}
options: {validate:true, name:bar}
settings after extend: {validate:true, limit:5, name:bar}
defaults after extend: {validate:true, limit:5, name:bar}
options after extend: {validate:true, name:bar}

七.测试工具函数

测试工具函数主要用于判断对象是否是某一种类型, 返回的都是 Boolean 值:

[jQuery.isArray\(obj \)](#)

[jQuery.isFunction\(obj \)](#)

同时别忘记了 javascript 中自带的 isNaN 和 isFinite:

```
var test = "123";

alert(isNaN(test));
```



```
alert(isFinite(test));
```

isNaN 函数判断参数是否是非数字. 如果是数字则返回 false.

isFinite 函数检查其参数是否是无穷大.如果参数是 NaN（非数字）,或者是正、负无穷大的数,则返回 false.否则返回 true.

八.字符串处操作工具函数

目前核心类库中只有一个字符串工具函数:

[jQuery.trim\(str \)](#)

返回值: string

说明:去掉字符串起始和结尾的空格。

举例:

去掉字符串起始和结尾的空格:

```
$.trim("  hello, how are you?  ");
```

结果:

```
"hello, how are you?"
```

九.Url 操作工具函数

[jQuery.param\(obj \)](#)

返回值:string

说明:

将表单元数组或者对象序列化。是.serialize()的核心方法。

数组或 jQuery 对象会按照 name/value 对进行序列化，普通对象按照 key/value 对进行序列化

举例:

```
var params = { width:1680, height:1050 };
```

```
var str = jQuery.param(params);
```

```
$("#results").text(str);
```

结果:

```
width=1680&height=1050
```

jQuery 将其归为 `Urls` 分类, 因为此方法通常用于发送 `GET` 请求时将对象作为 `urls` 参数传递给服务端.

十. 扩展工具函数与 jQuery 包装集函数

扩展工具函数只需要对 jQuery(即"\$")进行扩展. 通常开发工具函数或者插件的人希望在开发时使用"\$", 但因为"\$"有可能和其他脚本库冲突, 所以通常我们使用下面的语法开发工具函数:

```
(function($)  
  
    {  
  
        $.myExtendMethod = function(o)  
  
            {  
  
                alert(0);  
  
            };  
  
    })(jQuery);
```

在函数体内的"\$"能保证是代表 jQuery 对象.

然后使用这种方式开发不能享受到智能感知的便利. 一般我们将扩展工具函数和扩展 jQuery 包装集函数都放在一个单独的文件中.

下面这个示例演示如何添加自定义的 jQuery 工具方法和 jQuery 包装集方法:

```
/// <reference path="jquery-1.3.2-vsdoc2.js" />
```

```
jQuery.myExtendMethod = function(o)
```

```

{

    ///    <summary>

    ///        扩展方法注释.

    ///    </summary>

    ///    <param name="o" type="String">参数提示文字</param>

    ///    <returns type="string" >返回值提示文字</returns>

    alert(0);

};

```

jQuery.fn.myExtendMethod = **function**(o)

```

{

    ///    <summary>

    ///        扩展方法注释.

    ///    </summary>

    ///    <param name="o" type="String">参数提示文字</param>

    ///    <returns type="string" >返回值提示文字</returns>

    alert(0);

};

```

通过第一行 reference, 我们可以在此 js 文件中继续使用 jQuery 脚本智能感知.

jQuery.myExtendMethod 方法扩展的工具函数.

jQuery.fn.myExtendMethod 方法扩展的是 jQuery 包装集函数, 即为使用\$()获取到的对象添加了方法.

同理使用 XML 注释, 比如<summary> 还可以为自定义方法添加智能感知提示.脚本中的 XML 注释和.NET中的一样, 有关.NET中的 XML 注释可以参考我的另外一篇文章:

[使用.NET中的 XML 注释\(一\) -- XML 注释标签讲解](#)

十一.总结

jQuery 提供了许多的工具函数, 在一般情况下可以满足我们的需要. 但是对于像 JSON 格式化一类的操作, 需要我们自己扩展, 现有的各种扩展组件资源将提高我们的开发效率, 本系列 Ajax 章节就介绍的一个 JSON 序列化的组件 jQuery.json. 更多的组件需要大家在工作中挖掘.

Tag 标签: [jQuery](#),[jQuery 教程](#)

jQuery (十) jQueryUI 常用功能实战

一.摘要

本系列文章将带您进入 jQuery 的精彩世界, 其中有很多作者具体的使用经验和解决方案, 即使你会使用 jQuery 也能在阅读中发现些许秘籍.

本文是实战篇. 使用 jQueryUI 完成制作网站的大部分常用功能.

二.前言

经过公司内部收集, 整理了一些经常使用 javascript 实现的功能. 实现这些功能的主角不是让人眼花缭乱的 jQuery 插件, 而是 jQuery UI.

如果你还在为了一个小小的特效而去下载并安装插件, 发现 Bug 还没有人替你解决. 记住插件是我们没有办法的最后选择.

使用插件有太多的坏处:

- 1.不利于维护**
- 2.增加页面大小**
- 3.不利于成员间交流共享,具有学习成本.**
- 4.不够健壮, 不能保证插件版本一直更新并修复所有问题.**

下面就引入今天的主角:jQuery UI

三.jQuery UI

jQuery UI 是 jQuery 的一部分, 是在 jQuery 之上的一套专门用于 UI 交互的类库. 使用 jQuery UI 可以实现底层交互, 各种高级特效, 动画, 并且可定制主题.

我们可以用它轻松的构建高度交互的 Web 应用程序.

官方首页:

<http://jqueryui.org/>

下载:

<http://jqueryui.com/download>

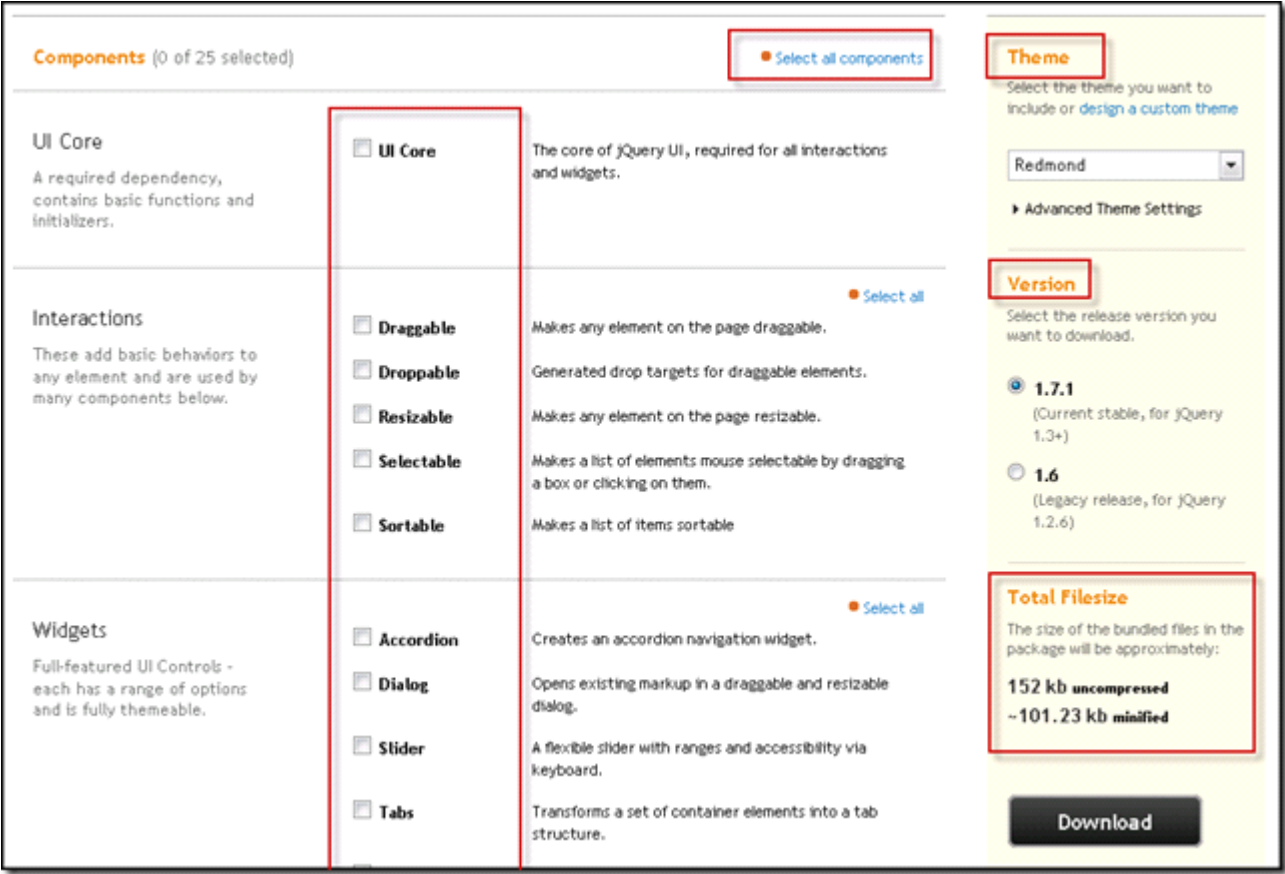
示例和文档:

<http://jqueryui.com/demos/>

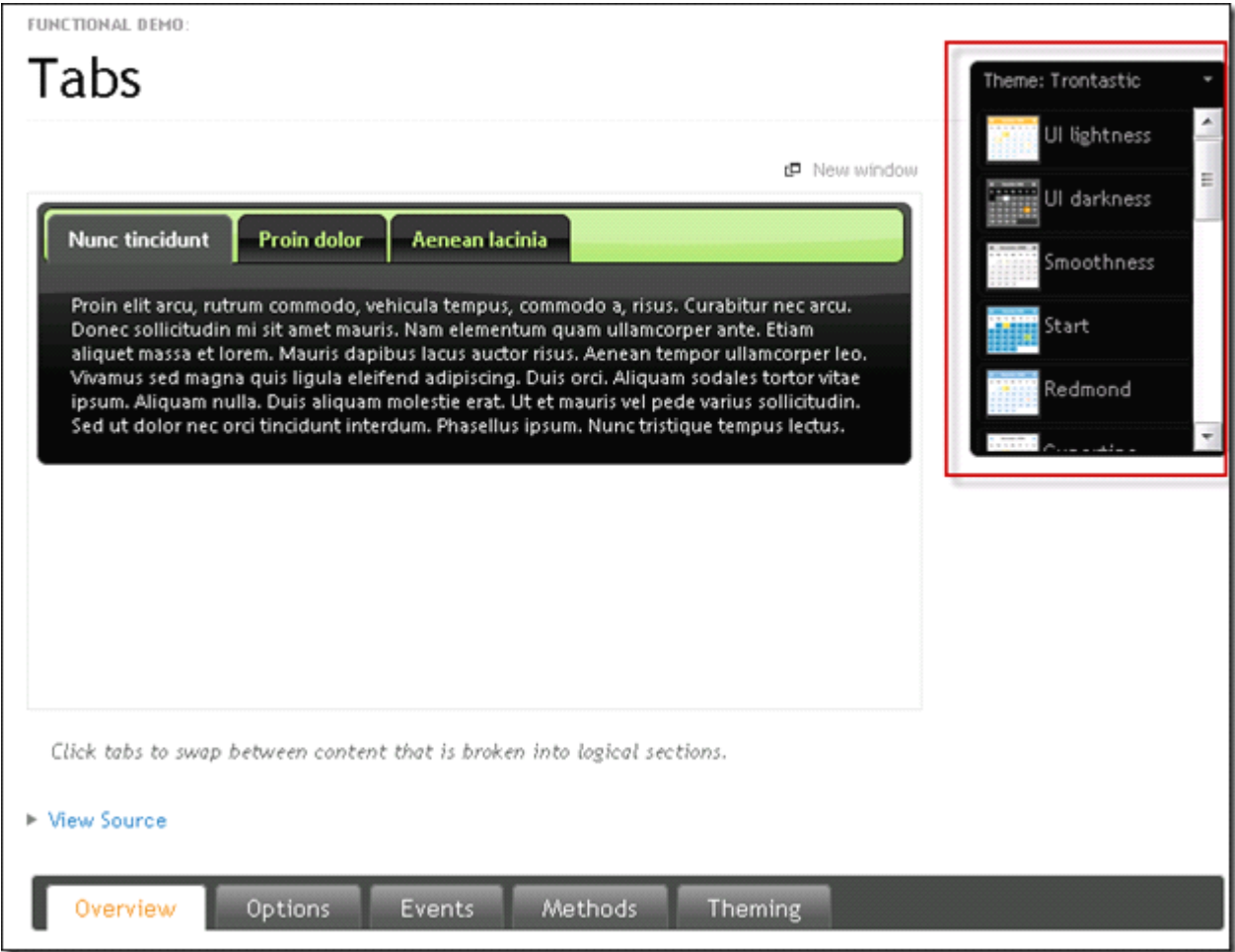
皮肤:

<http://jqueryui.com/themeroller/>

jQuery UI 的在线网站十分强大. 首先就是在下载时可以组装自己想要的功能定制下载:



并且针对各种控件不仅提供了详细的参数说明和实例, 还可以随时更换实例的皮肤:



本文主要讲解实例, 大家可以通过实例代码快速上手 jQuery UI. 使用 jQuery UI 我们可以再不借助其他插件的情况下完成大部分页面应用, 说其是一个官方的超级 UI 插件也不为过. 包含所有功能的 jQuery UI 类库文件为188K, 启用 Gzip 压缩后是45K. 虽然较大但是一次加载全网站获益. 而且45K 大小在当今的互联网时代还算可以接受.

目前还没有 jQuery UI 的中文教程. 因为本文是实战篇, 所以不会仔细讲解基础内容. 在后面的章节中我会加入 jQuery UI 的基础教程. 争取成为中文 jQuery UI 第一教程.

四. 准备工作

我将所有相关的文件, 包括各种类库文件, Theme 模板放置在如下路径:

<http://www.dotnetapi.com/JSLib/>

此路径开通了目录浏览, 可以直接查找需要的文件. 目录组织结构按照本系列: [\(八\) 插播:jQuery 实施方案](#) 中介绍的方案组织.

另外也可以从 Google 上引用文件, Google 的 CDN 速度更快也更有保证, 参见: [Google's CDN](#)

本文的实例的所有引用都使用 WebConfig.ResourceServer 这个属性:

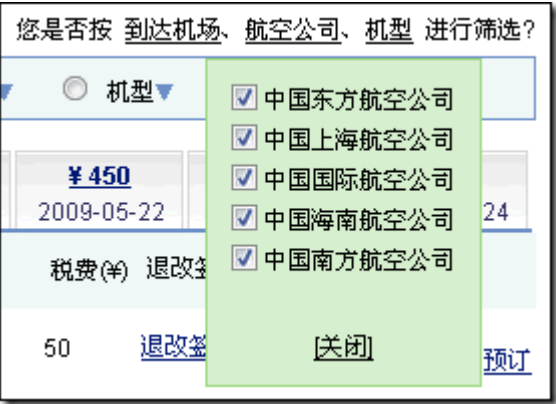
```
public class WebConfig
{
    public static string ResourceServer = @"http://www.dotnetapi.com/";
}
```

五.弹出层对话框

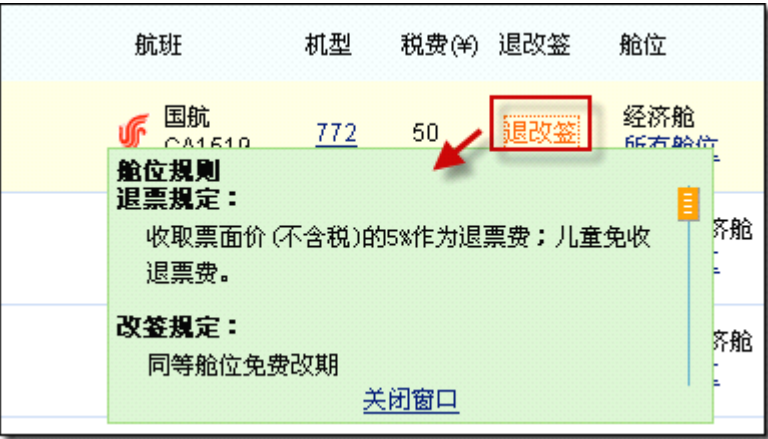
弹出框是最常用最实用的功能. 先来看一下艺龙网上的一些应用场景.

1. 艺龙网应用场景举例

(1) 静态提示类弹出层. 弹出层的内容是固定的.



(2) 动态提示类弹出层. 弹出层内容是根据事件源变化的.



(3) 遮罩类弹出层. 弹出时背景变灰并不可点击.



2. 应用实例

使用 jQuery UI 的 Dialog 组件. 我以轻松实现上面三种效果.

Dialog 组件的主要特点是可以拖动([Draggable](#)), 可以改变大小([Resizable](#)) .

示例完整代码如下:

```
<%@ Page Language="C#" %>

<!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 id="Head1" runat="server">

<title>jQuery UI - 弹出层应用实例 Dialog</title>

<!--black-tie,blitzer,blitzer,dot-luv,excite-bike,hot-sneaks,humanity,mint-
choc,redmond,smoothness,south-street,start,swanky-purse,trontastic,ui-
darkness,ui-lightness,vader-->

<link rel="stylesheet" type="text/css"
href="<%=WebConfig.ResourceServer
+ "/JsLib/jquery/themes/redmond/style.css"%>" />

<script type="text/javascript" src="<%=WebConfig.ResourceServer
%>/JsLib/jquery/jquery-min-lastest.js"></script>

<script src="<%=WebConfig.ResourceServer %>/JsLib/jquery/ui/jquery-
ui-all-min-lastest.js"

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

<% if (false)
```

```
{%}<script      src=~"/js/jquery-vsdoc-lastest.js"  
  
type="text/javascript"></script>  
  
<% }%>  
  
<script type="text/javascript">  
  
/*===== 必须放在头部加载的语句块。尽量避免使用  
=====*/  
  
</script>  
  
<style type="text/css">  
  
</style>  
  
</head>  
  
<body>  
  
<!-- Demo 静态提示类弹出层 -->  
  
<div class="ui-widget ui-widget-content ui-corner-all" style="width:  
700px; padding: 5px;">  
  
<h3>Demo. 共享同一个静态弹出层，弹出层内容固定:</h3>  
  
<div>  
  
<span id="spanShowTip1">显示提示</span>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
<span id="spanShowTip2">显示提示</span>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
  
<span id="spanShowTip3">显示提示</span>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
<span id="spanShowTip4">显示提示</span>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
  
</div>
```



```
<div>

    <input                type="button"                id="btnShowIframe"

name="btnShowIframe" value="显示弹出层"/>

</div>

</div>

<!-- 提示类弹出层 -->

<div id="divTip" title="自定义标题">

    <p>弹出层</p>

</div>

<!-- 遮罩类弹出层 -->

<div id="divIframe" title="iFrame 弹出层" style="text-align:center;">

    <iframe                src="http://www.elong.com"                width="450px"

height="230px"></iframe>

</div>

<script type="text/javascript">

    /*=====用户自定义方法=====*/


    /*=====事件绑定=====*/
```

```
$(function()

{

    //静态提示类弹出层

    $("span[id^=spanShowTip]").css("cursor",
"pointer").click(function(event)

    {

        $("*").stop();

        event.stopPropagation();

        var top = $(event.target).offset().top + 20;

        var left = $(event.target).offset().left;

        $("#divTip").dialog("option", "position", [left, top]);

        $("#divTip").dialog("open");

    });

    //动态提示类弹出层

    $("span[id^=spanShowDataTip]").css("cursor",
"pointer").click(function(event)

    {

        $("*").stop();

        $("#divTip").dialog("close");

        event.stopPropagation();
```

```
var top = $(event.target).offset().top + 20;

var left = $(event.target).offset().left;

$("#divTip").html($(event.target).attr("data"));

$("#divTip").dialog("option", "position", [left, top]);

$("#divTip").dialog("open");

});

//遮罩类弹出层

$("#btnShowIframe").click(function(event)

{

    event.preventDefault();

    event.stopPropagation();

    $("#divIframe").dialog("open");

});

//单击自身取消冒泡

$("#divTip, #divIframe").bind("click", function(event)

{

    event.stopPropagation();

});
```

//document 对象单击隐藏所有弹出层

```
$(document).bind("click", function(event)

{

    $("#divTip").dialog("close");

    $("#divIframe").dialog("close");

});

});
```

/*=====加载时执行的语句=====*/

```
$(function()

{
```

//初始化提示类弹出层

```
$("#divTip").dialog({

    show: null,

    bgiframe: false,

    autoOpen: false

});
```

```
//初始化遮罩类弹出层
```

```
$("#divIframe").dialog({
```

```
    show: null,
```

```
    bgiframe: false,
```

```
    autoOpen: false,
```

```
    width: 500,
```

```
    height: 300,
```

```
    draggable: true,
```

```
    resizable: false,
```

```
    modal: true
```

```
});
```

```
});
```

```
</script>
```

```
</body>
```

```
</html>
```

效果如图:

静态提示类弹出层

Demo. 共享同一个静态弹出层, 弹出层内容固定:

显示提示 显示提示 显示提示 显示提示

自定义标题

Demo. 每个弹出层内 弹出层

动态提示类弹出层:

Demo. 每个弹出层内容不同, 弹出层内容存储在事件源的元素属性中:

红色 绿色

自定义标题

Demo 颜色是绿色

遮罩类弹出层:

Demo. 每个弹出层内容不同, 弹出层内容存储在事件源的元素属性中:

红色 绿色

Demo. 弹出IFrame

显示弹出层

iFrame弹出层

会员登录

帐号:

登录名不能为空

密码:

密码不能为空

找回密码及卡号?

登录并继续

非会员注册

☒ 注册成为艺龙会员

3.关键点讲解

(1)计算弹出层位置

提示类弹出框最重要的是计算弹出框的位置. 通过事件对象获取到事件源, 使用 offset()函数计算事件源相对于 document 的位置:

```
var top = $(event.target).offset().top;

var left = $(event.target).offset().left;
```

因为是相对于 document, 即页面左上角的位置, 所以需要将弹出层放在 **Body** 元素中的第一层.

即父类就是 body. 如果包含在其他元素中, 需要确定任何一个父类的 **position** 样式设置为了 **relative**.

计算出来的 top 和 left 是事件源的位置, 在此位置显示会将事件源对象遮盖住. 所以通常需要手工做一些偏移, 比如 top+20.

(2) 取消冒泡和浏览器默认行为

如果我们为 document 对象绑定了单击后关闭弹出层的事件, 那么就一定要取消事件的冒泡. 使用 event 对象的 stopPropagation()方法可以取消冒泡.

`event.stopPropagation();`

对于具有默认行为的元素,比如提交按钮的提交表单行为, <a>元素的超链接行为等, 我们如果在这些元素上应用事件, 还需要取消它们的默认行为:

`event.preventDefault();`

(3) 设置动画效果与取消动画

通过设置 dialog 的配置项的 show 属性, 可以设置显示 dialog 时候的动画效果:

`$('.selector').dialog({ show: 'slide' });`

show 默认为 null 即无动画, 可以是使用下列值:

`'blind', 'clip', 'drop', 'explode', 'fold', 'puff', 'slide', 'scale', 'size', 'pulsate'.`

对于这些动画的效果, 可以在此页观看:

<http://jqueryui.com/demos/show/>

当一个动画效果执行时, 如果用户在此对这个元素进行操作, 就会出现各种问题, 比如定位不准确等. 所以如果应用了动画, 我们在对其操作时需要使用 stop()函数来停止动画, 通常是停止虽有元素的动画:

`$("*").stop();`

但是即使停止了动画再进行操作, 如果操作的太快也容易产生问题. 所以至于是否使用动画需要经过权衡决定.

(4) 动态提示类弹出层的数据传递

动态提示类弹出层的数据是不同的, 本文实例使用的是将数据存储在元素属性 data 上:

`红色`

这是一种简单直观的方式. 比较容易编程实现(尤其是在使用 MVC 编程模型的时候.)

还有一种常用方式是使用 javascript 变量存储数据.这两种方式在第5章时有过讲解:

<http://www.cnblogs.com/zhangziqui/archive/2009/05/06/jQuery-Learn-5.html>

(5)更换主题

大家注意实例中的弹出层没有为其编辑任何样式, 但是显示出来后已经被美化过了. 这是因为我引用了 jQuery UI 的主题:

```
<!--black-tie,blitzer,blitzer,dot-luv,excite-bike,hot-sneaks,humanity,mint-
choc,redmond,smoothness,south-street,start,swanky-purse,trontastic,ui-
darkness,ui-lightness,vader-->

<link rel="stylesheet" type="text/css"
href="<%=WebConfig.ResourceServer
+ "/JsLib/jquery/themes/redmond/style.css"%>" />
```

注释中有很多的主题, 只需要将引用路径中的"redmond"改为其中任何一个, 弹出层的样式会立刻发生变化.

VS 中有一个 Bug, 就是针对 link 标签, href 中的语句块编译有问题, 某些情况下<%%>不被编辑解析. 所以我使用上面代码中的方式构造 href 属性值.

可以在下面的地址查看各个主题的效果:

<http://jqueryui.com/themeroller/#themeGallery>

六.Tab 标签

不刷新页面, 在页面中的不同标签间切换:



本实例通过 jQuery UI 的 Tabs 组件实现. Tabs 组件的使用与 dialog 一样十分简单, 默认的配置即可实现最简单的 tab, 通过设置更多的 options 可以实现更复杂的应用.

1.应用实例

源代码:

```
<%@ Page Language="C#" %>
```

```
<!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 id="Head1" runat="server">
```

```
<title>jQuery UI - 弹出层应用实例 Dialog</title>
```

```
<!--black-tie,blitzer,blitzer,dot-luv,excite-bike,hot-sneaks,humanity,mint-  
choc,redmond,smoothness,south-street,start,swanky-purse,trontastic,ui-  
darkness,ui-lightness,vader-->
```

```
<link rel="stylesheet" type="text/css"  
href="<%=WebConfig.ResourceServer  
+ "/JsLib/jquery/themes/redmond/style.css"%>" />
```

```
<script type="text/javascript" src="<%=WebConfig.ResourceServer  
%/>/JsLib/jquery/jquery-min-lastest.js"></script>
```

```
<script src="<%=WebConfig.ResourceServer %>/JsLib/jquery/ui/jquery-  
ui-all-min-lastest.js"
```

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

```
<% if (false)
```

```
{%><script src="~/js/jquery-vsdoc-lastest.js"  
type="text/javascript"></script>
```

```
<% }%>
```

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

```
/*===== 必须放在头部加载的语句块，尽量避免使用
=====*/

</script>

<style type="text/css">

</style>

</head>

<body>

<!--Demo.默认 Tab 与 Ajax Tab -->

<div id="tabs1" style="width:300px;">

<ul>

<li><a href="#tabs1-1">One</a></li>

<!-- Ajax Tab -->

<li><a href="TabData.htm">Two</a></li>

<li><a href="#tabs1-3">Three</a></li>

</ul>

<div id="tabs1-1">

<p>Tab1内容</p>

</div>

<div id="tabs1-3">
```

<p>Tab3内容</p>

</div>

</div>

<!--Demo. 可折叠的 Tab -->

<div id="tabs2" style="width: 300px;">

One

Two

Three

<div id="tabs2-1">

<p>Tab1内容</p>

</div>

<div id="tabs2-2">

<p>Tab2内容</p>

</div>

<div id="tabs2-3">

<p>Tab3内容</p>

</div>

</div>

<!--Demo. 鼠标滑动即切换的 Tab -->

<div id="tabs3" style="width: 300px;">

One

Two

Three

<div id="tabs3-1">

<p>Tab1内容</p>

</div>

<div id="tabs3-2">

<p>Tab2内容</p>

</div>

<div id="tabs3-3">

<p>Tab3内容</p>

</div>

</div>

<script type="text/javascript">

/*=====用户自定义方法=====*/

/*=====事件绑定=====*/

\$(function()

{

});

/*=====加载时执行的语句=====*/

\$(function()

{

//默认 Tabs


```
$("#tabs1").tabs();

//可折叠的 Tabs

$("#tabs2").tabs({

    collapsible: true

});

//鼠标滑动即切换的 Tabs

$("#tabs3").tabs({

    event: "mouseover"

});

});

</script>

</body>

</html>
```

效果:

1. 默认设置的 **Tabs**, **Two** 标签内容使用 **Ajax** 获取





2.再折叠 tab



3.鼠标滑动即切换的 tab



2.要点讲解

(1) 注意 **Tabs** 中的 **HTML** 结构.

使用 **ul** 构建标签. 内容 **div** 一定要和标签关联, 没有关联的 **div** 将不被处理直接显示.

(2) 使用 **Ajax** 可以不指定内容容器, 但是也可以将 **Ajax** 内容放入指定容器中.

```
<li><a href="hello/world.html" title="Todo Overview"> ... </a></li>
```

```
<div id="Todo_Overview"> ... </div>
```

(3) 活用事件

tab 有很多事件:

select, load, show, add, remove, enable, disable

使用这些事件可以完成很多复杂任务. 需要注意事件的签名:

```
$('#example').bind('tabsselect', function(event, ui) {  
  
    // Objects available in the function context:
```

```
        ui.tab      // anchor element of the selected (clicked) tab

        ui.panel    // element, that contains the selected/clicked tab contents

        ui.index    // zero-based index of the selected (clicked) tab

    });
```

第一个是事件对象, 第二个 `ui` 对象是传递的额外参数, 我们可以获取 `tab` 对象, `tab` 所在容器和 `tab` 的索引值.

比如我们可以在事件中做验证:

```
$('#example').tabs({

    select: function(event, ui) {

        var isValid = ... // form validation returning true or false

        return isValid;

    }

});
```

或者当添加一个 `tab` 时立刻切换到选中状态:

```
var $tabs = $('#example').tabs({

    add: function(event, ui) {

        $tabs.tabs('select', '#' + ui.panel.id);

    }

});
```

活学活用, 更多应用大家也可以参见 tab 组件的官方文档:

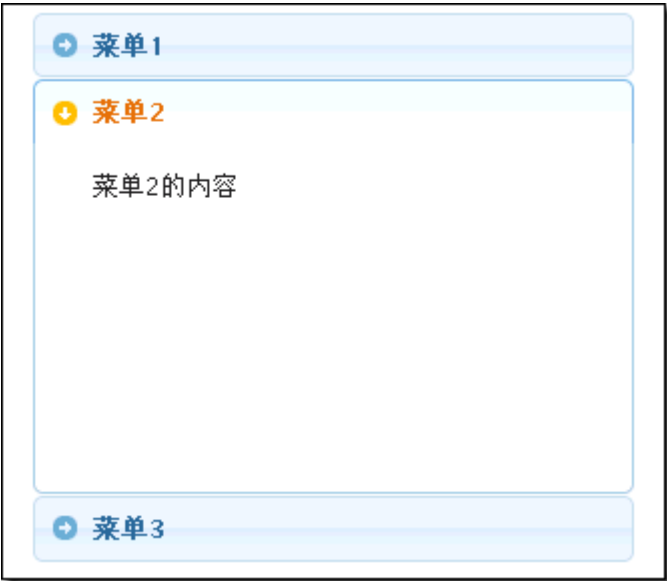
<http://jqueryui.com/demos/tabs>

七. 手风琴菜单

使用 jQuery UI 的 accordion 组件可以实现手风琴菜单. 效果见下图.

accordion 文档地址: <http://jqueryui.com/demos/accordion/>

1.实例效果



2.实例代码

```
<%@ Page Language="C#" %>
```

```
<!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 id="Head1" runat="server">
```

```
<title>jQuery UI - 手风琴菜单应用实例 Accordion </title>
```

```
<!--black-tie,blitzer,blitzer,dot-luv,excite-bike,hot-sneaks,humanity,mint-
choc,redmond,smoothness,south-street,start,swanky-purse,trontastic,ui-
darkness,ui-lightness,vader-->

<link rel="stylesheet" type="text/css"
href="<%=WebConfig.ResourceServer
+"/JsLib/jquery/themes/redmond/style.css"%>" />

<script type="text/javascript" src="<%=WebConfig.ResourceServer
%>/JsLib/jquery/jquery-min-lastest.js"></script>

<script src="<%=WebConfig.ResourceServer %>/JsLib/jquery/ui/jquery-
ui-all-min-lastest.js"

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

<% if (false)

{><script src="~/js/jquery-vsdoc-lastest.js"
type="text/javascript"></script>

<% }%>

<script type="text/javascript">

/*===== 必须放在头部加载的语句块。 尽量避免使用
=====*/

</script>

<style type="text/css">

body
```

```
{

    font-size: 12px;

}

</style>

</head>

<body>

    <!-- Demo. 默认配置的 Accordion 菜单 -->

    <div style="width: 300px; float:left; margin-left:20px;">

        <div id="accordion1">

            <h3><a href="#">菜单1</a></h3>

            <div>

                菜单1的内容<br />

                菜单1的内容<br />

                菜单1的内容<br />

                菜单1的内容<br />

                菜单1的内容<br />

                菜单1的内容<br />

                菜单1的内容<br />

            </div>

        </div>

    </div>

</body>

</html>
```

<h3>菜单2</h3>

<div>

菜单2的内容

</div>

<h3>菜单3</h3>

<div>

菜单3的内容

</div>

</div>

</div>

<!-- Demo. 取消自动高度, 可折叠 -->

<div style="width: 300px; float: left; margin-left: 20px;">

<div id="accordion2">

<h3>菜单1</h3>

<div>

菜单1的内容

菜单1的内容

菜单1的内容

菜单1的内容

菜单1的内容

菜单1的内容

菜单1的内容

</div>

<h3>菜单2</h3>

<div>

菜单2的内容

</div>

<h3>菜单3</h3>

<div>

菜单3的内容

</div>

</div>

</div>

<!-- Demo. 鼠标滑动触发, 自定义图标 -->

<div style="width: 300px; float: left; margin-left: 20px;">

<div id="accordion3">

<h3>菜单1</h3>

<div>

菜单1的内容

菜单1的内容

菜单1的内容

菜单1的内容

菜单1的内容

菜单1的内容

菜单1的内容

</div>

<h3>菜单2</h3>

<div>

菜单2的内容

</div>

<h3>菜单3</h3>

<div>

菜单3的内容

</div>

</div>

</div>

<script type="text/javascript">

```
/*=====用户自定义方法=====*/
```

```
/*=====事件绑定=====*/
```

```
$(function()
```

```
{
```

```
});
```

```
/*=====加载时执行的语句=====*/
```

```
$(function()
```

```
{
```

```
    //默认配置的 Accordion 菜单
```

```
    $("#accordion1").accordion();
```

```
    //取消自动高度, 可折叠
```

```
    $("#accordion2").accordion({
```

```
        autoHeight:false,
```

```
        collapsible: true
```

```
    });
```

```
//鼠标滑动触发, 自定义图标
```

```
$("#accordion3").accordion({  
  
    icons: {  
  
        header: "ui-icon-circle-arrow-e",  
  
        headerSelected: "ui-icon-circle-arrow-s"  
  
    },  
  
    event: "mouseover"  
  
});  
  
});
```

```
</script>
```

```
</body>
```

```
</html>
```

3. 关键点讲解

(1) 注意高度设置过小问题

当包含 accordion 控件的容器高度设计过小时, 在 FireFox3中在此容器后面的内容会被 accordion 控件部分遮盖. 在 IE 中没有此问题. 经检查是因为容器高度小于菜单高度导致. 所以我们在应用时应当注意不要将容器高度设置过小.

(2) 部分关键属性

autoHeight: 设置是否自动将内容高度设置为容器高度.

collapsible: 设置是否可折叠

一般上面两个配合使用, 以为折叠后肯定会改变菜单高度, 会导致 autoHeight 设置为 true 无效.

更多属性和事件使用请参阅官方文档.

八.总结

本章简单介绍了 jQueryUI, 并且使用 jQuery UI 完成了弹出层,tabs,手风琴菜单的应用实例. 使用 jQuery UI 可以不需要额外寻找插件. 并且实现简单.

但是有些功能是必须使用插件完成的. 下一张讲解两个插件实例: 自动完成插件 AutoComplete 和 表单验证插件 jQuery Validate.

本章源代码下载:

<http://files.cnblogs.com/zhangziqu/Code-jQueryStudy-10.rar>

Tag 标签: [jQuery](#),[jQuery 教程](#),[jQueryUI](#)

jQuery (十一) 实战表单验证与自动完成提示插件

一.摘要

本系列文章将带您进入 jQuery 的精彩世界, 其中有很多作者具体的使用经验和解决方案, 即使你会使用 jQuery 也能在阅读中发现些许秘籍.

本文是介绍两个最常用的 jQuery 插件. 分别用于表单验证和自动完成提示(类似 google suggest).

二.前言

研究别人的作品真是一件花时间而且痛苦的过程. 当然也和本人英文不好有关. 总觉得控件作者写了很多文档但是都不够系统, 需要深入研究很多的实例后才能了解作者的思路. 所以学习和研究一个插件需要很高成本, 如果发现了 Bug 并修复需要的成本也是未知数(本次我花了较少的时间解决了自动完成提示插件的一个中文 bug, 但是如果复杂的 bug 就不会这么简单了.).

对于简单应用我首先推荐上文中的 jQuery UI. 但是 jQuery UI 解决的问题有限. 使用 jQuery 插件是我们最后的一个好办法---还算是好办法, 起码比自己开发要好吧?

很多 jQuery 的插件编码异常优美, 看一看艺龙首页现在的城市输入框控件, 除了需要为输入框手工添加很多很多属性(onkeyup, onkeydown 等等), 而且还不够通用, 占用服务器资源和网络资源. 但是当初也是花费了很久的时间完成的作品.

站在巨人的肩膀上, 让我感觉写脚本和写设计 C#程序一样, 都有高度和深度可以挖掘. 除了使用作者开发好的功能, 还可以学习如何开发和封装 javascript 控件. 看过优秀的 jQuery 插件作者的代码和设计思想后, 常常自叹设计水平差距居然如此之大, 增加自认为脚本高手, 比较过后就是 C#程序员和架构师之间的差距.

希望大家通过本章节介绍的两个插件, 除了学会如何使用, 还能够略微领悟到如何封装和设计 javascript 控件.

三.表单验证插件 **validate**

在提交表单前常要对用户输入进行校验. ASP.NET 的验证控件就是用于此目的, 可以同时进行客户端和服务端验证. 但是验证控件并没有被所有项目采用. 而且在 MVC 项目中经常使用自己的客户端验证框架.

在比较了若干表单验证插件后, 决定采用 **validate** 插件. 因为其使用简单并且灵活.

插件首页:

<http://bassistance.de/jquery-plugins/jquery-plugin-validation/>

插件文档:

<http://docs.jquery.com/Plugins/Validation>

配置说明:

<http://docs.jquery.com/Plugins/Validation/validate#options>

1.应用实例

实例效果:

表单验证

* 姓名: a

请输入一个长度最少是 2 的字符串

* E-Mail: abc

电子邮箱格式不正确

网址:

* 内容:

必选字段

提交

实例代码:

```
<%@ Page Language="C#" %>
```

```
<!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 id="Head1" runat="server">

    <title>jQuery PlugIn - 表单验证插件实例 Validate </title>

    <!--black-tie,blitzer,blitzer,dot-luv,excite-bike,hot-sneaks,humanity,mint-
choc,redmond,smoothness,south-street,start,swanky-purse,trontastic,ui-
darkness,ui-lightness,vader-->

    <link rel="stylesheet" type="text/css"
```

href="<%=WebConfig.ResourceServer

+"/JsLib/jquery/themes/redmond/style.css"%>" />

<script type="text/javascript" src="<%=WebConfig.ResourceServer
%>/JsLib/jquery/jquery-min-lastest.js"></script>

<script type="text/javascript" src="<%=WebConfig.ResourceServer
%>/JsLib/jquery/ui/jquery-ui-all-min-lastest.js"></script>

<script type="text/javascript" src="<%=WebConfig.ResourceServer
%>/JsLib/jquery/plugin/jquery.validate/jquery.validate.min.js"></script>

<script type="text/javascript" src="<%=WebConfig.ResourceServer
%>/JsLib/jquery/plugin/jquery.validate/localization/messages_cn.js"></script>

<% if (false)

{%><script src="~/js/jquery-vsdoc-lastest.js"
type="text/javascript"></script>

<% }%>

<script type="text/javascript">

/*===== 必须放在头部加载的语句块。 尽量避免使用
=====*/

</script>

<style type="text/css">

body

```
{

    font-size:12px;

}

/* form 中显示文字的 label */

.slabel

{

    width:100px;

    display: -moz-inline-box;

    line-height: 1.8;

    display: inline-block;

    text-align:right;

}

/* 出错样式 */

input.error, textarea.error

{

    border: solid 1px #CD0A0A;

}

label.error

{
```



```
        color:#CD0A0A;

        margin-left:5px;

    }

    /* 深红色文字 */

    .textred

    {

        color:#CD0A0A;

    }

</style>

</head>

<body>

    <form id="commentForm" method="get" action="">

        <fieldset style="width:500px;"><legend>表单验证</legend>

            <p><label                for="cname"                class="slabel"><em

class="textred">*</em> 姓名:</label>

                <input id="cname" name="name" size="25" class="required"

minlength="2" />

            </p>

            <p><label                for="cemail"                class="slabel"><em

class="textred">*</em> E-Mail:</label>
```

```
<input id="cemail" name="email" size="25"/>

</p>

<p><label for="curl" class="slabel">网址:</label>

    <input id="curl" name="url" size="25" class="url" value=""

/>

</p>

<p><label          for="ccomment"          class="slabel"><em

class="textred">*</em> 内容:</label>

    <textarea      rows="2"      id="ccomment"      name="comment"

cols="20" class="required" style="height:80px;"></textarea>

</p>

<p style="text-align:center;">

    <input class="submit" type="submit" value="提交" />

</p>

</fieldset>

</form>

<script type="text/javascript">

    /*=====用户自定义方法=====*/
```

```
/*=====事件绑定=====*/
```

```
$(function()
```

```
{
```

```
});
```

```
/*=====加载时执行的语句=====*/
```

```
$(function()
```

```
{
```

```
    $("#commentForm").validate({
```

```
    {
```

```
        errorClass: "error",
```

```
        submitHandler: function(form)
```

```
        {
```

```
            //如果想提交表单，需要使用 form.submit()而不要使用
```

```
            $(form).submit()
```

```
            alert("submitted!");
```

```
        },
```

```
        rules: {
```

```

        //为 name 为 email 的控件添加两个验证方法:required()和
        email()

        email: { required: true, email: true }

    },

    messages: {

        //为 name 为 email 的控件的 required()和 email()验证方法
        设置验证失败的消息内容

        email: {required:"需要输入电子邮箱", email:"电子邮箱
        格式不正确"}

    }

});

});

</script>

</body>

</html>

```

2. 实例讲解

(1) 验证方法

验证方法是验证某一个控件是否满足某些规则的方法, 返回一个 boolean 值. 比如 [email\(\)](#) 方法验证内容是否符合 email 格式, 符合则返回 true. 下面是类库中 email 方法的源代码:

```

// http://docs.jquery.com/Plugins/Validation/Methods/email

```

```
email: function(value, element) {
```

```
// contributed by Scott Gonzalez:
```

```
http://projects.scottsp playground.com/email\_address\_validation/
```

```
return this.optional(element) || /^[^([a-z]|\d|[\!#\$\%&'\*\+\-\
\/=\?\^_\{\}~]][\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-
\uFFEF])+(\.([a-z]|\d|[\!#\$\%&'\*\+\-\
\/=\?\^_\{\}~]][\u00A0-
uD7FF\uF900-\uFDCF\uFDF0-
\uFFEF])+)*)(((\x22)((((\x20|\x09)*(\x0d\x0a))?(\x20|\x09)+)?(((\x
01-\x08\x0b\x0c\x0e-\x1f\x7f)|\x21|[\x23-\x5b]|[\x5d-
\x7e]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(\x01-
\x09\x0b\x0c\x0d-\x7f)|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-
\uFFEF]))))*(((\x20|\x09)*(\x0d\x0a))?(\x20|\x09)+)?(\x22))@((([a-
z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|([a-
z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-
|\.|\_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-
z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF]))\.)+((([a-
z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|([a-z]|[\u00A0-
uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-z]|\d|-|\.|\_|~|[\u00A0-
uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])*([a-z]|[\u00A0-
uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))\.$/i.test(value);

},
```

我们在:

<http://docs.jquery.com/Plugins/Validation>

中的 **List of built-in Validation methods** 一节中列出了所有内置的验证方法。同时插件还提供了 additional-methods.js 文件，里面包含了更多的验证方法，引入后既可启用。

(2) 验证消息

验证消息就是验证方法失败后显示的文字内容。验证消息一定关联在某一个验证方法上，并且

全局的验证消息保存在 jQuery.validator.messages 属性中.

默认的 validate 类库自带英文验证消息:

```
messages: {  
  
    required: "This field is required.",  
  
    //...  
  
});
```

上面说明当 required 验证方法验证失败是, 显示"This field is required."这条消息.

在下载文件的 localization 文件夹中, 包含了各国语言的基本验证消息, 如同本实例一样引入不同的语言文件即可实现语言切换:

```
<script      type="text/javascript"      src="<%      =WebConfig.ResourceServer  
%>/JsLib/jquery/plugin/jquery.validate/localization/messages_cn.js"></script>
```

语言文件的内容举例:

```
jQuery.extend(jQuery.validator.messages, {  
  
    required: "必选字段",  
  
    //...  
  
});
```

现在必填项的问题提示就变成了中文.

除了全局默认的验证消息, 也可以为某一个表单元素设置特有的验证消息, 比如本文实例中, 为 email 元素设置了特有的验证消息:

```
messages: {  
  
    //为 name 为 email 的控件的 required()和 email()验证方法
```

设置验证失败的消息内容

```
email: {required:"需要输入电子邮箱", email:"电子邮箱  
格式不正确"}
```

options 的 messages 属性可以针对某一个表单元素设置验证消息，第一个 email 表示 email 元素，值是一个集合, required 就表示 required 验证函数，第二个 email 表示是 email 验证函数.

(3)验证规则

验证规则就是这样的语意语句: 在元素 A 上, 使用 验证方法 A 和 验证方法 B 进行验证.

验证规则将元素与验证方法关联起来, 因为验证方法同时也关联了验证消息, 所以元素与消息也关联了起来.

为一个元素添加验证规则有多种方式.

本实例的"姓名"元素使用了 CSS 样式规则和元素属性规则:

```
<input id="cname" name="name" size="25" class="required" minlength="2"  
/>
```

class 元素属性设置元素的 CSS 样式类, 因为样式类中添加了 required 类, 所以会和 required() 验证函数关联. 这种规则叫做 CSS 样式规则.

minlength 元素属性也会自动和 minlength()验证函数关联, 这种规则叫做元素属性规则.

另外还可以通过编程的方式进行关联:

```
rules: {  
  
    //为 name 为 email 的控件添加两个验证方法:required()和  
    email()  
  
    email: { required: true, email: true }  
  
    },
```

上面的语句表名为 email 表单对象添加了 required()和 email()验证函数.

(4) 表单提交

默认情况下, 当验证函数失败时表单不会提交.

但是可以通过添加 class="cancel"的方式让提交按钮跳过验证:

```
<input type="submit" class="cancel" name="cancel" value="Cancel" />
```

当表单提交时, 会触发 options 中 submitHandler 属性设置的函数:

```
submitHandler: function(form)

{

    //如果想提交表单, 需要使用 form.submit()而不要使用
    $(form).submit()

    alert("submitted!");

},
```

此函数的签名同上. 我们可以在这个函数中, 编写在表单提交前需要处理的业务逻辑.

需要注意当最后以编程的方式提交表单时, 一定不要使用 jQuery 对象的 submit()方法, 因为此方法会触发表单验证,并且再次调用 submitHandler 设置的函数, 会导致递归调用.

此函数的参数 form 就是表单对象, 用途就是不进行验证提交表单:form.submit()

(5) DEBUG 模式

在开发阶段我们通常不希望表单被真正提交, 虽然可以通过本实例中重写 submitHandler 函数来实现, 但是还有更好的方式, 我们可以在 submitHandler 函数完成正式提交的逻辑, 然后通过设置 options 的 debug 属性, 来达到即使验证通过也不会提交表单的目的:

```
$(".selector").validate({

    debug: true

})
```

(6) 多表单验证

有时会在一个页面上出现多个 Form, 因为 validate 控件是针对 form 对象进行包装的, 所以我们可以控制哪些 form 对象需要验证.

同时为了方便一次设置页面上所有的应用了 validate 控件的 form 对象, 提供了 jQuery.validator.setDefaults 函数让我们可以一次设置所有的默认值:

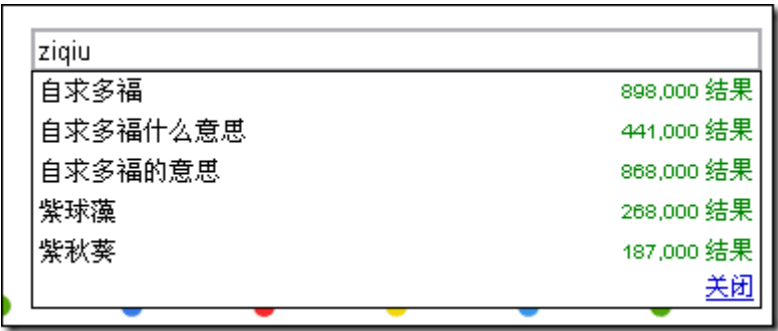
```
jQuery.validator.setDefaults({
```


debug: true

});

四.自动完成插件 autocomplete

autocomplete 插件能帮助我们实现类似于 Google Suggest 的效果:



插件首页:

<http://bassistance.de/jquery-plugins/jquery-plugin-autocomplete/>

插件文档:

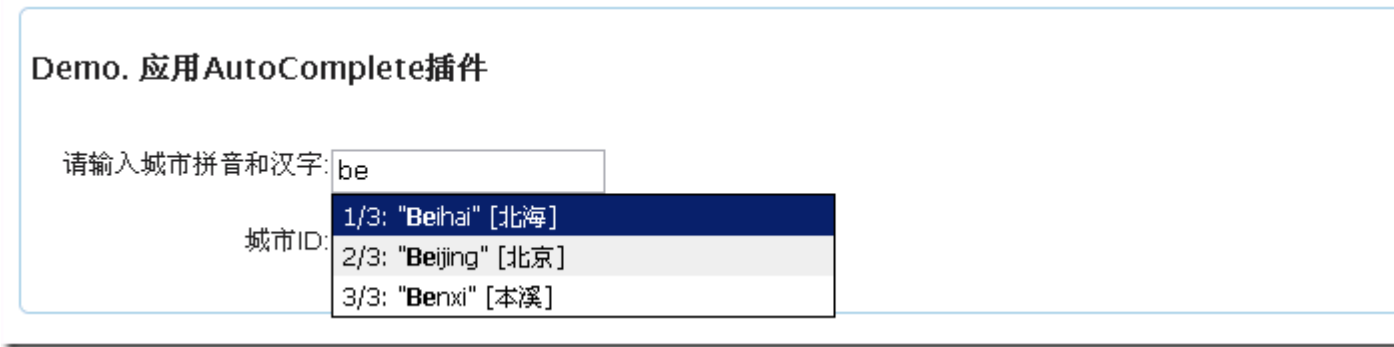
<http://docs.jquery.com/Plugins/Autocomplete>

配置说明:

<http://docs.jquery.com/Plugins/Autocomplete/autocomplete#toptions>

1.应用实例

本实例演示的是使用 autocomplete 完成对输入城市的自动提示效果,如图:



实例代码:

```
<%@ Page Language="C#" %>
```

```
<!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 id="Head1" runat="server">
```

```
<title>jQuery PlugIn - 自动完成插件实例 AutoComplete </title>
```

```
<!--black-tie,blitzer,blitzer,dot-luv,excite-bike,hot-sneaks,humanity,mint-  
choc,redmond,smoothness,south-street,start,swanky-purse,trontastic,ui-  
darkness,ui-lightness,vader-->
```

```
<link rel="stylesheet" type="text/css"  
href="<%=WebConfig.ResourceServer  
+ "/JsLib/jquery/themes/redmond/style.css"%>" />
```

```
<link rel="stylesheet" type="text/css"  
href="<%=WebConfig.ResourceServer  
+ "/JsLib/jquery/plugin/jquery.autocomplete/jquery.autocomplete.css"%>" />
```

```
<script type="text/javascript" src="<%=WebConfig.ResourceServer  
%>/JsLib/jquery/jquery-min-lastest.js"></script>
```

```
<script type="text/javascript" src="<%=WebConfig.ResourceServer  
%>/JsLib/jquery/ui/jquery-ui-all-min-lastest.js"></script>
```

```
<script type="text/javascript" src="<%=WebConfig.ResourceServer  
%>/JsLib/jquery/plugin/jquery.autocomplete/jquery.autocomplete.min.js"></scri  
pt>
```

```
<% if (false)
```

```
{%}<script                                src="~/js/jquery-vsdoc-lastest.js"
type="text/javascript"></script>

<% }%>

<script type="text/javascript">

    /*===== 必须放在头部加载的语句块。尽量避免使用
=====*/

</script>

<style type="text/css">

    body

    {

        font-size: 12px;

    }

    .formLabel{float: left; width: 150px; text-align:right;}

    .formInput{float: left;}

</style>

</head>

<body>

    <!-- Demo. 应用 AutoComplete 插件 -->

    <div class="ui-widget ui-widget-content ui-corner-all" style="width:
```

700px; padding: 5px;">

<h3>

Demo. 应用 AutoComplete 插件 </h3>

<br style="clear: both" />

<div class="formLabel">

<label for="inputCityName">请输入城市拼音和汉字:</label>

</div>

<div class="formInput">

<input id="inputCityName" name="inputCityName" type="text"

/>

</div>

<br style="clear:both" />

<br style="clear: both" />

<div class="formLabel">

<label for="inputCityName">城市 ID:</label></div>

<div class="formInput">

<input id="inputCityId" name="inputCityId" type="text"

/></div>

<br style="clear: both" />

<br style="clear: both" />

</div>

<script type="text/javascript">

/*=====用户自定义方法=====*/

//城市数据

var cityList;

//autocomplete 选项

var options = {

minChars: 1,

max: 500,

width: 250,

matchContains: true,

formatItem: function(row, i, max)

{

return i + "/" + max + ": \" + row.CityNameEn +

\" [\" + row.CityName + \"]\";

},

formatMatch: function(row, i, max)

{

return row.CityNameEn + " " + row.CityName;

```
    },

    formatResult: function(row)

    {

        return row.CityName;

    }

};

//autocomplete 初始化函数

function initAutoComplete(data)

{

    cityList = data;

    $("#inputCityName").autocomplete(cityList, options);

    $("#inputCityName").result(function(event, data, formatted)

    {

        $("#inputCityId").val(data.ElongCityId);

    });

}

/*=====事件绑定=====*/
```

```
$(function()

{

});

/*=====加载时执行的语句=====*/

$(function()

{

    //加载城市数据，并在回调函数中用返回的数据初始化
autocomplete

$.getJSON("cityinfo.htm", null, initAutoComplete)

});

</script>

</body>

</html>
```

2. 实例讲解

(1)准备数据源

首先要准备实现自动建议的数据源。本实例是通过发送 Ajax 请求获取 JSON 对象。autocomplete() 方法支持两个参数，第一个是 data，第二个是 options。

其中 data 参数可以使本实例中的一个数据变量，也可以是一个 url。如果是 url 则会每次都调用 Ajax 请求获取数据。

为了效率我倾向于在数据量允许的情况下，在页面加载后使用 Ajax 获取全部的数据，然后使用传递数据变量给 autocomplete 组件。如实例中所示。除非数据特别巨大无法再客户端加载，则只能每次都使用发送 Ajax 请求从服务器端获取部分数据。但是这会对服务器造成负担。

(2) 设置关键函数

虽然 options 是可选项, 但是对于我们的数据源 cityList 是一个多属性对象, 所以必须设置下面几个关键的配置项后才能够使用:

formatItem

对匹配的每一行数据使用此函数格式化, 返回值是显示给用户的数据内容.

函数签名:

function(row, rowNum, rowCount, searchItem)

参数说明:

row: 当前行. the results row,

rowNum: 当前行号,从1开始.(注意不是索引,索引从0开始) the position of the row in the list of results (starting at 1),

rowCount: 总的行号 the number of items in the list of results

searchItem: 查询使用的数据, 即 formatMatch 函数返回的数据格式的内容. 我们在 formatMatch 函数中会设置程序内部搜索时使用的数据格式,这个格式和给用户展示的数据是不同的.

formatMatch

对每一行数据使用此函数格式化需要查询的数据格式. 返回值是给内部搜索算法使用的. 实例中用户看到的匹配结果是 formatItem 中设置的格式, 但是程序内部其实只搜索城市的英文和中文名称, 搜索数据在 formatMatch 中定义:

return row.CityNameEn + " " + row.CityName;

函数签名:

function(row, rowNum, rowCount,)

参数说明同上

formatResult

此函数是用户选中后返回的数据格式. 比如实例中只返回城市名给 input 控件:

return row.CityName;

函数签名:

function(row, rowNum, rowCount,)

参数说明同上

(3) 为控件添加 Result 事件函数

上面3个函数无法实现这类要求: 虽然只返回城市名称, 但是查询时使用城市 ID, 选中一个城市后需要将城市 ID 存储在一个隐藏域中.

所以 autocomplete 控件提供了 result 事件函数, 此事件会在用户选中某一项后触发:

```
$("#inputCityName").result(function(event, data, formatted)

{

    $("#inputCityId").val(data.ElongCityId);

});
```

函数签名:

```
function(event, data, formatted)
```

参数列表:

Result 事件会为绑定的事件处理函数传递三个参数:

event: 事件对象. event.type 为 result.

data: 选中的数据行.

formatted: 虽然官方的解释应该是 formatResult 函数返回的值, 但是实验结果是 formatMatch 返回的值. 在本实例为: "Beijing 北京".

(4) 匹配中文

当前版本的 autocomplete 控件对中文搜索存在 Bug, 原因是其搜索事件绑定在 keydown 事件上, 当使用中文输入法输入"北"字时没有任何提示. 我对原库做了修改, 将 keydown 事件修改为 keyup 事件, 即可完成对中文的智能提示搜索. 另外主要需要将"matchContains"配置项设置为 "true", 因为我们的搜索格式是"Beijing 北京", 默认只匹配开头的字符.

(5) 更多配置项

关于更多的配置项, 请参考官方文档:

<http://docs.jquery.com/Plugins/Autocomplete/autocomplete#toptions>

(6) 更多事件

除了上面介绍的 autocomplete()和 result()函数, 还有如下函数:

[search\(\)](#) : 激活 search 事件

[flushCache\(\)](#) : 清空缓存

[setOptions\(options \)](#) : 设置配置项

五.总结

本文详细介绍了表单验证插件和自动完成插件，目前大家可以搜索到很多的插件应用，或者上千的插件列表，但是却找不到详细的使用文档。插件用起来简单但是真正的灵活应用却不容易，除了要翻越英文文档学习基本的使用，还要花很长时间了解各个参数的作用，如何配合使用等。并且在上面做二次开发的难度相对较大，插件的核心代码多没有注释并且复杂，要在其中寻找逻辑关系要花费很多时间和精力。本文介绍的两个插件更多的细节请参考官方文档，地址都在一开始为大家提供了。

后续文章我决定先进行 jQuery 技巧和 javascript 必备知识的讲解，我们很少开发自定义插件所以将开发插件篇放在最后。

本章节代码下载:

<http://files.cnblogs.com/zhangziqiu/Code-jQueryStudy-11.rar>

Tag 标签: [jQuery](#),[jQuery 教程](#),[jQueryUI](#),[validate](#),[autocomplete](#)

jQuery Ajax 全解析

本文地址: [jQuery Ajax 全解析](#)

本文作者: [QLeelulu](#)

转载请标明出处!

jQuery 确实是一个挺好的轻量级的 JS 框架，能帮助我们快速的开发 JS 应用，并在一定程度上改变了我们写 JavaScript 代码的习惯。

废话少说，直接进入正题，我们先来看一些简单的方法，这些方法都是对 jQuery.ajax() 进行封装以方便我们使用的方法，当然，如果要处理复杂的逻辑，还是需要用到 jQuery.ajax() 的(这个后面会说到)。

1. load(url, [data], [callback]) : 载入远程 HTML 文件代码并插入至 DOM 中。

url(String): 请求的 HTML 页的 URL 地址。

data(Map): (可选参数) 发送至服务器的 key/value 数据。

callback(Callback): (可选参数) 请求完成时(不需要是 success 的)的回调函数。

这个方法默认使用 GET 方式来传递的，如果[data]参数有传递数据进去，就会自动转换为 POST 方式的。jQuery 1.2 中，可以指定选择符，来筛选载入的 HTML 文档，DOM 中将仅插入筛选出的 HTML 代码。语法形如 "url #some > selector"。

这个方法可以很方便的动态加载一些 HTML 文件，例如表单。

示例代码:

```
$("#ajax.load").load("http://www.cnblogs.com/QLeelulu/archive/2008/03/30/1130270.html .post",
```

```
function (responseText, textStatus, XMLHttpRequest){
```

```
this;//在这里 this 指向的是当前的 DOM 对象，即$("#ajax.load")[0]
```

```
//alert(responseText);//请求返回的内容
```

```
//alert(textStatus);//请求状态: success, error
```

```
//alert(XMLHttpRequest);//XMLHttpRequest 对象
```

```
});
```

这里将显示结果。

注: 不知道为什么 URL 写绝对路径在 FF 下会出错，知道的麻烦告诉下。下面的 get() 和 post() 示例使用的是绝对路径，所以在 FF 下你将会出错并不会看到返回结果。还有 get() 和 post() 示例都是跨域调用的，发现传上来后没办法获取结果，所以把运行按钮去掉了。

2. `jQuery.get(url, [data], [callback])`: 使用 GET 方式来进行异步请求

参数:

url(String): 发送请求的 URL 地址.

data(Map): (可选) 要发送给服务器的数据, 以 Key/value 的键值对形式表示, 会做为 QueryString 附加到请求 URL 中。

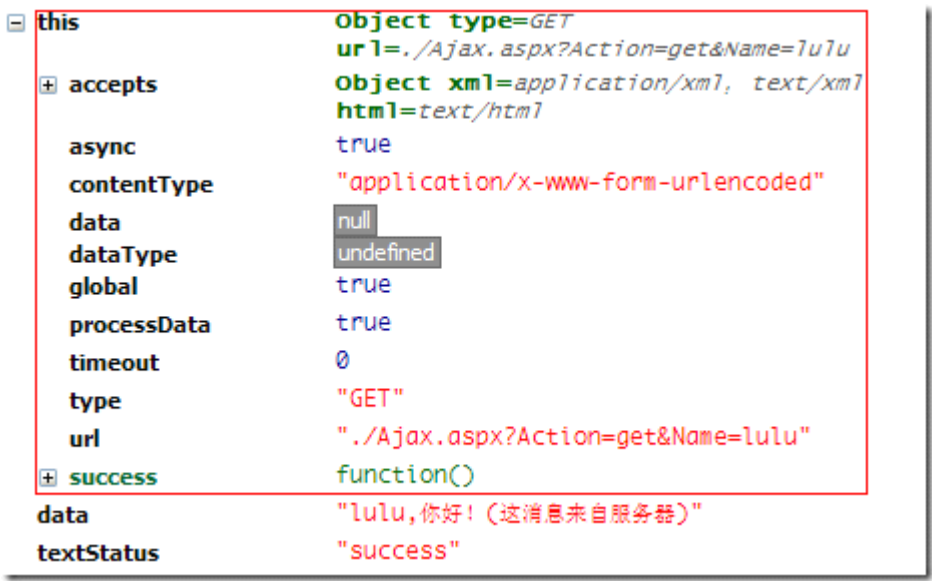
callback(Function): (可选) 载入成功时回调函数(只有当 Response 的返回状态是 success 才是调用该方法)。

这是一个简单的 GET 请求功能以取代复杂 `$ajax` 。请求成功时可调用回调函数。如果需要在出错时执行函数, 请使用 `$ajax`。示例代码:

```
$get("./Ajax.aspx", {Action:"get",Name:"lulu"}, function (data, textStatus){  
  
    //返回的 data 可以是 xmlDoc, jsonObj, html, text, 等等.  
  
    this; // 在这里 this 指向的是 Ajax 请求的选项配置信息, 请参考下图  
  
    alert(data);  
  
    //alert(textStatus); //请求状态: success, error 等等。  
    当然这里捕捉不到 error, 因为 error 的时候根本不会运行  
    该回调函数  
  
    //alert(this);  
  
});
```

点击发送请求:

`jQuery.get()`回调函数里面的 `this` , 指向的是 Ajax 请求的选项配置信息:



3. `jQuery.post(url, [data], [callback], [type])` : 使用 POST 方式来进行异

步请求

参数:

url(String): 发送请求的 URL 地址.

data(Map): (可选) 要发送给服务器的数据, 以 Key/value 的键值对形式表示。

callback(Function): (可选) 载入成功时回调函数(只有当 Response 的返回状态是 success 才是调用该方法)。

type (String): (可选)官方的说明是: Type of data to be sent。其实应该为客户端请求的类型 (JSON,XML,等等)

这是一个简单的 POST 请求功能以取代复杂 \$.ajax 。请求成功时可调用回调函数。如果需要
在出错时执行函数, 请使用 \$.ajax。示例代码:

Ajax.aspx:

```
Response.ContentType = "application/json";
```

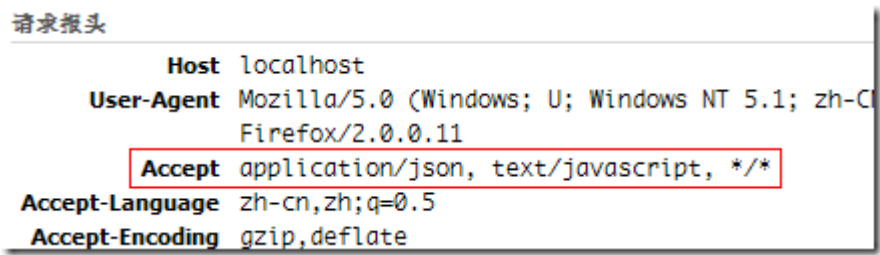
```
Response.Write("{result: '" + Request["Name"] + "',你好! (这消息来自服务器)}");
```

jQuery 代码:

```
$.post("Ajax.aspx", { Action: "post", Name: "lulu" },  
  
    function (data, textStatus){  
  
        // data 可以是 xmlDoc, jsonObj, html, text, 等等.  
  
        //this; // 这个 Ajax 请求的选项配置信息, 请参考 jQuery.get()说到的 this  
  
        alert(data.result);  
  
    }, "json");
```

点击提交:

这里设置了请求的格式为"json":



如果你设置了请求的格式为"json", 此时你没有设置 Response 回来的 ContentType 为: Response.ContentType = "application/json"; 那么你将无法捕捉到返回的数据。

注意一下, alert(data.result); 由于设置了 Accept 报头为“json”, 这里返回的 data 就是一个对象, 并不需要用 eval()来转换为对象。

4. jQuery.getScript(url, [callback])：通过 GET 方式请求载入并执行一个 JavaScript 文件。

参数

url(String)：待载入 JS 文件地址。

callback(Function)：(可选) 成功载入后回调函数。

jQuery 1.2 版本之前，getScript 只能调用同域 JS 文件。1.2中，您可以跨域调用 JavaScript 文件。注意：Safari 2 或更早的版本不能在全局作用域中同步执行脚本。如果通过 getScript 加入脚本，请加入延时函数。

这个方法可以用在例如当只有编辑器 focus()的时候才去加载编辑器需要的 JS 文件.下面看一些示例代码：

加载并执行 test.js。

jQuery 代码：

```
$.getScript("test.js");
```

加载并执行 AjaxEvent.js ，成功后显示信息。

jQuery 代码：

```
$.getScript("AjaxEvent.js", function(){
    alert("AjaxEvent.js 加载完成并执行完成.你再点击上面的 Get 或 Post 按钮看看有什么不同? ");
});
```

加载完后请重新点击一下上面的 Load 请求看看有什么不同。

jQuery Ajax 事件

Ajax 请求会产生若干不同的事件，我们可以订阅这些事件并在其中处理我们的逻辑。在 jQuery 这里有两种 Ajax 事件：局部事件 和 全局事件。

局部事件就是在每次的 Ajax 请求时在方法内定义的，例如：

```
$.ajax({
    beforeSend: function(){
        // Handle the beforeSend event
    },
```

```
complete: function(){

    // Handle the complete event

}

// ...

});
```

全局事件是每次的 Ajax 请求都会触发的，它会向 DOM 中的所有元素广播，在上面 `getScript()` 示例中加载的脚本就是全局 Ajax 事件。全局事件可以如下定义：

```
$("#loading").bind("ajaxSend", function(){

    $(this).show();

}).bind("ajaxComplete", function(){

    $(this).hide();

});
```

或者：

```
$("#loading").ajaxStart(function(){

    $(this).show();

});
```

我们可以在特定的请求将全局事件禁用，只要设置下 `global` 选项就可以了：

```
$.ajax({

    url: "test.html",

    global: false, // 禁用全局 Ajax 事件.

    // ...

});
```

下面是 jQuery 官方给出的完整的 Ajax 事件列表：

- **ajaxStart** (Global Event)
This event is broadcast if an Ajax request is started and no other Ajax requests are currently running.
- **beforeSend** (Local Event)
This event, which is triggered before an Ajax request is started, allows you to modify the XMLHttpRequest object (setting additional headers, if need be.)
- **ajaxSend** (Global Event)
This global event is also triggered before the request is run.
- **success** (Local Event)
This event is only called if the request was successful (no errors from the server, no errors with the data).
- **ajaxSuccess** (Global Event)

This event is also only called if the request was successful.

- **error** (Local Event)

This event is only called if an error occurred with the request (you can never have both an error and a success callback with a request).

- **ajaxError** (Global Event)

This global event behaves the same as the local error event.

- **complete** (Local Event)

This event is called regardless of if the request was successful, or not. You will always receive a complete callback, even for synchronous requests.

- **ajaxComplete** (Global Event)

This event behaves the same as the complete event and will be triggered every time an Ajax request finishes.

- **ajaxStop** (Global Event)

This global event is triggered if there are no more Ajax requests being processed.

具体的全局事件请参考 API 文档。
好了，下面开始说 jQuery 里面功能最强的 Ajax 请求方法 \$.ajax();

jQuery.ajax(options): 通过 HTTP 请求加载远程数据

这个是 jQuery 的底层 AJAX 实现。简单易用的高层实现见 \$.get, \$.post 等。

\$.ajax() 返回其创建的 XMLHttpRequest 对象。大多数情况下你无需直接操作该对象，但特殊情况下可用于手动终止请求。

注意： 如果你指定了 dataType 选项，请确保服务器返回正确的 MIME 信息，(如 xml 返回 "text/xml")。错误的 MIME 类型可能导致不可预知的错误。见 [Specifying the Data Type for AJAX Requests](#) 。
当设置 datatype 类型为 'script' 的时候，所有的远程(不在同一个域中)POST 请求都回转换为 GET 方式。

\$.ajax() 只有一个参数: 参数 key/value 对象，包含各配置及回调函数信息。详细参数选项见下。

jQuery 1.2 中，您可以跨域加载 JSON 数据，使用时需将数据类型设置为 [JSONP](#)。使用 [JSONP](#) 形式调用函数时，如 "myurl?callback=?" jQuery 将自动替换 ? 为正确的函数名，以执行回调函数。数据类型设置为 "jsonp" 时，jQuery 将自动调用回调函数。(这个我不是很懂)

参数列表:

参数名	类型	描述
url	String	(默认: 当前页地址) 发送请求的地址。
type	String	(默认: "GET") 请求方式 ("POST" 或 "GET"), 默认为 "GET"。注意: 其它 HTTP 请求方法, 如 PUT 和 DELETE 也可以使用, 但仅部分浏览器支持。
timeout	Number	设置请求超时时间 (毫秒)。此设置将覆盖全局设置。
async	Boolean	(默认: true) 默认设置下, 所有请求均为异步请求。如果需要发送同步请求, 请将此选项设置为 false。注意, 同步请求将锁住浏览器, 用户其它操作必须等待请求完成才可以执行。
beforeSend	Function	发送请求前可修改 XMLHttpRequest 对象的函数, 如添加自定义 HTTP 头。XMLHttpRequest 对象是唯一的参数。 function (XMLHttpRequest) {

		<pre>this; // the options for this ajax request }</pre>
cache	Boolean	(默认: true) jQuery 1.2 新功能, 设置为 false 将不会从浏览器缓存中加载请求信息。
complete	Function	请求完成后回调函数 (请求成功或失败时均调用)。参数: XMLHttpRequest 对象, 成功信息字符串。 <pre>function (XMLHttpRequest, textStatus) { this; // the options for this ajax request }</pre>
contentType	String	(默认: "application/x-www-form-urlencoded") 发送信息至服务器时内容编码类型。默认值适合大多数应用场合。
data	Object, String	发送到服务器的数据。将自动转换为请求字符串格式。GET 请求中将附加在 URL 后。查看 processData 选项说明以禁止此自动转换。必须为 Key/Value 格式。如果为数组, jQuery 将自动为不同值对应同一个名称。如 {foo:["bar1", "bar2"]} 转换为 '&foo=bar1&foo=bar2'。
dataType	String	预期服务器返回的数据类型。如果不指定, jQuery 将自动根据 HTTP 包 MIME 信息返回 responseXML 或 responseText, 并作为回调函数参数传递, 可用值: "xml": 返回 XML 文档, 可用 jQuery 处理。 "html": 返回纯文本 HTML 信息; 包含 script 元素。 "script": 返回纯文本 JavaScript 代码。不会自动缓存结果。 "json": 返回 JSON 数据 。 "jsonp": JSONP 格式。使用 JSONP 形式调用函数时, 如 "myurl?callback=?" jQuery 将自动替换 ? 为正确的函数名, 以执行回调函数。
error	Function	(默认: 自动判断 (xml 或 html)) 请求失败时将调用此方法。这个方法有三个参数: XMLHttpRequest 对象, 错误信息, (可能) 捕获的错误对象。 <pre>function (XMLHttpRequest, textStatus, errorThrown) { // 通常情况下 textStatus 和 errorThrown 只有其中一个有值 this; // the options for this ajax request }</pre>
global	Boolean	(默认: true) 是否触发全局 AJAX 事件。设置为 false 将不会触发全局 AJAX 事件, 如 ajaxStart 或 ajaxStop 。可用于控制不同的 Ajax 事件
ifModified	Boolean	(默认: false) 仅在服务器数据改变时获取新数据。使用 HTTP 包 Last-Modified 头信息判断。
processData	Boolean	(默认: true) 默认情况下, 发送的数据将被转换为对象(技术上讲并非字符串) 以配合默认内容类型 "application/x-www-form-urlencoded"。如果要发送 DOM 树信息或其它不希望转换的信息, 请设置为 false。

success	Function	请求成功后回调函数。这个方法有两个参数：服务器返回数据，返回状态 function (data, textStatus) { // data could be xmlDoc, jsonObj, html, text, etc... this; // the options for this ajax request }
---------	----------	--

这里有几个 Ajax 事件参数：**beforeSend**, **success**, **complete** , **error** 。我们可以定义这些事件来很好的处理我们的每一次的 Ajax 请求。注意一下，这些 Ajax 事件里面的 **this** 都是指向 Ajax 请求的选项信息的(请参考说 `get()` 方法时的 `this` 的图片)。
请认真阅读上面的参数列表，如果你要用 jQuery 来进行 Ajax 开发，那么这些参数你都必需熟知的。

示例代码，获取博客园首页的文章题目：

```
$.ajax({

    type: "get",

    url: "http://www.cnblogs.com/rss",

    beforeSend: function(XMLHttpRequest){

        //ShowLoading();

    },

    success: function(data, textStatus){

        $(".ajax.ajaxResult").html("");

        $(".item",data).each(function(i, domEle){

            $(".ajax.ajaxResult").append("<li>"+$(domEle).children("title").text()+"</li>");

        });

    },

    complete: function(XMLHttpRequest, textStatus){

        //HideLoading();

    },

    error: function(){

        //请求出错处理

    }

})
```

```
});
```

这里将显示首页文章列表。

其他

jQuery.ajaxSetup(options)：设置全局 AJAX 默认选项。

设置 AJAX 请求默认地址为 `"/xmlhttp/"`，禁止触发全局 AJAX 事件，用 POST 代替默认 GET 方法。其后的 AJAX 请求不再设置任何选项参数。

jQuery 代码：

```
$.ajaxSetup({  
  
    url: "/xmlhttp/",  
  
    global: false,  
  
    type: "POST"  
  
});  
  
$.ajax({ data: myData });
```

serialize() 与 serializeArray()

`serialize()`：序列表表格内容为字符串。

`serializeArray()`：序列化表格元素 (类似 `'serialize()'` 方法) 返回 JSON 数据结构数据。

示例：

HTML 代码：

```
<p id="results"><b>Results:</b></p>  
  
<form>  
  
    <select name="single">  
  
        <option>Single</option>  
  
        <option>Single2</option>  
  
    </select>  
  
    <select name="multiple" multiple="multiple">  
  
        <option selected="selected">Multiple</option>
```

```
<option>Multiple2</option>

<option selected="selected">Multiple3</option>

</select><br/>

<input type="checkbox" name="check" value="check1"/> check1

<input type="checkbox" name="check" value="check2"

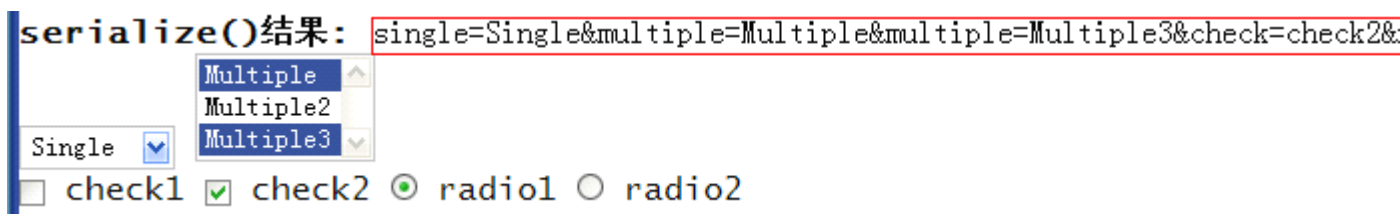
checked="checked"/> check2

<input type="radio" name="radio" value="radio1"

checked="checked"/> radio1

<input type="radio" name="radio" value="radio2"/> radio2

</form>
```



serializeArray() 结果为:

json	[Object name=single value=Single, Object name=multiple value=Multiple, Object name=multiple value=Multiple3, 2 more...]
0	Object name=single value=Single
name	"single"
value	"Single"
1	Object name=multiple value=Multiple
name	"multiple"
value	"Multiple"
2	Object name=multiple value=Multiple3
name	"multiple"
value	"Multiple3"
3	Object name=check value=check2
name	"check"
value	"check2"
4	Object name=radio value=radio1
name	"radio"
value	"radio1"

一些资源

一个 jQuery 的 Ajax Form 表单插件: <http://www.malsup.com/jquery/form/>

一个专门生成 Loading 图片的站点: <http://ajaxload.info/> 大家觉得那些 Loading 比较炫的可以在这里跟帖晒一下, 方便大家取用, 嘎嘎

=====

=====

自己的例子

=====

jq.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>$.ajax({})</title>
<script type="text/JavaScript" src="js/jquery-1.3.2-min.js"></script>
</head>
<body>
<script type="text/JavaScript">
$(function(){

$("#msg_div").ajaxStart(function(){
    $(this).html("<img src='images/loading.gif' />");
});

$("#loadgo").click(function(evt){
    $("#msg_div").load("test1.php",{name:"小林",age:"20"},function(msg){
        $(this).html("load 方法: "+msg);
    }
    );
});

$("#getgo").click(function(evt){
    $.get("http://hi.baidu.com/cnletian/blog",{action:"get",name:"小林",age:"20"},function(msg){
        $("#msg_div").html("get 方法: "+msg);
    }
    );
});

$("#postgo").click(function(evt){
    $.post("test1.php",{name:"小林",age:"20"},function(msg){
        $("#msg_div").html("post 方法: "+msg);
    }
    );
});

$("#ajaxgo").click(function(evt){
    $.ajax({
        type:"POST",
        url:"test1.php",
        data:{name:"小林",age:"20"},
        success:function(msg,statu_txt)
        {
```

```
        $("#msg_div").html("ajax 方法: "+msg);
    }
    });

});

});
</script>
<div id="msg_div"></div>
<input type="button" value="load 开始" id="loadgo" />
<input type="button" value="get 开始" id="getgo" />
<input type="button" value="post 开始" id="postgo" />
<input type="button" value="ajax 开始" id="ajaxgo" />
</body>
</html>
```

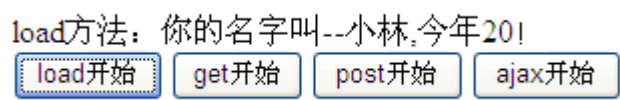
test1.php

```
<?php
switch ($_GET["action"])
{
case "post":
    $name=$_POST["name"];
    $age=$_POST["age"];
    break;
case "get":
    $name=$_GET["name"];
    $age=$_GET["age"];
    break;
default:
    $name=$_POST["name"];
    $age=$_POST["age"];
}

for($i=0;$i<10000000;$i++){

echo "你的名字叫--".$name.",今年".$age."! "
?>
```

运行效果：



打造自己的插件

首先你要理解插件用来为一些操作增加一些方便的“函数”方法的，

两个简单的不同形式的插件例子

一、描述:在 jQuery 命名空间上增加两个函数，可直接使用

```
jQuery 代码:
jQuery.extend({
  min: function(a, b) { return a < b ? a : b; },
  max: function(a, b) { return a > b ? a : b; }
});
```

```
结果:
jQuery.min(2,3); // => 2
jQuery.max(4,5); // => 5
```

二、描述:增加两个插件方法，可以在选择器包装集上使用，查找到相关内容时，调用此方法

```
jQuery 代码:
jQuery.fn.extend({
  check: function() {
    return this.each(function() { this.checked = true; });
  },
  uncheck: function() {
    return this.each(function() { this.checked = false; });
  }
});
结果:
$("input[type=checkbox]").check();
$("input[type=radio]").uncheck();
```

综合制作插件

(在插件中可能会用到 **return this.each()**)

```
jQuery.fn.foobar = function(options) {

  var settings = {      //可选的:默认的初始参数配置
    value: 5,
    name: "pete",
    bar: 655
  };

  if(options) {        //如果存在自定义参数，则加上自定义参数
    jQuery.extend(settings, options);
  }

  this.sPosition=function(obj){.....}      //只能在内部使用
  sPosition(12);
```



```
        return this;
    };
```

在文档中使用插件

1) 无参引用:

```
$("#...").foobar();
```

2) 有参引用:

```
$("#...").foobar({
    value: 123,
    bar: 9
});
```

如:

```
jQuery.fn.Log=function(message) {
    Return this.each(function() {
        $('<div class="log-message" />')
            .text(message)
            .hide()
            .appendTo(this)
            .fadeIn();
    });
};
```