**JQuery**

# jQuery是什么？

jQuery是一个JavaScript函数库。

jQuery是一个轻量级的"写的少，做的多"的JavaScript库。

# jQuery解决的问题

1、浏览器兼容问题

2、代码简洁：代码量少，不乱

3、实现动画效果

4、代码容错性

5、window.onload事件避免覆盖

# jQuery的使用

## 版本问题



min：它是压缩过的版本

区别：我们开发过程中，会用未压缩的版本，或者压缩的。

项目上线的时候，我们要用压缩过的版本。

版本问题：

1. 1.xxx 版本 jQuery-1.11.1.js
2. 2.xxx 版本 不再支持IE6、7、8

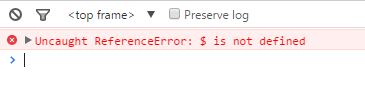
## 引包

### 引用本地文件

* 先将jQuery文件保存至项目中
* 在页面中引用jQuery

<script src="js/jquery-1.11.1.js"></script>

如果出现下列问题则标志jQuery引用未成功



### 使用CDN（内容分发网络），在线使用

需从 Staticfile CDN、又拍云、新浪、谷歌或微软引用 jQuery

例如以下代码：

<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js">

</script>

</head>

（不大推荐使用Google CDN来获取版本，因为Google产品在中国很不稳定）

## 多库共存

同一个页面，引入包含$变量的其他js库，保证它们不发生命名冲突。

// other\_lib.js 包含$

<script src="other\_lib.js"></script>

<script src="jquery-1.11.1.js"></script>

<script>

$.noConflict();

// 可以使用其他库的$符号了

// 使用jQuery

jQuery(document).ready(function(){});

</script>

# jQuery语法

## 基础语法

$(selector).action()

## 文档就绪事件

<script>

$(document).ready(function(){

});

//等价于

$(function(){

});

</script>

### JS和jQuery入口函数的区别

Js的window.onload事件是等到所有内容，以及我们的外部图片之类的文件加载完了之后，才回去执行

jQuery的入口函数 是在 html所有标签都加载之后，就回去执行。

当页面加载后点击里面的按钮无法执行，就是因为js在页面加载完之前无法执行

# jQuery选择器 selector



## 回顾CSS选择器

|  |  |  |
| --- | --- | --- |
| **CSS选择器回顾** | | |
| **符号** | 说明 | 用法 |
| **#id** | Id选择器 | #id{ color:red; } |
| **.class** | 类选择器 | .class{ //} |
| **Element** | 标签选择器 | P { //} |
| **\*** | 通配符选择器 | 配合其他选择器来使用 |
| **,** | 并集选择器 | div,p{} |
| **空格** | 后代选择器 | div span{}  选择div下面所有后代的span |
| **>** | 子代选择器 | div > span{} |
| **+** | 紧邻选择器 | div+p  选择div紧挨着的下一个p元素 |

## 基本选择器

|  |  |  |
| --- | --- | --- |
| **基本选择器** | | |
| **符号** | 说明 | 用法 |
| **$(“#demo”)** | 选择id为demo的第一个元素 | $(“#demo”).css(“background”,”red”) |
| **$(“.liItem”)** | 选择所有类名（样式名）为liItem的元素 | $(“.liItem”). css(“background”,”red”); |
| **$(“div”)** | 选择所有标签名字为div的元素 | $(“div”). css(“background”,”red”); |
| **$(“\*”)** | 选择所有元素  少用或配合其他选择器来使用 | $(“\*”). css(“background”,”red”) |
| **$(“.liItem,div”)** | 选择多个指定的元素，这个地方是选择出了 .liItem元素和div元素 | $(“.liItem,div”). css(“background”,”red”) |

## 层级选择器

|  |  |  |
| --- | --- | --- |
| **层级选择器** | | |
| **符号** | 说明 | 用法 |
| **空格** | 后代选择器  选择所有的后代元素 | $(“div span”). css(“background”,”red”); |
| **>** | 子代选择器  选择所有的子代元素 | $(“div > span”). css(“background”,”red”) |
| **+** | 紧邻选择器  选择紧挨着的下一个元素 | $(“div + p”). css(“background”,”red”) |
| **~** | 兄弟选择器  选择后面的所有的兄弟元素 | $(“div ~ p”). css(“background”,”red”) |

## 过滤选择器

|  |  |  |
| --- | --- | --- |
| **基本过滤选择器** | | |
| **符号** | 说明 | 用法 |
| **:eq(index)** | index是从0开始的一个数字，选择序号为index的元素。选择第一个匹配的元素。 | $(“li:eq(1)”). css(“background”,”red”) |
| **:gt(index)** | Index 是从0开始的一个数字，选择序号大于index的元素 | $(“li:gt(2)”). css(“background”,”red”) |
| **:lt(index)** | Index是从0开始的一个数字，选择小于index 的元素 | $(“li:lt(2)”). css(“background”,”red”) |
| **:odd** | 选择所有序号为奇数行的元素 | $(“li:odd”). css(“background”,”red”) |
| **:even** | 选择所有序号为偶数的元素 | $(“li:even”). css(“background”,”red”) |
| **:first** | 选择匹配第一个元素 | $(“li:first”). css(“background”,”red”) |
| **:last** | 选择匹配的最后一个元素 | $(“li:last”). css(“background”,”red”) |

## 属性选择器

|  |  |  |
| --- | --- | --- |
| **属性选择器** | | |
| **符号** | 说明 | 用法 |
| **$(“a[href]”)** | 选择所有包含href属性的元素 | $(“a[href]”). css(“background”,”red”) |
| **$(“a[href=’itcast’]”)** | 选择href属性值为itcast的所有a标签 | $(“a[href=’itcast’]”). css(“background”,”red”) |
| **$(“a[href!=’baidu’]”)** | 选择所有href属性不等baidu的所有元素，包括没有href的元素 | $(“a[href!=’baidu’]”). css(“background”,”red”) |
| **$(“a[href^=’web’]”)** | 选择所有以web开头的元素 | $(“a[href^=’web’]”). css(“background”,”red”) |
| **$(“a[href$=’cn’]”)** | 选择所有以cn结尾的元素 | $(“a[href$=’cn’]”). css(“background”,”red”) |
| **$(“a[href\*=’i’]”)** | 选择所有包含i这个字符的元素，可以是中英文 | $(“a[href\*=’i’]”). css(“background”,”red”) |
| **$(“a[href][title=’我’]”)** | 选择所有符合指定属性规则的元素，都符合才会被选中。 | $(“a[href][title=’我’]”). css(“background”,”red”) |

## 筛选选择器（方法）

<script>

$(document).ready(function () {

// .eq()

// 类比 ":eq(index)" 选择器

//$("span").eq(1).css("font-size","40px");

// .first()

// 类比 ":first" 选择器

//$("span").first().css("font-size", "40px");

// .parent()

// 选择父亲元素

//$("#sp").parent().css("background", "gold");

//$("p").parent("#div").css("background", "gold");

// .siblings()

// 选择所有的兄弟元素

//$("#li04").siblings(".liItem").css("background", "gold");

// .find()

// 查找所有后代元素

$("div").find("#li04").css("background", "gold");

});

</script>

# jQuery事件 action

## 事件概览



事件

click/mouseenter/blur/keyup

绑定事件

bind delegate on

触发一次

one

解绑

unbind undelegate

触发

click trigger triggerHandler

## 简单事件绑定

* .click(hander) .click() //绑定事件 或者触发 click事件
* .blur() //失去焦点事件，同上
* .hover(mousein, mouseleave) //鼠标移入，移出
* mouseout： 当鼠标离开元素及它的子元素的时都会触发。
* mouseleave: 当鼠标离开自己时才会触发，子元素不触发。
* .dbclick() 双击
* change 改变,比如：文本框发送改变，下来列表发生改变等...
* focus 获得焦点
* keyup, keydown, keypress : 键盘 键被按下。
* mousedown, mouseover
* 其他参考：http://www.w3school.com.cn/jquery/jquery\_ref\_events.asp

## 绑定事件的方式 bind方式（不推荐，1.7以后的jQuery版本被on取代）

语法格式：.bind( eventType [, eventData ], handler )

参数说明

* 第一个参数：事件类型
* 第二个参数：传递给事件响应方法的数据对象，可以省略。
* 事件响应方法中获取数据方式： e.data
* 第三个参数：事件响应方法
* 第二个参数可以省略。

$("p").bind("click", function(e){

//事件响应方法

});

## delegate方式（推荐，性能高，支持动态创建的元素）

语法格式：$(selector).delegate( selector, eventType, handler )

语法说明：

* 第一个参数:selector， 子选择器
* 第二个参数：事件类型
* 第三个参数：事件响应方法

$(".parentBox").delegate("p", "click", function(){

//为 .parentBox下面的所有的p标签绑定事件

});

优势：效率较高

## on方式（最现代的方式，兼容zepto，强烈建议使用的方式）

jQuery1.7版本后，jQuery用on统一了所有的事件处理的方法

语法格式：$(selector).on( events [, selector ] [, data ], handler )

参数介绍：

* 第一个参数：events，事件名
* 第二个参数：selector,类似delegate
* 第三个参数: 传递给事件响应方法的参数
* 第四个参数：handler，事件处理方法

//绑定一个方法

$( "#dataTable tbody tr" ).on( "click", function() {

console.log( $( this ).text() );

});

//给子元素绑定事件

$( "#dataTable tbody" ).on( "click", "tr", function() {

console.log( $( this ).text() );

});

//绑定多个事件的方式

$( "div.test" ).on({

click: function() {

$( this ).toggleClass( "active" );

}, mouseenter: function() {

$( this ).addClass( "inside" );

}, mouseleave: function() {

$( this ).removeClass( "inside" );

}

});

## one绑定一次事件的方式

.one( events [, data ], handler )

$( "p" ).one( "click", function() {

alert( $( this ).text() );

});

## 解绑事件

### unbind undelegate

* unbind解绑 bind方式绑定的事件( 在jQuery1.7以上被 on和off代替)

$(selector).unbind(); //解绑所有的事件

$(selector).unbind("click"); //解绑指定的事件

* undelegate解绑delegate事件

$( "p" ).undelegate(); //解绑所有的delegate事件

$( "p" ).undelegate( "click" ); //解绑所有的click事件

var foo = function () {

// Code to handle some kind of event

};

// ... Now foo will be called when paragraphs are clicked ...

$( "body" ).delegate( "p", "click", foo );

// ... foo will no longer be called.

$( "body" ).undelegate( "p", "click", foo );

### off

* off解绑on方式绑定的事件

$( "p" ).off();

$( "p" ).off( "click", "\*\*" ); // 解绑所有的click事件，两个\*表示所有

$( "body" ).off( "click", "p", foo );

案例1：

var foo = function() {

// Code to handle some kind of event

};

// ... Now foo will be called when paragraphs are clicked ...

$( "body" ).on( "click", "p", foo );

// ... Foo will no longer be called.

$( "body" ).off( "click", "p", foo );

案例2：（了解）解绑命名空间的方式：

var validate = function() {

// Code to validate form entries

};

// Delegate events under the ".validator" namespace

$( "form" ).on( "click.validator", "button", validate );

$( "form" ).on( "keypress.validator", "input[type='text']", validate );

// Remove event handlers in the ".validator" namespace

$( "form" ).off( ".validator" );

## 触发事件

* 简单事件触发

$(selector).click(); //触发 click事件

* trigger方法触发事件

$( "#foo" ).trigger( "click" );

* triggerHandler触发 事件响应方法，不触发浏览器行为

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

## event对象的简介

* event.data //传递的额外事件响应方法的额外参数
* event.currentTarget //在事件响应方法中等同于this，当前Dom对象
* event.target //事件触发源，不一定===this
* event.pageX //The mouse position relative to the left edge of the document
* event.pageY
* event.stopPropagation()//阻止事件冒泡
* event.preventDefault(); //阻止默认行为
* event.type //事件类型：click，dbclick...
* event.which //鼠标的按键类型：左1 中2 右3

# DOM操作

## 隐式迭代（自动循环遍历）

//所有的li的字体大小都变为50px

$("li").css("font-size","50px");

## li标签字体逐步变大

$("li").css("font-size",function(index,value){

return (index+1)\*parseInt(value)+"px";

})

## mouseover & mouseenter区别

mouseover/mouseout事件，鼠标经过的时候会触发多次，每遇到一个子元素就会触发一次。

mouseenter/mouseleave事件，鼠标经过的时候只会触发一次

## DOM对象与jQuery对象互相转换

### jQuery对象转换成DOM对象:

方式一：$(“#btn”)[0]

方式二：$(“#btn”).get(0)

### DOM对象转换成jQuery对象：

$(document) -> 把DOM对象转成了jQuery对象

var btn = document.getElementById(“bt n”);

btn -> $(btn);

## 元素操作

### 主要方法

.css()、addClass(“className”)、removeClass()、toggleClass

.attr()、removeAttr()、.val()、.html(“<p></p>”)、text()

node.append(“<p>我是追加的内容</p>”)、prePend()

### 样式改变

$(document).ready(function () {

//设置样式

$("li").css("font-size","40px");

$("li").css({

"font-size":"40px",

"color":"red"

});

//获取样式

var bgc = $("li").css("background-color");

alert(bgc);

console.log(bgc);

//添加样式

//给元素添加类样式，注意：只有类名，不加点

$("li").addClass("divItem");

$("li").eq(2).removeClass("divItem");

//如果不加参数，会清除所有类样式

$("li").eq(2).removeClass();

//切换样式

$("li").eq(2).toggleClass("divItem");

});

### jQuery获取元素的高和宽

height ： $(selector).height(); //带参数设置，不带参数获取,参数是number类型

width ： $(selector).width(200); //带参数设置，不带参数获取

innerHeight //只能获取：内边距+内容的高度

innerWidth //同上.........宽度

outerHeight

outerWidth //获取：左右内边距+内容+左右边框

**.height()方法和.css(“height”)的区别：**

返回值不同，.height()方法返回的是 数字类型(20)，.css(“height”)返回的是字符串类型(20px)，因此.height()方法常用在参与数学计算的时候

### jQuery元素坐标操作

#### offset

$(“div”).offset(); // 获取或设置坐标值 设置值后变成相对定位

offset() 获取或设置元素相对于文档位置的方法

返回一个object，包含left和top属性，值是相对于document的位置。

如果传入一个参数，则是对元素重新设置相对于document的位置。

传入参数必须包括：left和top属性。比如： {left：100，top：150}

例如：$(selector).offset({left:100, top: 150});

#### position

$(“div”).position(); // 获取坐标值 子绝父相 只能读取不能设置

position() 获取相对于其最近的定位的父元素的位置。

只能获取，不能设置。

相对与其最近的定位元素

返回一个object，包含left和top属性

例如： $(selector).position();

#### 滚动条（滚动事件）

$(“div”).scrollTop(); // 相对于滚动条顶部的偏移

$(“div”).scrolllLeft(); // 相对于滚动条左部的偏移

例如： $(selector).scrollLeft(100);

scroll() 事件触发或者绑定滚动事件

scroll(hander) 绑定滚动事件

scroll() 触发滚动事件 $(selector).scroll(function(){ //当选择的元素发生滚动的时候触发 });

## map函数

* $.map(arry,function(object,index){}) 返回一个新的数组
* $("li").map(function(index, element){}) 注意参数的顺序是反的

var newArr = $.map($("li"), function(i, e) {

return $(e).text() + i;//每一项返回的结果组成新数组

});

var newArr = $("li").map(function(elem, index){

console.log("elem:" + elem);

console.log("index:" + index);

retrun index;

});

## each函数

* $.each(array, function(index, object){})
* $("li").each(index, element )
* 参数的顺序是一致的。

$( "li" ).each(function() {

$( this ).addClass( "foo" );

});

$( "li" ).each(function( index ) {

console.log( index + ": " + $( this ).text() );

});

$( "div" ).each(function( index, element ) {});

## data

数据缓存

$(“div”).data(“index”); // 获取数据index的值

注意：

html里面的data 属性，例如：data-ROLE，jQuery获取的时候用：$(“div”).data(“role”);

当使用jQuery设置data属性的时候，例如：$(“div”).data(“UPCASE”,123); ，那么获取的时候，要使用：$(“div”).data(“UPCASE”);

<div data-role="page" data-last-value="43" data-hidden="true" data-options='{"name":"John"}'>注意HTML属性不区分大小写</div>

$( "div" ).data( "lastValue" ) === 43;

lastValue -> data-last-value

.data()跟.attr() 方法的区别：

1.获取数据的时候，attr方法需要传入参数，data方法可以不传入参数，这时候获取到的是一个js对象，即使没有任何data属性。

2. 获取到的数据类型不同，attr方法获取到的数据类型是字符串(String)，data方法获取到的是相应的类型。

3. data方法获取到数据之后，我们使用一个对象来接收它，那么就可以直接操作(设置值或获取值)这个对象，而attr方法不可以。

并且通过这种方式，要比.data(key,value)的方式更高效！

4. data-attribute属性会在页面初始化的时候放到jQuery对象，被缓存起来，而attr方法不会。

# jQuery效果

## 元素的显示与隐藏

//hide隐藏时，该元素不占位置

$("#hide").click(function(){

$("p").hide();

});

$("#show").click(function(){

$("p").show();

});

语法

$(selector).hide(speed,callback);

$(selector).show(speed,callback);

* 可选的 speed 参数规定隐藏/显示的速度，可以取以下值："slow"、"fast" 或毫秒。
* 可选的 callback 参数是隐藏或显示完成后所执行的函数名称。

<script>

$(document).ready(function(){

$(".hidebtn").click(function(){

$("div").hide(1000,"linear",function(){

alert("Hide() 方法已完成!");

});

});

});

</script>

第二个参数是一个字符串，表示过渡使用哪种缓动函数。

（jQuery自身提供"linear /匀速" 和 "swing /先快后慢"，其他可以使用相关的插件）。

## 淡入淡出

jQuery的四种fading方法

* fadeIn 淡出

$(selector).fadeIn(speed,callback);

* fadeOut 淡入

$(selector).fadeOut(speed,callback);

* fadeToggle 淡入或淡出

$(selector).fadeToggle(speed,callback);

* fadeTo 渐变为给定透明度

$(selector).fadeTo(speed,opacity,callback);

注：

callback：fade函数执行完后执行的函数

speed： 取值 fast/slow(需要加双引号)或毫秒数

opacity： 取值0-1之间

## 滑动

三种slide方法

* slideDown() 向下滑动
* slideUp() 向上滑动
* slideToggle() 向上或向下滑动

## 动画

$(selector).animate({params},speed,callback);

params：CSS属性

（默认情况下，所有的 HTML 元素有一个静态的位置，且是不可移动的。

如果需要改变为，我们需要将元素的 position 属性设置为 relative, fixed, 或 absolute!）

animate动画：不支持背景的动画效果

//使用绝对值

$("button").click(function(){

$("div").animate({

left:'250px',

opacity:'0.5',

height:'150px',

width:'150px'

});

});

//使用相对值

$("button").click(function(){

$("div").animate({

left:'250px',

height:'+=150px',

width:'+=150px'

});

});

//使用预定义值（"show"、"hide" 或 "toggle"）

$("button").click(function(){

$("div").animate({

height:'toggle'

});

});

//使用队列功能（按顺序执行）

$("button").click(function(){

var div=$("div");

div.animate({height:'300px',opacity:'0.4'},"slow");

div.animate({width:'300px',opacity:'0.8'},"slow");

div.animate({height:'100px',opacity:'0.4'},"slow");

div.animate({width:'100px',opacity:'0.8'},"slow");

});

## 停止动画

$(selector).stop();（默认为false）

$(selector).stop(stopAll,goToEnd); （默认为false）

stopALL：是否应该清除动画队列。默认是 false，即仅停止活动的动画，允许任何排入队列的动画向后执行。

goToEnd：是否立即完成当前动画。默认是 false。

stop（false）：停止当前动画，后续动画继续执行

stop（true）：停止当前动画，后续动画队列全部清空

stop（false，true）：当前动画立即执行完毕，后续队列的动画继续执行

stop（true，true）：当前动画立即执行完毕，后续队列的动画全部清空

# jQuery实例

## 自动显示下拉列表

html样式：

<div class="wrap">

<ul>

<li style="background-color: red;">

<a href="#">一级菜单1</a>

<ul style="display: none;">

<li><a href="#">二级菜单1</a></li>

<li><a href="#">二级菜单2</a></li>

<li><a href="#">二级菜单3</a></li>

</ul>

</li>

<li style="background-color: green;">

<a href="#">一级菜单2</a>

<ul style="display: none;">

<li><a href="#">二级菜单1</a></li>

<li><a href="#">二级菜单2</a></li>

<li><a href="#">二级菜单3</a></li>

</ul>

</li>

<li style="background-color: yellow;">

<a href="#">一级菜单3</a>

<ul style="display: none;">

<li><a href="#">二级菜单1</a></li>

<li><a href="#">二级菜单2</a></li>

<li><a href="#">二级菜单3</a></li>

</ul>

</li>

</ul>

</div>

jQuery代码：

<script>

$(document).ready(function () {

// mouseenter事件 是 鼠标经过时触发的事件

//方法一：使用mouseenter和mouseleave函数

/\*$(".wrap li").mouseenter(function () {

$(this).children("ul").show();

});

$(".wrap li").mouseleave(function () {

$(this).children("ul").hide();

});\*/

//方法二：使用hover函数的两个回调函数

/\*$(".wrap li").hover(function () {

$(this).children("ul").show();

}, function () {

$(this).children("ul").hide();

});\*/

//方法三：使用hover函数的一个回调函数

/\*$(".wrap li").hover(function () {

var $this = $(this).children("ul");

var isShow = $this.css("display");

if(isShow === "block") {

$this.hide();

} else {

$this.show();

}

});\*/

//使用一个方法进行显示和隐藏：slideToggle

$(".wrap li").hover(function () {

$(this).children("ul").stop(false,true).slideToggle();

});

});

</script>

## 懒加载

## 怎么写插件

演示jQuery.lxCfbg.js （自己的插件）

(function ($) {

$.fn.lxCfbg = function (options) {

// 合并参数

var opts = $.extend({

"color": "#000",

"font-size": "16px",

"background-color": "#fff"

}, options);

$this.css({

"color": opts.color,

"font-size": opts["font-size"],

"background-color": opts["background-color"]

});

return $this;

};

})(jQuery);

## jQuery UI

<http://www.runoob.com/jqueryui/jqueryui-intro.html>