**原生javascript学习**

## Javascript基础(来自微软)

资料来源: <https://msdn.microsoft.com/zh-cn/library/6974wx4d(v=vs.94).aspx>

* Javascript支持各种数据类型，String、number、boolean、object、array等
* javascript使用原型，而不是类(定义一个方法obj，这个obj是类。Obj.prototype.show=function(){}这是原型)

### 变量

--可以在不使用 var 关键字的情况下在声明中声明一个变量，并为该变量分配一个值。 这是一个隐式声明。(age = 10;)

--强制转换，数值+字符串=字符串，布尔类型+字符=字符，布尔类型+数字=数字

--字符转数值使用parseInt(整数)，parseFloat(小数点)

### 数据类型

--主数据类型：字符串、数字、boolean

--复合数据类型：对象、数组

--特定数据类型：null、undefined

--判断undefined两种方式

Var x; if(x == undefined){}  
 var x; if(typeof(x) == “undefined”){}

--判断对象是否有某属性

Var obj = {}; obj.prop = null;  
 if(“prop” in obj){}

### 运算符

其它运算符：typeof、void、instanceof、new、in

运算优先级

### 程序控制流

if/else while/for/for in/ switch

--尽可能使用短路运算实际控制条件，条件’&&’将最可能为false条件放前面，条件’||’将最可能为true放前面，这样可能省掉一个表达式运算

如：var n = 10; if((n == 10) || (n == 20)){}

Var n=10; if((n == 20) && (n == 10)){}

--for…in遍历的是对象或数组的索引，即[{name: value}]中的name

### 函数

JavaScript 支持两种类型的函数：内置于语言的函数和你自行创建的函数。

### 对象和数组

四种对象：

内部对象(如Array和String)

自己创建的对象

宿主对象(如window和document)

ActiveX对象

--数组也是一个对象

### 内部对象

--Array、Boolean、Date、Error、Function、Global、JSON、Math、Number、Object、RegExp、String

--var arr = new Array(12) / var arr = new Array(‘one’, ‘two’, ‘three’)

--JSON对象方法：【JSON.stringify将对象和数组序列化为JSON文本

对象转json={“name”:”yingdi”, “age”:”25”}

数组转json=[“zhangsan”, “lishi”, “wangwu”]

【JSON.parse将JSON文本反序列化为内存对象

Json转对象={name:”yingdi”, age:25}

Json转数组=(3)[“zhangsan”, “lishi”, “wangwu”]

### Creating对象

--直接实例化=var obj = new Object(); obj.name = “yingdi”;

--对象文本表示=var obj = {name: “yiingdi”};

### 计算日期和时间

--当前日期

var today = new Date();

console.log(today.getFullYear()+'-'+(today.getMonth()+1)+'-'+today.getDate());  
--返回毫秒数=today.getMilliseconds()

--返回秒数= today.getMilliseconds() / 1000

--比较日期= if(day1.getTime() == day2.getTime){ }

--String转Date= var timeSec = Date.parse(“2019-11-19”); var dayTime = new Date(timeSec);

## JavaScript基础(来自Github)

https://github.com/smyhvae/Web

### Javascript的组成

* ECMAScript：JavaScript的语法标准。包括变量、表达式、运算符、函数、if语句、for语句等
* DOM：文档对象模型，操作网页上的元素的API。比如让盒子移动、变色、轮播图等
* BOM：浏览器对象模型，操作浏览器部分功能的API。比如让浏览器自动滚动

### 数据类型

* 基本数据类型（值类型）：String 字符串、Number 数值、Boolean 布尔值、Null 空值、Undefined 未定义
* 引用数据类型（引用类型）：Object 对象

*注意：内置对象function、Array、Date、RegExp、Error等都是属于Object类型。也就是说，除了那五种基本数据类型之外，其他的，都称之为 Object类型*

### 重点类型String的常见操作

* 引号不能嵌套：双引号里不能再放双引号，单引号里不能再放单引号。但是单引号里可以嵌套双引号
* 转义字符：[\" 表示 "][\' 表示 '][ \n 表示换行][ \r 表示回车][ \\ 表示\]
* prompt()就是专门用来弹出能够让用户输入的对话框，用户不管输入什么，都是字符串
* 其它数值转成String的三个方法：加空串+””、toString()、new String(x)

### 重点类型Number类型的常见操作

* 字符串转整数：parseInt(String)
* 字符串转浮点数：parseFloat(String)

### fn() 和 fn 的区别

* fn()：调用函数。相当于获取了函数的返回值
* fn：函数对象。相当于直接获取了函数对象
* object.fn() 和 fn()效果一致，都是执行函数

### for..in遍历对象属性

for(var n in object){}，对象object有几个属性就会遍历几次

### 作用域=全局作用域和函数作用域

### 全局作用域

直接编写在script标签中的JS代码，都在全局作用域

* 全局作用域在页面打开时创建，在页面关闭时销毁
* 在全局作用域中有一个全局对象window，它代表的是一个浏览器的窗口，它由浏览器创建我们可以直接使用

在全局作用域中：

* 创建的变量都会作为window对象的属性保存
* 创建的函数都会作为window对象的方法保存

变量的声明提前：

* var a = 1，会在所有的代码执行之前被声明（但是不会赋值），在定义前调用=undefined
* a = 1，变量声明不会提前，在定义前调用报错

函数的声明提前

* function foo(){}，会被声明提前
* var foo = function(){}，不会被声明提前

### 函数作用域

* 调用函数时创建函数作用域，函数执行完毕以后，函数作用域销毁
* 每调用一次函数就会创建一个新的函数作用域，他们之间是互相独立的
* 在函数作用域中可以访问到全局作用域的变量，在全局作用域中无法访问到函数作用域的变量
* 函数作用域操作一个变量时，先在自身作用域查找、再往上一级作用域找、再到全局作用域、还没有报ReferenceError
* 全局作用域和函数作用域都定义了变量a，如果想访问全局变量，可以使用window.a
* 在函数中，没有var声明的变量都会成为全局变量，而且并不会提前声明

### This

* 以函数的形式调用时，this永远都是window。比如fun();相当于window.fun();
* 以方法的形式调用时，this是调用方法的那个对象。比如object.fun();

### 原型prototype和原型链

* 我们所创建的每一个函数，解析器都会向函数中添加一个属性prototype，可以通过【方法.prototype】或【实例.\_\_prototype\_\_】来访问
* 使用一个对象的属性或方法时，会现在自身中寻找，如果没有则去原型对象中寻找，再找不到再往上原型找，找不到返回undefined

### 数组

* 创建语法：var arr = [a, b] / var arr = new Array(a, b) / var arr = new Array(10)
* 判断是否数组：[布尔值= A instanceof Array]或[布尔值 = Array.isArray(A)]
* 将数组中的元素用符号连接成字符串：字符串 = 数组.join(参数)
* push()：在数组最后面插入项，返回数组的长度[数组改后的长度 = 数组.push(元素);]
* pop()：取出数组中的最后一个元素，返回被删除的元素[被删除的元素 = 数组.pop();]
* unshift()：在数组最前面插入项，返回数组的长度[数组改后的长度 = 数组.unshift(元素);]
* shift()：取出数组中的第一个元素，返回被删除的元素[被删除的元素 = 数组.shift();]
* reverse()：翻转数组（返回值是反转后的数组，而且原数组也已经被反转了）[反转后的数组 = 数组.reverse();]
* 伪数组：arguments，用在函数中获取实参的值和个数[arguments] [arguments.length]
* sort()：给数组排序，返回排序后的数组（排序的规则看参数）[从小到大排序后的数组 = 数组.sort(function(a,b){ return a-b; });]无参：按照数组元素的首字符对应的Unicode编码值排列；带参：必须为函数（回调函数：callback）
* concat() ：把参数拼接到当前数组（原数组不会被修改）[新数组 = 数组1.concat(数组2);]
* slice()：从当前数组中截取一个新的数组（不影响原来的数组）[新数组 = 原数组.slice(开始位置index，结束位置index);]
* splice()：删除当前数组的某些元素（原数组会被改变）[新数组 = 数组1.splice(起始索引index，需要操作的个数，弥补的值);]
* 清空数组的几种方式：[array.splice(0);] [array.length = 0;] [array = [];推荐]

### 内置对象

Arguments、

Array、

Boolean、

Date、

Error、

Function、

Math、

Number、

Object、

RegExp、

String

### DOM操作

DOM: 文档对象模型

* 解析过程：HTML加载完毕，渲染引擎会在内存中把HTML文档，生成一个DOM树，getElementById是获取内中DOM上的元素节点。然后操作的时候修改的是该元素的**属性**。在DOM中一切都是节点[**元素节点**：HMTL标签] [**文本节点**：标签中的文字（比如标签之间的空格、换行）] [**属性节点**：：标签的属性]

**JS是以事件驱动为核心的一门语言。**

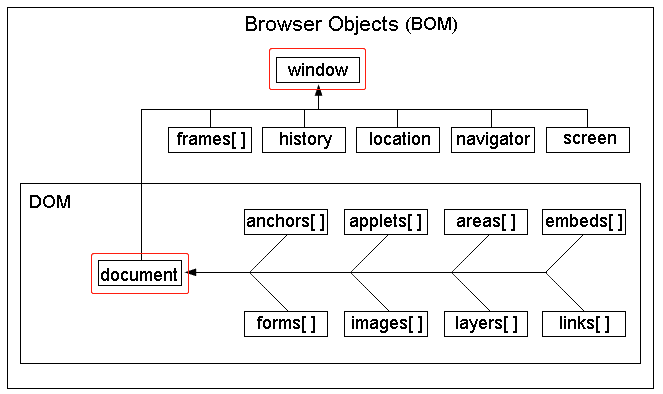
* 事件源：引发后续事件的html标签
* 事件：js已经定义好了
* 事件驱动程序：对样式和html的操作。也就是DOM
* 获取事件源：document.getElementById(“box”);
* 绑定事件： 事件源box.事件onclick = function(){ 事件驱动程序 };
* 书写事件驱动程序：关于DOM的操作

**总结即：事件源.事件=function(){事件驱动程序}**

### BOM操作

BOM: 浏览器对象模型

结构图如下



#### 系统对话框：

[alert()] [confirm()] [prompt]

#### 打开/关闭窗口

打开窗口：window.open(url,target,param)

**参数解释：**

* url：要打开的地址
* target：新窗口的位置。可以是：\_blank 、\_self、 \_parent 父框架
* param：新窗口的一些设置
* 返回值：新窗口的句柄

**param**这个参数，可以填各种各样的参数（），比如

* name：新窗口的名称，可以为空
* features：属性控制字符串
* fullscreen= { yes/no/1/0 } 是否全屏，默认no
* toolbar= { yes/no/1/0 } 是否显示工具条，默认no

等等。。。各个参数之间用逗号隔开就行，但我们最好是把它们统一放到json里

关闭窗口：window.close()

#### 新窗口相关：

* 新窗口.moveTo(5,5)
* 新窗口.moveBy()
* 新窗口.resizeTo()
* window.resizeBy()

举例：

var newWin = window.open("demo.html", "\_blank", json);

newWin.moveTo(500, 500);

#### location对象

window.location可以简写成location。location相当于浏览器地址栏，可以将url解析成独立的片段

**location对象的属性**

* **href**：跳转
* hash 返回url中#后面的内容，包含#
* host 主机名，包括端口
* hostname 主机名
* pathname url中的路径部分
* protocol 协议 一般是http、https
* search 查询字符串
* location.reload()：重新加载

举例：5秒后自动跳转到百度

setTimeout(function () {

location.href = "http://www.baidu.com";

}, 5000);

#### Navigator对象

* Window.navigator.userAgent：客户端浏览器信息
* Window.navigator.platform：客户端浏览器平台-win32

#### History对象

* 后退：[history.back()] [history.go(-1)：0是刷新]
* 前进：[history.forward()] [history.go(1)]

## jQuery(来自Github)

[https://github.com/smyhvae/Web](https://github.com/smyhvae/Web/blob/master)

* 官网：<http://jquery.com/>
* 官网API文档：<http://api.jquery.com/>
* 中文汉化API文档：<http://www.css88.com/jqapi-1.9/>

### jQuery特点

* 浏览器兼容
* 链式编程，比如.show()和.html()可以连写成.show().html()。链式编程原理：return this
* 隐式迭代，在方法的内部会为匹配到的所有元素进行循环遍历，不用再手动遍历

### 入口函数

* 原生JS入口函数：window.onload = function() {};// 页面文本和图片加载完成才执行
* jQuery入口函数1：$(document).ready(function () {};// 文本加载完毕可执行，不等图片加载
* jQuery入口函数2：$(function () {};// 写法1的简写版
* jQuery入口函数3：$(window).ready(function () {};// 文本和图片加载完才执行

### $和jQuery两个对象

console.log($);

console.log(jQuery);

console.log($===jQuery);// true

$和jQuery实际是个函数名，它们完全相等

### js中的DOM对象 和 jQuery对象 比较

通过 jQuery 获取的元素是一个数组，数组中包含着原生JS中的DOM对象。

总结：jQuery 就是把 DOM 对象重新包装了一下，让其具有了 jQuery 方法

**两者的互换：**

DOM 对象 转为 jQuery对象：$(js对象);

jQuery对象 转为 DOM 对象：

jquery对象[index]; // 方式1（推荐）

jquery对象.get(index); // 方式2

### jQuery转字符串为JSON对象

var jsonStr = “{“code”:”200”,”name”:”yingdi”}”;  
var jsonObj = $.parseJSON(jsonStr);// jsonObj={code:”200”,name:”yingdi”}

### jQuery转form元素为JSON

var $(‘#formID’).toJson();// 结果是将一个form表单的元素根据name和值构成一个JSON对象

$('#updateForm').form('clear').form('load',updaterow);// 这里是将一个JSON对象的值根据name赋值给一个form表单

### 基本选择器

* $(“#id”);// 选取第一个id属性为id的元素
* $(“.class”);// 选取类名为class的所有元素
* $(“element”);// 选取所有element标签元素
* $(\*);// 选取所有元素
* $(“#id, .class”);// 并集选择，选取id=id，class=class的所有元素

### 层级选择器

* $(“#id > p”);// 选取id=id的元素下的直接子元素p
* $(“#id p”);// 选取id=id的元素下的p元素，包括子元素和孙子元素

### 基本过滤选择器

* :eq(Index) // 选择序号为index的元素，index从0开始，例：$(“li:eq(0)”)
* :gt(Index) // 选择序号大于index的元素
* :lt(Index) // 选择序号小于index的元素
* :odd // 选择奇数行元素
* :even // 选择偶数行元素
* :first // 选择第一个元素
* :last // 选择最后一个元素

### 属性选择器

* $(“a[href]”) // 选择所有包含href属性的元素
* $(“a[href = ‘cast’]”) // 选择所有href属性等于cast的a标签，不等于!=同理
* $(“a[href^=’web’]”) // 选择所有以web开头的元素
* $(“a[href$=’cn’]”) // 选择所有以cn结尾的元素
* $(“input[name=’yingdi’]”) // 选择name等于yingdi的input元素

### 筛选选择器

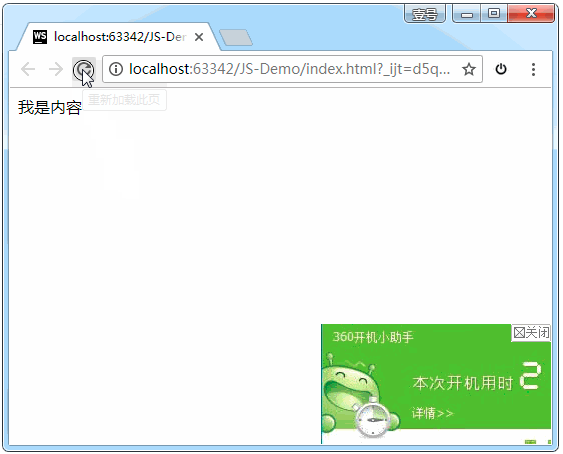
* find(selector) // 查找指定元素的所有后代元素(包含子子孙孙)，例：$(“#swap”).find(“li”)
* children(selector) // 查找指定元素的直接子元素，例：$(“#swap”).children(“ul”)
* siblings() // 查找指定元素的兄弟元素(不包括自己)，例：$(“#swap”).sublings()
* parent() // 查找直接父元素
* eq(index) // 查找指定元素集合的第index个元素，index从0开始，例：$(“li”).eq(2)

### this

谁正在调用函数，this就指向谁

### 右下角弹出广告的例子

https://github.com/smyhvae/Web/blob/master/05-jQuery /02-jQuery动画详解.md



### Each

**大部分情况下是不需要使用each方法的，因为jQuery的隐式迭代特性。但是，如果要对每个元素做不同的处理，这时候就用到了each方法。**

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

参数解释：

* index表示当前元素在所有匹配元素中的索引号
* element参数二表示当前元素（是js 中的DOM对象，而不是jQuery对象）

### jQuery操作样式和类

**样式操作(样式较少时可以使用)：**

* 获取样式：$("div").css("width")
* 设置单个样式：$("div").css("background-color","red");
* 设置多个样式：$("div").css({"width":100,"height":100,"background-color":"pink"});

**类操作(样式较多或较少都可以使用，推荐)：**

* 添加样式：$(selector).addClass("liItem"); //为指定元素添加类className
* 移除类样式：$(selector).removeClass("liItem"); // 不带参数时移除全部
* 判断有无类样式：$(selector).hasClass("liItem");
* 切换类样式：$(selector).toggleClass(“liItem”); //为指定元素切换类 className，该元素有类则移除，没有指定类则添加

### jQuery操作属性

* 获取属性：$(selector).attr("title");
* 设置属性：$(selector).attr("title", "生命壹号");
* 移除属性：$(selector).removeAttr("title");
* 针对checked、selected、disabled属性，要使用 prop()方法，而不是其他的方法

### jQuery节点操作

**动态创建元素：**

* 方法一：var $spanNode1 = $("<span>我是一个span元素</span>"); // 返回的是 jQuery对象，此方法类似于 原生 js 中的document.createElement("标签名");
* 方法二：var node = $("#box").html("<li>我是li</li>");此方法类似于 原生 js 中的innerHTML。

**添加元素：**

在目标元素的最后插入内容

* $(selector).append($node); //参数是 jQuery对象
* $(selector).append('<div></div>'); //参数是 htmlString
* $(selector).prepend(node); // 在元素的第一个子元素前面添加节点
* $(selector).after(node); //在被选元素之后，作为兄弟元素插入内容或节点
* $(selector).before(node); //在被选元素之前，作为兄弟元素插入内容或节点

**注意**

* 如果是页面中存在的元素，那调用append()后，会把这个元素放到相应的目标元素里面去；但是，原来的这个元素，就不存在了
* 如果是给多个目标追加元素，那么方法的内部会复制多份这个元素，然后追加到多个目标里面去

**清空元素：**

$(selector).empty(); / $(selector).html(""); / $(selector).remove();

**复制元素：**

复制的新元素 = $(selector).clone();

## BootStrap前端框架

官方网站：<https://getbootstrap.com/>

中文网站：<http://www.bootcss.com/>

因为 bootstrap.js依赖jQuery，所以要先引用jquery.js 然后引用bootstrap.js

## 高级Javascript(来自微软)

### 原型和原型继承

### 变量作用域、闭包

--全局变量=函数之外声明 / 变量位于函数内赋值语句左侧且没有用var声明

--局部变量=函数内用var声明的变量

--闭包=函数嵌套(函数中调用函数，如function1调用function2)中，子函数function2会保存function1传入的变量，该变量在function1外改变，不影响function2的调用结果

## 进阶Javascript(来自Github)

### 作用域和闭包

* 闭包就是能够读取其他函数内部数据（变量/函数）的函数
* 当一个嵌套的内部(子)函数引用了嵌套的外部(父)函数的变量或函数时, 就产生了闭包

**闭包的作用：**

* (1)使用函数内部的变量在函数执行完后, 仍然存活在内存中(延长了局部变量的生命周期)
* (2)让函数外部可以操作(读写)到函数内部的数据(变量/函数)

// 如下实例代码------------------------------------

function fn1() {

var a = 2

function fn2() {

a++

console.log(a)

}

return fn2;

}

var f = fn1(); //执行外部函数fn1，返回的是内部函数fn2

f() // 3 //执行fn2

f() // 4 //再次执行fn2

**作用1分析：**

上方代码中，外部函数fn1执行完毕后，变量a并没有立即消失，而是保存在内存当中

**作用2分析：**

函数fn1中的变量a，是在fn1这个函数作用域内，因此外部无法访问。但是通过闭包，外部就可以操作到变量a

**闭包的生命周期：**

* 产生: 嵌套内部函数fn2被声明时就产生了(不是在调用)
* 死亡: 嵌套的内部函数成为垃圾对象时，比如f = null时

**闭包的应用：定义具有特定功能的js模块**

将所有的数据和功能都封装在一个函数内部(私有的)，只向外暴露一个包含n个方法的对象或函数。模块的使用者, 只需要通过模块暴露的对象调用方法来实现对应的功能。

**闭包的缺点及解决：**

缺点：函数执行完后, 函数内的局部变量没有释放，占用内存时间会变长，容易造成内存泄露

解决：能不用闭包就不用，及时释放。比如：f = null

### 深入解读this

### 创建对象和继承

### 浅拷贝和深拷贝