**一、JS语法**

**数字（Number）字面量** 可以是整数或者是小数，或者是科学计数(e)

**字符串（String）字面量** 可以使用单引号或双引号:

**表达式字面量** 用于计算：5 + 6 5 \* 10

**数组（Array）字面量** 定义一个数组：[40, 100, 1, 5, 25, 10]

**对象（Object）字面量** 定义一个对象：{firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}

**函数（Function）字面量** 定义一个函数：function myFunction(a, b) { return a \* b;}  
JavaScript 使用关键字 **var** 来定义变量， 使用等号来为变量赋值，变量是一个**名称**。字面量是一个**值**

JavaScript使用 **算术运算符** 来计算值: (5 + 6) \* 10

JavaScript使用**赋值运算符**给变量赋值：

x = 5  
y = 6  
z = (x + y) \* 10

|  |  |  |
| --- | --- | --- |
| 赋值，算术和位运算符 | =  +  -  \*  / | 在 JS 运算符中描述 |
| 条件，比较及逻辑运算符 | ==  != <  > | 在 JS 比较运算符中描述 |

Js语句是用分号分隔

**JavaScript 注释**

不是所有的 JavaScript 语句都是"命令"。双斜杠 **//** 后的内容将会被浏览器忽略：

// 我不会执行

## JavaScript 数据类型

JavaScript 有多种数据类型：数字，字符串，数组，对象等等，

## 数据类型的概念

编程语言中，数据类型是一个非常重要的内容。

为了可以操作变量，了解数据类型的概念非常重要。

## JavaScript 字母大小写

JavaScript 对大小写是敏感的。

当编写 JavaScript 语句时，请留意是否关闭大小写切换键。

函数 **getElementById** 与 **getElementbyID** 是不同的。

同样，变量 **myVariable** 与 **MyVariable** 也是不同的。

**二、标识符:**

**一、（变量、函数、属性的名字和函数的参数）书写规范：**

1、区分大小写

2、标识符第一个字符必须是英文字母、下划线\_或美元符号$，其后的字符可以是字母、数字或下划线、美元符号；

3、自定义的标识符不可以 JavaScript 中的关键字、保留字作为标识符

4、标识符不能包含空格；

5、标识符不能包含+、-、@、#等特殊字符；

**二、由多个单词组成的复合标识符命名主要有两种方式：**

一是使用下划线连接各个单词，每个单词全部小写，例如：dept\_name。

二是使用驼峰式，其中又分大驼峰和小驼峰。大驼峰的格式是每个单词的首字母大写，其余字母小写，例如：DeptName；小驼峰的格式是第一个单词全部小写，第二单词开始的

**三、程序结构：**

**一：程序运行的三大结构：顺序结构、选择结构、循环结构**

1. **顺序结构：代码逐行执行，自上而下**
2. **选择结构：通常是和关系比较一起用**

①单分支选择：if语句

语法：if(条件){

代码块

}

②双分支选择：if...else语句

语法：if(条件){

//为true时执行的代码

}else{

//为fasle时执行的代码

}

③多分支语句：if..else if..else

语法：if(条件1){

//条件1为true时执行的代码

}else if(条件2){

//条件2为true时执行的代码

}else{

//条件1条件2都不为true时执行的代码

}

④switch语句是if..else if语句的简写方式

语法：switch(表达式){

case 1:

语句1

break;

case 2:

语句2

break;

..........

default:

语句n //默认其他都不是的情况下执行

break;

}

注：1)每一个case中都需要加一个break

1. default可以写在switch任意的位置，写在switch末尾不需要加break
2. 每一个case值不能重复
3. **循环结构：重复执行一段代码**

作用：简化代码，处理重复的代码

①while循环：不知道具体执行次数的时候使用，先判断后执行

语法：while(循环条件){

循环体；

迭代部分；

}

②do...while循环：先执行后判断，无论结果正误都执行一次

语法：do{

循环体；

迭代部分；

}while(循环部分)；

③for循环：比while语法更清晰，可读性更高

语法：for(初始值；循环体；迭代部分){

输出体；

}

**4、break、continue**

break:用在switch和循环中，表示结束switch和结束当前循环

continue：用在循环中，表示跳出本次循环，继续执行下次循环

**5、循环嵌套的应用：外层循环控制行，内层循环控制列**

**四、运算符**

经常见的运算符有：算数运算符，赋值运算符，比较运算符，逻辑运算符，三元运算符。

1. 算数运算符

+ 加：只要符号任意一边是字符串类型，就会进行字符串拼接

* 减；\* 乘；/ 除；% 取余（模运算）；++ 递增（累加）；-- 递减

1. 赋值运算符

= 直接赋值

+=，-= 加，减一个数后再赋值

\*=，/=，%= 乘，除，取余后再赋值

3、比较运算符

进行比较后，会返回一个布尔值

< 小于；> 大于；

>= 比较左边的值是否 大于或等于 右边的值；

<= 比较左边的值是否 小于或等于 右边的值;

== 比较符号两边的值是否相等，不管数据类型,

Eg: 1 == '1' 两个的值是一样的，所以得到 true;

=== 比较符号两边的值和数据类型是否都相等,

Eg: 1 === '1' 两个值虽然一样，但是因为数据类型不一样，所以得到 false;

!= 比较符号两边的值是否不等,

Eg: 1 != '1' 因为两边的值是相等的，所以比较他们不等的时候到 false;

!== 比较符号两边的数据类型和值是否不等

Eg:1 !== '1' 因为两边的数据类型确实不一样，所以得到 true;

“=” “==” “===”有什么去别？

= 是赋值运算符，把右边的值赋值给左边。

== 判断 判断两边值是否相等。

=== 全等 判断两边的值和数据类型是否完全相同。

1. 逻辑运算符

&& （and） 与，符号左边必须为 true并且右边也是true，才会返回true，只要有一边不是 true，那么就会返回false。

|| （or）或，符号的左边为 true，或者右边为true，都会返回true,只有两边都是 false 的时候才会返回 false。

！ （not）非，本身是 true 的，会变成 false，本身是 false 的，会变成 true。

复合逻辑表达式:

（user==”小星”）&&（age>=18||addr==”中国”&&（pwd==123））;

1. 三元运算符

三元运算符是一种需要三个操作数的运算符，运算的结果根据给定条件决定。

条件表达式：？表达式1：表达式2；（先求表达式的值，如果为true是表达式1，为false是表达式2）。

**五、数据类型**

1、六种不同数据类型

string（字符串）  
number（数字）  
boolean（布尔）  
object（对象）  
function（函数）  
symbol（ES6的数据类型之一）

2、三种对象类型

Object（对象）  
Date（日期）  
Array（数组）

3、有 2 个不包含任何值的数据类型：

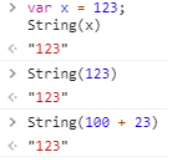
null  
undefined

注意：NaN 的数据类型是 number  
数组(Array)的数据类型是 object  
日期(Date)的数据类型为 object  
null 的数据类型是 object  
未定义变量的数据类型为 undefined

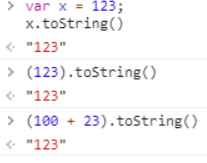
4、将数字转换为字符串

全局方法 String() 可以将数字转换为字符串。

此方法可用于任何类型的数字，字母，变量，表达式：

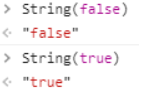


Number方法toString()也可以完成转换。

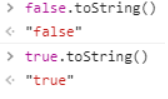


1. 将布尔值转换为字符串

全局方法 String() 可以将布尔值转换为字符串。



Boolean 方法 toString() 也可以完成转换。



1. 将日期转换为字符串

Date() 返回的是字符串。

全局方法 String() 可以将日期对象转换为字符串。



Date 方法 toString() 也可以完成转换



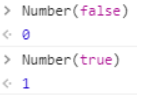
7、将字符串转换为数字

全局方法 Number() 可以将字符串转换为数字。  
字符串包含数字(如 “3.14”) 转换为数字 (如 3.14).  
空字符串转换为 0。  
其他的字符串会转换为 NaN (不是个数字)



8、将布尔值转换为数字

全局方法 Number() 可将布尔值转换为数字。



9、将日期转换为数字

全局方法 Number() 可将日期转换为数字。

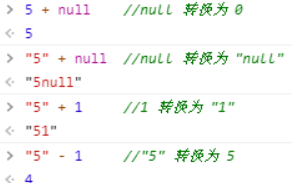


日期方法 getTime() 也有相同的效果。



10、自动转换类型

当 JavaScript 尝试操作一个“错误”的数据类型时，会自动转换为“正确”的数据类型。



11、自动转换为字符串

当尝试输出一个对象或一个变量时,JavaScript会自动调用变量的 toString()方法。

类似像当输出的变量或对象是数字和布尔值也是会转换成字符串。

**六、事件**

1.注册事件（绑定事件）

* 1. 注册事件概述
  2. 事件监听
  3. addEventListener 事件监听方式

1. 删除事件

2.1 删除事件的方式

3.DOM 事件流

4.事件对象

4.1 事件分类

4.2 什么是事件对象

4.3事件对象的常见属性和方法

4.4事件对象的相关属性

5.阻止事件冒泡

5.1 阻止事件冒泡的兼容性解决方案

6.事件委托（代理、委派）

7.常见鼠标事件

7.1 常见鼠标事件

7.2 鼠标事件对象

8.常见的键盘事件

1.注册事件（绑定事件）

1.1注册事件概述

给元素添加事件，称为注册事件或者绑定事件。

注册事件有两种方式：①传统方式②方法监听注册方式

①传统方式

利用on开头的事件onclick

<button onclick=”alert(hello word)”></button>

btn.onclick = function(){}

特点：注册事件的唯一性

同一个元素同一个事件只能设置一个处理函数，最后注册的处理函数将会覆盖前面注册的处理函数。

②方法监听注册方式

w3c 标准 推荐方式

addEventListener() 它是一个方法

IE9 之前的IE不支持此方法，可以使用attachEvent() 代替

特点：同一个元素同一个事件可以注册多个监听器

按注册顺序依次执行

1.2 事件监听

可以为一个元素，同时绑定多个事件

obj.addEventListener(事件，回调函数，冒泡/捕获) ;

btn.addEventListener("click",function(ev){ //ev 事件对象

alert(1212);

},false) //false 冒泡

注意:

1. 事件监听只能通过 JS 方式绑定在 DOM 对象上，不能通过标签事件绑定(标签上只有 onclick 等原始事件) • 阻止默认事件（DOM2 级事件）
2. 通过事件监听绑定的事件,不能用 return false 阻止默认事件，必须通过 ev 事件对象来阻止:

ev.preventDefault();//W3C 事件对象方式阻止默认事件

1.3 addEventListener 事件监听方式

eventTarget.addEventListener(type, listener[, useCapture])

eventTarget.addEventListener() 方法将指定的监听器注册eventTarget（目标对象）上，当该对象触发指定的事件时就会执行事件处理函数

该方法接收三个参数：

type:事件类型字符串，比如click、mouseover，注意这里不要带on

listener：事件处理对象，事件发生时，会调用该监听函数

useCapture：可选参数，是一个布尔值，默认时false

2.删除事件

2.1 删除事件的方式

①传统注册方式

eventTarget.onclick = null;

②方法监听注册方式

Ⅰ：eventTarget.removeEventListener(type, listener[, useCapture]);

Ⅱ：eventTarget.datachEvent(eventNameWithOn, callback);

1. DOM事件流

事件流描述时从页面中接收事件的顺序。

事件发生时会在元素节点之间按照特定的顺序传播，这个传播过程DOM事件流

①事件默认行为

* 很多的网页元素都会有默认的行为，比如说当你点击一下超链接a标签时，它会有一个跳转的行为；当你在网页上点击鼠标右键时会出现一个右键菜单；这些行为并不是绑定函数来实现的，这类行为被称为对象的默认行为
* 阻止默认行为：event.preventDefaut()

②事件流

w3c 标准的事件流方向：捕获-----事件目标-----冒泡

* 事件捕获
* 事件目标
* 事件冒泡 与 阻止事件冒泡

③事件目标

* ev.target
* ev.target.nodeName
* ev.target.tagName

④事件委派

* 原理

将事件绑定在父级上，利用事件冒泡原理，通过判断事件的“目标元素”来触发父级上绑定的事件

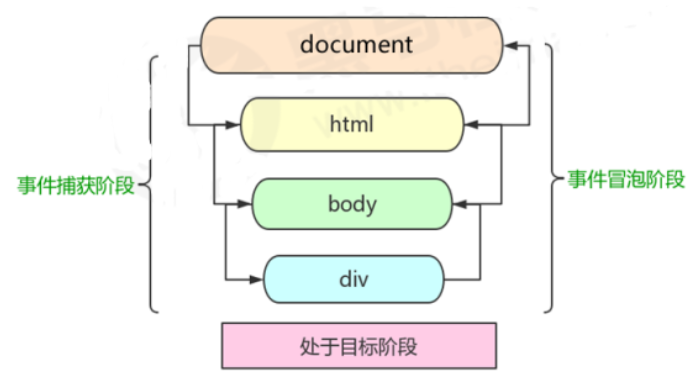
* 作用

不用分别为子元素绑定事件

为未知元素绑定事件

⑤事件冒泡：IE最早提出，事件开始时是由最具体的元素接收，然后逐级向上传播到DOM最顶层节点的过程

⑥事件捕获：由DOM最顶层节点开始，然后逐级向下传播到最具体的元素接收的过程

例如：我们向水里面扔一块石头，首先它会有一个下降的过程，这个过程就可以理解为从最顶层向事件发生的最具体元素（目标点）的捕获过程；之后会产生泡泡，会在最低点（ 最具体元素）之后漂浮到水面上，这个过程相当于事件冒泡。

注意：

1. JS 代码中只能执行捕获或者冒泡其中的一个阶段。
2. onclick 和 attachEvent 只能得到冒泡阶段。
3. addEventListener(type, listener[, useCapture])第三个参数如果是 true，表示在事件捕获阶段调用事件处理程序；如果是 false（不写默认就是false），表示在事件冒泡阶段调用事件处理程序。
4. 实际开发中我们很少使用事件捕获，我们更关注事件冒泡。
5. 有些事件是没有冒泡的，比如 onblur、onfocus、onmouseenter、onmouseleave
6. 事件冒泡有时候会带来麻烦有时候又会帮助很巧妙的做某些事件。
7. 事件对象
   1. 事件分类

– window 事件

• onload 加载（某个页面或图像被完成）

• onunload 用户退出页

• onresize 窗口或框架被调整尺寸

• onscroll 滚动条事件 – 鼠标事件

• onclick 鼠标点击

• ondblclick 鼠标双击

• onmousedown 鼠标按键按下

• onmouseup 鼠标按键被松开

• onmouseout 鼠标从某元素移开

• onmouseover 鼠标被移到某元素之上

• onmouseenter 鼠标进入某元素

• onmouseleave 鼠标离开某元素

• onmousemove 鼠标移动

• oncontextmenu 右键菜单

– input 事件

• onblur 元素失去焦点。文本输入框案例 →判断数字 isNaN → js 全局函数

• onfocus 元素获得焦点。input 输入框

• onchange 内容改变时触发。select 选择，select.value 默认选项

– 键盘事件

• onkeydown 按键按下

• onkeypress 按键按下并释放

• onkeyup 按键释放

– form 事件

• onreset 表单重置（重置按钮）

• onsubmit 表单提交(form 内有 text 被聚焦，直接回车可触发 onsubmit)

* 1. 什么是事件对象

鼠标在页面上滑动，肯定的是鼠标的onmouseemve事件，如何得到鼠标的坐标值呢？一定会有个地方存储着这些值，方便我们去读取。获取事件数据，不同的事件会有不同数据

绑定在事件上的匿名函数，只可以有一个参数，并且这个参数就代表事件对象

oDiv.onclick = function(ev){

//ev就是事件对象

}

//区别在于事件对象和和函数传参

eventTarget.onclick = function(event) {}

eventTarget.addEventListener('click', function(event) {}）

// 这个 event 就是事件对象，我们还喜欢的写成 e 或者 evt

* 官方解释：event 对象代表事件的状态，比如键盘按键的状态、鼠标的位置、鼠标按钮的状态。
* 简单理解：事件发生后，跟事件相关的一系列信息数据的集合都放到这个对象里面，这个对象就是事件对象event，它有很多属性和方法。
* 这个 event 是个形参，系统帮我们设定为事件对象，不需要传递实参过去。当我们注册事件时， event 对象就会被系统自动创建，并依次传递给事件监听器（事件处理函数）。

比如：

1. 谁绑定了这个事件。
2. 鼠标触发事件的话，会得到鼠标的相关信息，如鼠标位置。
3. 键盘触发事件的话，会得到键盘的相关信息，如按了哪个键。
   1. 事件对象的常见属性和方法

e.target 和 this 的区别：

this 是事件绑定的元素， 这个函数的调用者（绑定这个事件的元素）

e.target 是事件触发的元素。

e.type：返回事件的类型，比如click mouseover ，没有on

e.preventDefault()：阻止默认事件，比如不让链接跳转，不让表单按钮提交

* 1. 事件对象的相关属性

|  |  |
| --- | --- |
| 键盘属性 | 说明 |
| ev.keyCode | 键盘编号 |
| ev.ctrlKey | Ctrl键按下ev.ctrlKey==1 |
| ev.altKey | Alt键按下ev.altKey==1 |
| ev.shiftKey | Shift键按下ev.shiftKey==1 |
| 鼠标属性 | 说明 |
| ev.button | 鼠标按钮 0： 左键 1： 中键 2： |
| ev.clientX | 鼠标指针的水平坐标 |
| ev.clentY | 鼠标指针的垂直坐标 |
| ev.target | 触发事件的目标元素 |

1. 阻止事件冒泡
   1. 阻止事件冒泡的两种方式

事件冒泡：开始时由最具体的元素接收，然后逐级向上传播到到 DOM 最顶层节点。

事件冒泡本身的特性，会带来的坏处，也会带来的好处，需要我们灵活掌握。

阻止事件冒泡

1. 标准写法：利用事件对象里面的 stopPropagation()方法e.stopPropagation()
2. 非标准写法：IE 6-8 利用事件对象 cancelBubble 属性　e.cancelBubble = true;
   1. 阻止事件冒泡的兼容性解决方案

if(e && e.stopPropagation){

e.stopPropagation();

}else{

window.event.cancelBubble = true;

}

6.事件委托（代理、委派）

<ul>

　　<li>知否知否，应该有弹框在手</li>

　　<li>知否知否，应该有弹框在手</li>

　　<li>知否知否，应该有弹框在手</li>

　　<li>知否知否，应该有弹框在手</li>

　　<li>知否知否，应该有弹框在手</li>

</ul>

假如要实现点击每个li都会弹出对话框，以前需要给每个li注册事件按，是很繁琐的，而且访问DOM的次数越多，就会延长整个页面的交互就绪世间。这时就可以通过事件委托解决。

事件委托也称为事件代理， 在 jQuery 里面称为事件委派。

具体做法：

在ul上添加事件监听器而不是li上，这样点击每个li都会弹出对话框，要对具体的li进行操作，可用target对象实现。event.target就是你点击的那个li元素

7.常见鼠标事件

除了那些常见的on开头的，如onblur，onmouseover，onmouseout等，还有：

① 禁止鼠标右键菜单

contextmenu主要控制应该何时显示上下文菜单，主要用于程序员取消默认的上下文菜单

document.addEventListener('contextmenu', function(e) {

e.preventDefault();

})

② 禁止鼠标选中（selectstart 开始选中）

document.addEventListener('selectstart', function(e) {

e.preventDefault();

})

7.2 鼠标事件对象

event对象代表事件的状态，跟事件相关的一系列信息的集合。现阶段我们主要是用鼠标事件对象MouseEvent 和键盘事件对象KeyboardEvent。

|  |  |
| --- | --- |
| 鼠标属性 | 说明 |
| ev.button | 鼠标按钮 0： 左键 1： 中键 2： |
| ev.clientX | 鼠标指针的水平坐标 |
| ev.clentY | 鼠标指针的垂直坐标 |
| ev.target | 触发事件的目标元素 |
| ev.pageX | 返回鼠标相对于文档页面的X坐标 ，IE9+支持 |
| ev.pageY | 返回鼠标相对于文档页面的Y坐标 ，IE9+支持 |
| e.screenX | 返回鼠标相对于电脑屏幕的Y坐标 |
| e.screenY | 返回鼠标相对于电脑屏幕的Y坐标 |

1. 常用的键盘事件
2. onkeyup：某个键盘按键松开时触发；
3. onkeydown：某个键盘按键按下时触发；按住则一直触发；
4. onkeypress：也是按下触发，按住一直触发，但是区别于onkeydown的是，它不识别功能键，比如箭头，shift等。

注意： onkeydown 和 onkeyup 不区分字母大小写，onkeypress 区分字母大小写

|  |  |
| --- | --- |
| 键盘属性 | 说明 |
| ev.keyCode | 键盘编号 |
| ev.ctrlKey | Ctrl键按下ev.ctrlKey==1 |
| ev.altKey | Alt键按下ev.altKey==1 |
| ev.shiftKey | Shift键按下ev.shiftKey==1 |

--------------补

事件类型：

页面事件：如load、unload、error、resize、scroll、select、DOMActive，是用户与页面上的元素交互时触发的

|  |  |  |
| --- | --- | --- |
| **事件** | **浏览器支持** | **描述** |
| **onAbort** | HTML: 2 | 3 | 3.2 | 4  Browser:**IE4**|**N3**| O | 图片在下载时被用户中断 |
| **onBeforeUnload** | HTML: 2 | 3 | 3.2 | 4  Browser:**IE4**| N | O | 当前页面的内容将要被改变时触发的事件 |
| **onError** | HTML: 2 | 3 | 3.2 | 4  Browser:**IE4**|**N3**| O | 捕抓当前页面因为某种原因而出现的错误，如脚本错误与外部数据引用的错误 |
| **onLoad** | HTML: 2 | 3 | 3.2 |**4** Browser:**IE3**|**N2**|**O3** | 页面内空完成传送到浏览器时触发的事件，包括外部文件引入完成 |
| **onMove** | HTML: 2 | 3 | 3.2 | 4  Browser: IE |**N4**| O | 浏览器的窗口被移动时触发的事件 |
| **onResize** | HTML: 2 | 3 | 3.2 | 4  Browser:**IE4**|**N4**| O | 当浏览器的窗口大小被改变时触发的事件 |
| **onScroll** | HTML: 2 | 3 | 3.2 | 4  Browser:**IE4**| N | O | 浏览器的滚动条位置发生变化时触发的事件 |
| **onStop** | HTML: 2 | 3 | 3.2 | 4  Browser:**IE5**| N | O | 浏览器的停止按钮被按下时触发的事件或者正在下载的文件被中断 |
| **onUnload** | HTML: 2 | 3 | 3.2 |**4** Browser:**IE3**|**N2**|**O3** | 当前页面将被改变时触发的事件 |

。

焦点事件：如blur、DOMFocusIn、DOMFocusOut、focus、focusin、focusout，在元素获得或失去焦点的时候触发，这些事件当中，最为重要的是blur和focus，有一点需要引起注意，这一类事件不会发生冒泡！

表单事件：

|  |  |  |
| --- | --- | --- |
| **事件** | **浏览器支持** | **描述** |
| **onBlur** | HTML: 2 | 3 | 3.2 |**4** Browser:**IE3**|**N2**|**O3** | 当前元素失去焦点时触发的事件 [鼠标与键盘的触发均可] |
| **onChange** | HTML: 2 | 3 | 3.2 |**4** Browser:**IE3**|**N2**|**O3** | 当前元素失去焦点并且元素的内容发生改变而触发的事件 [鼠标与键盘的触发均可] |
| **onFocus** | HTML: 2 | 3 | 3.2 |**4** Browser:**IE3**|**N2**|**O3** | 当某个元素获得焦点时触发的事件 |
| **onReset** | HTML: 2 | 3 | 3.2 |**4** Browser:**IE4**|**N3**|**O3** | 当表单中RESET的属性被激发时触发的事件 |
| **onSubmit** | HTML: 2 | 3 | 3.2 |**4** Browser:**IE3**|**N2**|**O3** | 一个表单被递交时触发的事件 |

鼠标与滚轮事件：如click、dblclick、mousedown、mouseenter、mouseleave、mousemove、mouseout、mouseover、mouseup，是当用户通过鼠标在页面执行操作时所触发的

。

|  |  |
| --- | --- |
| 鼠标事件类型 | |
| **事件类型** | **说明** |
| click | 单击鼠标左键时发生，如果右键也按下则不会发生。当用户的焦点在按钮上并按了 Enter 键时，同样会触发这个事件 |
| dblclick | 双击鼠标左键时发生，如果右键也按下则不会发生 |
| mousedown | 单击任意一个鼠标按钮时发生 |
| mouseout | 鼠标指针位于某个元素上且将要移出元素的边界时发生 |
| mouseover | 鼠标指针移出某个元素到另一个元素上时发生 |
| mouseup | 松开任意一个鼠标按钮时发生 |
| mousemove | 鼠标在某个元素上时持续发生 |

早期版本IE浏览器 DOM对象.attachEvent(type,callback); type指的是DOM对象绑定的事件类型

标准浏览器DOM对象。addEventListener(type,callback,[capture]);

事件对象：当发生事件时，都会产生一个事件对象event，这个对象包含所有与事件相关的信息

常用的属性和方法 type返回当前事件的类型 target返回触发此事件的元素 currentTarget返回其事件监听器触发该事件的元素

bubbles表示事件是否是冒泡事件类型 cancelable表示事件是否取消默认动作 eventPhase返回事件传播的当前阶段 1表示捕获阶段 2表示处于目标阶段 3表示冒泡阶段

stopPropagation阻止事件冒泡 preventDefault()阻止默认行为

------------------------补

# 3. 文档事件

* DOMNodeInserted：在元素中添加子元素触发
* DOMNodeRemoved：元素中任意子元素被删除时触发
* DOMSubtreeModified：元素结构中发生任何变化时触发
* DOMNodeRemovedFromDocument：元素从文档中移除之前被触发
* DOMNodeInsertedIntoDocument：元素从文档中被添加之前触发
* hashchange：只能给window添加，指URL#后面的值变化的时候触发

一、  什么是javaScript对象？

        对象是javaScript中最重要的API。

        javaScript包含多种对象有：内置对象、外部对象（window对象、dom对象）、自定义对象。

二、  如何使用对象？

        对象包含属性和方法。

        访问对象属性：对象.属性

        访问对象方法：对象.方法（）

三、  常用内置对象

        JS中常用内置对象有：String对象、Number对象、Boolean对象、Array对象、Math对象、Date对象、RegExp对象、Function对象。

1.   String对象

（1）创建String对象有两种方式：

var str1=”hello word”;

var str2=new String(“hello word”);

（2）String对象的属性：

str.length;

（3）String对象的常用方法

##大小写转换方法

str1.toLowerCase()、str1.toUpperCase()

##获取指定字符

srt1.charAt(index):返回指定位置的字符

str1.charCodeAt(index)：返回指定位置字符的Unicode编码。（index表示字符位置）

##查询指定字符串

str1.indexOf(findstr,[index])、str1.lastIndexOf(findstr,[index])

PS:findstr表示要查询的字符串，lastIndexOf表示从最后开始找起，如果没有找到则返回-1.

##获取子字符串(含头不含尾，下标均是从0开始)

str1.substring(start,[end])

##替换子字符串

str1.replace(findstr,tostr)

PS:findstr为要找的子字符换，tostr替换为的字符串，返回替换后的字符串。

##拆分子字符串

Str1.split(bystr,[howmany])

PS:bystr分割用的字符串，howmany指定返回的数组的最大长度，可以省略，返回分割后的字符串。

注意：字符串是不可变的，一些可以修改的字符串的API，实际上是返回一个新的字符串，并不改变原始字符串.

2.Number对象

（1）Number是数值对象，创建Number对象如下：var a=1.1;

（2）Number对象的常用方法

##toFixed(num):转换为字符串，并保留小数点后一定位数。如果必要该数字可以被舍弃，也可以用0补足。

3.Array对象

##JS中的数组均是Object，如var a1=[1,25,"男朋友",true];

数组的长度是可变的。

##JS中的数组可以像集合一样操作

var a2=new Array();

   a2.push(8);

   a2.push(false);

   console.log(a2[2]);

##访问数组元素console.log（a1[0]）

##数组中元素倒转：a1.reverse();

4.Math对象

Math对象用于执行数学任务，无需创建，直接使用即可。如下所示：

console.log(Math.PI);console.log(Math.E);

console.log(Math.round(Math.PI));

PS：Math.round（）表示为取整。

5.Date对象

Date对象用于处理日期和时间，封装了系统时间毫秒数。

##创建Date对象(Date对象会自动把当前的日期和时间保存为初始值)

Datedate1=new Date();

Datedate2=new Date(“2015/1/1 1:12:11”);

##Date对象常用方法

读写时间毫秒数：date1.getTime()、date1.setTime(毫秒)

读写时间分量：

date1.getDate()、date1.getDay()、date1.getFullYear()等。

date1.setDate()、date1.setDay()、date1.setFullYear()等。

转化为字符串：

toString()、toLocaleTimeString()、toLocaleDateString()

6.RegExp对象

RegExp对象表示正则表达式,作用也就是用来检查字符串。

##创建RegExp对象

VarrgExp=new RegExp(“pattern”,[“flags”]);

VarrgExp=/pattern/flags;

PS:flags的标识有两种

（1）   g:设定当前匹配为全局模式。

（2）   i:忽略匹配中的大小写检测。

注意：全局模式（第N次调用则找出第N个匹配的部分），非全局模式（找出首次匹配的部分）。

##RegExp对象的常用方法

RegExpObject.test(string)\*重点\*

如果字符串string中含有与RegExpObject匹配的文本，则返回true,否则返回false.

reg.exec(str):检测字符串中指定的值，返回指定的值。

reg.test(str):检测字符串中指定的值，返回false或true.

##String对象与正则表达式（regexp代表正则表达式或字符串）

--x.replace(regexp,tostr),返回替换后的结果。

--x.match(regexp)，返回匹配字符串的数组。

--x.search(regexp)，返回匹配字符串的首字符位置索引。

7.Function对象

JS中的函数就是Function对象。

函数名就是指向Function对象的引用，使用函数名可以访问函数对象，函数名（）是调用函数。

##定义函数对象

函数的返回值默认为undefined,也可以使用return返回具体的值。

##函数的参数

JS的函数没有重载。

调用时只要函数名一样，无论传入多少个参数，调用的都是同一个函数。

没有接收到实参的参数值是undefined.

所有的参数传递给arguments数组对象。

##arguments对象\*重点\*..

Arguments是一种特殊的对象，在函数代码中，表示函数的参数数组。

在函数代码中，可以使用arguments访问所有参数。

arguments.length:函数的参数个数

arguments[i]:第i个参数

可以利用arguments实现可变参数的函数。

四、  外部对象

     浏览器提供的对象，浏览器的API，用来操作浏览器。

1.BOM(Browser Object Model)：

     #浏览器对象模型，用来访问和操纵浏览器窗口，使JAVAScript有能力与浏览器“对话”。（浏览器与JS是两个不相关的内容，JS依附于浏览器上）

#通过使用BOM，可移动窗口、更改状态栏文本，执行其他不与页面内容发生直接联系的操作。

2.DOM（Document Object Model）:

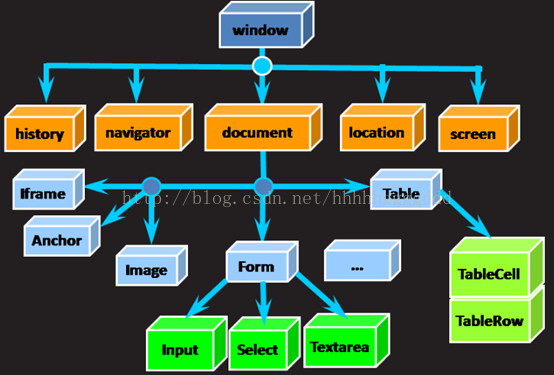
     #文档对象模型，用来操作文档。定义了访问和操作HTML文档的标准方法。

     #应用程序通过对DOM树的操作，来实现对HTML文档数据的操作。

      PS:window代表浏览器，它的子对象（属性）代表浏览器的不同部位。

外部对象分为两个部分BOM与DOM，其中BOM包含DOM.

BOM与DOM关系如下图示：



3.window对象（BOM）

      window表示浏览器窗口，所有JAVAScript全局对象、函数以及变量均自动成为window对象的成员。

##常用属性：

#history:浏览过窗口的历史记录对象。

       history对象包含用户（在浏览器窗口中）访问过的URL，常用方法：

--back()

--forward()

--go(num):num可以取正数或者负数，正数表示前进，负数表示后退。

#location:窗口文件地址对象。

location对象常用于获取和改变当前浏览器的网址。

href:当前窗口正在浏览器的网页地址,也可以用于修改网址。

--reload():重新载入当前网址，就像按下刷新按钮。

PS：location与超链接都可以改变当前窗口，两者的区别是---location改变当前窗口时可以添加条件，比如可以在页面上显示“您确定要离开此页面吗？”，而超链接不能添加条件，不灵活，一般建议采用location。

显示小代码：

functiontarena() {

   var b=confirm("您确定要离开此页面吗？");

   if(b){

        //确认为true时，跳转。

        //location=地址栏

        //修改href就是修改网址。

        location.href="http://www.tmooc.cn";

    }

     }

 #screen:当前屏幕对象。

        screen对象包含有关客户端显示屏幕（比如说ipad、笔记本电脑、手机）的信息，常用于获取屏幕的分辨率和色彩，常用属性：

--width/height

--availWidth/availHeight

#navigator：浏览器相关信息。

常用于获取客户端浏览器和操作系统信息。

Navigator.userAgent;

##常用方法：

(1)alter(str),提示对话框，显示str字符串的内容。

(2)confirm()，确认对话框，显示str字符串的内容，按确定按钮返回true，其他操作返回false.

4.document对象

4.1  DOM(文档对象模型)

当网页被加载时，浏览器会创建页面的文档对象模型。

DOM节点树（DOM模型被构造为对象的树，这棵树的根就是document对象）

#DOM操作：查找节点、读取节点信息、修改节点信息、创建新节点、删除节点。

4.2          读取、修改

##读取节点的名称和类型

#nodeName:节点名称

--元素节点和属性节点：标签或属性名称。

--文本节点：永远是#text.

--文档节点：永远是#document

#nodeType:节点类型（返回数值）

元素节点返回1，属性节点返回2，文本节点返回3，注释节点返回8，文档节点返回9.

##读写节点的内容（<x>这就是内容</x>),双标签中间的文本。

#innerHTML：设置或获取位于对象起始和结束标签内的HTML（带标签）。

#innerText:设置或获取位于对象起始和结束标签内的文本（不带标签）。

##读写节点的属性

#通用API,任何版本浏览器都支持

getAttribute()方法：根据属性名称获取属性的值。

setAttribute(),removeAttribute().

#新的API，低版本的浏览器不支持（只要增和改）

节点.属性名（class例外，需要写className）

PS：凡是在funciton外的方法，读取时均是按照顺序读取。

4.3查询

##通过id查询（标准）

document.getElementById()，通过制定的ID来返回元素节点，忽略文档的结构，如果ID错误，则返回null。

##根据标签名查询（标准）

#在整个页面上根据标签名查询节点。

document.getElementsByTagName(“li”),

#在某节点下根据标签名查询其子节点。

ul. getElementsByTagName(“li”),

PS:”li”为要查询的子节点。

##根据name查询（标准）

document.getElementsByName().

##根据class查询某节点（非标准）

getElementsByClassName(),

##根据层次（关系）查询

#查询父亲（标准）

parentNode

#查询孩子（标准）

childNodes.

getElementsByTagName(“li”)，建议使用此根据标签名查询孩子的方法。

#查询孩子（非标准）

children.

#查询兄弟（标准）

//某节点.父亲.孩子[n]

jd.parentNode. getElementsByTagName(“li”)[1];

#查询兄弟（非标准）

jd.previousElementSibling;

jd.nextElementSibling;

4.4增删节点

##创建节点

document.createElement(“li”);

li为要创建的元素标签名称，返回新创建的节点。

##增加节点

#追加到所有子节点之后

parentNode.appendChild(newNode);新节点作为父节点的最后一个子节点添加。

 #插入到某子节点之前

parentNode.insertBefore(newNode,refNode);refNode为参考节点，新节点位于此节点之前添加。

##删除节点

#父亲删孩子

node.removeChild(childNode);node为childNode的父节点。

#自己删自己（非标准）

（1）sz.parentNode.removeChild("sz");

（2）var tj=document.getElementById("tj");

      tj.remove();

小案例：联动菜单代码：

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>联动菜单</title>

</head>

<body>

       省：

   <select id="province" οnchange="chg();">

        <option value="-1">请选择</option>

        <option value="0">河北省</option>

        <option value="1">山东省</option>

        <option value="2">山西省</option>

   </select>

   市：

   <select id="city">

        <option>请选择</option>

   </select>

   <script>

      //模拟加载城市

      function loadCities() {

     return [

         ["石家庄","廊坊","保定"],

         ["济南","青岛","德州"],

         ["太原","大同","阳泉"]

             ];

      }

     var cities=loadCities();

     function chg() {

    //获取省份的下拉选

    var sel1=document.getElementById("province");

    //获取省份的序号

    var pindex=sel1.value;

    //获取该省份对应的城市

    var pcities=cities[pindex];

    //先删除城市下拉选内所有的城市

    var sel2=document.getElementById("city");

    //

    //sel2.innerHTML="<option>请选择</option>";

    var options=sel2.getElementsByTagName("option");

    for(var i=options.length-1;i;i--){

    sel2.removeChild(options[i]);

    }

    //再增加该省份的城市。

    if(pcities){

    for(var i=0;i<pcities.length;i++){

    var opn=document.createElement("option");

    opn.innerHTML=pcities[i];

    sel2.appendChild(opn);

    }

    }

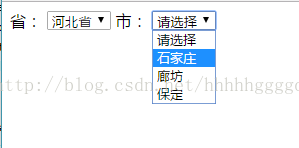
     }

   </script>

</body>

</html>

 网页显示效果如下：



五、自定义对象

自定义对象是一种特殊的数据类型，由属性和方法封装而成。

---属性指与对象有关的值:对象名.属性名。

---方法指对象可以执行的行为或者可以完成的功能：对象名.方法名（）

1.创建自定义对象

（1）采用直接量（JSON"JavaScript Object Notation"是一种轻量级的数据交换格式）的方式创建自定义对象

名称需要使用“”引起来，多对定义之间使用，隔开，名称可以是属性。

function f1() {

  var stu= {"name":"zs","age":25,"work":function(){alert("我学Java")}};

  alert(stu.name);

  alert(stu.age);

  //work的function方法无返回值，则不需要加alert。

  stu.work();

   }

（2）采用构造器创建对象

##采用自定义构造器创建对象（自定义了一个构造器Coder，然后每次写时都可以调用Coder构造器）

function f3() {

  var c=new Coder("lisi",26,function(){alert("我写Java")});

  alert(c.name);

  alert(c.age);

  c.work();

   }

   function Coder(name,age,work) {

  //this=创建出来的对象

  //this.name给该对象声明一个属性

  this.name=name;

  this.age=age;

  this.work=work;

   }

*JavaScript*对象知识点总结

（*1*）核心对象：

|  |  |
| --- | --- |
| 对象 | 说明 |
| *Array* | 表示数组 |
| *Boolean* | 表示布尔值 |
| *Date* | 表示日期 |
| *Math* | 提供了基本的数学常量和函数 |
| *Number* | 表述实数数值 |
| *Object* | 包含了由所有 *Javascript* 对象共享的基本功能 |
| *RegExp* | 表述了一个正则表达式，同时包含了由所有正则表达式对象共享  的静态属性 |
| *String* | 表述了一个 *Javascript* 字符串 |

对于这些对象，我们可以使用*new*操作符创建相应的对象实例，也可以隐含地声明*/*创建这些对象。例如*: var name=”wzk”;//*等价于：*var name=new String(“wzk”);*

（*2*）*Window*对象属性

|  |  |
| --- | --- |
| 属性 | 说明 |
| *defaultStatus* | 默认的状态条信息 |
| *document* | 当前显示的文档（该属性本身是一个 *document* 对象） |
| *frame* | 窗口里的一个框架（该属性本身也是一个对象） |
| *frames* | *array* 列举窗口的框架对象的数组，按照这些对象在文档中  出现的顺序列出（该属性本身也是一个对象） |
| *history* | 窗口的历史列表（该属性本身也是一个对象，即 *history* 对象） |
| *length* | 窗口内的框架数 |
| *location* | 窗口所显示文档的完整 */* 绝对 *URL* 对象 |
| *name* | 窗口打开时，赋予该窗口的名字 |
| *parent* | 包含当前框架的窗口的同义词 |
| *self* | 当前窗口或框架的同义词 |
| *status* | 状态条中的消息 |
| *top* | 包含当前框架的最顶层浏览器窗口的同义词 |
| *window* | 当前窗口或框架的同义词，与 *self* 相同 |

*<?xml:namespace prefix = o ns = "urn:schemas-microsoft-com:office:office" />*

（*3*）*Window*对象的方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| *alert()* | 打开一个 *Alert* 消息框 |
| *clearTimeout()* | 用来终止 *setTimeout* （）方法的工作 |
| *close()* | 关闭窗口 |
| *confirm()* | 打开一个 *Confirm* 消息框，用户可以选择 *OK*  或者 *Cancel* ，如果用户单击 *OK* 按钮，则该方法  返回 *true* ，单击 *Cancel* 按钮，则返回 *false* |
| *blur()* | 把焦点从指定窗口移开 |
| *focus()* | 把指定的窗口带到前台 |
| *open()* | 打开一个新窗口 |
| *prompt()* | 打开一个 *Prompt* 对话框，用户可向该框输入文本，  并把输入的文字返回到脚本 |
| *setTimeout()* | 等待一段指定的毫秒数时间，然后运行指令事件处理程序 |

（*4*）*document*对象属性

|  |  |
| --- | --- |
| 属性 | 说明 |
| *alinkColor* | 活动链接的颜色 |
| *anchor* | 一个 *HTML* 锚点。使用 *<a name=>*  标记创建（该属性本身也是一个对象） |
| *anchors* | *array* 列出文档锚点对象的数组 |
| *bgColor* | 文档的背景颜色 |
| *cookie* | 存储于 *Cookie* 文件内的一段消息，它是该  文档对象的一个属性 |
| *fgColor* | 文档的文本颜色 |
| *form* | 文档中的一个窗体，有 *<form>* 标记创建（该属性本身也是  一个对象） |
| *forms* | *array* 按照其出现在文档中的顺序列出窗体对象的一个数组 |
| *lastModified* | 文档最后的修改日期 |
| *linkColor* | 文档链接的颜色 |
| *link* | 文档中一个 *<a href>* 标记 |
| *links* | *array* 文档中 *link* 对象的一个数组，按照他们出现的文档中  顺序排列（该属性本身也是一个对象） |
| *location* | 当前显示文档的 *URL* |
| *referrer* | 包含链接的文档的 *URL* ，用户单击该链接可到达当前文档 |
| *title* | 文档标题 |
| *vlinkColor* | 指定用户已观察过的文档链接文本颜色 |

# string

### 方法一：charAt()：返回在指定位置的字符

var str = 'nihaome wozhendexiangxeuhaoqianduan'var str2 = str.charAt(4);//字符串的下标是从1开始的,所以返回ovar str3 = str.charAt(25); //25位置是a;var str4 = str.charAt(5,26); //传入两个参数，不会报错，但是第二个参数默认无效，//所以返回的是位置5的字符m

console.log(str2);

console.log(str3);

console.log(str4);

### 方法二：charCodeAt()：返回咋指定位置的字符的Unicode编码

var str = "llinihao mezaigama";var str2 = str.charCodeAt(5);//位置5的字符是h，h的Unicode编码是104var str3 = str.charCodeAt(3);

console.log(str3); //110

console.log(str2);

### 方法三：conca()：连接字符串

var str1 = "lixiaoqi";var str2 = 'zaijian';

console.log(str2.concat(str1));//把str1连接到str2上

### 方法四： indexOf()： 检索字符串，返回的是字符在字符串的下标

var str1 = "lixiaoqi";var num1 = str1.indexOf('o'); //检索字符串，返回的是字符在字符串的下标//字符串的下标是从0开始计数var num2 = str1.indexOf('lllll'); //没有找到匹配的值，返回-1var num3 = str1.indexOf('ni'); //没有找到匹配的值，返回-1

console.log(num1);

console.log(num2);

console.log(num3);

### 方法五：match()：在字符串内检索指定的值或找到一个或多个正则表达式的匹配，返回的是值而不是值的位置。

var str1 = "lixiaoqi";var str2 = str1.match('iao'); //检索字符串，返回的是字符var str3 = str1.match('lll');//没有指定的字符，返回的是null

console.log(str2);

console.log(str3);

### 方法六：replace()：替换匹配的字符串

var str1 = "nihaome";var str2 = "lixiaoqi";var str3 = str1.replace(str1.match('hao'),str2.match('lixi'));//找到str1的hao，找到str2的lixi，用str2替换掉str1

console.log(str3);

### 方法七：search()：检索与字符串匹配的子串，返回的是地址

var str1 = "lixiaoqi nihaoya zaiganma";var str2 = "nihaoya";var str3 = "wobuzai";var num1 = str1.search(str2);//返回的是第一个字符所在的位置var num2 = str1.search(str3); //未检测到返回-1

console.log(num1);

console.log(num2);

### 方法8：slice()：提取字符串片段，并在新的字符串中返回被提取的部分

var str1 = 'nihaoya wobuhaoya'

console.log(str1.slice(4,9)); //4位置开始，到9的前一个位置结束

console.log(str1.slice(2,14));//start位置开始，end前一个位置结束

console.log(str1);//原字符串不变

console.log(str1.slice(-1,0)); //返回的是空字符串

console.log(str1.slice(30,100)); //长度超过字符串的长度，返回空字符串

console.log(str1.slice(-1,10));//返回的是空字符串

### 方法9：split()：把字符分割成数组

var str1 = "nihao-zaijian-dajiahao";

console.log(str1.split(''));var str2 = str1.split('-');

console.log(str2);

console.log(str1);//原字符串不变

### 方法10：toLocaleLowerCase()：把字符串转换成小写

var str = "NIHAOME WOHENHAOnizaiganmane";

console.log(str.toLocaleLowerCase());var str1 = str.toLocaleLowerCase();

console.log('-----------');

console.log(str1);

console.log(str);//原字符串不变

### 方法11：toLocaleUpperCase()：把字符串准换成大写

var str = "deawwWWEzaiganmane";

console.log(str.toLocaleUpperCase());var str1 = str.toLocaleUpperCase();

console.log('-----------');

console.log(str1);

console.log(str);//原字符串不变

### 方法12：toLowerCase()：把字符串转换成小写

var str = "deawwWWEzaiganmane";

console.log(str.toLowerCase());var str1 = str.toLowerCase();

console.log('-----------');

console.log(str1);

console.log(str);//原字符串不变

### 方法13：toUpperCase():把字符串转换成大写

var str = "deawwWWEzaiganmane";

console.log(str.toUpperCase());var str1 = str.toUpperCase();

console.log('-----------');

console.log(str1);

console.log(str);//原字符串不变

### 方法14：substr()：从起始索引号提取字符串中指定书目的字符

var str = "nihaoya wohenhao dajiahao"var str3 = str.substr(3,19);//字符串中空格不占位置，从3开始，到19结束

console.log(str3);

### 方法15：subString()：提取字符串中两个指定索引号之间的字符

var str = "nihaoya wohenhao dajiahao"var str3 = str.substring(3,19);//字符串中空格占位置，从3开始，到18结束var str4 = str.substr(3,19);//空格不占位。从3开始到19结束

console.log(str4);

console.log('-------------');

console.log(str3);

# 数组array

### 方法1：slice[start,end]：返回从原数组中指定开始下表到结束下表之间的项组成的新数组（原数组不变）

* 1个参数：n，即n到末尾的所有
* 2个参数：[start,end]

var arr = [1,2,3,4,5,6,7,8];var arr1 = arr.slice(2,6);

console.log(arr1);//数组下表从0开始，从2开始，到6的前一个位置结束var arr3 = arr.slice(4); //从4开始到结束

console.log(arr3);

### 方法2：splice()：

* 删除，2个参数，起始位置，删除的项数

var arr = [1,2,3,4,5];var arr1 = arr.slice(2,3); //实际得到的新数组的项数是end-strat；

console.log(arr1);var arr2 = arr.slice(1,5);

console.log(arr2);

console.log(arr);

* 插入：3个参数，起始位置，删除的项数，插入的项。
* 替换，任意参数，起始位置，删除的项数，插入任意数量的项数

### 方法3：pop()：删除数组的最后一个元素，减少数组的长度，返回被删除的值

var arr = [1,2,3,4,5];

console.log(arr.length);

console.log('-----------');var arr1 = arr.pop();

console.log(arr1); //5

console.log(arr.length);

### 方法4：push()：将参数添加到数组的最后，返回新数组的长度

var arr = [1,2,3,4,5];

console.log(arr.length);

console.log('-----------');var num = arr.push(6,7,8);

console.log(num); //返回的是新数组的长度

console.log(arr); //原数组被改变成新数组

### 方法5:shift()：删除数组的第一个参数，数组的长度减1，无参

var arr = [1,2,3,4,5];

console.log(arr.length);

console.log('-----------');var arr1 = arr.shift();

console.log(arr1); //返回的是删除的数组的值

console.log(arr); //原数组被改变成新数组

### 方法6：unshift()：像数组的开头添加一个或更多的元素，并返回新的长度。（参数不限）

var arr = [1,2,3,4,5];

console.log(arr.length);

console.log('-----------');var arr1 = arr.unshift(2,4,5,{name:'liqi'});

console.log(arr1); //返回的是新数组的长度

console.log('------------');

console.log(arr); //原数组被改变成新数组

### 方法7：sort():按指定的参数对数组进行排序，返回的值是经过排序后的数组（无参，函数）

var arr = [1,'q',3,6,2,'a',9];

console.log(arr);//原数组var arr1 = arr.sort();

console.log(arr1);//返回的是排序后的数组

console.log('-------------------');

console.log(arr);//原数组变成新数组

### 方法8：concat(3,4)：把两个字符串连接起来，返回的值是一个副本（参数不限）

var arr1 = [1,'q',3,6,2,'a',9];var arr2 = [{name:'niho'},1,4];

console.log(arr1.concat(3,arr1));//新创建一个数组，原数组不变

console.log(arr1);

console.log(arr2);//原数组不变

### 方法9：join()：将数组的元素组成一个字符串，以separator（分割器）

var arr1 = [1,'q',3,6,2,'a',9];var arr2 = arr1.join(',');//以逗号分隔var arr3 = arr1.join('');//以空格分隔

console.log(arr1);//原数组不变

console.log(arr2);

console.log(arr3);

### 方法10：indexOf():从数组的开头向后查找，接受两个参数，要查找的项和查找起点的位置索引

var arr = [1,2,3,4,5,6,78];var num = arr.indexOf(78,3);//查找78所在的位置

console.log(num); //返回的项的索引var num1 = arr.indexOf(3,1);

console.log(num1);var num2 = arr.indexOf(2);

console.log(num2);

console.log(arr);

### 方法10 ：foeEach():对数组的每个元素执行一次提供的函数。

var array1 = ['a', 'b', 'c'];

array1.forEach(function(element) {

console.log(element);});

### 方法11：对数组的每一项运行给定的函数，返回没戏函数调用的结果组成的数组

var array1 = [1, 4, 9, 16];const map1 = array1.map(x => x \* 2); //对数组的每项\*2

console.log(map1);

### for in方法：遍历数组

var arr = [1, 4, 9, 16];for (var i = 0; i < arr.length; i++){

console.log(arr[i]);}

### ES5-every和some方法：

1. every是所有函数的每个回调都返回true的时候才会返回true，当遇到false的时候终止执行，返回false。
2. some函数是存在有一个函数返回true的时候终止执行并返回true，否则返回false。

// everyvar arr = [1,6,8,-2,-5,7,-4]var isPositive = arr.every(function(value){

return value > 0;})

console.log(isPositive) // false//somevar arr = [1,6,8,-2,-5,7,-4]var isPositive = arr.some(function(value){

return value > 0;})

console.log(isPositive) // true

### ES5-Array.isArray(obj)

这是Array对象的一个静态函数，用来判断一个对象是不是数组

var arr = [3,4,5]

console.log(Array.isArray(arr)); // true// ES3的判断方法

console.log(typeof arr); // object

console.log(arr.toString()); // 3,4,5

console.log(arr instanceof Array); //true

### ES5-filter(function(element))

返回数组的一个子集，回调函数用于逻辑判断是否返回，返回true则把当前元素加到返回数组中，false则不加  
新数组只包含返回true的值，索引确实的不包括，原数组保持不变。

var arr = [1,6,8,-2,-5,7,-4]var positiverArr = arr.filter(function(value){

return value > 0})

console.log(positiverArr); // [1, 6, 8, 7]

console.log(arr); // [1, 6, 8, -2, -5, 7, -4]

### reduce(function(v1,v2),value) / .reduceRight(functio(v1,v2),value)

遍历数组，调用回调函数，将数组元素组合成一个值，reduce从索引最小值开始，reduceRight反向，方法有两个参数

1. 回调函数，把两个值合成一个，返回结果
2. value,一个初始值，可选

var arr = [1,2,3,4,5,6]

console.log(arr.reduce(function(v1,v2){

return v1 + v2;})) // 21//开始是1+2 = 3，之后3+3 =6，之后6+4 =10，之后//10+5 =15，最后15+6 =21

console.log(arr.reduce(function(v1,v2){

return v1 - v2;},100)) // 79//开始的时候100-1=99，之后99-2=97，之后97-3=94，之后//94-4=90，之后90-5=85，最后85-6=79

**七、对象**

**一、什么是javaScript对象？**

对象是javaScript中最重要的API。

javaScript包含多种对象有：内置对象、外部对象（window对象、dom对象）、自定义对象。

**二、如何使用对象？**

对象包含属性和方法。

访问对象属性：对象.属性

访问对象方法：对象.方法（）

**三、常用内置对象**

JS中常用内置对象有：String对象、Number对象、Boolean对象、Array对象、Math对象、Date对象、RegExp对象、Function对象。

1. String对象

（1）创建String对象有两种方式：

var str1=”hello word”;

var str2=new String(“hello word”);

（2）String对象的属性：

str.length;

（3）String对象的常用方法

##大小写转换方法

str1.toLowerCase()、str1.toUpperCase()

##获取指定字符

srt1.charAt(index):返回指定位置的字符

str1.charCodeAt(index)：返回指定位置字符的Unicode编码。（index表示字符位置）

##查询指定字符串

str1.indexOf(findstr,[index])、str1.lastIndexOf(findstr,[index])

PS:findstr表示要查询的字符串，lastIndexOf表示从最后开始找起，如果没有找到则返回-1.

##获取子字符串(含头不含尾，下标均是从0开始)

str1.substring(start,[end])

##替换子字符串

str1.replace(findstr,tostr)

PS:findstr为要找的子字符换，tostr替换为的字符串，返回替换后的字符串。

##拆分子字符串

Str1.split(bystr,[howmany])

PS:bystr分割用的字符串，howmany指定返回的数组的最大长度，可以省略，返回分割后的字符串。

注意：字符串是不可变的，一些可以修改的字符串的API，实际上是返回一个新的字符串，并不改变原始字符串.

2.Number对象

（1）Number是数值对象，创建Number对象如下：var a=1.1;

（2）Number对象的常用方法

##toFixed(num):转换为字符串，并保留小数点后一定位数。如果必要该数字可以被舍弃，也可以用0补足。

3.Array对象

##JS中的数组均是Object，如var a1=[1,25,"男朋友",true];

数组的长度是可变的。

##JS中的数组可以像集合一样操作

var a2=new Array();

a2.push(8);

a2.push(false);

console.log(a2[2]);

##访问数组元素console.log（a1[0]）

##数组中元素倒转：a1.reverse();

4.Math对象

Math对象用于执行数学任务，无需创建，直接使用即可。如下所示：

console.log(Math.PI);console.log(Math.E);

console.log(Math.round(Math.PI));

PS：Math.round（）表示为取整。

5.Date对象

Date对象用于处理日期和时间，封装了系统时间毫秒数。

##创建Date对象(Date对象会自动把当前的日期和时间保存为初始值)

Datedate1=new Date();

Datedate2=new Date(“2015/1/1 1:12:11”);

##Date对象常用方法

读写时间毫秒数：date1.getTime()、date1.setTime(毫秒)

读写时间分量：

date1.getDate()、date1.getDay()、date1.getFullYear()等。

date1.setDate()、date1.setDay()、date1.setFullYear()等。

转化为字符串：

toString()、toLocaleTimeString()、toLocaleDateString()

6.RegExp对象

RegExp对象表示正则表达式,作用也就是用来检查字符串。

##创建RegExp对象

VarrgExp=new RegExp(“pattern”,[“flags”]);

VarrgExp=/pattern/flags;

PS:flags的标识有两种

（1）g:设定当前匹配为全局模式。

（2）i:忽略匹配中的大小写检测。

注意：全局模式（第N次调用则找出第N个匹配的部分），非全局模式（找出首次匹配的部分）。

##RegExp对象的常用方法

RegExpObject.test(string)\*重点\*

如果字符串string中含有与RegExpObject匹配的文本，则返回true,否则返回false.

reg.exec(str):检测字符串中指定的值，返回指定的值。

reg.test(str):检测字符串中指定的值，返回false或true.

##String对象与正则表达式（regexp代表正则表达式或字符串）

--x.replace(regexp,tostr),返回替换后的结果。

--x.match(regexp)，返回匹配字符串的数组。

--x.search(regexp)，返回匹配字符串的首字符位置索引。

7.Function对象

JS中的函数就是Function对象。

函数名就是指向Function对象的引用，使用函数名可以访问函数对象，函数名（）是调用函数。

##定义函数对象

函数的返回值默认为undefined,也可以使用return返回具体的值。

##函数的参数

JS的函数没有重载。

调用时只要函数名一样，无论传入多少个参数，调用的都是同一个函数。

没有接收到实参的参数值是undefined.

所有的参数传递给arguments数组对象。

##arguments对象\*重点\*..

Arguments是一种特殊的对象，在函数代码中，表示函数的参数数组。

在函数代码中，可以使用arguments访问所有参数。

arguments.length:函数的参数个数

arguments[i]:第i个参数

可以利用arguments实现可变参数的函数。

**四、外部对象**

浏览器提供的对象，浏览器的API，用来操作浏览器。

1.BOM(Browser Object Model)：

#浏览器对象模型，用来访问和操纵浏览器窗口，使JAVAScript有能力与浏览器“对话”。（浏览器与JS是两个不相关的内容，JS依附于浏览器上）

#通过使用BOM，可移动窗口、更改状态栏文本，执行其他不与页面内容发生直接联系的操作。

2.DOM（Document Object Model）:

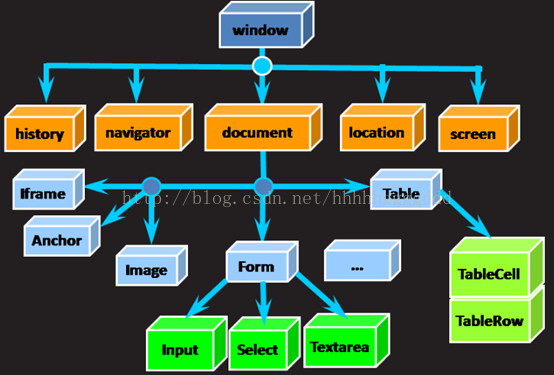
#文档对象模型，用来操作文档。定义了访问和操作HTML文档的标准方法。

     #应用程序通过对DOM树的操作，来实现对HTML文档数据的操作。

      PS:window代表浏览器，它的子对象（属性）代表浏览器的不同部位。

外部对象分为两个部分BOM与DOM，其中BOM包含DOM.

BOM与DOM关系如下图示：



3.window对象（BOM）

      window表示浏览器窗口，所有JAVAScript全局对象、函数以及变量均自动成为window对象的成员。

##常用属性：

#history:浏览过窗口的历史记录对象。

       history对象包含用户（在浏览器窗口中）访问过的URL，常用方法：

--back()

--forward()

--go(num):num可以取正数或者负数，正数表示前进，负数表示后退。

#location:窗口文件地址对象。

location对象常用于获取和改变当前浏览器的网址。

href:当前窗口正在浏览器的网页地址,也可以用于修改网址。

--reload():重新载入当前网址，就像按下刷新按钮。

PS：location与超链接都可以改变当前窗口，两者的区别是---location改变当前窗口时可以添加条件，比如可以在页面上显示“您确定要离开此页面吗？”，而超链接不能添加条件，不灵活，一般建议采用location。

显示小代码：

functiontarena() {

   var b=confirm("您确定要离开此页面吗？");

   if(b){

        //确认为true时，跳转。

        //location=地址栏

        //修改href就是修改网址。

        location.href="http://www.tmooc.cn";

    }

     }

 #screen:当前屏幕对象。

        screen对象包含有关客户端显示屏幕（比如说ipad、笔记本电脑、手机）的信息，常用于获取屏幕的分辨率和色彩，常用属性：

--width/height

--availWidth/availHeight

#navigator：浏览器相关信息。

常用于获取客户端浏览器和操作系统信息。

Navigator.userAgent;

##常用方法：

(1)alter(str),提示对话框，显示str字符串的内容。

(2)confirm()，确认对话框，显示str字符串的内容，按确定按钮返回true，其他操作返回false.

4.document对象

4.1  DOM(文档对象模型)

当网页被加载时，浏览器会创建页面的文档对象模型。

DOM节点树（DOM模型被构造为对象的树，这棵树的根就是document对象）

#DOM操作：查找节点、读取节点信息、修改节点信息、创建新节点、删除节点。

4.2          读取、修改

##读取节点的名称和类型

#nodeName:节点名称

--元素节点和属性节点：标签或属性名称。

--文本节点：永远是#text.

--文档节点：永远是#document

#nodeType:节点类型（返回数值）

元素节点返回1，属性节点返回2，文本节点返回3，注释节点返回8，文档节点返回9.

##读写节点的内容（<x>这就是内容</x>),双标签中间的文本。

#innerHTML：设置或获取位于对象起始和结束标签内的HTML（带标签）。

#innerText:设置或获取位于对象起始和结束标签内的文本（不带标签）。

##读写节点的属性

#通用API,任何版本浏览器都支持

getAttribute()方法：根据属性名称获取属性的值。

setAttribute(),removeAttribute().

#新的API，低版本的浏览器不支持（只要增和改）

节点.属性名（class例外，需要写className）

PS：凡是在funciton外的方法，读取时均是按照顺序读取。

4.3查询

##通过id查询（标准）

document.getElementById()，通过制定的ID来返回元素节点，忽略文档的结构，如果ID错误，则返回null。

##根据标签名查询（标准）

#在整个页面上根据标签名查询节点。

document.getElementsByTagName(“li”),

#在某节点下根据标签名查询其子节点。

ul. getElementsByTagName(“li”),

PS:”li”为要查询的子节点。

##根据name查询（标准）

document.getElementsByName().

##根据class查询某节点（非标准）

getElementsByClassName(),

##根据层次（关系）查询

#查询父亲（标准）

parentNode

#查询孩子（标准）

childNodes.

getElementsByTagName(“li”)，建议使用此根据标签名查询孩子的方法。

#查询孩子（非标准）

children.

#查询兄弟（标准）

//某节点.父亲.孩子[n]

jd.parentNode. getElementsByTagName(“li”)[1];

#查询兄弟（非标准）

jd.previousElementSibling;

jd.nextElementSibling;

4.4增删节点

##创建节点

document.createElement(“li”);

li为要创建的元素标签名称，返回新创建的节点。

##增加节点

#追加到所有子节点之后

parentNode.appendChild(newNode);新节点作为父节点的最后一个子节点添加。

 #插入到某子节点之前

parentNode.insertBefore(newNode,refNode);refNode为参考节点，新节点位于此节点之前添加。

##删除节点

#父亲删孩子

node.removeChild(childNode);node为childNode的父节点。

#自己删自己（非标准）

（1）sz.parentNode.removeChild("sz");

（2）var tj=document.getElementById("tj");

      tj.remove();

小案例：联动菜单代码：

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>联动菜单</title>

</head>

<body>

       省：

   <select id="province" οnchange="chg();">

        <option value="-1">请选择</option>

        <option value="0">河北省</option>

        <option value="1">山东省</option>

        <option value="2">山西省</option>

   </select>

   市：

   <select id="city">

        <option>请选择</option>

   </select>

   <script>

      //模拟加载城市

      function loadCities() {

     return [

         ["石家庄","廊坊","保定"],

         ["济南","青岛","德州"],

         ["太原","大同","阳泉"]

             ];

      }

     var cities=loadCities();

     function chg() {

    //获取省份的下拉选

    var sel1=document.getElementById("province");

    //获取省份的序号

    var pindex=sel1.value;

    //获取该省份对应的城市

    var pcities=cities[pindex];

    //先删除城市下拉选内所有的城市

    var sel2=document.getElementById("city");

    //

    //sel2.innerHTML="<option>请选择</option>";

    var options=sel2.getElementsByTagName("option");

    for(var i=options.length-1;i;i--){

    sel2.removeChild(options[i]);

    }

    //再增加该省份的城市。

    if(pcities){

    for(var i=0;i<pcities.length;i++){

    var opn=document.createElement("option");

    opn.innerHTML=pcities[i];

    sel2.appendChild(opn);

    }

    }

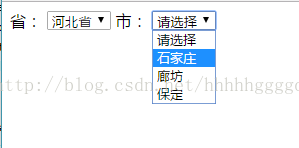
     }

   </script>

</body>

</html>

 网页显示效果如下：



五、自定义对象

自定义对象是一种特殊的数据类型，由属性和方法封装而成。

---属性指与对象有关的值:对象名.属性名。

---方法指对象可以执行的行为或者可以完成的功能：对象名.方法名（）

1.创建自定义对象

（1）采用直接量（JSON"JavaScript Object Notation"是一种轻量级的数据交换格式）的方式创建自定义对象

名称需要使用“”引起来，多对定义之间使用，隔开，名称可以是属性。

function f1() {

  var stu= {"name":"zs","age":25,"work":function(){alert("我学Java")}};

  alert(stu.name);

  alert(stu.age);

  //work的function方法无返回值，则不需要加alert。

  stu.work();

   }

（2）采用构造器创建对象

##采用自定义构造器创建对象（自定义了一个构造器Coder，然后每次写时都可以调用Coder构造器）

function f3() {

  var c=new Coder("lisi",26,function(){alert("我写Java")});

  alert(c.name);

  alert(c.age);

  c.work();

   }

   function Coder(name,age,work) {

  //this=创建出来的对象

  //this.name给该对象声明一个属性

  this.name=name;

  this.age=age;

  this.work=work;

   }