JavaScript

编程语言：

机器语言、汇编语言、高级语言

机器语言和汇编语言本质相同，都属于低级语言，汇编是机器语言加好记的英文标识符。高级语言在运行前需要翻译器翻译成二进制代码才能运行

高级语言分为编译性语言和解释性语言，编译性语言：C、C++、C#、Java；解释性语言：python、JS

标记性语言：HTML

编程语言和标记性语言的区别

编程语言有很强的逻辑和行为能力，并向计算机主动发出指令

标记语言不向计算机发出指令，常用语格式化和链接，被动被读取的

JS初识

js是一种运行在客户端的脚本语言

脚本语言：不需要编译，在运行过程中由JS解释器逐行进行解释并执行

现在也可以基于Node.js技术进行服务器端编程

JS的作用

1. 表单动态校验（密码强度检测）（JS产生最初的目的）
2. 网页特效
3. 服务端开发（node.js）
4. 桌面程序（Electron）
5. App（Cordova）
6. 控制硬件-物联网（Ruff）
7. 游戏开发（Cocosd-js）

浏览器执行JS简介

浏览器分为两部分：渲染引擎和JS引擎

渲染引擎：用来解析HTML和CSS，如chrome浏览器的blink，老版本的webkit

JS引擎：也称为JS解释器，用来读取网页中的JavaScript代码，对其处理后 运行，比如chrome浏览器的V8

浏览器本身不会执行JS代码，而是通过内置JS引擎来执行JS代码。JS引擎执行代码时逐行解释每一句源码，然后由计算机执行。

JS组成

ECMAScript：JS语法

DOM：页面文档对象模型

BOM：浏览器对象模型

ECMAScript：是由ECMA国际进行标准化的一门编程语言，这种语言在万维网上应用广泛，被称为JavaScript或JScript，实际上是ECMAScript语言的实现和扩展。ECMAScript规定了JS的编程语法和基础核心知识，是所有浏览器厂商共同遵守的一套JS语法工业标准

JavaScript由网景公司发布

JScript由微软公司发布

DOM：是W3C组织推荐的处理可扩展标记语言的标准编程接口。通过DOM提供的接口可以对页面上的各种元素进行操作（大小、位置、颜色等）

BOM：提供了独立于内容的、可以与浏览器窗口进行互动的对象结构。通过BOM可以操作浏览器窗口，比如弹出框、控制浏览器跳转、获取分辨率等

JS书写位置：

1. 行内：直接写到元素内部
2. 可以将单行或少量JS代码写在标签元素的事件属性中（以on开头的属性），如onclick
3. 注意单双引号的使用：HTML中推荐使用双引号，JS中推荐使用单引号
4. 行内编写可读性差，引号易错
5. 特殊情况下使用
6. 内嵌：在<head>标签内添加<script></script>标签，并在内部编写JS代码
7. 外部
8. 新建后缀为.js的文件
9. 在其中编写JS代码
10. 在html页面中引入：<script scr=’url’></script>
11. 引用外部JS文件的<script>标签内部不可以写代码

JS注释

单行注释：// 快捷键：ctrl+/

多行注释：/\* \*/ 快捷键：shift+alt+a

注释快捷键再按取消注释

JS输入输出语句

alter(‘xxx’)：

浏览器弹出警示框，归属浏览器

console.log(‘xxx’)：

浏览器控制台打印输出信息，归属浏览器，类似python的print

prompt(info)：

浏览器弹出输入框，用户可以输入，归属浏览器，返回值为字符串

变量

本质是程序在内存中申请的一块用来存放数据的空间

声明变量：

var xxx; //var关键字声明变量

赋值变量：

xxx=aaa; //与其他编程语言一样

变量初始化：声明并赋值

var xxx=aaa; //声明赋值放在一行内，简洁

多变量声明赋值：

var xxx=a, nnn=b, mmm=c;

注：

1.JS变量可直接赋值使用不用必须声明

2.区分大小写

3.不能以数字开头，不能包含空格

4.不能是关键字

5.类似python

6.尽量不用name作为变量名

7.特殊字符开头只能是$和\_

8.var a=b=c=10 此声明方法等效于var a=10；b=10；c=10，若在函数中a为局部变量，b、c为全局变量

数据类型

数据类型本质是为不同的数据分配大小合适的内存空间，这样可以有效节省内存空间。JS和python一样自动定义数据类型

简单数据类型

Number：数字型，包含整型和浮点型

Boolean：布尔型，True，False，等价于1,，0

String：字符串型，”xxxxxx”

Undefined：var a; 声明了变量没赋值，则a为Undefined

Null：var a=null; 声明变量a为空值

Number：

八进制表示：值前加0，如8 ：010

十六进制表示：值前加0x，如21 ：0x15

最大值：Number.MAX\_VALUE

最小值：Number.MIN\_VALUE

无穷大：Infinity，代表无穷大，大于任何数值

无穷小：-Infinity，代表无穷小，小于任何数值

NaN：not a number，代表任何一个非数值

isNaN(xxx)：用来判断一个变量是否为非数字类型，是非数字返回true，不是返回false

String：

使用单引号或双引号定义，JS推荐优先使用单引号

可用引号嵌套，外双内单或者外单内双都行

转义符：

都是\开头

\n：换行符

\\：斜杠\

\’：单引号

\”：双引号

\t：tab缩进

\b：空格

字符串长度

使用length属性可以获取整个字符串的长度，str.length

字符串拼接

用+号进行拼接，字符串和其他任何类型的值拼接都按字符串处理，操作类似python

Boolean：

true和false，在运算中true为1，false为0

Undefined和Null

Undefined和数字相加等于NaN

获取变量数据类型

typeof(xxx)，返回变量的数据类型，不用括号加空格也可

数据类型转换

1. 转换为string类型

toString()：转成字符串，xxx.toString()

String(xxx)：转成字符串

加号拼接字符串：和字符串拼接

1. 转换为Number型

parseInt(string)：将string型转换成整数型，取整，必须开头为数字，并且会自动去掉后面的字母

parseFloat(string)：将string型转换为浮点型，保留小数，必须开头为数字，并且会自动去掉后面的字母

Number(string)：将string型强制转换成数值型

隐式转换：利用算术运算符隐式转换为数值型

1. 转换为布尔型

Boolean(xxx)，空字符串，0，NaN，null，undefined会转换成false，其他的都会转换成true

标识符

开发人员为变量、属性、参数、函数起的名字

关键字

js所使用的字，有特殊含义了，不能作为标识符

保留字

未来可能会成为关键字的字，不能作为标识符

运算符

+、-、\*、/、%（取余）

浮点数存在精度问题不能直接判断相等与否

递增

++m、m++，类似c，递增1

前置递增++m，先递增再返回值

后置递增m++，先返回值在递增

递减

--m、m--，类似c，递减1

前置递减--m，先递减再返回值

后置递减m--，先返回值再递减

比较运算符

和python一样

==：等于，会给变量转换数据类型，如38==’38’true

===：全等，数值和数据类型必须都一样才行，如38===’38’false

逻辑运算符

&&：与

||：或

！：非，取反

赋值运算符

=：直接赋值

+=：先加一个数再赋值（类似python，该角度解释更确切）

-=：先减一个数再赋值

\*=、/=、%=：乘、除、取模后再赋值

流程控制

分三种结构：顺序结构、分支结构、循环结构

if语句

挨条判断，速度相对慢

if (条件表达式) {

执行语句2;

} else if {

执行语句2;

} else {

执行语句3;

}

三元表达式

条件表达式？表达式1：表达式2

若条件表达式为true，返回表达式1的值，若为false返回表达式2的值

switch语句

直接跳转符合的执行语句，分支多的时候速度比if快

switch (表达式) {

case 值1:

执行语句1;

break;

case 值2:

执行语句2;

break;

...

default:

执行最后的语句;

}

注：如果不写break将不判断直接执行下一个case

for循环

语法类似c++

for (i=1; i<10; i++) {

执行语句

}

数组array

arr=[1,2,3,4,5]，可以存放任意类型的数据

创建数组

1. 利用new关键字

var 数组名=new Array(x);

x为一个数时，表示创建的数组的长度

x为两个以上数时，表示添加进数组的值

1. 利用数组字面量创建数组

var arr=[ ];

访问数组

arr[索引]

数组长度

arr.length

数组扩容修改

arr=[1,2,3]

1. 修改数组索引

arr[3] // 相当于把数组扩容了一个，arr[3]=undefined

1. 修改索引并赋值

arr[3]=xxx // 直接扩容并赋值为xxx

1. 修改数组内数据

arr[0]=4 // 修改索引为0的数据为4

注：不要给数组名直接赋值，会把数组抹掉并重新赋值，如arr=5，数组arr会重置为一个普通变量并赋值为5

检测是否为数组

1. instanceof运算符，用来检测是否为数组

var arr=[ ]

console.log(arr instanceof A)

1. Array.isArray(arr) 返回true或false

添加数组元素

1. push() 在数组末尾添加一个或者多个数组元素

arr.push(1,’xxx’)

push()的返回值是新数组的长度

1. unshift() 在数组的开头添加一个或者多个数组元素

unshift()的返回值是新数组的长度

删除（弹出）数组元素

1. pop() 弹出数组最后一个元素

返回值即被弹出的元素，类似python

1. shift() 弹出数组第一个元素

返回值即被弹出的元素

函数

封装了一段可以被重复调用的代码块，为了让代码更好的复用

声明函数

常用动名短语作为函数名

1. 关键字声明

function 函数名(参数名) {

函数体

}

1. 匿名函数

var 变量名=function(形参) {

函数体

}

调用函数

1. 函数名(参数)
2. 匿名函数：变量名(实参)

参数传递

只有位置传参，多个参数之间用英文逗号分隔

1. 实参比形参多，无影响，多的实参不会被接收
2. 实参比形参少，缺少的形参会被定义为undefined

函数返回值

return 变量名

1. return只能返回一个值，若想返回多个值可以使用数组

return [1,2,3,4,5]

1. return后的代码不会被执行，可以退出循环，并且直接结束当前函数
2. 函数没有return返回undefined

arguments的使用

一个存储了所有传入的参数值的伪数组，具有length属性，没有pop、push方法

作用域

变量在某个范围内起作用和效果，目的是为了提高程序的可靠性，减少命名冲突

全局作用域

整个<script>标签内，或者单独的一个js文件

局部作用域

函数内部，只在函数内部起效果和作用

块级作用域

带花括号的{ }就是块级作用域

全局变量

1. 在全局作用域下声明的变量为全局变量
2. 在函数内不声明直接赋值的变量也是全局变量

全局变量只有在浏览器关闭的时候才会销毁，比较占内存资源

局部变量

函数内部声明的变量为局部变量

局部变量在程序执行完毕就会销毁，比较节约内存资源

案例：var a=b=c=10 此声明方法等效于var a=10；b=10；c=10，若在函数中a为局部变量，b、c为全局变量

作用域链

内部函数访问外部函数的变量，采取链式查找的方式来决定取哪个值，这种结构我们称为作用域链（就近原则）

预解析

js引擎运行js代码先预解析，再执行

js预解析会把所有var和function提升到当前作用域的最前面

代码执行会按照从上到下的顺序

预解析分为变量预解析（变量提升）和函数预解析（函数提升）

变量提升就是把素有的变量声明提升到当前的作用域最前面，不提升赋值操作

函数提升就是把函数声明提升到当前作用域的最前面，不调用函数

对象

对象是一组无序的相关属性和方法的集合，所有事物都是对象，例如字符串、数值、数组、函数等

属性：事物的特征，在对象中用属性来表示（常用名词）

方法：事物的行为，在对象中用方法来表示（常用动词）

创建对象（object）

1. 字面量创建对象

花括号（大括号）创建对象{ }

var obj={

uname: ‘张三丰’,

age: ‘18’,

sex: ‘男’,

sayHi: function() {

函数体

}

}

1. 里面的属性或者方法采取键值对形式：属性名: 属性值 方法名: 匿名函数
2. 多个属性或者方法中间用逗号隔开
3. 方法冒号后面跟的是一个匿名函数
4. 利用new Object创建对象

var obj=new Object();

obj.uname=’xxx’ //追加属性

obj.age=18 //追加属性

obj.sayHi=function() {

函数体;

} //追加方法

1. 构造函数创建对象（类）

构造函数就是把对象里面一些相同的属性和方法抽象出来封装到函数里，方便重复使用大量创建对象

构造函数语法

function 构造函数名(形参名) {

this.属性名=值;

this.方法名=function() {}

}

调用构造函数创建对象（对象的实例化）

new 构造函数名(实参);

使用对象

1. 调用对象的属性方法1，对象名.属性名
2. 调用对象的属性方法2，对象名[‘属性名’]
3. 调用对象的方法，对象名.方法名()

遍历对象

for (var k in obj) {

console.log(k); //输出属性名

console.log(obj[k]); //输出属性值

}

内置对象

js自带的一些对象，供开发者使用，提供了一些常用或是最基本而必要的功能Math、Date

查文档MDN

开发者网络（MDN）提供了有关开放网络技术（Open Web）的信息，包括HTML、CSS和万维网及HTML5应用的API

Date对象

var date=new Date()

不传入参数返回当前系统时间

格式化日期



时间戳（毫秒数）

获取从1970.1.1到现在的毫秒数，具有永远不会重复的特性

date.valueOf() //返回时间戳

date.getTime() //返回时间戳

var date1=+new Date() //返回时间戳

Date.now() //直接返回时间戳，H5新增

实现倒计时

使用时间戳来做，用未来的时间戳减去当前时间戳，并把结果转换成天、时、分、秒

时间戳转换工具

总秒数=（未来时间戳—当前时间戳）/1000

s=parseInt(总秒数%60) //计算当前秒数

m=parseInt(总秒数/60%60) //计算分钟

h=parseInt(总秒数/60/60%24) //计算小时

d=parseInt(总秒数/60/60/24) //计算天数

字符串对象

字符串具有不可变性，表面的变化本质是内存地址指向的变化，若频繁的更改字符串变量的内容和拼接字符串会占用大量的内存空间

字符的位置

str.indexOf(‘要查找的字符’, 起始的位置（选填）)

字符串的拼接和截取

拼接

str.concat(str1, str2, str3)方法，用于拼接多个字符串，等效于+

截取

str.substr(截取的起始位置, 截取几个字符)

数据类型

分为简单数据类型和复杂数据类型，简单数据类型又称基本数据类型或者值类型，复杂数据类型又叫做引用类型

值类型：在存储时变量中存储的是值本身

引用类型：在存储时变量中存储的仅仅是地址（引用），因此叫做引用类型通过new关键字创建的对象（系统对象、自定义对象），如Object、Array、Date

堆和栈

栈：由操作系统自动分配释放存放函数的参数值、局部变量的值等。其操作方式类似于数据结构中的栈，存放简单数据类型的数据

堆：一般由程序员分配释放，若程序员不释放，由垃圾回收机制回收，存放复杂数据类型的数据

简单类型内存分配

在栈内开辟一块内存空间，存储实质的数据，变量名指向该内存空间

复杂类型内存分配

在栈内开辟一块内存空间，存储一条地址，该地址指向堆内的一块内存空间，该空间存储实质的数据

简单类型传参

本质为在栈内复制一份数据给形参，因此函数内操作形参不会影响外部实参

复杂类型传参

本质为在栈内复制一份地址给形参，因地址都指向同一块堆内空间，所以函数内操作形参会影响到外部实参的值

Web APIs

页面交互功能，以JS为基础

API（application programming interface）应用程序编程接口，是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件得以访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节。API是给程序员提供的一种工具，以便能轻松的实现想要实现的功能

web API是浏览器提供的一套操作浏览器功能和页面元素的API（BOM和DOM）

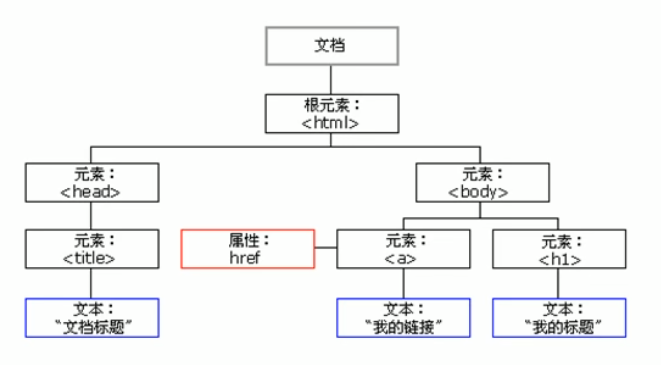
MDN内有详细的API文档

DOM

文档对象模型（Dcument Object Model），是W3C组织推荐的处理标记语言（HTML或XML）的标准编程接口

主要用来操作元素，可以改变网页的内容、结构和样式

DOM树



文档：一个页面就是一个文档，DOM中使用document表示

元素：页面中的所有标签都是元素，DOM中使用element表示

节点：网页中的所有内容都是节点（标签、属性、文本、注释等），DOM中使用node表示

DOM把以上内容都看做是对象

DOM操作流程

1. 获取元素

1>ID获取

2>标签名获取

3>html5新方法获取

4>特殊元素获取

1. 注册事件

1>传统注册

2>监听注册

1. 操作元素

操作函数

获取元素

因为文档页面从上往下加载，所以先得有标签，故<script>标签要写在最下面

1. 根据ID获取

1>使用getElementById()方法可以获取带有ID的元素对象

2>参数id是大小写敏感字符串

3>返回的是完整的被选标签对象

4>document.getElementById(‘xxxx’)

5>console.dir(对象)打印元素对象，更好的查看里面的属性和方法

1. 根据标签名获取
2. 使用getElementsByTagName()方法可以返回带有指定标签名的对象的 集合（数组）取用：elements[x].属性/操作=xxx
3. 参数是大小写敏感的字符串
4. 选取不同父元素下的标签元素：element.getElementsByTagName(‘xxx’)

element为单一指明父元素

1. 通过HTML5新增的方法获取
2. 通过类名获取元素

document.getElementsByClassName(‘类名’)

返回元素对象集合

1. 通过选择器获取元素对象（单个）

document.querySelector(‘选择器’)

返回选择器元素指向的第一个对象

1. 通过选择器获取元素对象（多个）

document.querySelectorAll(‘选择器’)

返回指定选择器的所有对象集合（数组）

1. 特殊元素获取
2. 获取body标签

document.body

返回body元素对象

1. 获取html标签

document.documentElement

返回html元素对象

事件基础

js有能力创建动态页面，事件是可以被js侦测到的行为

触发—响应机制

网页中的每个元素都可以产生某些可以触发js的事件，例如单机按钮产生一个事件，然后去执行某些操作

事件三要素

事件源、事件类型、事件处理程序

事件源：事件被触发的对象

事件类型：如何触发什么事件，例：鼠标点击、鼠标经过、键盘按下等

事件处理程序：通过一个函数赋值方式

btn.onclick=function() {

函数体

}

执行事件过程

1. 获取事件源
2. 绑定事件 注册事件
3. 添加事件处理程序

添加注册事件方案

1. 获取元素
2. element.事件，再编写操作

例：

var div=document.querySelector(‘div’);

div.onclick=function() {

div.placeholder=’xxx’;

div.属性=值

}

不添加注册事件方案

1. 获取元素
2. element.操作，直接编写操作

例：

var div=document.querySelector(‘div’);

div.innerText=’xxx’;

div.属性=值



操作元素

操作元素的属性以下都相当于元素对象的属性

1. innerText改变元素内容，不识别html标签，按字符串展示，非标准（IE）

element.innerText=xxx //xxx可以是固定的字符串，也可以是函数

1. innerHTML改变元素内容，识别html标签，标准方法（W3C）

element.innerHTML=xxx //xxx可以是固定的字符串，也可以是函数

1. 以上两个方法还可以用来获取元素的内容，同样innerText不识别标签；innerHTML不仅识别html标签，还保留空格和换行

var p=document.querySelector(‘p’)

console.log(p.innerText)

console.log(p.innerHTML)

1. 修改标签元素的属性

如src、title、href等

element.属性=新值

1. 修改表单属性

element.属性=新值

还可以

btn.onclick=function() {

this.属性=新值

}

其中this指向的是调用该函数的对象（btn）（常用）

1. 修改css样式属性

1>element.style.属性=’xxx’

通过该方法修改的css样式是行内样式，优先级最高

2>element.className=’类名’

修改当前元素的类名，以此来更改元素样式，用于修改样式较多的情况

element.className=’类名 类名’

可以应用多个类的样式

1. 焦点事件

onfocus获得焦点

onblur失去焦点

1. 全选按钮实现

核心思路：复选按钮跟着全选按钮的checked属性走

1. 获取元素属性值
2. element.属性 //获取内置属性值（元素本身自带的属性）
3. element.getAttribute(‘属性’) //主要获取自定义属性值
4. 设置属性值
5. element.属性=’值’
6. element.setAttribute(‘属性’, ‘值’)
7. 移除属性

element.removeAttribute(‘属性’)

1. H5新增自定义属性方法

自定义属性的目的是为了保存并使用数据，有些数据可以保存到页面中而不用保存到数据库中

H5中设置自定义属性以data-开头

例：<div data-index=”1”> </div>

1. H5新增获取属性方法

element.dataset.属性

element.dataset[‘属性’]

dataset是一个集合（数组），存放了所有以data-开头的属性的值

如果属性名中有多个横杠‘-’链接的单词，获取时采取驼峰命名法

例：<div data-list-name=”xxx”> </div>

div.dataset.listName

div.dataset[‘listName’]

注：getArrtribute()方法没有兼容性问题，dataset方法IE11以后才支持

节点操作

以上获取元素的方法比较繁琐且逻辑性不强，节点操作可以弥补这些缺陷

节点概述

网页中的所有内容都是节点（标签、属性、文本、注释等）所有html元素（节点）均可以修改、创建、删除

一般节点至少拥有nodeType（节点类型）、nodeName（节点名称）、nodeValue（节点值）这三个基本属性。

元素节点nodeType为1

属性节点nodeType为2

文本节点nodeType为3（文本节点包含文字、空格、换行等）

实际开发中，节点操作主要操作元素节点

节点层级

利用DOM树可以把节点划分为不同的层级关系，常见的是父子兄层级关系

父级节点

node.parentNode

取离元素最近的父级节点，若找不到就返回null

子级节点

node.childNodes

取节点下的所有子节点（包含文本、元素），返回一个集合（数组），标准方法，不常用

node.children[index]

是一个只读属性，返回所有的子元素节点（数组），它只返回子元素节点，其余节点不返回，非标准方法，但是各浏览器都支持，不存在兼容问题

node.firstChild

获取第一个子节点，不管是文本节点还是元素节点

node.lastChild

获取最后一个子节点，不管是文本节点还是元素节点

node.firstElementChild

获取第一个子元素节点，找不到返回null

node.lastElementChild

获取最后一个子元素节点，找不到返回null

注：这两个方法存在兼容问题，IE9以上才支持

实际开发时使用node.children[index]的方式来获取准确的子元素，以此解决兼容性问题

兄弟节点

node.nextSibling

返回下一个兄弟节点，包含元素节点、文本节点等全部节点，没有则返回null

node.previousSibling

返回上一个兄弟节点，包含元素节点、文本节点等全部节点，没有则返回null

node.nextElementSibling

返回下一个兄弟元素节点

node.previousElementSibling

返回上一个兄弟元素节点

注：这两个方法存在兼容性问题，IE9以上支持

利用两个获取兄节点的方法封装一个解决兼容性问题的函数

function getNextElementSibling(element) {

var el=element;

while (el=el.nextSibling) {

if (el.nodeType===1) {

return el;

}

}

return null;

}

创建节点

document.createElement(‘tagName’)

该方法创建由tagName指定的html元素。因为这些元素原先不存在，是根据我们的需求动态生成的，所以我们也称为动态创建元素节点

添加节点

node.appendChild(chid)

node是父级，child是子级，该插入方法类似于python里的append，在末尾插入

node.insertBefore(child, 指定元素)

node是父级，child是子级，该方法可以在指定元素的前面插入新元素

删除节点

node.removeChild(child)

node是父级，child是子级

js阻止a标签跳转

两种写法

<a href=”javascript:;”> </a>

<a href=”javascript:void(0);”> </a>

复制节点

node.cloneNode()

该方法返回调用该方法的节点的一个副本，也称为复制节点

1. 若括号内参数为空或false，则是浅复制，即只复制节点本身，不复制其子节点
2. 若括号内为true为深拷贝，复制标签和里面的子节点

三种动态创建元素区别

1. document.write()

会导致页面全部重绘

1. element.innerHTML()

该方法用拼接字符串的方式传参创建新元素，拼接过多会显著影响速度

但使用数组形式拼接，其速度是最快的，结构稍微复杂

1. document.createElement()

创建多个元素效率低一些，但结构清晰

注册事件（绑定事件）

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：传统方式和方法监听注册方式

传统方式

利用on开头的事件，如onclick，特点是注册事件的唯一性。

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

方法监听注册方式

w3c标准推荐方式

addEventListener()是一个方法

IE9之前的IE不支持此方法，可使用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

addEventListener事件监听方式

监听目标.addEventListener(type, listener, [useCapture])

将指定的监听器注册到监听目标上，当该目标对象触发指定事件时，就会执行事件处理函数

type：事件类型，如click、mouserover，不带on，格式为字符串’click’

listener：事件处理函数，事件发生时，会调用该监听函数

useCapture：可选参数，是一个布尔值，默认是false

attachEvent事件监听方式

监听目标.attachEvent(eventNameWithOn, callback)

该方法将制定的监听器注册到目标对象上，当该对象出发制定的事件时，指定的回调函数就会被执行

eventNameWithOn：时间类型字符串，比如onclick、onmouseover，这里要带on

callback：事件处理函数，当目标触发事件时回调函数被调用

注：IE9之前才支持

删除事件（解绑事件）

传统方式解绑

element.onclick=null

方法监听注册方式解绑

监听目标。removeEventListener(type, listener, [useCapture])

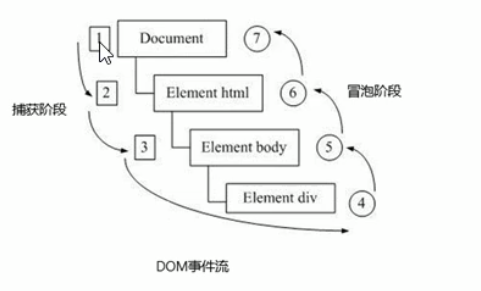
attachEvent方法注册方式解绑

监听目标.detachEvent(‘onclick’, 函数)

DOM事件流

事件流描述的是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，这个传播过程即DOM事件流



事件对象

element.onclick=function(event) {}

1. event就是一个事件对象，写到我们侦听函数的小括号里面，当形参来看
2. 事件对象只有有了事件才会存在，它是系统自动创建的，不需要手动传递参数
3. 事件对象是事件的乙烯类相关数据的集合，跟事件相关的比如鼠标点击里面包含了鼠标的相关信息
4. 事件对象可以自定义名字，如event、evt、e
5. 存在兼容性问题，IE678不支持此写法，必须使用window.event，兼容性写法：e=e||window.event

事件对象常见属性和方法



e.target和this的区别

e.target指向触发事件的对象

this指向绑定事件的对象

e.target存在兼容性问题，IE678不支持，兼容性写法：target=e.target||window.event.srcElement

事件委托

也称为事件代理，事件委派

事件委托原理

不是每个子节点单独设置事件监听器，而是事件监听器设置在其父节点上，然后利用冒泡原理影响设置每个子节点

即给父元素设置监听器，点击子元素其实是父元素的监听器触发，若需获取具体的被点击子元素，则可以使用e.target

优点：可以减少对子元素的操作，提升程序性能

常用鼠标事件

1. 禁止鼠标右键

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();

})

1. 禁止鼠标选中（selectstart开始选中）

document.addEventListener(‘selectstart’, function(e) {

e.preventDrefault()

})

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，现阶段主要使用鼠标事件对象MouseEvent和键盘事件对象KeyboardEvent



键盘事件



keyup和keydown不区分大小写，keypress区分大小写

keydown和keypress触发事件的时候，字符还未落入文本框中，取不到

keyup事件触发的时候，字符已经落入文本框中，能取到

e.keyCode返回用户所按键的ASCII码

e.code

e.key返回用户按的键

BOM

浏览器对象模型（Browser Object Model），提供独立于内容而与浏览器窗口进行交互的对象，核心对象时window

BOM由一系列相关的对象构成，并且每个对象都提供了很多方法和属性

DOM与BOM对比



BOM的构成

BOM比DOM更大，它包含DOM

window对象时浏览器的顶级对象，它具有双重角色

1. 它是JS访问浏览器窗口的一个接口
2. 它是一个全局对象。定义在全局作用域中的变量、函数都会成为window对象的属性、方法，在调用时可以省略window，前面学习的对话框都属于window对象方法，如alert()、prompt()等

注：window下有个特殊属性name，所以声明变量时最好避开name

页面加载事件（onload、load）

1. window.onload=function() { }
2. window.addEventListener(‘load’, function() { })

onload内的函数会在加载完页面所有内容后生效，函数写在onload中就不用考虑代码执行顺序的问题

注：window.onload传统注册事件方式只能写一次，如果有多个，会以最后一个window.onload为准。若使用addEventListener则没有限制

1. document.addEventListener(‘DOMContentLoaded’, function() { })

DOMContentLoaded事件触发时，仅当DOM加载完成，不包括样式表，图片，flash等。IE9以上支持。

如果页面的图片很多的话，从用户访问到onload触发可能需要较长的时间，交互效果就不能实现，必然影响用户的体验，此时用DOMContentLoaded事件比较合适

触发load事件的三种情况

1. a标签的超链接
2. F5或者刷新按钮（强制刷新）
3. 前进后退按钮

注：火狐浏览器的后退不会刷新页面，即不会触发load事件，应该用pageshow事件

调整窗口大小事件

window.onresize=function() { }

window.addEventListener(‘resize’, function() { })

window.onresize是调整窗口大小加载事件，当触发时就调用处理函数

注：1. 只要窗口大小发生像素变化就会触发该事件

2. 经常利用这个事件完成响应式布局。window.innerWidth返回当前屏幕的宽度

两种定时器

1. window.setTimeout(调用函数, [延迟的毫秒数])

1>window在调用时可以省略

2>延时事件单位时毫秒，可以省略，默认是0

3>调用函数可以直接写函数，也可以写函数名，还可以写字符串 ’ 函数名( ) ’（不提倡）

setTimeout(fn, 2000) setTimeout(‘fn()’, 2000) setTimeout(function() { }, 2000)

4>页面中可能有很多定时器，经常给定时器加标识符（名字）

5>调用函数也称回调函数callback，监听器和注册事件中的函数也可以称为回调函数

停止setTimeout()定时器

window.clearTimeout(timeoutID)

window可以省略

timeoutID是定时器的标识符

1. setInterval(回调函数, [间隔毫秒数])

该方法重复调用一个函数，每隔设定时间就去调用一次回调函数

1>window在调用时可以省略

2>延时事件单位时毫秒，可以省略，默认是0

3>调用函数可以直接写函数，也可以写函数名，还可以写字符串 ’ 函数名( ) ’（不 提倡）

4>页面中可能有很多定时器，经常给定时器加标识符（名字）

停止setInterval()定时器

window.clearInterval(intervalID)

window可以省略

intervalID是定时器的标识符

this指向问题

1. 全局作用域或者普通函数中this指向全局对象window，定时器的this指向window
2. 方法中谁调用指向谁
3. 构造函数中this指向构造函数实例

JS执行机制

js是单线程

同一时间只能做一件事，因为js是为处理页面中的用户交互以及操作DOM诞生的，所以只能一步步按顺序执行，先添加再删除。

单线程就意味着所有任务需要排队，前一个任务结束才会执行后一个任务。这样所导致的问题是：如果js执行的时间过长，会造成页面渲染不连贯。

为了解决该问题，利用多核CPU的计算能力，HTML5提出Web Worker标准，允许JS脚本创建多个线程。于是JS出现了同步和异步

同步任务

同步任务都在主线程上执行，形成一个执行栈

异步任务

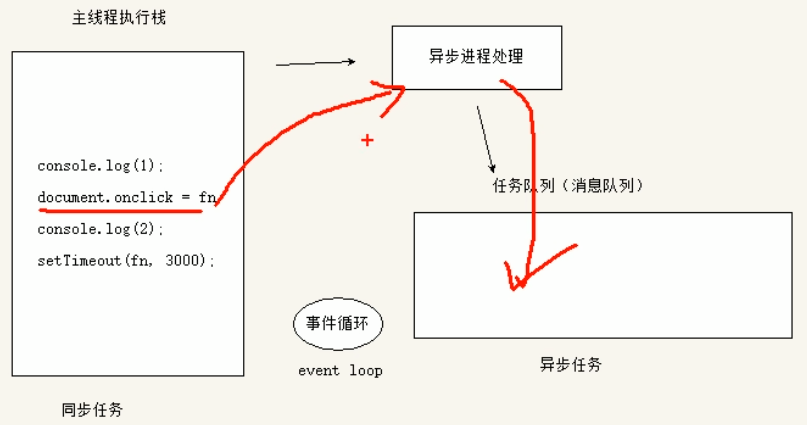
js的异步任务是通过回调函数实现的

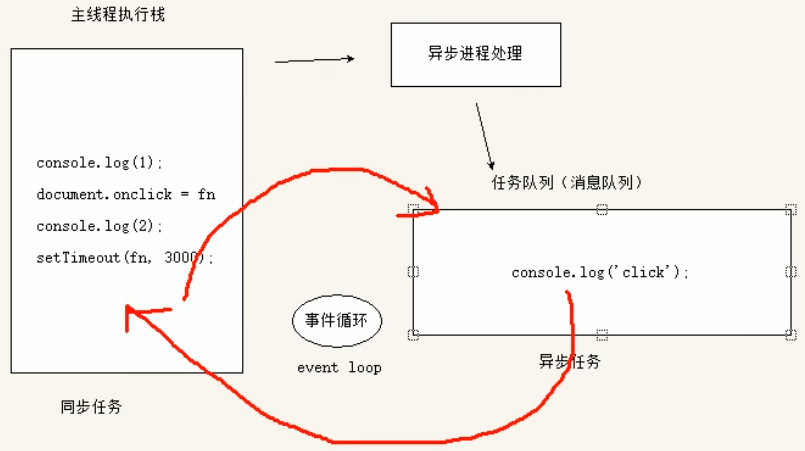
一般，异步任务有三种类型

1. 普通事件，如click、resize等
2. 资源加载，如load、error等
3. 定时器，包括setInterval、setTimeout等

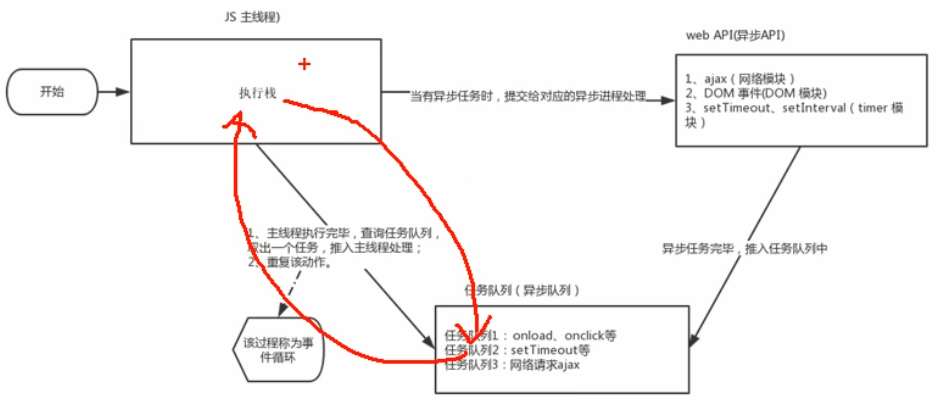
执行机制

1. 先执行执行栈中的同步任务
2. 异步任务（回调函数）放入任务队列中
3. 一旦执行栈中的所有同步任务执行完毕，系统就会按次序读取任务队列中的异步任务，于是被读取的异步任务结束等待状态，进入执行栈，开始执行





由于主线程不断的重复获取任务、执行任务、再获取任务、再执行，所以这种机制被称为事件循环（event loop）



location对象

window对象提供了一个location属性，用于获取或设置窗体的URL，并且可以用于解析URL。这个属性返回的是一个对象，也称为location对象。

URL（统一资源定位符）是互联网上标准资源的地址。互联网上的每个文件都有一个唯一的URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它



location对象属性



location.search获取查询字符串内参数的流程

1. location.search返回的是完整的查询字符串：?xxx=xx 使用时搭配substr(‘起始位置’, 截取几个字符) 截取有用部分
2. location.search.substr(1) 该语句截掉了查询字符串的? 返回xxx=xx 再利用split(‘=’)分割键和值
3. location.search.substr(1).split(‘=’) 该语句返回包含键和值的数组：[‘xxx’, ‘xx’] arr[1] 即需要的值

location对象方法



location.assign(‘url’) 记录浏览历史，可以实现后退功能

location.replace(‘url’) 不记录浏览历史

location.reload() 相当于刷新，填入true参数时相当于强制刷新（ctrl+F5）

navigator对象

navigator.userAgent 返回终端的版本和浏览器版本，用来判断PC端还是移 动端

history对象

window对象提供了一个history，与浏览器历史记录进行交互。该对象包含用户（在浏览器窗口）访问过的URL

相当于浏览器中的“前进”、“后退”按钮



history一般情况用的不多；OA系统中会用到。

PC端网页特效

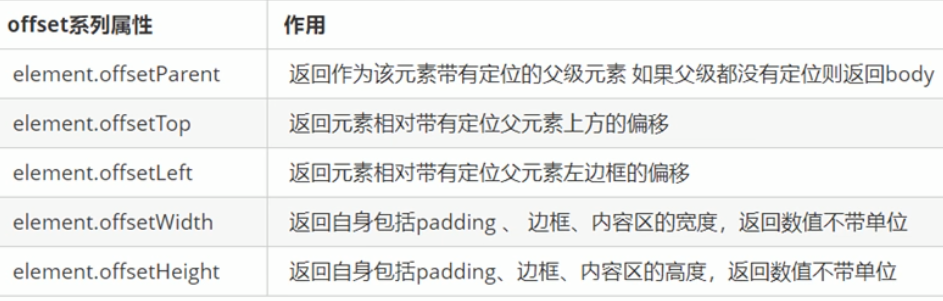
offset概述

offset翻译为偏移量，使用offset相关属性可以动态的得到元素的位置（偏移）、大小等

获得元素自身的大小（宽度高度）

注：返回的数值不带单位（px）

offset系列常见属性



offsetParent

offsetTop

offsetLeft

获得元素的偏移（位置），返回不带单位（px）的数值，以带有定位的父亲为准，如果元素没有父亲或父亲没有定位则以body为准

offsetWidth

offsetHeight

获得元素的大小：宽度、高度，是包含padding+border+width的宽和高

offsetParent

返回带有定位的父亲，若父级都没有定位返回body

offset和style的区别



元素可视区client系列

client翻译过来就是客户端，使用client系列的相关属性来获取元素可视区的相关信息。通过client系列的相关属性可以动态的得到该元素的边框大小、元素大小等



立即执行函数

不需要调用，立马能够自己执行的函数

语法：

(function() { }) () 或 (function() { } ())

后的括号相当于调用的意思

立即执行函数最大的的作用是独立创建了一个作用域，里面的var 变量是局部变量，不会有命名冲突的情况

元素滚动sroll系列

使用scroll系列的相关属性可以动态的得到该元素的大小、滚动距离等（内容的大小）



scroll属性所指的具体地方



滚动事件onscroll、scroll

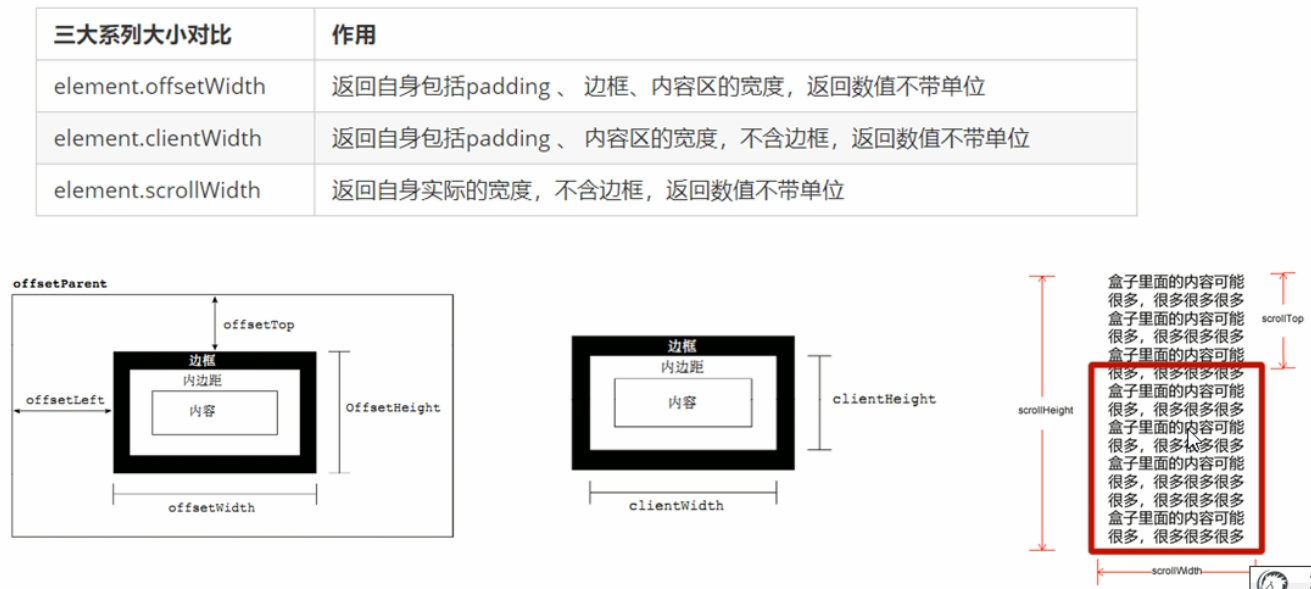
当滚动条活动时触发的事件

页面被卷去的头部可以通过window.pageYOffset获得，被卷去左侧用window.pageXOffset

存在兼容问题，IE9以上支持

注：元素被卷去头部用element.scrollTop，页面被卷去头部用window.pageYOffset

三大系列对比



mouseenter和mouseover的区别

mouseenter、mouseover鼠标事件

当鼠标移动到元素上时就会触发mouseenter、mouseover事件

mouseover鼠标经过自身盒子会触发，经过子盒子还会触发。

mouseenter只会经过自身盒子触发，因为mouseenter不会冒泡

mouseleave鼠标离开触发，同样不会冒泡。

动画函数

核心原理：通过定时器setInterval()不断移动盒子位置

实现步骤：

1. 获得盒子当前位置
2. 让盒子在当前位置上加上1个移动距离
3. 用定时器不断重复这个操作
4. 加一个结束定时器的条件
5. 注意此元素需要添加定位，才能使用element.style.left

动画函数的封装

参数：动画对象、结束位置

function animate(obj, target) {

clearInterval(obj.timer)

obj.timer=setInterval(function() {

if (obj.offsetLeft >= target) {

clearInterval(obj.timer);

}

obj.style.left=obj.offsetLeft + 1 + ’px’

}, 30);

}

·obj.timer使得定时器私有化，成为obj的属性，不用var开辟新空间存放定时器，性能优化

·开头的clearInterval用来解决多次点击造成的定时器多开元素移动加速问题

缓动效果原理

缓动动画就是让元素运动速度有所变化，最常见的是让速度慢慢停下来

思路：

1. 让盒子每次移动的距离慢慢变小，速度就会慢慢落下来
2. 核心算法：（目标值-现在的位置）/10 作为每次移动的距离步长
3. 停止条件：让当前盒子位置等于目标位置就停止定时器

缓动函数封装

动画函数在多个目标间移动

function animate(obj, target) {

clearInterval(obj.timer);

obj.timer=setInterval(function() {

var step=(target - obj.offsetLeft) / 10;

step = step > 0 ? Math.ceil(step) : Math.floor(step);

if (obj.offsetLeft == target) {

clearInterval(obj.timer);

}

obj.style.left=obj.offsetLeft + step + ’px’;

}, 30);

}

步长值避免出现小数，正数时使用Math.ceil()方法向上取整（取大数），负数时使用Math.floor()方法向下取整（取小数）

动画函数添加回调函数（callback）

回调函数原理：函数可以作为一个参数，将这个函数作为参数传到另一个函数里面，当那个函数执行完毕之后，再执行传进去的这个函数，这个过程叫回调

function animate(obj, target) {

clearInterval(obj.timer);

obj.timer=setInterval(function() {

var step=(target - obj.offsetLeft) / 10;

step = step > 0 ? Math.ceil(step) : Math.floor(step);

if (obj.offsetLeft == target) {

clearInterval(obj.timer);

if (callback) {

callback();

}

}

obj.style.left=obj.offsetLeft + step + ’px’;

}, 30);

}

动画函数封装到单独的JS文件内

新建JS文件，将代码粘贴到文件内，使用时引用文件

引用js文件：<script src=”xxx.js”> </script>