#s是跨平台的脚本语言

脚本语言：不能独立运行，依赖网页

HTML依赖浏览器

JS 依赖HTML

jjJS组成： 1 ECMAScript 3 4 5 6 7

2 DOM 文档

3 BOM 浏览器

所有事在<script>中完成

属性：1 type=’text/javascript’ 文本格式

2 src=’dome.js’

可以有多个<script> 自上而下

一个script只能专注做一件事

#页面输出内容：

1 document.write() &lt :< &gt:> 标签解析<1> 转义字符<2>

2 alert()

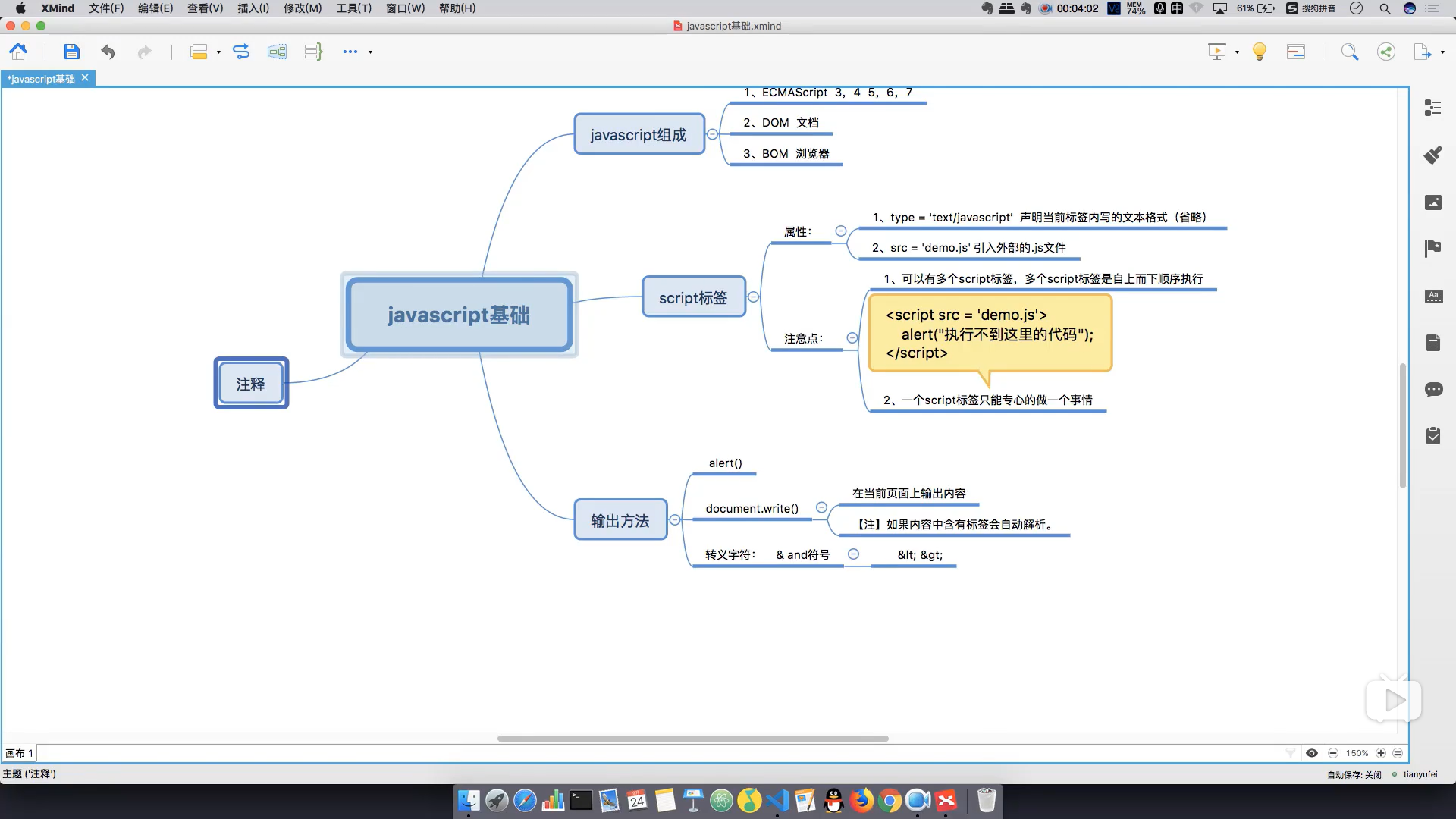
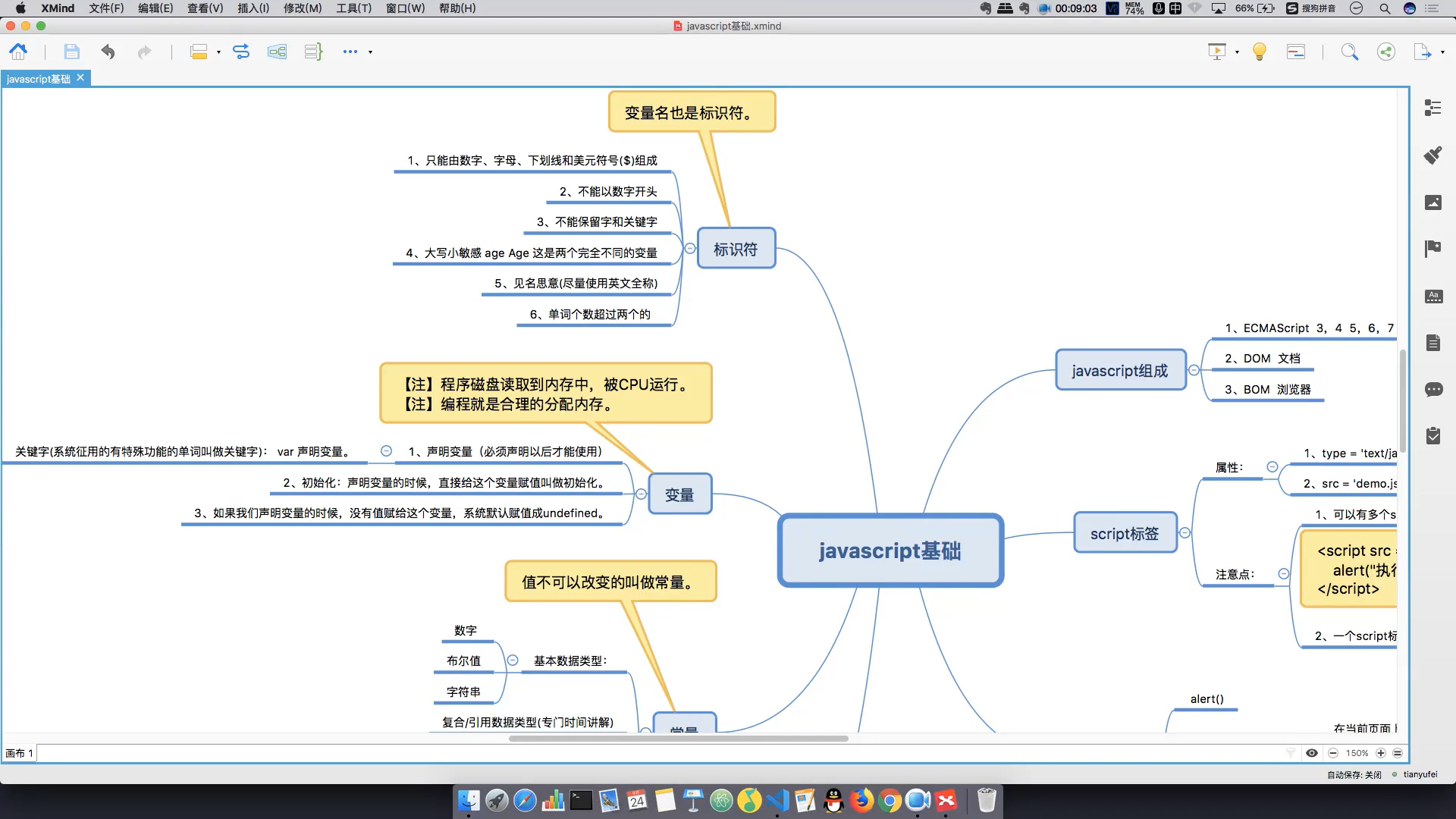
3 console.log() 调试面板控制台



#引用数据类型 尽量不要改变变量 混淆

Typeof() 变量，数据 输出数据类型

Alert(typeof typeof 100)

1. Number
2. String  
   
3. 

Shift()

数组.shift（） 没参数 从头部取下一个元素 返回：那个元素

Unshift() 数组.unshift(1,2,3..) 从头部插入 返回：数组长度

Concat（）

拷贝原数组

合并数组· 数组。Concat（1，2，3） 返回：合并生成的数组，原数组不变

Slice（）

数组。Slice（start，end）；【start，end ）

功能：基于当前数组获取指定区域元素[ ）,生成新的数组

返回：生成的新数组

Splice 截取

数组。Splice（start，length，1，2.。。。）；

功能：start 开始截取的位置 length：截取元素的长度

Join（）

数组。Join（字符串）

用了严格模式，有什么变量：

1 全局变量声明，加var

2 函数内重名属性

3 argument对象不允许被动态改变

indexOf()

数组.indexoFtem,start) item数据，start 下标 返回-1 没找到

forEach

arr.forEach(fuction(item.idex,arr)) 数组内容，索引，本身

#map映射

arr.map(function(tiem,idex,arr))

newarr= arr.map(function(tiem,idex,arr){

return item\*1.3;

})

#filter 过滤

var newArr=arr.filter(function(item,index,arr){

Return item>30;

});

#some every 符合条件，返回true，后面循环停止

var res=arr.some(function(item,index,arr){

alert(item);

return item>30;

});

# reduce 归并

Var arr={10,20,30,40,50}

Var res=arr.reduce(function(prev,next,index,arr){ //prev 第一次下标为0 第二次 上次的return

Alert(prev+next)

Return prev+next

});

#字符串概念 （带‘’ “”）

字符串声明：1 new 2 字符串常量赋值 3 省略new声明

var str=new string(100);

var str=’10086’;

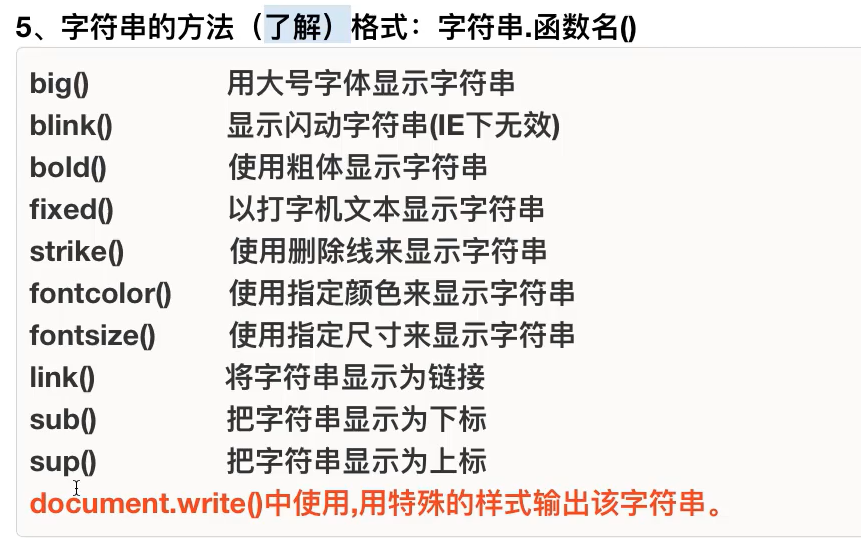
var str=string(100);

字符串是只读的 无法修改

charAt(3)

charCodeAt3() // 字符串调用

fromCharCode(3) 编码转字符//string静态方法 String调用



# 正则表达式：修饰符： i 忽略大小写 g全局匹配

Search() 返回字符串在原始字符串首次出现的位置。忽略大小写

Search(find\_text,within\_text,start\_num)

A=’你是什么大学’ search(“是”,a)

supStr=’ABCDefg’

subStr=/abc/i;

var index=supStr.search(subStr);

# substring(start,end) substr(start,length)

提取字符串，生成新数组

Slice(start,end):对数组进行部分截取，返回新数组

# replace()

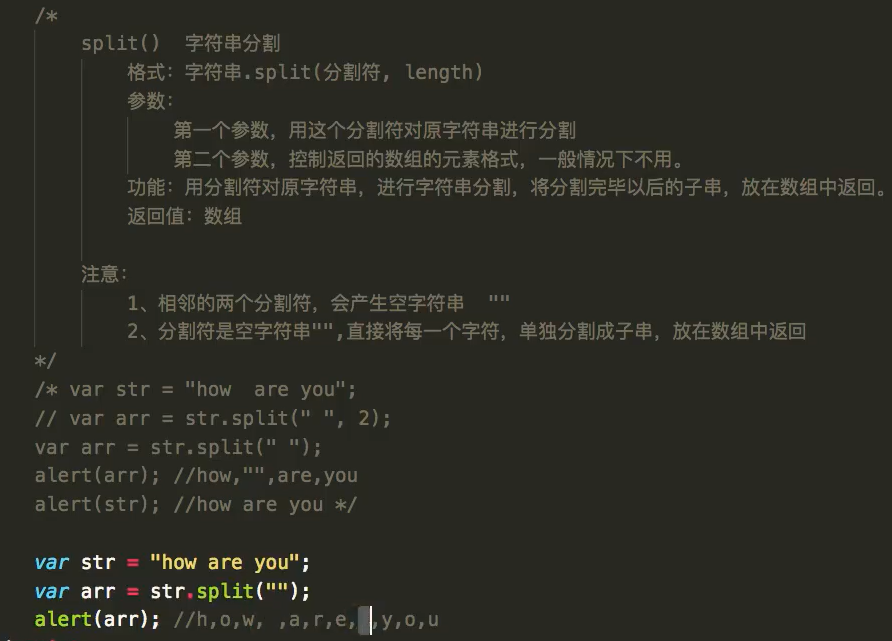
supStr.replace(oldStr,newStr)

第一个参数：正则表达式 /\*\*\*/ig

#字符串切割 split()

Str.toLowercase() 转小写

Str.toUppercase() 转大写



该文档说，从结构上看，所有的数据(data)最终可以分解成三种类型：

第一种类型是标量scalar，也就是一个单独的字符串string或数字numbers，比如“成都”这个单独的词。

第二种类型是序列sequence，也就是若干个相关的数据按照一定顺序并列在一起，又叫做数组array，或者列表list，比如“成都，重庆”。

第三种类型是映射mapping，也就是一个名/值name/value，即数据有一个名称，还有一个与之相对应的值，这又称作散列hash或字典dictionary，比如“蓉城：成都”。

json  
1.并列的数据之间用逗号(,)分隔

2.映射用冒号(:)表示

3.并列数据的集合(数组)用方括号([])表示

4.映射的集合(对象)用大括号({})表示

验证码

<script>

// 验证码

function testCode(n){

var arr=[];

for(i=0;i<n;i++)

{

var num=parseInt(Math.random()\*10);

arr.push(num);

}

return arr.join('');

}

testCode(6);

</script>

面对对象：生活映射到程序中

面对过程：数学逻辑

1 分析哪些实体

2 实体有哪些属性功能

3 实体间作用

# 对象

声明对象

1 new 2 省略new 3 对象常量赋值（大括号，代表对象）

对象属性（婚前） 普通变量（婚后）

Obj3={

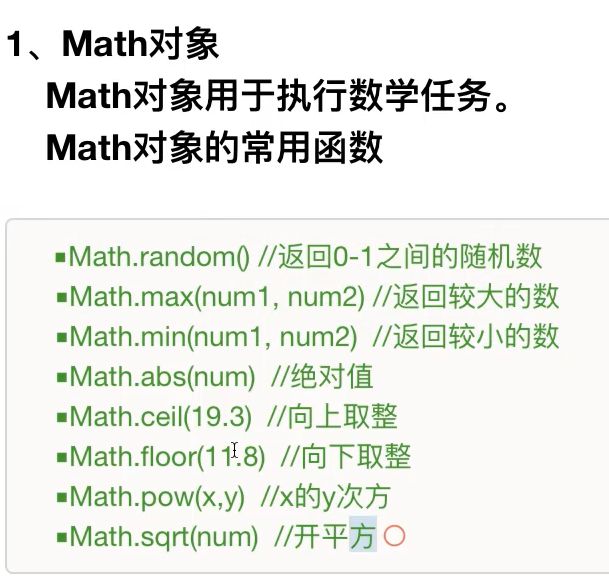
Username:‘钢铁侠’

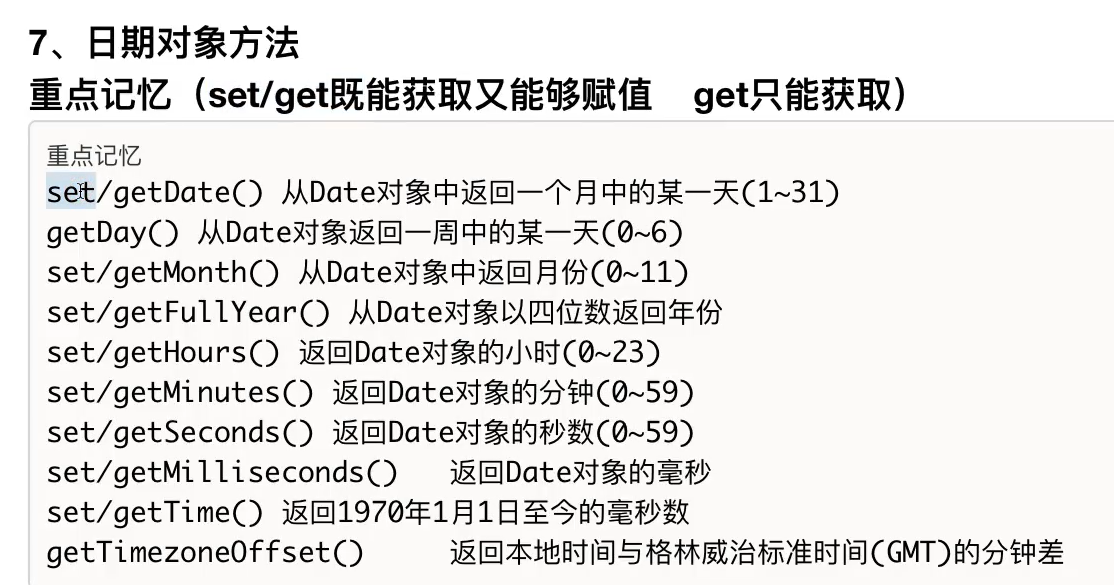
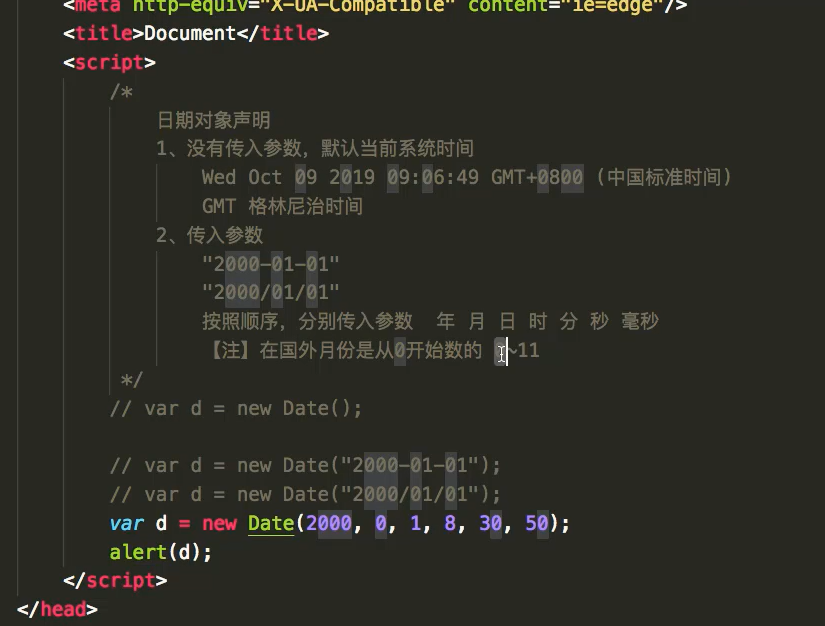
Age:’13’

Show:function(){

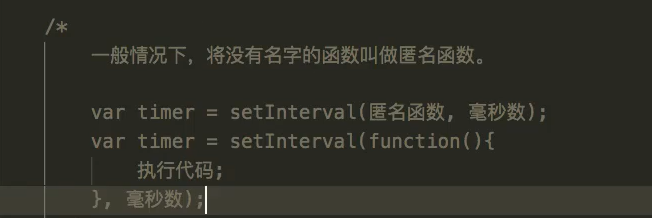
}

}













什么是执行上下文顺序？

就是当前的JavaScript代码被解析和执行时所在环境的抽象概念 js所有代码都在上下文中执行

执行上下文有3个类型

全局执行上下文：只有一个，这是默认的,基础的执行上下文。不在任何函数的代码就是全局执行上下文 两作用：创建全局变量，指向windows下的变量 另一个是讲this指向全局

函数执行上下文：有无数个，每个函数都有自己的执行上下文，每个函数只有被调用时，才会被创建，每次调用时，都会为该函数创建一个新的执行上下文

Eval函数执行上下文：运行在eval函数中的代码，

执行上下文的生命周期

1, 创建阶段

当函数被调用，但没有执行任何其内部代码前，会选确定这3个步骤：

1. 创建变量对象：首先初始化函数的参数 arguments，提升函数声明和变量声明
2. 创建作用域链:在执行期上下文的创建阶段，作用域是在变量对象之后创建的，作用域链本身包含变量对象。作用域用于解析变量。当被要求解析变量时，javascript始终从代码嵌套的最内层开始，如果最内层没有找到变量，就会跳转到上一层父作用域中查找，直到找到该变量
3. 确定this指向，有多种情况

2 执行阶段

执行变量赋值，代码执行

3 回收阶段

执行上下文出栈等待虚拟机回收执行上下文

三. 变量提升和this的指向

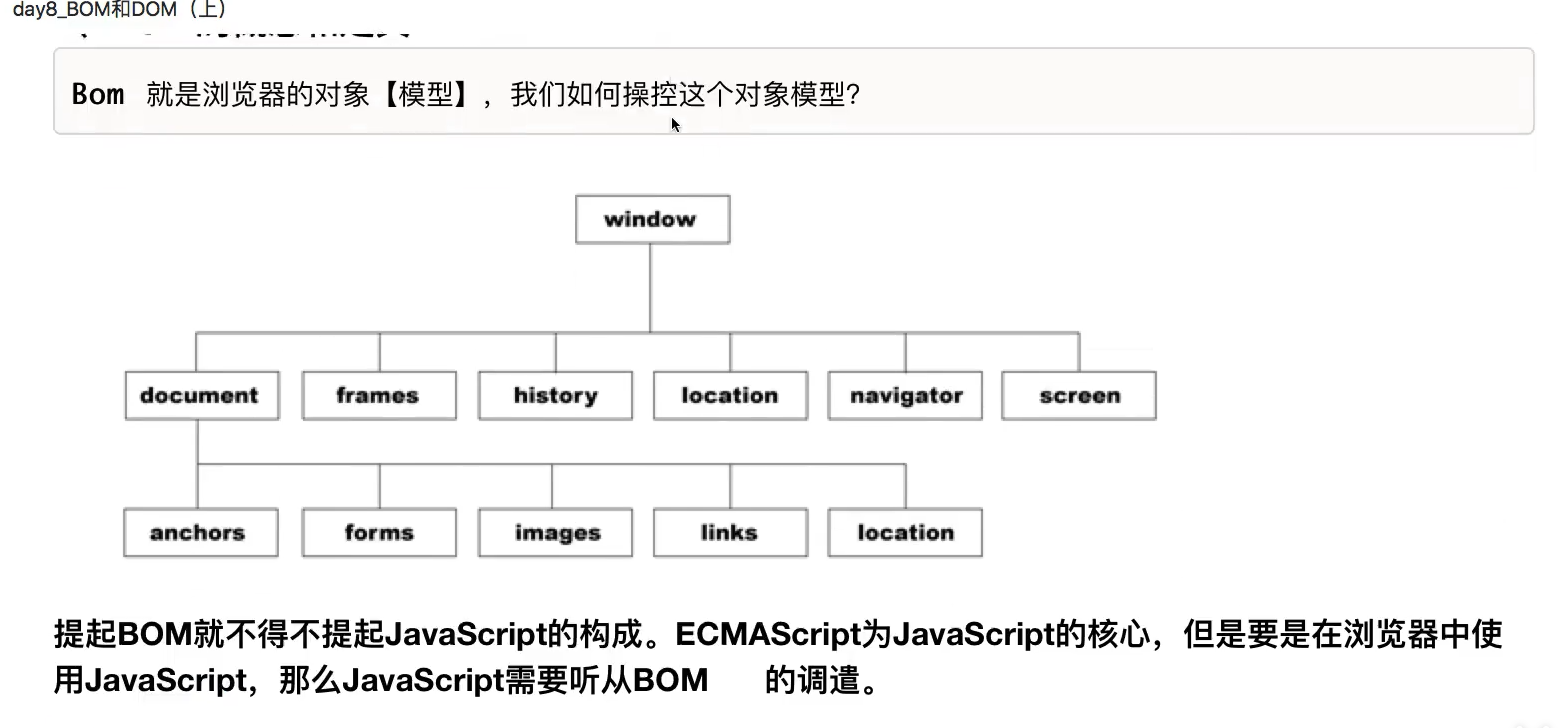
1 变量声明提升：

大部分编程语言都是先声明变量在使用，但在js中，事情有些不一样：

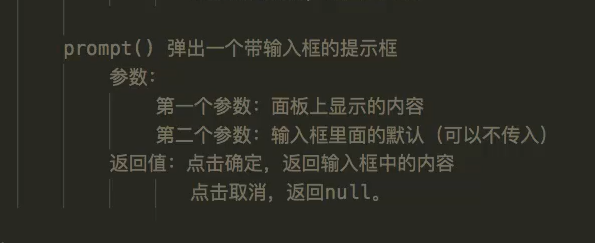
Console.log(a);//undefined

Var a=10;

上述代码正常输出undefined而不是报错uncaught referece









<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <!-- history对象   windows.history 掌管的时，当前窗口（不是浏览器） 历史记录（只要加载url不一样，就会产生历史记录）

    属性

     history.length

     方法

     history.back()   返回上一条记录

     history.forward() 前进到下一条的历史记录

     history.go()

         参数    0   刷新当前页面

                正整数  前进

                负      后退n条记录 -->

        <button onclick="alert(history.length)">获取历史记录的条数  </button>

        <button onclick="history.back();">back</button>

        <button onclick="history.forward();"> forward</button>

        <button onclick="histroy.go(2)">go</button>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

     <!-- 协议：//IP（域名）/：端口好/路径/？查询字符串#锚点

     location.protocol  file  本地磁盘文件访问

                        http

                        https  证书认证协议

     location.hostname   主机名  IP    全球范围

                                域名  IP别称

     location.port      端口号

        0-65535   每个app随机分配

      hostname.port    使用协议

                    浏览器  8080

                    http   80

                    https 443

      location.pathname    路径

      location.search    查询字符串  （前后端交互）

            ？name1=value1&哪么=value2

        location.hash -->

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <!-- js 获取顶层doucument节点获取：

    （1）

            document.getElementByld(id)

            通过节点的ID，可以准确获得需要的元素，是比较简单快捷的方法。

            如果页面上含有多个相同的ID的节点 ，只返回第一个节点

            document.getElementsByTagName("标签的名字")

                    .getElementsByName("name属性的值")

                    .getElementByclassName("类样式的名字")

                    .querySelector("选择器") -->

            获取节点.parentNode--> 获取到节点的父节点

            获取节点.children-->   获取到节点的子节点集合

            获取节点.childNodes--> 获取到节点的子节点集合（带有前后两个空白的文本节点）

            创建插入节点

            1 document.createElement("div")  创建一个元素节点

            2 document.createTextNode("文本文本")   创建一个文本节点

            被插入的节点.appendChild(创建的节点，被插入的节点)   在已知父节点的某个孩子前面添加内容改变文本内容

            选中的元素.innerText=''      直接将HTML代码当作字符来处理

            选中的元素.innerHTML=''      可以识别HTML代码

            删除：直接设置为空("")

            替换节点：

            父节点.replaceChild(新节点，老节点)

            克隆（复制节点）

            选中的元素.cloneNode(ture/false)

            当clone参数为true的时候：选中元素里面所有懂得内容克隆

                         false  ：元素本身克隆

            删除节点.removeChild(子节点)

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

<meta name="viewport" content="width=device-width, i

nitial-scale=1.0">

    <title>Document</title>

</head>

<body>

       // alert(windows.location==weindowd.Document.location)

       <!-- location   地址栏

       属性

       方法：

            location.assign(url)

            在当前窗口跳转带这个url

            location.replace(url)

            在当前窗口替换新的url，不会产生历史记录

            Location.reload()

            刷新窗前窗口

            location。reload（true）

            不经过浏览器缓存强制从服务器重载 -->

</body>

</html>