Mosaic，是互联网上第一个普遍使用的能显示图片的浏览器

浏览器组成

1. shell部分
2. 内核部分

渲染引擎（语法规则和渲染）

Js引擎

其他模块

翻译有两种形式：

1. 编译

编译集成为一个文件 .obj .exe .class等

优点：速度快，适合底层开发，操作系统，桌面应用

缺点：可移植性非常差

1. 解释

JavaScript php jsp等

优点：跨平台

缺点：速度慢

JavaScript语言：

1. 解释性语言，浏览器端的脚本语言
2. 单线程
3. ECMA标注，为了统一规格JavaScript兼容与ECMA标准，因此也称为ECMASCRIPT

JS执行队列：轮流执行（还是单线程运行）

JS三大部分：ECMAScript，DOM，BOM

ECMAScript：基础语法部分，能干的非常有限,但非常难，非常重要

DOM：Document Object Model

Bom: Browser Object Model

如何引入JavaScript：

1. 页面内嵌<script></script>
2. 外部引入<script src=”location”></script>

运行到script标签时会停止加载html和css，专注加载script标签直到加载完毕并执行完毕

script标签可以写在head里也可以写在body里的最后一行，但绝对不能写在head标签与body标签中间，如果想用body里的标签控制script标签就得把script标签写在body里的最后一行

为符合web标准（w3c中的一项）结构，样式，行为相分离，通常采用外部引入

JS基本语法

变量

变量声明：

声明，赋值分解

单一var声明法：var （只有var）

命名规则

1.变量名必须以英文字母，‘\_’，$开头

2.变量名可以包括英文字母，‘\_’，$，数字

3.不可以用系统关键字，保留字做关键名

关键字：break，else，new，var，case，finally，return，

void，catch，for，switch，while，default，if，throw

，delete，in，try，do，instanceof，typeof

保留字：abstract，enum，int，short，boolean，export，

Interface，static，byte，extends，long，super，

char，final，native，synchronize，class，float，

Package，throws，const，goto，private，transient，

Debugger，implements，protected，volatile，double

，import，pubic

值类型

不可改变的原始值（栈数据）

Number（只有浮点型），String，Boolean，Undefined，null

一但被赋值无法改变，想改只能换地址，比如用字符串

截断再输出也无法改变输出的值

引用值（堆数据）

数组，对象，函数

语句后面要用“；”结束

Js语法错误会导致后续代码终止，但不会影响其他代码块

（有多个script标签）

“=，+，-”两边都应该有空格

Js运算符

“+”：1：数学运算，字符串连接

2：任何数据类型加字符串都等于字符串

“-”，“\*”，“/”，“%”，“=”，“（）”

优先级“=”最弱，“（）”优先级最高

“++”，“--”，“+=”，“-=”，“/=”，“\*=”，“%=”

比较运算符：“<”,“>”,“==”,“!=”“<=”,“>=”

比较结果为Boolean值（只有两个值，ture和false）

逻辑运算符：“&&”，“||”，“！”

运算结果为真实的值

被认定为false的值

Undefined，null，NaN，0，false，“”（空串）

NaN（not a number）-->但它是number类型的，NaN

不等于任何东西包括它自己

&&逻辑与：全对才为对，先转化为Boolean值，再判

断Boolean值如果第一个是false那么终止语句并返回第

一个表达式结果，如果为ture那么看第二个语句，同理

||逻辑或：全错才为错，先转化为Boolean值，再判

断Boolean值如果第一个是ture那么终止语句并返回第

一个表达式结果，如果为false那么看第二个语句，同理

！逻辑非

/是告诉编译器后面的字符是要的，而不是结束符什么的

If /else if/else

for( ; ; )，每个空里不是必须有但必须写分号

While（）成立的情况下一直循环，do{} while（）不管

while成不成立先do一下，开发一般不用。。

Switch（条件）{case xx：；break； default：；判断 }，几个

case后相同的语句可以只写一条，放在最后，再写一个

Break。

没有循环时不能用break，否则会报错

Continue继续执行下一条语句

数组，堆数据，数组2里可以存不同种类的数据，数组

第一位从0开始arr[0]为第一位

属性：var 。。={name:,用逗号分隔，其他还是用;}

编程形式的区别：面向过程（为主） 面向对象

Typeof：六种数据类型：number，string，Boolean，

Undefined，objet，function

值undefined的属性是undefined，null和数组的属性

是object

类型转换：

Number（mix）把括号中的转化为number类型

Parseint（string，2-36）把string转化为数字，还有把

2-36是指把string当成几进制的数换成十进制输出，如

过有非数字如abc等那么就到那为止输出前面是数字的

Parsefloat（string）转换为浮点型，没有基底，如过有

非数字如abc等那么就到那为止输出前面是数字的

Var num = 123； var output = num.Tostring(2-36);

undefined和null不能tostring，以十进制为准转化为目

标进制

Var num = 123.456；num.toFixed(X)保留X位小数点

String（mix）把括号中的转化为string类型

Boolean（）

偏向使用‘’，容易识别

隐式类型转换：

IsNaN（）；++/--，+/-；+；\*/%；&&，||，！；<，>，<=

，>=；==，！=

都隐藏了先转化为数字类型的步骤

0/0为NaN，1/0为infinity正无穷

===绝对等于，只能完全相同才行，!==绝对不等于

要使用变量前必须声明，只有typeof（）不用

作用域定义：变量和函数生效（能被访问）的区域

JavaScript只有函数可以产生作用域

全局，局部变量

var demo=123（全局变量）

function test（）{demo=123；（局部变量）}

在函数外面无法访问里面的局部变量，里面的可以访问外面的

作用域的访问顺序

作用域不同即使变量名相同也不能访问

作用域精解:每个JavaScript都是一个对象，对象中有些属性我们可以访问，有些不可以访问，这些仅供JavaScript引擎存取，如

[[scope]]，[[scope]]指的就是我们所说的作用域，其中存储了运行期的上下文集合。

作用域链：[[scope]]中所储存的执行期上下文对象的集合，这个集合成链式连接，我们管这个链接叫作用域链

运行期上下文：当函数执行时会创建一个称为“执行上下文”的内部对象。一个执行上下文定义了一个函数执行时的环境，函数每次执行所对应的函数执行上下文都是独一无二的，所以多次调用一个函数会导致创建多个执行上下文，当函数执行完毕后会销毁执行上下文。

查找变量：从作用域链的顶端向下查找

当一个函数里面套个函数时，外面函数初始就有一个作用域链，1：它外面函数的上下文，2：整体的上下文，0：运行时产生自己的上下文

function a(){

function b(){

Var b = 234;

}

Var a = 123;

b();

}

var glob = 100;

a();

闭包：当内部函数被保存到外部时，将会生成闭包。闭包会导致原作用域链不释放(被保存出来的函数执行完也不会释放作用域链)，造成内存泄露（往内存中装东西导致内存减少）

闭包例子：

function a(){

function b(){

var b = 234;

}

document.write(aaa);

var aaa = 123;

return b;

}

var glob = 100;

var demo = a();

demo();

闭包的作用：

实现共有变量： eg：函数累加器

可以做缓存： eg：eater

可以实现封装，属性私有化： eg：person()；

立即执行函数：(function (){})();或(function (){}());(最好这么写)

第二种写法可以传参，var a,num = 123; (function (b){a=b+1}(num))

; =>a=124

不需要声明不需要定义直接执行，执行完即释放

作用：如变量定义需要很多步骤，就用立即执行函数去定义变量

函数的声明不可以直接运行，但函数表达式可以

function a() {}();不能执行 var a = function(){}();可以执行

闭包的防范：用立即执行函数解决

Js三部曲：1.语法分析 2.预编译 3.解释执行

1. 语法分析：是通篇找一些低级语法错误

逻辑错误不影响上面的代码运行，语法错误全篇不能运行

2.预编译：通篇预编译 变量声明提升，函数声明整体提升

预编译前奏：（1）imply global暗示全局变量：即任何变量，如果变量未经声明就赋值，此变量为全局所有

eg: a = 123; eg: var a = b = 123;(b=123不算声明)

（2）一切声明的全局变量都是Windows属性

eg：var a = 123; ===> windows.a = 123;

预编译四部曲：（1）创建AO对象 （2）找形参和变量，将变量形参作为AO属性名，值为undefined （3）将实参值和形参统一 （4）在函数体里面找函数声明，值为函数体

1. 解释执行：解释一句执行一句

对象

1. 属性的增，删，改，查

直接在属性外面操作

var hgf = { name : ’’, age : , wife : {name : ‘’, age : }, eat :

function(){}}

增：hgf.anotherlover = ‘...’;

改：hgf.age = 20;

查：document.write(hgf.name);

删：delete hgf.age;

1. 对象的创建方法

字面量

构造函数：命名首字母大写

系统自带：new（Object();Array();Number();Boolean();Date();）

当以new操作符调用构造函数时，函数内部会发生：  
·创造一个空对象并且this变量引用了该对象，同时还继承了该对象的原型

·属性和方法被加入到this引用的对象中

·新创建的对象由this所引用，并最后隐式的返回this

var obj = new Object();

obj.name = hgf;

自定义

function person() {

This.name = hgf;

This.age = 19;

}

var Hgf = new person();

Var Hgf1 = new person();

new操作符只能返回对象,无法手动改变为返回原始值

Object.create(原形)方法

包装类

String(); Boolean(); Number();

每当需要访问属性时都会产生一个包装类，把变量隐饰为对象

原型

1.定义：原型是function对象的一个属性，它定义了构造函数的公共祖先。通过该构造函数产生的对象，可以继承原型的属性和方法，原型也是对象。

2.利用原型特点和概念可以提取公有属性。

3.对象如何查看原型---->隐式属性\_\_proto\_\_

4.如何查看对象的构造函数---->constructor

当构造函数没有这个属性时可以从构造函数的原型继承，如果有就用构造函数里的

原型链

1. 如何构成原型链？  
   2.原型链上属性的增删改查

操作原型链最下方，无法用delete删除父级属性只能删除原型链最下方自己的属性，原型链上属性不可以改，叫属性赋值，父级属性不变，原型链最下方属性或改或赋值

1. 绝多数对象都会继承自Object.prototype

所有原型都有对象是错误的

1. Object.create(原型)；

例：

GrandFoo.prototype.xing = “han”;

Function GrandFoo() {}

Var grandFoo = new GrandFoo

Foo.prototype = grandFoo;

Function Foo() {

This,name = ‘gf’;

}

Var foo = new Foo;

Son.prototype = foo;

Function Son() {

This.sex = ‘male’;

}

call/apply

作用：改变this指向

区别：改变完指向后面传的参数形式不同

Function test() {

Document.write(this.name);

}

Window.name = ‘abc’;

Var obj = {name: ‘hgf’};

Test.call(obj);//打印出hgf而不是abc，call可以改变this指向

Call(改变的指向，传递的参数，传递的参数)

Apply(改变的指向，参数以数组形式传进去)

继承发展史

1. 传统形式：原型链，只要是父级的属性全都继承

·过多的继承了没用的属性

1. 借用构造函数

用call来改变指向，用多个构造函数达到继承的目的

·不能继承借用构造函数的原型

·每次构造函数都要多走一个函数

1. 共享原型

·不能随意改动自己的原型

Function inherit(继承)(C, P) {

C.prototype = P.prototype;}

P.prototype.age = 19;

Function P() {

}

Function C() {

}

Inherit(P,C)

Document.write(new C().age);//输出age=19

1. 圣杯模式

Function inherit(C, P) {

Function F() {}

F.prototype = P.prototype;

C.prototype = new F();

C.prototype.constructor = C;

C.prototype.uber = P.prototype;(只是为了记录真正的父级，超类)

另一种写法(更好点)

var inherit = (function() {

var function F() {}

return function(C,P){

F.prototype = P.prototype;

C.prototype = new F();

C.prototype.constructor = C;

C.prototype.uber = P.prototype;}

}())

属性表示方法

obj.prop obj[‘prop’]

obj.prop在系统内部会自动转化为obj[‘prop’]

对象的枚举

遍历对象，for(var prop in obj)，定义一个变量，每次循环这个变量都代表一个属性，prop是变量为字符串，对象中的属性也是字符串，所以不能用obj.prop访问属性，直接用obj.prop会在对象中寻找prop属性，要用obj[prop]访问

1. hasOwnProperty:判断是不是本身的属性，剔除原型链上的属性

对象是谁就是谁调用oPerson.hasOwnProperty(prop)，是自己属性就返回true，用if()语句来区分是不是本身的属性

1. in:判断一个属性是不是在这个对象中，是的话返回true

name in oPerson ，name属性是否在oPerson对象中

1. instanceof:前面的对象是不是后面的构造出来的

辨别方法：前面的原型链里有没有后面的原型

oPerson instanceof Object

this

1. 函数预编译过程this-->window
2. 全局作用域里this-->window
3. Call/apply改变this指向
4. obj.func()，func()里this指向obj，谁调用的this，this就指向谁

This的指向在函数定义时无法确定，只有在执行时才会确定

例题：var name = “222”;

var a = {

name : “111”,

say : function () {

console.log(this.name);

}

}

var fun = a.say;

fun(); a.say();

var b = {

name : “333”,

say : function (fun) {

fun();

}

}

b.say(a.say); b.say = a.say; b.say();

答案：222 111 222 333

arguments

arguments.callee(实参)：表示本函数引用，调用函数自身

func.caller:看是谁调用了func函数

克隆

Object.prototype.toString.call()来区别变量是数组还是对象，是对象返回值是[object,Object]，是数组返回[object,Array]

浅层克隆：

Var obj = {

Name : “hgf”,

Age : 19;,

Sex : “male”

}

var obj1 = {}

function clone(parent,child) {

var child = child || {};

for(var prop in parent){

if(parent.hasOwnProperty(prop)){

child[prop] = parent[prop];

}

}

}

深层克隆：

数组：

1. 数组的定义：

·new Array(length/content);

Var arr = new Array(200,100,100);，只填一个值指的是数组长度，填多个才是数组

·字面量;

Var arr = [100];数组内容就是100

1. 数组的读和写

·arr[num]不可溢出读

Console.log(arr[100])==>undifined

·arr[num] = xxx;//可以溢出写

可以随便填数组值 arr[100] = 1;

数组常用方法

改变原数组

·reverse 反转数组 arr.reverse();

sort

push 在数组的最后一位加一个数多个数或一个数组都可以，数组的length+n arr.push(X,[Y,Z])

pop 从末位删且只能删一个，不是真的删，相当于剪切 arr.pop()

shift 打头删 arr.shift();

unshift 在数组的第一位加一个数多个数或一个数组都可以，数组的length+n arr.unshift(X,[Y,Z]);

splice 截取剪切 arr.splice(从第几位开始,截取的长度,在截取的位置插入的值)

不改变原数组

·concat,join--->split,toString

类数组

1. 可以利用属性名模拟数组的特性
2. 可以动态增长length属性
3. 如果强行让类数组调用push方法，则会根据length属性值的位置进行属性的扩充，push原码：

Array.prototype.push = function (xxx) {

This[this.length] = xxx;

This.length++;

}

Try catch

Try{放可能会出错的代码}catch(){}，放在try{}catch(e){}里的代码依然会出错但不会影响以外的代码运行，一但try里代码出错会跳出try进入catch，不继续运行错误代码后面的代码，系统在try里挑出的错误会以参数的形式传到catch里，e是对象，有name和message属性(错误名称和信息)  
企业级开发模块怎么写js

利用闭包使变量变为私有

逗号运算符，前面的只是解析一下，返回的是后面的

函数表达式在后面直接加小括号运行，函数定义就没了