一．变量分为原始值和引用值(由值决定类型)

1> 原始值 存在stack(栈)里面

(1)Number(数字类型)

(2)String(字符类型)

(3)Boolean(true/false)

(4)undefined(没有定义)

(5)null(无意义空 占个位置)

2>引用值 大部分放在heap(堆)里面

(1)array(数组)

(2)Object(对象)

(3)function(方法)

(4).......date regExp

1. 布尔值为false的情况(除了这些boolean都为true)
2. undefined(未定义，找不到值时出现)
3. null(代表空值)
4. false(布尔值的false，字符串"false"布尔值为true)
5. 0(数字0，字符串"0"布尔值为true)
6. NaN(无法计算结果时出现，表示"非数值"；但是typeof NaN==="number")
7. ""(双引号)或''(单引号) (空字符串，中间有空格时也是true)

7>实例 -->

打印出false 或者 使用Boolean(a)测试

三．与( || )或( && )非( ! )问题

1>或( && )从第一个往后依次判断是否位假, 是假则返回其值, 若全为真则返回最后一

个值 -->

打印出的结果为3

1. 与( || )从第一个往后依次判断是否位真, 是真则返回其值, 若全为假则返回最后一个

值 -->

打印出的结果为1

3>非( ! )表达式转化成布尔值取反

注意: !!就是取反再取反, 没有发生改变

四．typeof详细讲解(检验值类型)

1>typeof()返回值的六种类型

(1)number

(2)string

(3)boolean

(4)object(null数组对象都是返回这个)

(5)undefind

(6)function

2>typeof的显示类型转换(注意其大小写)

(1)Number(mix)返回值为mix, 返回类型为number类型

注意：若为字符返回NaN, 且mix有小数则报错

(2)parseInt(a, radix)返回值为a成radix为底的十进制数, 返回类型为数字类型

注意：若numb不是radix进制格式的数则返回值为NaN

(3)parsFloat(str)返回值为str(有小数点小数点原样返回) 返回类型为number类型

(4)String(str)返回值为str(原样返回), 返回类型为string类型

(5)str.toString(radix)返回值为str以radix的数, 返回类型为string类型

(6)Boolean(str)返回值为str的true/false, 返回类型为boolean类型

(7)实例 -->

## 值打印出65,

## 类型打印出number

3>隐式类型转化

(1)isNaN(a)a转化成number与NaN进行对比 返回类型为boolean

(2)++/-- +/-(一元正负)转化成Number在相加, 返回其值 返回类型类为number

(3)+ 转化成string在相加, 返回其值 返回类型为string

注意：两边有一个是字符串才转化

(4)-\*/% 转化成Number在相乘, 相减.. 返回类型为number

注意：若有字符串则会返回NaN

(5)&&||！数字优先原则, 返回值为true/false 返回类型为boolean

(6)<><=>= 转化成数字优先原则, 返回值为true/false 返回类型为boolean

(7)== !== 转化数字优先原则, 值为true或者false 返回类型为boolean

数字优先原则(全为数字返回数字)

注意：undefind和null不大于等于小于零

undefind=null是true,

NaN=NaN是false

===和！==不发生隐示类型转换

1. 定义函数

1>函数声明

(1)function () {}(匿名函数, 一般立即执行函数这样声明)

(2)function test() {} 执行函数直接test()

注意：函数声明开发规范写明命名函数需是小头封式(theFirstName)

2>函数表达(使用一个变量来接收这个函数)

(1)var test = function test () {} (命名函数表达式) test无意义

(2)var test = function abc () {} (匿名函数表达式) 常用

注意：两者唯一的区别就是查看test.name属性时写了属性数abc没写则是test

1. 实参和形参

1>实例演示 -->

function sun(a, b) { c = a + b console.log(c) } sun(1, 2)

a, b是形参 1, 2是实参

2>实参长度和形参长度

(1)形参长度 --> arguments.length

(2)实参长度 -->函数名.length

3>映射关系(形参和实参对应时才有) -->

function sum(a, b) { b = 2; console.log(arguments[1]);} sum(1);

打印出undefined, 打印不出2

sum(1)改为sum(1, 随便什么参数) 打印出2

1. return
2. return的作用

(1)函数有return时结束 函数没写return在函数的最后系统会隐示的自动加上return.

(2)能返回值。(用这个时候一般用一个变量接收这个值)

1. js语句
2. switch case条件判断语句

(1)用法的基本形式 -->

switch (表达式) case 条件1 执行函数 case 条件2 执行函数 ...

注意：##它是打印出符合条件之下的所有语句(穿透)。

##break可终止穿透, break只能放在循环里面, 不然报错。

##continue是继续 终止本次循环执行下一圈循环

2>while循环

while循环用于不确定循环次数, if循环可以控制次数

(满足条件无限循环里面的东西, 知道不满足条件) while(条件){循环的东西}

1. Js执行的三部曲(预编译)

1>语法解析(通篇扫描下有没有低级的语法错误 比如少了分号没有)

2>预编译(重点)

(1)预解析的详情讲解过程(函数整体提升, 变量声明提升)

##创建AO对象(Active Object)

##找形参和变量声名，将变量和形参名作为AO属性名，值为undefind

##将实参和形参相统一(实参值赋值给形参)

##在函数体里面找函数声明，赋值予函数体

注意： 函数是整个函数提升上去

变量只是变量名提升上去了而变量值没提升上去

3>解释执行(解析一行执行一行)

1. 作用域及作用域链(里面的可以访问外面的, 外面的不能访问里面的)

1>变量的作用域类型

(1)全局作用域(window, GO)

##未经声明的变量赋值, 此变量全局对象所有(window), 称暗示全局变量

##全局上的任何变量都是window所有

(2)局部作用域(AO)

2>[[scope]]属性, 即作用域

(1)属性其里面存储了执行期上下文(AO, GO)

(2)scope就是javascript函数(对象)中一个我们不能访问的属性, 仅供javascript引擎使用的属性

3>作用域链

[[scope]]中所存储的执行期上下文对象的集合，这个集合呈链式链接，我们把这种链式链接叫做作用域链。

4>执行期上下文

当函数执行时，会创建一个称为执行期上下文的内部对象。一个执行期上下文定义了一个函数执行时的环境，函数每次执行时对应的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，执行上下文被销毁。

5>闭包

闭包就是里面的函数被保释到了外部(在销毁之前保存到了外部)

1. 闭包

1>基本概念

闭包就是里面的函数被保释到了外部(在销毁之前保存到了外部)

2>闭包的私有化(留一个接口return里面的函数) -->



1. 立即执行行数(与普通函数唯一的区别是执行后马上释放)

1>写法

(1)( function (形参) {} (实参) )

(2)( function (形参) {}) (实参)

注意：function () {}是匿名函数

2>一般用法

(1)return出一个值, 使用一个变量接收

3>注意事项：

(1)只有表达式能被执行符号执行, 且被执行符号执行的函数会被永久放弃 -->

##var test = function () { console.log('a'); } () 输出a

##function test() { console.log('a');}() 输出不了a, test输出不了函数

(2)()里面是表达式 (function f() {})里面的函数变成了函数表达

1. 对象

1>构建对象的方式(常用, 解决了下面所毛病)

(1)自定义构造函数创建对象

##function 大头峰式命名 () {} var 变量(小写) = new 前面声明的函数() -->



##new对象时候详细讲解



(2)调用系统构造函数创建对象(无法识别是属于具体的哪个对象)

##var 变量名 = new Object() //用个变量接收叫实例化对象 -->



(3)工厂模式创建对象(能批量生产对象,且能识别类型，但繁琐) -->

##function 变量() { var obj = new Object(); 添加属性方法在此添加 return obj;}

var 变量接收 = 变量() (函数执行); (接收变量和变量可以不同) -->



注意:return obj一定要写不然打印出的对象是undefined

(4)通过字面量对象创建对象var obj = {}

(5)Object.create()创建对象

##语法

Object.create(proto[,propertiesObject])

-- proto: 对象, 要继承的原型为null值时

--propertiesObject: 对象, 可选参数, 为新创建的对象指定属性对象。

注意：第一个参数为null时, 是唯一一个继承终端不是Object.prototype

2>循环对象的方法

(1)for prop(随便取, 其值为对象的key) in 对象

3>对象的增删改查

(1)增 对象.东西(里面没有的东西) = "值" -->

mrZhu.age = "21" 则在1的例子里面加上了 age : 21,属性。

(2)查 对象.东西(里面有的东西) -->

mrZhu.sex 里面就会显示male

(3)改 对象.东西(里面有的东西) = "值" -->

mrZhu.smoke = "1年" 对象里面smock就会显示1年。

(4)删 delete 对象.属性(里面有的属性) -->

delete mrZhu.sex 对象里面就没有sex属性了。

4>构造函数的内部原理

(1)在函数题前面隐示的加上this = { }

(2)执行this.xxx = xxx

(3)隐示的返回this -->



注意：只有new Person()之后才发生构造函数的这三步，否则就是正常函数的执行。

5>对象小知识总结

(1)当一个变量没经过声名就使用会报错，但当一个属性没有使用会输出undefined

(2)调用属性的两种方式

##obj.属性 (使用obj.属性里面会隐示的转化下obj[name])

##obj['属性'] 应用(实现调用方法数字几就出几的名字)-->



1. 原型

1>原型的基本概念

(1)型是function对象的一个属性，它定义了构造函数制造出了对象的共有祖先。

(2)原型即prototype，函数刚出生原型就定义好了。它类似一个空对象。

(3)函数会继承原型上的属性和方法(可以利用其, 提取出共有属性)。

注意：若自己身上有原型上一模一样的属性时出现自己身上的属性。

2>原型的增删改查(四个差不多, 以增为例子) -->

Car.prototype.height = 1400;

function Car(color) { this.color = color } var car = new Car(）；

car.prototype.height = 1000;

注意：添加需要.prototype

3>constructor(构造器), prototype和 \_\_proto\_\_

(1)constructor(构造器)

##每一个原型对象都会自动获得一个constructor属性，该属性指向prototype所在函数的指针；

(2)prototype

##只要创建一个任意新函数，就会根据一个prototype属性，该属性指向函数的原

型对象；

(3)\_\_proto\_\_

##当调用构造函数创建实例时，该实例内部将包含一个指向构造函数原型对象的指针，在大部分浏览器中用\_\_proto\_\_标识；(继承就是上它指向的原型找)

(4)三者之间的联系

##person(实例化函数).\_\_proto\_\_ == = Person(构造函数).prototype

##person(实例化函数).constructor === Person(构造函数)

##Person(构造函数).prototype.constructor === Person(构造函数)

(5)总结

##\_\_proto\_\_和constructor是对象独有的属性

(在js中对象是由函数构成的所以有这两个属性)

##prototype是函数独有的属性

(函数也是对象的一种，是并集关系不是父子集关系)

##hasOwnProperty(属性的字符串形式), 判断是不是原型上的属性

##in 使用形式 -->"属性名" in obj

注意：in判断原型上的属性和hasOwnProperty一样, 但不同的是它会把自己的原型也算上来

4>构造函数内部原理 -->



5>原型小知识总结

(1)object.prototype是所有原型链的终端除了Object.create(null)创建的对象

(2)原型链不能增删改查, 只能自己修改自己原型

(3)Object.creat( 对象或者null )

##对象的话原型就指向这个对象

##null对象不最终继承Object.prototype除了这个全是最终继承Object.prototype

(4)人为设的\_\_proto\_\_不会继承(写了Object.creat(null)截断了继承对象，然后人为设个\_\_proto\_\_指向是不会继承的)

(5)constructor是Object()就是原型链的顶端

(6)因为原始值undefind和null不是对象，也没有原型链所以不可以经过包装类所以不可以添加属性和方法

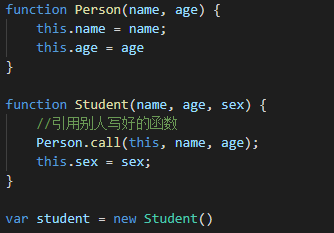
1. 继承发展发展史
2. 传统模式 --- 原型链

缺点：过多的继承了没用的属性, 全部都会继承

2>借用构造函数(开发时常用, 不能算标准的继承)

缺点：每次构造函数都要多走一个函数, 原理是方法借用, 原型是自己

(1)实例 -->



3>共享原型

(1)实例 -->



(2)共享原型继承封装成方法(Target继承Oringin的) -->



缺点：不能修改自己的原型, 修改自己的原型会影响父级。

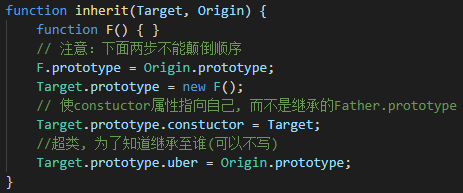
4>圣杯模式

原理：创建一个中间函数, 通过new 中间函数连接两个函数

(1)实例 -->



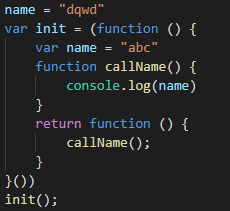
(2)封装方法 -->



1. 命令空间(闭包应用, 模块化开发, 防止污染全局变量)

1>目的：管理变量, 防止污染全局变量(模块化开发)

2>实例 -->

 打印出abc, 不会污染全局变量

1. 包装类

1>包装类基本概念

(1)普通变量不能直接调用属性和方法, 而对象可以直接调用属性和方法

这些普通类型的变量会默认会调用包装

包装类: 普通类型的变量经过包装可以直接调用属性和方法 -->



(2)基本包装类型：数字(number), 字符串(string), 布尔(boolean)

2>包装类详解

加属性方法时候内部会隐式的添加, 执行完毕之后会销毁 -->

var num = 4; num.len = 3 new Number(num).len(隐式的, 执行后销毁)

注意：字符串赋length属性不会影响length数值, 数组会截断数组 -->

var str = “abcd” str.length = 2

访问str.length值为4(包装类的length属性) str = “abcd”

var arr = [1, 2, 3, 4] arr.length = 2

访问arr.length = 2 arr = [1, 2]

3>包装类小知识总结

(1)原始值不能有属性和方法的为什么能调用？

经过包装类(不能加属性但又能访问属性)

1. 常见的字符方法(都是静态方法)及字符串小知识

1>.charAt(index) 返回对应字符串的第index位字符 -->

 返回值为h

注意: 超过字符串长度(length-1)返回空字符串,空格也占一个字符索引位。

2>String.fromCharCode(ussc码, ussc码, ...)返回参数对应的USSC码对应的字符串

 返回值为ABC

3>.indexOf(要找的字符串, [从某个位置开始找(索引值)])

找参数字符串的索引位(没找到返回-1) -->

注意区分：lastIndexOf()从后先前找，获取的索引仍然是从左往右数

 返回值为0

3>.replace(“old string”, “nwe string”)方法 把old string替换成nwe string

 返回值为olllll

注意：不用正则只替换一次

4>.slice(“从第几位”, “到第几位”)返回提取后的字符串(第几位都是索引)

 返回值为h

5>.split(“删除什么字符串”, [“切割后留下几个”] )按什么字符串拆分 返回多个数组

 返回值为[“a”, “b”]

6>.substr(“开始的位置”, [“个数”])返回截取后的新字符串

 返回值为el

7>.substring(“开始的索引”, “结束的索引”), 返回截取后的字符串，不包含结束的索引字符串

 返回值为he

8>.toLocaleLowerCase(); 把字符串转为小写的 和toLocalecase();一模一样

 返回值为hello world

9>.toUpperCase(); 把字符转为大写

 返回值为HELLO WORLD

10>.trim(); 去除字符串两端的空格



返回值为 

1. 字符串小知识

(1)字符串string和字符串对象(new String())

-->

 输出还是hello 无法改变(只可以读，不可写)

-->

栈指向了新的堆

输出的结果是”小苏好邪恶”

1. 数组

1>构建数组的方式

(1)var arr = [];

(2)var arr = new Array();

注意：这个括号和(1)构建的对象[]添加值一样唯一的区别就是只传一个值这个代表数组长度。

2>数组小知识

(1)当给数组第十位赋值时会将数组撑长为11 -->

arr[10] = abc 打印出 [undefined\*10, abc], 且length值为11

(2)数组除了可以用for循环遍历外还可以使用for item in arr(数组是特殊的对象)

3>数组方法

(1)可以改变原数组的方法

##.push(); 把值追加贷数组中(最后面), 返回值为追加后数组的长度 -->



数组为 打印arr为 5

##.pop()把删除数组中最后一个元素的值, 返回值为删除的这个值



数组为 打印arr为400

##.shift()删除数组中第一个元素, 返回值为删除的这个值



数组为 打印arr为 100

##.unshift(值)向数组中的第一个元素前插入值, 返回的是插入后数组的长度



数组为 打印arr为5

##.sort(function () {})排序不稳定, 可传入一个比较函数可以进行排序

function () { 排序规则 } 升序原理(return a-b;)降序(return b-a)

注意：1形参一定要写两个值

2看返回值

1)当返回值为负数,那么前面的数放在前面 (return 1和return -1)

2)当为正数,那么后面的数在前面

3)为0,不动

##.reverse()数组反向



数组为 打印arr为[25, 16, 4, 1]

##.splice(“开始”, “要删除的个数”, [“截取后添加的元素”])截取切在截取位置添加元素



数组为 打印arr为4

注意：截0位(在数组中间添加元素)

(2)不改变原数组(一般用个函数接收不然没意义)

##.concat()连接两个数组



数组为 打印arr为原数组arr

##.join("东西"按东西样式连接(注意需要时字符串形式))



数组为 打印arr为原数组arr

##.split("东西"按照东西样式拆分数组)



数组为 打印arr为

##.slice(“开始”, “结束“);截取数组从开始到结束(不包括结束那个元素)



数组为 打印arr为原数组arr

4>其他数组方法

(1)A instanceof B(为Array或者Object) 检测A是否属于B -->





注意：数组是特殊的对象上例子中写 [] instanceof Object 返回值为true

(2)Array.isArray(数组) 判断是否是数组 -->





(3)every(function () {}) 可用于数组判断

##数组中的每个元素都会执行这个函数

##函数的第一个参数代表元素的值

##第二个代表元素的索引

##第三个是原数组(谁调用的就是谁一般用不到第三个参数)

##返回的是布尔值, 所有都满足条件才返回true -->





(4)filter(function () {})筛选

##返回符合设定条件的值成数组(新数组)形式 数组中的每个元素都会执行这个函数##函数的第一个参数为元素的值

##第二个元素的索引

##第三个参数为遍历的数组(原数组) -->





(5)forEach(function () {}) 循环数组(相当于一个for循环)

##第一个参数为元素的值

##第二个为元素的索引

##第三个参数为原数组

(6)indexOf(元素值) 返回的是元素的索引, 没有这个元素值则返回-1 -->





(7)map(function () {}) 数组中每个元素都会执行这个函数，执行后的结构重新放到一

个新数组中返回出来





1. 类数组

1>是类数组的条件

(1)属性要为索引(数字)属性

(2)必须要有length属性(必要条件),做好加上push属性

2>类数组注意事项

(1)类数组长得像数组但又不能用数组的方法 -->

arguments是个类数组,不能用push数据进数组里面会报错

(2)强行让类数组调用push方法,则会根据length属性值的位置进行扩充 -->



打印出

##其原理就是push方法的封装 -->



1. this问题

1>this指向问题

(1)函数的预编译过程this执向window

new test();new完之后会预编译第一时候this = Object.creat(test.prototype)

(2)全局作用域this指向window

(3)call/aplay可以改变this指向。

(4)obj.fun(); fun()里面的this指向obj(谁调用这个方法this指向谁) -->

var obj = { a : function () { console.log(this.name) }, name : "abc" }

obj.a(); 这个this指向obj

2>this小知识总结

(1)函数的连续调用return this(函数不屑return默认的值是undefined)

(2)call的用法 --> .call(需要指向的函数, 参数)

(2)apply的用法 --> .apply(需要指向的函数, 数组(参数的集合))

注意：只能是数组

1. 区分数组[]和对象{}的几种方法

1>.obj.constructor打印function Object(){[native code]}

2>.[] instanceof Array 打印ture ([]对象\_\_poto\_\_上有没有Array的prototype)

3>Object.prototype.toString.call([]); 打印出'[object array]'

注意：var obj = {} (直接在控制台上打印{}不可以会报错)

1. Data对象

1>不需要先new Date的方法

(1).new Date() 获取当前时间 -->





(2).new Date(string) 传入字符串参数的时间 8:00 -->





注意:需要是字符串，获取的是时间类型的值

(4)计算时间 -->





注意：是数字类型 为了计算，时间值和时间值不可计算可以用其计算时间



2>需先new Date的方法(常用)

(1).getFullYear()获取年的方法用 -->





注意区分：.getYear()方法不常用(1900到现在的年数)

(2).getMonth() 获取月份方法 -->





注意：获取的月份从0开始要 +1才是当前的月份 例子中真实是7月

(3).getDay- 获取星期几 -->





(4).getDate() 获取日期 -->





(5).获取小时,分钟，秒



(6)其他方法

##日期









##小时分钟秒









##毫秒数





注意：和Date.now()区别

-- Date.now和new Date是同级， 而value需先new Date在使用

1. 内置对象Math(大部分用来处理数字)
2. 以查看js文档百度搜索MDN

2>Math不是一个构造函数不像Object -->

(1)不能使用new Math实例化一个方法(报错)

(2)Math.PI();(PI方法在里面已经定义好了)

3>封装一个Math.max()方法-->



1. 静态方法(Math.radaio())和实例方法(new Date().方法)



分割线

1. 定时器(属于BOM部分)

1>基本定时器

(1)格式 --> setInterval( function () {}, time );

(2)扩展：

##每隔time时间执行一次function(执行无限次), 且time只识别一次

##定时器会返回一个唯一标识

##可用一个变量接收定时器唯一标识, 用于清除定时器(clearInterval) -->

var timer = setInterval(function () {}, 1000) clearInterval(timer)

2>一次性定时器

(1)格式 --> steTimeout(function() {}, time)

(2)扩展：

##每隔time时间执行一次function(只执行一次)

##清除定时器clearTimeout()

1. 脚本化css
2. 读写CSS属性

(1)domEle.style.prop

##可读写行间样式,没有兼容性问题

(2)注意格式

##float这样的关键字属性应在前面加css -->float --> cssFloat

##复合属性必须拆解，组合单词采用小驼峰式写法

##写入值必须是字符串格式

1. 查询计算样式

(1)window.getComputedStyle(ele,null)[prop]

(2)注意事项：

##计算样式只读

##返回的计算样式的值都是绝对值，没有相对单位

##IE8及其以下的版本不兼容

##()里第二个参数可以获取伪元素的样式

(3)IE计算样式

##ele.currentStyle[prop]

++计算样式只读

++返回的计算样式值不是经过转换的绝对值

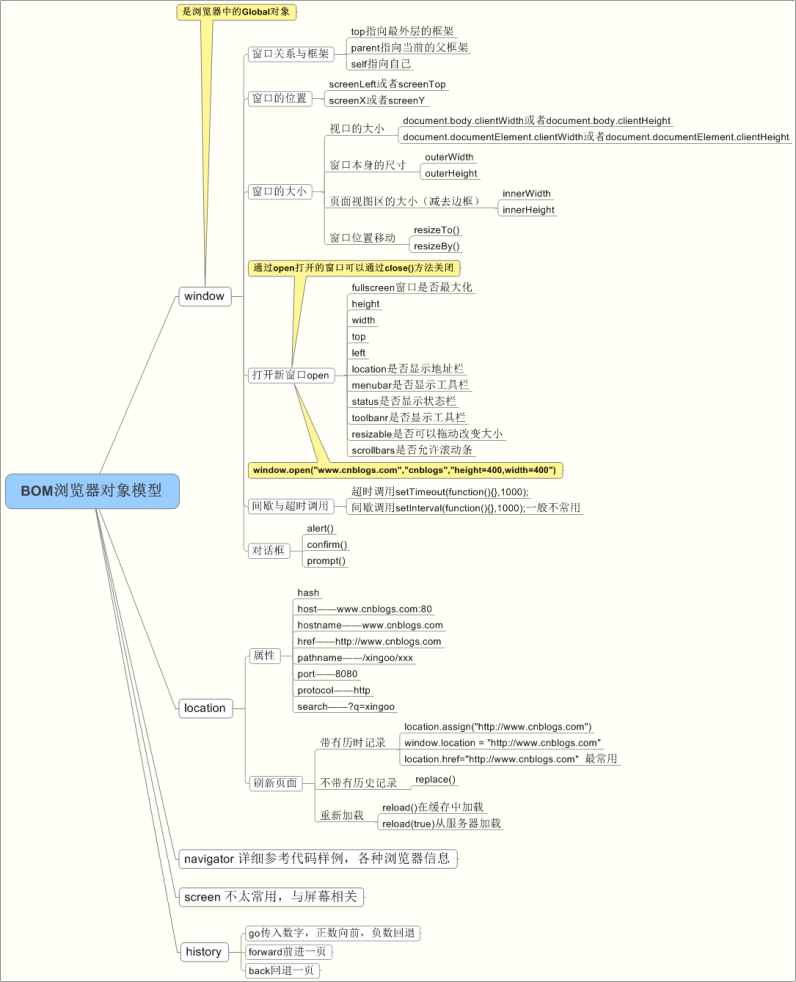
++IE独有的属性

1. BOM部分

1>基本概念

是浏览器对象模型, 可以与浏览器窗口进行互动的对象结构, BOM由多个对象结构构成, 其中代表浏览器窗口的window对象是BOM的顶层对象, 其他对象是该对象的子对象

2>思维导图



3>BOM对象

(1)window对象, 是JS的最顶层对象, 其他的BOM对象都是window对象的属性

##详细说名

window对象双重角色

++是javascript访问浏览器窗口的一个接口,

++又是ECMscript规定的Global对象

##常见用法

++弹框类方法, 前面省略window

--alert(“提示信息”)

--confirm(“确认信息")

--prompt(“弹出输入框”)

--open(“url地址”, “打开方式(可以是-self或-black)”, “新窗口的大小”)

注意：如果url为空，则默认打开一个空白页面，如果打开方式为空，则默认为新窗口方式打开页面。

返回值为：返回新打开窗口的window对象

--close()关闭当前页面

注意：存在兼容问题 谷歌(默认直接关闭)IE(询问用户)

++其他方法

--window.moveTo() 移动当前窗口

--window.resizeTo() 调整当前窗口尺寸

++定时器方法(更详细内容参考 一. 定时器)

--setTimeout(函数, 时间)只执行一次

--setInterval(函数, 时间)执行无数次

(2)document对象, 文档对象

(3)location对象, 浏览器当前URL信息

##详细说明

window.location对象：用于获得当前页面的地址 (URL)，并把浏览器重定向到新的页面。在编写时可不使用 window 这个前缀。

##方法

--location.href = “url地址”

--hash 返回#号后面的字符串，不包含散列，则返回空字符串

--host 返回服务器名称和端口号

--pathname 返回目录和文件名。 /project/test.html

--search 返回？号后面的所有值

--port 返回URL中的指定的端口号，如URL中不包含端口号返回空字符串

--portocol 返回页面使用的协议。 http:或https:

(4)navigator对象, 浏览器本身信息

##详细说明

window.navigator 对象包含有关访问者浏览器的信息。在编写时可不使用 window 这个前缀。

##方法

--navigator.platform：操作系统类型；

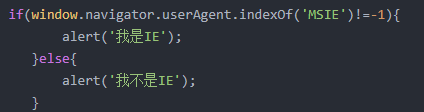
--navigator.userAgent：浏览器设定的User-Agent字符串。

--navigator.appName：浏览器名称

--navigator.appVersion：浏览器版本

--navigator.language：浏览器设置的语言；

--userAgent是最常用的属性，用来完成浏览器判断(配合ondexOf) -->



(4)screen对象, 客户端屏幕信息

##属性

--window.screen 对象包含有关用户屏幕的信息。

--screen.availWidth 属性

返回访问者屏幕的宽度，以像素计，减去界面特性，比如窗口任务栏。

-- screen.availHeight 属性

返回访问者屏幕的高度，以像素计，减去界面特性，比如窗口任务栏。

(5)history对象, 浏览器访问历史信息

##方法

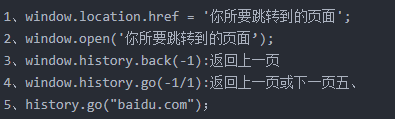
--window.history 对象包含浏览器的历史。

为了保护用户隐私，对 JavaScript 访问该对象的方法做出了限制。

--history.back() - 加载历史列表中的前一个 URL。返回上一页。

--history.forward() - 加载历史列表中的下一个 URL。返回下一页。

1. 综合页面跳转方法 -->



4>window对象常用方法补充

(1)window.innerHeight(innerWidth)：

表示可见区域的高度(宽度)如果底部(右侧)有滚动条, 会把底部(右侧)滚动条的高度(宽度)计算进去，此方法适用于标准浏览器及IE9+

(2)document.documentElement.clientHeight(clientWidth)：

表示可见区域的高度(宽度)不会把滚动条的高度宽度计算进去, 此方法适用于标准浏览器及低版本IE标准模式。

(3)document.documentElement.offsetHeight(offsetWidth)：

整个文档的高度(宽度)offsetWidth获得的值与clientWidth一样，而offsetHeight获得的是整个文档的高度。

(4)document.documentElement.scrollHeight(scrollWidth)：

获得网页正文全文高(宽), 包含边线宽度。

(5)document.body.clientHeight(clientWidth)：

整个文档的高度(宽度)clientWidth获得的值等于

document.documentElement.offsetWidth剪去body边线宽度，而clientHeight获得的是整个body的高度减去边线宽度。

(6)document.body.offsetHeight(offsetWidth)：

获得的值等于document.body.clientHeight（clientWidth）加上body的边线(border)宽度，即这两个值取到到的包含边线宽度。

(7)document.body.scrollHeight(scrollWidth)：

获得网页正文全文高和宽, 包含边线宽度。

(8)document.body.scrollTop（scrollLeft）：

浏览器窗口顶部与文档顶部之间的距离(浏览器窗口与文档左侧之间的距离)，也就是滚动条滚动的距离。

(9)window.pageYOffset(pageXOffset)：

浏览器窗口顶部与文档顶部之间的距离(浏览器窗口与文档左侧之间的距离), 也就是滚动条滚动的距离。

(10)window.screen.height（width）：

屏幕分辨率的高(宽)

(11)window.screen.availHeight（availWidth）：

屏幕可用工作区高(宽)。

(12)以上方法区别

##offsetTop、clientTop、scrollTop分别代表什么

--offsetTop:距离上方或上层控件的位置, 整型, 单位像素。

--clientTop:元素的内容区距离元素边缘的距离, 即元素的边线宽度。

--scrollTop:元素窗口顶部与元素顶部之间的距离, 即元素内部滚动条滚动的距离

四．DOM部分

1>选中dom

(1).getElementsById();通过ID名选择出元素 不区分id的大小写

(2).getElementsByTagName();通过标签名选择出元素（表单元素）

(3).getElementsByName();通过name选择出元素

(4).gtElementsByClassName();ie9一下不能用 通过class民选择出元素

下面的是静态的是副本，上面的是动态的，实时的。

(5).querySelector("里面css怎么选这里怎么选");

(6).querySelectorAll("里面css怎么选这里怎么选");选择出一组

2>遍历节点树(注意是节点)

(1)parentNode 父节点(最顶端是#document)；

(2)childNodes[第几个节点，可不写]--子节点们（所有的子节点）

(3)firstChild-第一个子节点

(4)lastChild-最后一个子节点

(5)nextSibling-后一个兄弟节点

(6)previousSibing 前一个兄弟节点

3>节点类型

(1)元素节点 -----1

(2)属性节点-----2

(3)文本节点-----3

(4)注释节点----8

(5)document节点-------9

(6)DocumentFragment-----11

4>基于元素点的遍历(除了chidren其他的都是IE9一下不兼容的)

(1)parentElement（顶端是html）返回单前元素父元素节点

(2)chidern 返回当前元素的子元素节点

(3)firstElementChild 返回第一个元素节点

(4)lastElementChild 返回最后一个元素节点

(5)nextElementSibLing后一个元素节点

(6)previousElementSibLing前一个元素节点

5>节点的四个属性

(1)nodeName

##元素的标签名称, 以大写的形式表示，只读

##属性节点显示属性名称, 文本节点显示#text, 文档节点显示#document

(2)nodeValue 只有文本节点和注释节点属性节点，有(text,comment)可读可写

(3)nodeType 返回节点的类型(>3), 只可读

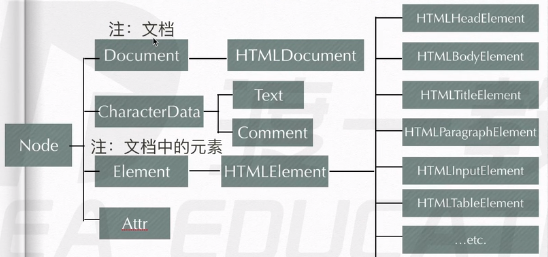
(4)attributes属性节点的集合

6>检测方法

(1)hasChildNodes();方法 检测有没有子节点有turn没有false。

7>DOM结构树(继承关系)

(1)基本结构



补充 document继承HTMLDocument和XMLDocument(不常用)继承Document

Node的\_\_proto\_\_(继承)为EventTarget的\_\_proto\_\_(继承)为Object

(2)继承关系实例演示

##getElementsById定义在Document.prototype上 ,

即Element节点上不能使用

##getElementByName定义在了HTMLDocument.prototype上,

即不能在xml document和Element上使用

##getElementByTagName定义在Document.prototype上和Element上,

即都能使用

##HTMLDocument定义了一些常用的属性, body和head分别指其文档内的标签

--document.body表示<body>标签

--document.head表示<head>标签

##Document.prototype上定义了documentElement属性, 指代文档的根标签

--document.documentElement表示<html>标签

9>操作html

(1)增(实在js手里拿着不会显示在页面, 需要显示在页面需先插入appendChild())

##.createElement("标签名");创建元素节点（标签）

##.createTextNode("");创建节点(文本)

##.creatComment("")注释节点

##.createDocumentFragment()；创建文案碎片节点

(2)插

##appendChild(可以理解成push) 页面上已经有的剪切操作 如，

##div.appendChild(span); 原来div和span是兄弟结构的变成父子结构了。

##.insertBefore(a, b); 读insert a Before b (a在b前面) 要选中他的父元素 ##ParentNode.insertBefore(a, b);

(3)删

##.parent.removeChild();选中父节点删除括号里面的子标签 不需要""

##child.remove(); 删掉选中的节点。

(4)改

##parentNode.replaceChild(new(新节点), origin(老节点))new代替origin

10>操作html和文本的方法

(1)ele.innerHTML = (覆盖)

(2)ele.innerHTML += (在内容后面加东西)

(3)ele.innertext = 文本 (覆盖) 小心有span标签

11>操作节点的属性方法

(1)setAttribute("(属性名)class","on(属性值)") -->

class="on" 添加行间属性

(2)getAttribute("标签") 取出行间属性

12>操作节点样式

(1)HTML元素.style.样式属性＝"值"

(2)HTML元素.className="样式名称(定义好的class)"

13>获取元素位置的一些属性

(1)offsetLeft/Top 返回当前元素左边界到它上级元素的左边界的距离，只读属性

(2)offsetHeight/Width 返回元素的高度

(3)offsetParent 返回元素的偏移容器，即对最近的动态定位的包含元素的引用

(4)scrollTop/Left 返回匹配元素的滚动条的垂直位置

(5)clientWidth/Height 返回元素的可见宽度

五．事件部分

1>绑定事件方式

(1)ele.onxxx = function(event) {}

##一个元素同一个事件只能绑定一次

##解除事件：ele.onxxx = null/false

##this指向dom元素本身

(2)obj.addEventListener("type", fn, false);

## IE9下不兼容

##可以一个事件绑定多个处理函数，同一个元素执行同一个函数只能执行一次

##ele.removeListener(type, fn, false)里面的全部值要是同一个

++fn需要是一个函数的引用不然找不到上面说说的同一个

##this指向dom元素本身

(3)obj.attachEvent("on+ type", fn)

##IE独有的

##可以一个事件绑定多个处理函数，同一个元素执行同一个函数可以执行多次

##和ele.removeListener(type, fn, false)一样全部要是同一个

##this指向window

2>事件处理模型(一个对象的一个事件类型只能出现一个模型(要么冒泡要么捕获))

(1)事件冒泡

##结构上(非视觉上)嵌套关系的元素，会存在事件冒泡的功能，即同一事件上，自子元素冒泡向父元素。(自底向上)

(2)事件捕获(不常规模型)(只有谷歌浏览器实现了)

##结构上(非视觉上)嵌套关系的元素，会存在事件捕获的功能，即同一事件上，自父元素捕获至子元素(事件源元素)。(自顶向下)

(3)注意事项：

##事件触发顺序, 先捕获后冒泡, 注意区分事件绑定的执行顺序

##事件的执行顺序是先绑定先执行(测试捕获和冒泡的先后顺序需注意绑定执行

顺序)

##focus, blur, change, submit, reset, select等事件不冒泡

3>取消事件冒泡

(1)w3c标准event.stopPropagation();IE9以下不支持

(2)event.cancelBubble = true; ie独有的

4>阻止默认事件

(1)return false；(句柄的方式阻止 只能取消句柄绑定的默认事件)

以对象属性的方式注册的事件才生效

(2)event.preventDefault(); W3C标准，IE9一下不兼容

(3)event.returnValue = false; 兼容IE

5>事件对象

event(就是function里面的形参e)非IE存在

IE存在于window.event--->event || window.event

6>事件源对象

(1)event.target 火狐只有这个

(2)event.srcElement IE只有这个

(3)chrome浏览器两个都有

7>鼠标事件

(1)click(敲击事件),

(2)mousedown

(3)mousemove

(4)mouseup

(5)contextmenu(右键出现菜单事件)

(6)mouseover(鼠标放进去出现什么)

(7)mouseout(鼠标出来出现什么)

(8)mouseenter

(7)mouseleave

注意：区分左右键只能用moseup和mousedown事件可以区分 里面的button属性。e.button ==2 右键 e.button == 0 左键

8>键盘事件

(1)keydown,

(2)keyup,

(3)keypress

触发顺序 keydown>keypress>keyup

##keydown和keypress的区别

++keydown能监听所有键盘类事件且charcode值为0(fn不能，fn键是辅助键)

++keypress能监听所有字符类按键charcde为ask码(能区分字母的大小写)

9>文本类事件

(1)input事件(改变就会触发该事件),

(2)focus(聚焦触发),

(3)blur(失去焦点触发),

(4)change(状态位发生改变才触发)(聚焦改变失去焦点才触发)

10>窗体操作类事件(window上的事件)

(1)scroll(滚动条滚动触发该事件),

(2)load(效率最低,等randTree加载绘制完成之后才触发, 常用)

11>总结

(1)事件一般全部都是小写

分割线

1. JSON

1>概念

是一种数据传输格式, 以对象的形式(键值对), 与对象的区别是定义的对象建必须有引号(认为规定) --> { “name”: “zhu”}

2>数据传输常用方法

(1)JSON.stringify(obj) 从一个对象中解析出字符串(传递数据给后端时使用)

-->

使用方法放回结果”{"{"a":"1","b":"2"}"}”

(2)JSON.parse(str) 从一个字符串中解析出JSON对象(前端接收后端数据时使用)

-->

使用方法放回结果Object{a:"1",b:"2"}

1. randerTran绘制页面

1>加载规则

深度优先原则(html --> head -->meta(head里面所有标签) --> body )

2>加载顺序

(1)先解析html(解析完) --> domTree

##解析完毕(DOM节点树的解析完毕)

src里面的资源和href啥的还没有加载出来

##加载完毕

src里面的资源和href啥的加载完毕

##符合加载规则(深度优先原则)

(2)在解析css --> cssTree

--符合加载规则(深度优先原则)

(3)拼接domTree和cssTree形成randerTree, 开始绘制页面

3>randerTree重新绘制(重排reflow)

(1)dom节点的删除, 添加

(2)dom节点的宽高变化, 位置变化, display: none -->block等

(3)offsetWidth, offsetLeft获取宽高位置(重新绘制randerTree获取的数据才是最新的)

4>randerTree不重新绘制, 只重新修改修改的部分(repaint重绘)

修改 如字体颜色, 背景颜色等

1. 异步加载

1>方法

(1)defer异步加载, dom文档解析完毕执行, 代码可以在内部, 只有IE9--可以使 -->



(2)aysnc异步加载, 加载完毕执行, 只能加载外部脚本(js只能是外部引入的模式)

(3)创建script, 插入DOM中, 加载完毕后callback(回调函数) (常用)

执行时, 不阻塞页面(可实现异步加载, 也能实现按需加载)

##基本形式 ->

var script = document.createElement(“script”);

script.src = “加载的文件(异步下载, 并不会执行)”

document.body.appendChild(script) (执行这行才会执行src里面的代码)

##实例 -->



注意：script.src = “demo.js”应放在if循环的后面, 防止电脑加载资源过快, 先加载完成了资源在触发事件

##封装(完整形式) -->



1. 时间线(加载, 解析顺序)
2. 创建Document对象, 开始解析web页面, 解析HTML元素和他们的文本内容后添

加Element对象和Text节点到文档中。这个阶段Document。readyState = "loading"

2>遇到link外部css，创建线程加载，并继续解析文档。

3>遇到script外部js，并且没有设置async ， defer ，浏览器加载，并阻塞，等待js加载完成并执行该脚本，然后继续解析文档

4>遇到script外部js，并且设置有async，defer 浏览器创建线程加载，并继续解析文档，对于async属性的脚本，脚本加载完成后立即执行(异步禁止使用document.write())。

5>遇到img标签等，先正常解析dom结构，然后浏览器异步加载src，并继续解析文档

6>当文档解析完成，document.readyState = "interactive"；

7>文档解析完成后，所有设置有defer的脚本会按照顺序执行。

8>当文档解析完成之后，document对象触发DOMContentLoaded事件，这也标志着程序执行从同步脚本执行阶段，转化为事件驱动阶段

注意：DOMContentLoaded事件只能绑定在addEventListener绑定

9>当所有saync的脚本加载完成并执行后，img等加载完成后，document.readyState = "complete" window对象触发load事件

10>从此，页面以异步响应方式处理用户输入，网络事件等。

11>注意document.write()问题(覆盖里面全部内容)

(1)异步加载

(2)window.load = function () { document.write() }

1. 总结区分

1>$(document).ready(function () {}) 文档解析完毕在执行

2>window.load = function () {} 文档加载完毕在执行

3>原生js模拟$(function () {})需利用DOMContentLoaded事件

或者监听document.readyState状态为interactive

注意：DOMContentLoaded事件只能绑定在addEventListener绑定

4>解析完毕randerTree绘制完毕, 加载完毕randerTree绘制完毕且外部引入的文件全部加载完毕(src里面的资源和href啥的加载完毕, img等全部加载完毕)

分割线

1. \ 转义字符(常用的)

1>\ 把后面一个字符转化成正常文本显示

2>\n 换行显示

3>\r 行结束符(window操作系统一般回车键都是 /r/n)

4>\t 一个缩进

1. 正则表达式(RegExp)
2. 创建正则表达式对象

(1)var reg = /abc(匹配规则)/i属性 (判断字符串是否含有abc)

(2)var reg = new RegExp(“abc(匹配规则)”, “i(属性)”)

1. 属性(修饰符)

(1)i 忽略大小写

(2)g 全局匹配(全部匹配规则的都返回不管是不是一样的)

(3)m 执行多行匹配(一般配合匹配第一个最后一个)

1. 表达式([], ())

(1)区间匹配

var reg = /[1-9], [A-z], [cd]/

匹配第一位1到9, 第二位A到z, 第三位c或者d的字符串

(2)查找指定选项

var reg = /(abc|bcd)[0-9]/g

匹配abc加上0到9或者bcd加上0到9随便一位数的字符串

4>元字符(表达式的简写)

(1)/\w/ 0到9, A到z, \_ /\W/ ^\w(除了\w的其他全部)

(2)/\d/ 0到9 /\D/ 除了\w的其他全部

(3)/\s/ 空白字符(空格符, 制表符(\t), 回车符(\r), 换行符(\n)等) S同上

(4)/\b(加上边界是什么字符)/ 单词边界

5>量词

(1)n+