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）

这样测试 -->

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

测试方法 -->

(1)值打印出65,

(2)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,相同true,不同false 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(全为数字返回数字), 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

注意：undefind和null不大于等于小于零且undefind=null是true,NaN=NaN是false

===和！==不发生隐示类型转换

1. 定义函数
2. 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)

1. 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>概念

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. switch case

switch case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：(1)它是打印出符合条件之下的所有语句(穿透)。

(2)break可终止穿透, break只能放在循环里面, 不然报错。

(3)continue是继续 终止本次循环执行下一圈循环

1. while循环的用法(注意和if循环区分)

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)
2. 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)
3. 预编译

(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

1. 解释执行(解析一行执行一行)
2. 作用域及和其相关的东西(里面的可以访问外面的, 外面的不能访问里面的)
3. 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, 及作用域

(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. 立即执行函数(与普通函数唯一的区别是执行后马上释放)
2. 写法

(1)( function (形参) {} (实参) )

(2)( function (形参) {}) (实参)

注意：function () {}是匿名函数

1. 实用用法

(1)return出一个值, 使用一个变量接收

注意：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 -->

1. var test = function () { console.log('a'); } () 输出a
2. function test() { console.log('a');}() 输出不了a, test输出不了函数
3. 对象(常用3>)

1>调用系统构造函数创建对象(不能识别什么类型)

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



2>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

3>自定义构造函数创建对象(解决了前面所有毛病, 常用)-->

function 大写变量() { 添加的属性方法在此添加 } var 变量(小写) = new 大写变量(函数执行);-->



new对象时候详细说明



1. 通过字面量对象创建对象var obj = {}

十四．原型