# 2019.3.20

## JavaScript的组成



# 2019.3.21

### 关于<script>的引用与属性

**<head>**

**<title> script脚本测试 </title>**

**<script defer="defer" type="text/javascript" src="test.js" ></script> <!-- 外部引入式写法 -->**

**<!--**

**关于script属性:**

**defer="defer": 等加载完毕html文档在解析js脚本( 外部引用有效 )**

**async="async": 异步解析js脚本,并加载html页面( 外部引用有效,有此属性不要修改DOM )**

**type="text/javascript": 约定必备**

**src="js路径": 引入外部js脚本,可本地,可远程( 有src属性时,内部嵌入内容会忽略,只执行外部引入内容 )**

**-->**

**<script type="text/javascript" > // 内部写入方式**

**alert("233");**

**</script>**

**</head>**

#### 关于CDATA避免XML解释器在“<”和”&”出现错误

**<!-- <![CDATA[ 代码内容 ]]> 防止"<"和"&"在XML解释器,内入式js代码出现错误 -->**

**<!-- 如果浏览器不支持<![CDATA[ ]]>,则注释掉如下( 即可支持所有浏览器 ): -->**

**<script type="text/javascript" >**

**//<![CDATA[**

**function abs(a,b){**

**if(a < b){**

**alert("hello");**

**}**

**else{**

**alert("NONO");**

**}**

**}**

**abs(1,2);**

**//]]>**

**</script>**

### 如果浏览器不支持js

#### 隐藏js代码，如果浏览器不支持js则会显示出内入式的js代码，影响html的内容排序

**<title> script脚本测试 </title>**

**<!-- 如果浏览器不支持js,隐藏js代码方式为( 用html方式注释掉 ): -->**

**<script type="text/javascript" >**

**<!--**

**function abs(a,b){**

**if(a < b){**

**alert("hello");**

**}**

**}**

**abs(1,2);**

**//-->**

**</script>**

#### <noscript>不支持/禁用 - js进行提示

**<!--**

**关于<noscript></noscript>**

**如果浏览器:**

**1. 不支持js**

**2. 禁用js**

**满足以上1点,则<noscript>可以让用户看到一条消息**

**正常情况下,用户是无法看到noscript信息的**

**-->**

**<noscript>**

**<p>**

**此页面需要js支持,(启用)js.**

**</p>**

**</noscript>**

# 2019.3.22

### // 关于js的严格模式

**function test0(){**

**"use strict" // 严格模式,尊循ES3**

**var massage; // 可以赋给任何类型数据**

**alert(massage);**

**}**

### 关于全局变量与局部变量

**// 全局变量与局部变量**

**var mas0; // 全局变量**

**function test(){ // function定义函数**

**var mas1=101,mas2="hello"; // 局部变量**

**mas3 = 100; // 全局变量**

**}**

**test(); // 函数调用**

### typeof操作符( 非函数 ),可以判断数据类型

**// typeof操作符( 非函数 ),可以判断数据类型**

**/\***

**\* undefined: 未定义值,变量为赋值**

**\* boolean: 布尔值**

**\* string: 字符串类型**

**\* number: 阿拉伯数字**

**\* object: 对象或null**

**\* function: 函数类型**

**\*/**

**var a = "he",b;**

**alert(typeof a); // typeof是操作符,非函数,所以不加括号也可以**

**alert(typeof(a)); // 当然加括号也可以**

**alert( typeof null ); // object类型( Safari 5/Chrome7 之前浏览器会返回function,其它浏览器返回object )**

**alert( typeof b ); // undefined类型**

### // 关于null与undefined的关系

**// null为一个空指针**

**alert(typeof null); // 返回object对象**

**alert(null == undefined); // true,注意虽然他们有这个关系,但是应用方向却大不同**

#### // 布尔类型，Boolean()函数可以将变量转为布尔类型值

**var mes = "hello";**

**alert("---")**

**alert( Boolean(mes) ); //** **Boolean()函数可以将变量转为布尔类型值**

**/\***

**\* 注意:布尔类型 "true/false" 不等于 "1/0"**

**\* object true/任何对象 false/null**

**\* undefined 无真,只与false**

**\*/**

**if( mes ){ // 可以这样使用**

**alert("mes为真");**

**}**

#### // Number类型

**// 关于8进制**

**/\***

**\* 8进制以0开头,如果超过(0~7)则变为10进制,会忽略开头的0**

**\* 在严格模式下,8进制写法会报错**

**\*/**

**var octalNum0 = 06; // 8进制,0开头**

**var octalNum1 = 08; // 10进制,因为超出(0~7)**

**// 关于16进制**

**/\***

**\* 16进制0x开头(0~9/A~F)**

**\* 字母不区分大小写**

**\*/**

**var hexNum0 = 0xA; // 16进制,0x开头**

**var hexNum1 = 0xa; // 二者相同,不区分大小写**

#### // 浮点数类型

**/\***

**\* 如果点后,为0则依然认为是整数类型**

**\* 特别注意: 0.1+0.2 == 0.30000000000000004 // 这是个舍入误差,很多编程语言都存在这个问题**

**\* 0.12+0.3 == 0.42 // 这个到是没有问题**

**\*/**

**var floatNum0 = 10.0; // 依然会被认定为int类型"10"**

**// 不推荐写法**

**var floatNum1 = 1.;**

**var floatNum2 = .1;**

**alert(floatNum2); // 0.1**

**// 关于e/E计算方法**

**/\***

**\* e与E相同,不区分大小写**

**\* 例: 3.12e4 == 3.1200.0**

**\* 3.12e4 相当于: 3.12\*10^4 == 3.1200.0**

**\*/**

**var floatNum3 = 3.12e4; // 3.1200.0**

**var floatNum4 = 3e-3; // 数学计算, 3\*10^(-3) == 0.003**

**alert( floatNum4 ); // 0,003**

### // 数值范围，isFinite()查看值是否在最大最小值内

**/\***

**\* 最大值 Number.MAX\_VALUE == 1.7976931348623157e+308**

**\* 最小值 Number.MIN\_VALUE == 5e-324**

**\* isFinite() 查看数是否在最大最小范围内,返回布尔类型**

**\* 如果值超出范围,则数值变为正/负Infinity( 正负无穷 )**

**\*/**

**var result = Number.MAX\_VALUE + Number.MAX\_VALUE; // Number.MAX\_VALUE保存着最大值**

**alert( isFinite(result) );** **// isFinite()查看值是否在最大最小值内**

**alert(result);**

**alert( Number.NEGATIVE\_INFINITY ); // Number.NEGATIVE\_INFINITY/Number.POSITIVE\_INFINITY == -Infinity/+Infinity**

### // 非数值NaN

**/\***

**\* NaN为非数值**

**\* 数值/0 返回NaN,保证不出错**

**\* NaN/10 == NaN // NaN与如何数计算都等于NaN**

**\* NaN != NaN // 甚至它都不等于他本身**

**\*/**

**alert("----");**

**var a=10,b=0;**

**alert(a/b); // 默认情况是NaN,Chrome则为Infinity**

**alert(NaN/10); // 依然为NaN,NaN除任何数都为NaN**

# 2019.3.28

### // isNaN() 判断是否为非数值,返回( 非数值为真,数值为假 )

**var a = 10, b = true, c = "abv",d = "123" ;**

**alert(isNaN(a)); // false**

**alert(isNaN(b)); // false: 布尔值,可以转换为1/0**

**alert(isNaN(c)); // true: 字母或符号字符串,非数值**

**alert( isNaN(d) ); // false: 字符串数字,则也可以转换为数值**

### // 数值转换,Number(),parseInt(),parseFloat()

**var num1 = true, num2 = null;**

**var str0 = "", str1 = "012", str2 = "0xa", str3 = "13abc", str4 = "abc";**

**var str5 = 12.3,str6 = 1.2e3;**

#### **// Number()转换数字,有很多规矩,一般不推荐使用**

**// 非字符串类型**

**// 可以字符串-转-整数类型-或-浮点类型**

**alert(Number(num1)); // 1: 布尔类型转换为1/0**

**alert(Number(num2)); // 0: null空类型返回0**

**// 字符串类型**

**alert(Number(str0)) // 0:空字符串返回0**

**alert(Number(str1)) // 12:会忽略前数字的0**

**alert(Number(str2)) // 10:会16进制值-转换- 对应的10进制值**

**alert(Number(str3)) // NaN:非纯数字会认定为非数值**

**alert(Number(str4)) // NaN:非数值**

#### **// parseInt( 数据,进制类型 ) 转换为整数类型,推荐使用,更加合理化**

**var num1 = true, num2 = null;**

**var str0 = "", str1 = "012", str2 = "0xa", str3 = "13abc", str4 = "abc";**

**var str5 = 12.3,str6 = 1.2e3;**

**// 1. 无数字,都为NaN**

**// 2. 开头数字解析,非数字停止解析**

**// 非字符串类型**

**alert(parseInt(num1)) // NaN:非数值**

**alert(parseInt(num2)) // NaN**

**// 字符串类型**

**alert(parseInt(str0)) // NaN**

**alert(parseInt(str1,10)) // 12:认定内容为10进制,忽略0,输出10进制数值**

**alert(parseInt(str2,16)) // 10:认定内容为16进制-转换-10进制数值**

**alert(parseInt(str3,10)) // 13:认定为10进制,开头遇到数字则继续解析,遇到非数字则停止解析,输出10进制数值**

**alert(parseInt(str4)) // NaN:非数值**

#### **// parseFloat( 数据 ) 转换浮点类型**

**// 1. 与parseInt()规矩差不多,只不过无指定进制参数**

**alert(parseFloat(str5)) // 12.3**

**alert(parseFloat(str6)) // 1200 // 科学计算法也没有问题**

### **// Object类型**

**// 目前解析: 一组数据和功能的集合**

**var a = new Object(); // 创建Object类型**

#### **// 操作符**

**// 目前解析: 操作数据,如+-\*/,关系操作符**

**// 操作对象会调用valueOf()/toString()方法,方便获取值( valueOf()此方法不太理解 )**

### **// 字面量**

**// '\b': 空格**

**// '\r': 回车**

**// '\x': 16进制数**

**// '\u': Unicode字符**

**var str0 = "\u03a3"; // Unicode字符码**

**alert(str0);**

#### **// .length字符串长度,注意: 如果有双字节字符,长度测量会不准确**

**var str1 = "abc";**

**alert(str1.length); // 3:字符串长度数量**

#### **// 字符串相加**

**var str2 = "java";**

**str2 += "script"; // 字符串相加**

**alert(str2);**

### **// 数据.toString( 指定转换进制 )方法, String( 数据 )函数,字符串转换**

**// toString()方法可以指定转换进制**

**// 1.null类型时,toString()会报错**

**var str3 = 10, str4 = null, str5 = true;**

**alert(str3.toString()); // "10":直接转字符串类型**

**alert(str3.toString(16)); // "a":数据-转换-为指定进制,转变为字符串类型**

#### **// String()靠谱的字符串转换,**

**// 他与toString()区别在与,**

**// 1. 没有办法指定进制转换,**

**// 2. 但是String()兼容性更强,他可以兼容非实数类型转换成字符串类型**

**alert(String(str4)); // "null":字符串类型**

**alert(String(str3)); // "10":直接转不解释**

# **2019.3.30**

### // 一元操作符

// 只能操作一个值,叫一元操作符

**// --age/++age形式**

**var age = 29;**

**var other = --age + 2; // 先执行age = age -1; 在执行 other = age + 2;**

**alert( other ); // 30**

**alert( age ); // 28**

**// age++/age--形式**

**age = 29;**

**other = age-- + 2; // 先执行 other = age + 2; 在执行 age = age -1**

**alert( other ); // 31**

**alert( age ); // 28**

#### **// 一元操作符(++/--),javascript不严谨性计算( 个人强烈不推荐计算 )**

**// 1. 字符串类型++ ( 字符串转换成实数进行数值实数( 有字母或其它符号,则返回NaN类型 ) )**

**// 2. 布尔类型++ ( 会把布尔类型当成1/0进行数值计算 )**

**// 3. 浮点型++ ( 正常的数值计算,不过注意舍入BUG )**

**// 4. Object类型++**

**// 例:**

**// 字符串++;**

**var a1 = "2";**

**a1++;**

**alert(a1); // 3: 被转成实数类型**

**alert(typeof a1);**

**// 布尔类型++**

**var a2 = true;**

**a2++;**

**alert(a2);**

**// Object类型++**

**var o = {**

**valueOf: function(){**

**return -1;**

**}**

**}**

**o++;**

**alert(o);**

#### // 一元加减符(+/-)说白了就是数学里的正负号

**// 1. 一样有上方一元操作符计算不严谨性**

**// 2. 区别在与遇到实数时,就相当于+/-号处理**

**//例:**

**var a3 = "1";**

**alert(+a3); // 3:被转换成了实数**

**alert(typeof +a3);**

# 2019.4.2

### // 32位存储-转-二进制码

// 默认32位,有一位表示正/负

// 0表示正数,1表示负数,所以32位真正存储数字的有31位

**var num = -18;**

**alert( num.toString(2) ); // 更加人性化的转换显示**

### **// 按位对比变换**

**// 总体围绕32位二进制数进行推演**

#### **// 1. 按位非(NOT)**

**// 1变0,0变1**

**// 如:**

**var num2 = 25; // 二进制: 00000000 00000000 0000000 00011001**

**var num3 = ~num2 // -26: 二进制: 11111111 11111111 11111111 1100110**

**alert(num3);**

#### **// 2. 按位与(AND)**

**// 变换规则:**

**// 1 : 1 = 1**

**// 1 : 0 = 0**

**// 0 : 0 = 0**

**// 如:**

**var num4 = 25 & 3;**

**alert(num4);**

**// 25: 00000000 00000000 00000000 00011001**

**// 3: 00000000 00000000 00000000 00000011**

**// 对比后: 1: 00000000 00000000 00000000 00000001**

#### **// 3. 按位或(OR)**

**// 变换规则:**

**// 1 : 1 = 1**

**// 0 : 1 = 1**

**// 0 : 0 = 0**

**// 与上对比变换类似**

**Var num5 = 25 | 3;**

#### **// 4. 按位异或(XOR)**

**// 变换规则:**

**// 1 : 1 = 0**

**// 0 : 1 = 1**

**// 0 : 0 = 0**

**// 规则,二位比较中只有一个1时才返回1**

**var num6 = 25 ^ 3;**

**alert(num6);**

# **2019.4.3**

### // 位移符号( 直接操纵二进制位数 )

// 左移(<<),右移(>>):

// 注意: <</>>对正负没有影响,因为他们真正只操控31位,正负位不进行操纵

// 无符左移(<<<),无符右移(>>>):

// 注意: <<</>>>直接操控32位,因此会影响负数差距会非常大,不过无符位数用负数,也是他喵的神经病

**// 左移( << )**

**var test = 123;**

**alert(test.toString(2)); // 123 : 0 0000000 00000000 00000000 01111011**

**var test1 = test << 5; // 左位移**

**alert(test1.toString(2)); // 3936 : 0 0000000 00000000 00001111 01100000 :可以发现,在"左移"在末尾插入了5位0**

**alert(test1); // 3936**

**// 右移( >> )**

**var test2 = 12;**

**alert(test2.toString(2)); // 12 : 0 0000000 000000000 000000000 00001100**

**var test3 = test2 >> 3; // 右位移**

**alert(test3.toString(2)); // 1 : 0 0000000 000000000 000000000 00000001 :可以发现,在"右移"在开头插入3个0**

**alert(test3); // 1**

#### **// 无符左/右移动,直接操控32位,原理于上方类似**

**// 无符左移<<<**

**// 无符右移>>>**

**Var test4 = 123;**

**Var test5 = test4 >>> 3; // 直接操控32位，在开通加3零在结尾去除超出位数，即是位移结果**

### // 布尔操作符

#### // 逻辑非 ! --- 其实就是真变假,假变真

其实就是真变假,假变真

#### // 逻辑或 ||

// a || b

// 1. 如果第一个数为真则不会继续执行,与C语言逻辑或几乎相同

#### // 逻辑与 &&

// a && b

// 1. 如果第一个数为假则不会继续执行,与C语言逻辑与几乎相同

#### // 乘/除性操作符( 其实就是" \*,/ "数学的乘号 )

// 1. 正常数值计算相乘/除

// 2. 如果一个操作数是非数值,则在后台调用Number()函数转换为数值后在进行计算

// 3. 关于Infinity细节看原书p66

#### // 模( % )

// 1. 左边值 < 右边值 返回 左边值

**alert( 5%23 ); // 5**

// 2. 其实求的是余数( 与C语言的%相同 i)

**alert( 23%5 ); // 3**

// 3. 遇到非数值先使用Number()转换为数值,在进行计算

// 4. 关于Infinity细节看原书p67

#### // 加性操作符（优先转为字符串进行拼接）

// 1. 字符串+字符串 = 进行字符串拼接

// 2. 非实数字符串+数值 = 数值转换字符串，进行字符串拼接

// 3. 如果操作的是对象,或者布尔类型等非数值,则后台会进行toString()/String()转换为字符串在进行字符串拼接

// 4. 实数字符串+数值 = 字符串转换为数值进行计算

**// 如:**

**var a = 1,b = 'b',c='c',d=true,e=’2’;**

**alert(b+c); // "bc": 字符串拼接**

**alert(a+b); // "1b": 先将数值转换为字符串在进行字符串拼接**

**alert(d+b); // "trueb": 非数值类型先进行转换,在进行字符串拼接**

**alert(a+e); // 3: 实数字符串+数值 = 字符串转换为数值进行计算**

#### // 减性操作符（优先转为数值进行计算）

**看实例：**

**var a = 'a',b='2',c='1',d=5;**

**alert(a-b); // NaN: 非数值**

**alert(c-b); // -1: 字符串转数值进行计算**

**alert(d-b); // 3: 字符串转数值进行计算**

# **2019.4.7**

### **// 关系操作符( >,>=,<,<= )**

**// 关系操作符返回布尔值**

**// 并且拥有自动转换数据类型功能**

**// 比较对象时,对象先调用valueOf()或toString()函数进行数值转换,在进行比较**

**// 例:**

**var a = 1,b = 2,c = '3',d = 's',e = 'A';**

**a < b // true: 数值正常比较**

**d > e // true: 双方是字符串时,比较ascii码值大小**

**b > c // false: 数值与实数字符串比较,字符串先转为数值在进行比较**

**a < d // false: 数值与非实数字符串比较,字符串转为NaN类型( 与NaN比较,无论什么情况返回false )**

### **// toLowerCase()方法,字符串,大写转为小写形式**

**alert("ABCaaa".toLowerCase()); // "abcaaa"**

### **// 相等操作符( !=,==,===,!== )**

**// 相等操作符同样返回布尔类型**

**// 并且拥有,自动转换数据类型的功能( 除===,!== )**

**// 注意:**

**// NaN与如何数据类型比较都返回false**

**// 布尔类型会转为1/0**

**// 在进行比较时强烈推荐使用===或!==( 推荐使用 )**

**// 例:**

**alert( "5"==5 ); // true: 自动转换类型**

**alert( "5"===5 ); // false: "==="比较数值和类型是否一直,显然二数类型并不一致**

**alert( "5"!=5 ); // false: 因为字符串先自动转换为数值在进行比较**

**alert( "5"!==5 ); // true: 类型不一样**

### **// 条件操作符( 其实就是"C语言里的三元表达式" )**

**var b1,b2=true;**

**b1 = b2 ? 2 : 3; // 如果b2为真,2赋值给b1,否则3赋值给b1**

**alert(b1); // 2**

### **// 赋值操作符**

**// 快捷写法:**

**// 这种写法适用与:**

**// \*=**

**// /=**

**// %=**

**// -=**

**// +=**

**// <<=**

**// <<<=**

**var num = 10;**

**num += 5; // num = num + 5;**

**alert(num); // 15:**

### **// 逗号操作符**

**var a=1,b=2; // 逗号表达式可用于多个变量赋值**

**var num = (1,2,3,4); // 4: 逗号表达式会将最后值赋给变量**

**alert(num); // 4**

## **// 语句( 和C语法几乎差不多 )**

### **// if语句( 老盆友了不多说 )**

**var i=0;**

**// 单行写法( 不推荐可读性差 )**

**// if( i>0 ) alert("yes") else if(i==0) alert("0") else alert("no")**

**// 规范写法**

**if( i>0 ){**

**alert("yes");**

**}**

**else if( i==0 ){**

**alert("0");**

**}**

**else{**

**alert("no");**

**}**

### **// do while语句( 与C几乎差不多 )**

**var s = 0;**

**do{**

**s++;**

**}while( s<10 );**

**alert(s);**

### **// for语句( 老朋友不多说 )**

**var i,count = 3;**

#### **// 规范写法**

**for(i=0;i<count;i++){**

**alert(i);**

**}**

#### **// 无限循环写法**

**/\***

**for(;;){**

**// 内容**

**}**

**\*/**

#### **// for也可以代替while写法**

**i = 0;**

**for(;i<conut;){**

**i++;**

**}**

#### **// for-in 用法( 可以迭代出对象属性,用法有点像python的for-in语句 )**

**// 1. 对象属性是无序的,因此每一次迭代出的属性是随机幻化的,但每一个属性都会被迭代出一次**

**// 2. 如果迭代出的属性变量值是null或undefined,则会被报错,ES5标准则会停止迭代**

**/\***

**for( var propName in window ){ // 迭代出window对象所有属性名,并赋值给propName变量**

**document.write(propName); // 在浏览器中写出变量内容**

**document.write("<br>"); // 写换行符**

**}**

**\*/**

#### **// label语句( 跳出循环使用 )**

**// 通常与for循环配合**

**// break label 或 continue label方法使用**

**// 注意: break label 直接结束目标for的循环**

**// continue label 跳到标签点继续执行**

**// 例:**

**/\***

**var i,a,b;**

**labelEnd: for(i=0;i<5;i++){**

**alert("1-1-1");**

**labelState: for(a=0;a<3;a++){ // label标签用法( 标签点 )**

**alert("2-2-2");**

**for(b=0;b<2;b++){**

**alert("3-3-3");**

**continue labelState; // 直接跳出到标签点,然后继续执行**

**}**

**}**

**break labelEnd; // 结束labelEnd标签点匹配的for循环**

**}**

**// labelEnd 结束后,其它语句开始执行位置**

**alert("end!");**

**\*/**

**// break,continue 功能与C语言语法功能一样**

### **// with语句**

**// 主要作用是将,作用域设置到指定对象中**

**// with会使代码性能降低,不推荐使用**

**// with也不方便调试,并且with无法在严格模式使用,会直接报错**

# 2019.5.11

### 调用[Unicode](https://baike.baidu.com/item/Unicode" \t "_blank) 码

var a = "\u00e8"; // 调用u码

alert(a);

### // 关于switch的用法

**// 1. js的switch于c的功能用法几乎差不多,**

**// 唯一的不不同点在于,它可以判断,更多的数据类型**

**// 2. 可以代替多个if else if判断**

**// 3. 二个case之间如果不写break一定要写注释**

**// 4. switch使用的===全等符判断，所以不会转换数据类型( 我喜欢 )**

**var test = "233";**

**var test2 = 233;**

**switch( 123 ){**

**case "123":**

**alert("123");**

**// 这里故意不写break**

**case test2: // 支持多种类型判断,甚至是变量**

**alert(test2);**

**break**

**default: // 经证实,default没有break依然会往下执行**

**alert( "都没有" );**

**case "233":**

**alert(test);**

**break**

**}**

### **// 关于函数的用法**

**// 1. 老朋友不多解释**

**// 2. 要么有返回值，要么不写返回值**

**function abs( name,mes ){**

**alert(name+mes);**

**return false;**

**}**

**abs("zhao","233");**

#### **// 函数的参数调用**

**// 1. js因为没有函数签名,所以重载,只能进行模仿,**

**// 如检测传入的参数数量,进行判断执行**

**// 2. 其实这些插入的参数是以数组形式传送的,所以即使超出所需值，也可以通过arguments调用**

**// 3. 非严格模式下,通过arguments数组可以改变参数,严格模式下将无法改变,默认值为undefined**

**// 4. 核心: arguments**

**function argu( name,phone){**

**alert( arguments[0]+arguments[1] ); // 索引对应验证**

**arguments[2] = "OK!"; // 非严格模式甚至可以改变传入的参数**

**alert(arguments[2]);**

**alert( arguments.length ); // 查看输入参数数量( 使用length方法测取数组长度 )**

**}**

**argu("zhao",1385,1069);**

#### **// 重载模拟**

**// 1. js是默认是没有重载的，不过可以模拟出来**

**// 2. 何为重载?**

**// 重载，简单说，就是函数或者方法有相同的名称，但是参数列表不相同的情形，这样的同名不同参数的函数或者方法之间，互相称之为重载函数或者方法。**

**function cz( ){**

**if( arguments.length == 1 ){ // 传入1个参数时调用**

**alert( arguments[0]/2 );**

**}**

**else if( arguments.length == 2 ){ // 传入2个参数时调用**

**var result;**

**result = arguments[0] + arguments[1];**

**alert( result );**

**}**

**}**

**cz(22);**

**cz(22,22)**

# **2019.5.14**

## **// 变量与作用域和内存问题**

**// 0. 基本类型: undefined,null,boolean,number,string 5个基本类型**

**// 1. 引用类型: object(),引用类型不允许直接访问内存,在操作过程中为引用访问**

**// 2. new Object() 创建对象**

### **// 0. 引用类型可以被赋予属性,而非引用类型则不可以**

**var person = new Object(); // 引用类型**

**person.name = "Zhao";**

**var dog = "Li"; // 基本类型**

**dog.color = "pink"; // 虽然不会出错,但是确实没什么用处**

**alert( person.name ); // "Zhao"**

**alert( dog.color ); // "undefined"**

#### **// 1. 基本类型,变量复制( javascript特立独行 )**

**var test1 = 250;**

**var test2 = test1; // 在javascript中,这样已经算是复制了！！！( C语言可与它不一样 )**

**alert(test2); // 250**

**test1 = 300; // 上放因并非指针指向,已经属于复制**

**alert(test2); // 250**

#### **// 2. 引用类型,变量复制**

**var test3 = new Object();**

**test3.num = 15;**

**var test4 = test3; // 其实并没有真正的复制,只是指针指向**

**alert(test4.num);**

### **// 3. 传递参数**

**// 0. 按值传递( 真正的复制 )**

**// 1. 引用类型值传递( 指针引用 )**

**// 2. 参数只能'按值传递'**

#### **// 4. 按值传递与局部变量**

**function addnum( num ){**

**num += 10; // 局部变量( 老相识，不必多解释 )**

**return num;**

**}**

**var num = 10;**

**var result = addnum( num );**

**alert(num); // 10**

**alert(result); // 20**

#### **// 5. 伪引用类型值传递，全局变量,在局部变量的反应**

**// 0. 其实,全局变量可以在局部中使用，**

**// 1. 但是局部变量不可以在全局使用,函数一旦结束执行，局部变量也会立刻销毁**

**function setName( obj ){**

**obj.name = "TENG"; // 函数的局部环境可以使用'全局变量'**

**obj = new Object(); // 其实创建的是局部变量**

**obj.name = "TTTT"; // 因为局部变量，在函数结束执行后会立刻销毁**

**}**

**var obj = new Object();**

**setName(obj);**

**alert(obj.name); // "TENG"**

### **// 6. 检测,引用类型/基本类型**

**// 0. instanceof 专门用来检测引用类型,能精确到引用类型什么形式**

**// 1. 引用类型返回true**

**// 2. 基本类型返回false**

**var test5 = new Object();**

**var test6 = 15;**

**var test7;**

**alert( test5 instanceof Object ) // true**

**alert( test6 instanceof Object ) // false**

**alert( test7 instanceof Array ) // false**