# 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) 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**

# 2019.7.29

## Udemy-js-学习

### // 0. js-使用键盘输入字符串 - prompt()默认键入为string类型

var Q = prompt( 'what is his last name?' );

alert( typeof(Q) ); // 键入值为String类型

console.log( Q );

### // 1. 大于小于符号演示演示

var a=1,b=2;

console.log( a>b );

console.log( typeof( a>b ) );

### // 2. if 语句练习

age = 12;

if( age<18 ){

console.log(" 禁止访问 ");

}

else if ( age>=18 && age <30 ){

console.log(" 访问成功 ");

}

else{

console.log(" 查无此页面 ");

}

### /\*\* 3元表达式 \*\*/

// 一遍用于,简单的判断赋值，不适于复杂的判断

// 使用方法:

// 条件 ? 真,执行此语句 : 假,执行此语句;

var drinks;

// var age = prompt(" 你的年龄? ");

age >= 18 ? drinks = "wine" : drinks = "joice";

console.log( "ZTaer drinks " + drinks + "!" );

### /\*\* switch \*\*/

// 适用于,避免使用多个if else,使程序简洁性

// js的switch很自由,case的条件按照if的判断条件来就好

var man;

switch( true ){

case age < 18 && age >= 16:

man="青少年";

break; // 没有break后果很严重,会不经case处理就执行其他语句

case age >= 18:

man = "成年人";

break;

case age == "21": // case 也可以多条配合,来执行同一条语句

case age == "二十一":

man="21岁";

break;

default: // 如果条件都不符合会执行此语句

man = "你猜一猜";

break;

}

console.log( "ZTaer 已经是 " + man + "!" );

### 获取数组长度，.length

for( i=0; i < join.length i++ ){ // 获取数组长度

var ave = sum\_ave( join[i] )[1]; // 获取数组平均值

if ( ave > winner ){

winner = ave;

pos = i; // 记录最大数位置

}

}

### /\*\* 关于数组 \*\*/

#### // 创建数组

var name = ['abc','sss'];

var year = new Array( '2018','2019' );

#### // 操控数组元素,添加数组元素,删除数组元素,确定数组元素位置

year[ year.length ] = '2020'; // 在末尾添加元素,length可以确定数组中的数量

year.push('8848'); // 在数组末尾添加

year.unshift( 'ztaer' ); // 在数组开头添加元素

console.log(year);

year.pop(); // 删除数组末尾元素

year.shift(); // 删除数组开头元素

console.log(year);

console.log( year.indexOf('2019') ); // 确定数组元素位置，如果数组中无目标元素,则返回"数字类型-1"

### /\*\*\* Object对象 \*\*\*/

#### // 创建对象 - 方法1

var ztaer = {

year: 18,

name: 'oo7',

skills: [ 'h5/css3','js','python3','linux' ],

gf: false

};

var oo7 = new Object(); // 创建对象 - 方法 2

#### // 索引对象属性

console.log( ztaer.year );

console.log( ztaer[ 'name' ] );

console.log( ztaer.skills[1] );

#### // 修改对象属性

ztaer.year = 20;

ztaer['name'] = '\_\_OO7\_\_';

console.log( ztaer );

#### // 添加属性给对象

oo7.name = 'ztaer';

oo7['sex'] = 'man';

console.log( oo7 );

### /\*\*\*\*\* 函数 \*\*\*\*\*/

#### // 函数的创建 - 第一种

function test( x ){

return x;

}

#### // 函数的创建 - 第二种（ 函数表达式 ）

test2 = function( y ){

return y;

}

console.log( test(2) + ' ' + test2(3) );

#### /\*\*\*\*\* object关于this使用，以及对象中函数创建 \*\*\*\*\*/

// object的函数创建其实依据第二种函数创建方法

// this使用:

// a) this.变量，可以调用自身对象属性

// b) this.变量。 也可以创建自身属性

var boy = {

birthYear: 1998,

calcAge: function( year ){ // 创建函数

// this 可以调用自身已有属性，也可以创建新属性

this.age = year - this.birthYear; // 创建新属性

return this.age;

}

}

// 对象函数调用

console.log( boy.calcAge( 2019 ) );

### 关于.toFixed()保留小数,返回字符串类型

// .toFixed(2) 四舍五入保留2位小数,返回字符串类型结果，所以计算时需在转换为数字类型

var userMark = {

name:'Mark',

cale: function( mass,height ){

this.mass = mass;

this.height = height;

this.BMI = +( mass / height \*\* 2).toFixed(2) ; // .toFixed()保留二位小数返回字符串结果，+号字符串转为数字类型数据

return this.BMI;

}

}

/\*\*\*\*\*\*\* MAIN \*/

var winner,BMI;

BMI\_Mark = userMark.cale( 72,1.8 ); // 调用对象属性函数

B

## 融合，数组，对象，最近所学函数实战看，2019.7.28/home.js

## /\*\*\*\* 几种循环，以及continue,break \*/

### // for循环/while与C无异

for( var i=0; i<10; i += 2 ){ // 花式输出偶函数

if( i > 5 ) break; // break与C无异

console.log(i);

}

### // while,按照C正常使用即可，continue/break同样如此

var group = [ '1','2',3,'4','5' ];

var i = 0;

console.log( typeof( group[2] ) )

while( i < group.length ){

if ( typeof group[i] !== 'number' ){ // typeof 验证数据类型并返回的是字符串类型数据

i++;

continue

}

else{

console.log( group[i] );

break;

}

}

## 堆栈的工作顺序



# 2019.8.2

## // 0. 函数与函数表达式 - 工作方式

// a) 函数与函数表达式工作方式是有区别的

// b) 函数声明会直接存放在堆栈中，是全文可以调用，但函数表达式，必须创建之后使用

### // 函数工作方式:

// 函数声明会直接存放在堆栈中，是全文可以调用，但函数表达式，必须创建之后使用

console.log( abs(5) );

function abs( x ){

return x\*\*2;

}

### // 函数表达式工作方式:

// console.log( avg(1,2) ) // 不可用，因为“变量”影响工作方式必须按照顺序执行

var avg = function( x,y ){

var result = ( x+y )/2;

return result;

}

## // 1. 变量工作方式

// a) 变量在全文中有定义但是顺序不正确，默认会赋值变量为undeifne，

// b) 如果是全文中都没用定义, 去调用则会出现错误

console.log(s); // 虽然顺序不正确，但确实有定义此变量，所以会返回undefined

var s = 5;

// console.log(bb); // 此变量没有定义，后返回错误的信息

## // 2. 关于this的细节

// a) 在对像中，this默认无法进行但级函数访问属性，最多只能一级函数访问，如下案列

// b) 其它变量 = this; this因规则受限无法多级函数使用，但是把他赋值给其它函数即可进行多级属性访问,如下案列

jone = {

name:'jone',

tel: 123456,

cale: function(){

console.log( this.name ); // 可正常访问

var that = this; // 将this赋值给that及时闭包依然可以代替this可以进行多级函数，直接属性访问

function test2(){

console.log( this.name ); // 无法访问,因为this的调用是有限制的，this只能进行一级访问。

console.log( that.tel ); // 正常访问，因为that变量单独已被this赋值，this是受默认规则的影响无法进行多级函数属性访问,但其它变量不受限。

}

test2();

}

}

jone.cale();

### // 3. 关于对象的细节，对象间借用函数

var zhao = {

name:'zhao',

cale: function(){

console.log(this.name);

}

}

var li = {

name:'zhao',

cale: zhao.cale() // 内借用

}

li.cale2 = zhao.cale; // 外借用

li.cale; // 调用借来的函数( 正常调用 )

## DOM操作 – 2019.7.29实战的游戏而总结

### /\*\*\*\*\*\*\*\*\*\*\*\* es5-生成随机数，以及转为整数

#### // 0. 生成0~1随机数( 与乘法配合生成其它随机数 )

Math.random();

#### // 1. 非四舍五入取整数

Math.floor();

// 例如:

var ran = 0;

ran = Math.floor( Math.random() \* 6 + 1 ); // 生成1~6的随机数

console.log(ran);

\*/

### /\*\*\*\*\*\*\*\*\*\*\*\* es5-document.querySelector()HTML标签操控

#### // 0. 选中标签

document.querySelector(' #score-0 '); // 以CSS选择器选中即可

#### // 1. 改变标签内容

// a) 改变标签文本内容: <a id="score-0" > 文本内容 <a>

document.querySelector(' #score-0 ').textContent = ran ;

// b) 在标签中插入HTML代码: <a id="score-0" > <b> $ran </b> </a>

document.querySelector('#score-0').innerHTML= '<b>' + ran + '</b>' ;

#### // 2. 获取标签内容

// a) 获取标签中的文本内容( 字符串类型 )

var test0 = document.querySelector('#score-0').textContent;

console.log( test0 );

#### // b) 获取标签中，HTML标签

var test1 = document.querySelector('.player-0-panel').innerHTML;

console.log( test1 );

#### // 3. 改变标签中CSS样式

// a) 其实核心document.querySelector('CSS选择器').style.css属性 = 'css样式参数';

document.querySelector('.dice').style.display = 'none';

#### // 4. 改变标签属性 - 改变图片实例

// a) 选中的标签.属性 = '修改值';

var choose\_img = 3;

var dice\_img = document.querySelector('.dice');

dice\_img.src = 'dice-' + choose\_img + '.png'; // 核心

\*/

### /\*\*\*\*\*\*\*\*\*\*\*\*\* es5-.addEventListener()监听事件，并执行函数

// 监听事件参数参考网站: <https://developer.mozilla.org/zh-CN/docs/Web/Events/>

// 目前发现，他只能监听按钮

#### // 0. 正常调用函数 - 监听单机事件,执行test0()函数

function test0(){

var dice = Math.floor( Math.random() \* 6 + 1 ); // 随机数

document.getElementById('score-0').textContent = dice; // 抓取ID标签

}

var btn\_roll = document.querySelector('.btn-roll').addEventListener('click',test0);

#### // 1. 调用匿名函数( 匿名函数就是没有变量名称的函数 ) - 监听单击时间，并执行函数

// a) .addEventListener('监听事件', function(){ 函数内容 } );

var btn\_roll = document.querySelector('.btn-roll').addEventListener('click', function(){

var dice = Math.floor( Math.random() \* 6 + 1 ); // 随机数

document.getElementById('score-0').textContent = dice;

} );

\*/

### /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* es5-.getElementById('test')抓取ID ，选中ID

// 0. 抓取ID, 抓取速度比.querySelector()更快，但是功能单一不方便

document.getElementById('test');

\*/

### /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* es5-.classList标签多类名操控，删除，添加，切换HTML标签类名

#### // 删除指定类名,多个类名用逗号隔开

document.querySelector('.player-0-panel').classList.remove('active','test','test1');

#### // 添加类名,多个类名同样用逗号隔开

document.querySelector( '.player-0-panel' ).classList.add('active','name');

#### // 切换类名，所为切换: 标签含有类名时删除,无类名时添加

document.querySelector('.player-0-panel').classList.toggle('active');

\*/

## Js-初始化规则

Js的初始化参数一遍用init()函数来命名

# 2019.8.5、

## // 继承的目的，以及使用概念（ 理论概念 ）

// 0. 目的是减少重复代码

// 1. 使用方法

// a) 创建一个蓝图函数，命名开头大写，易于区分蓝图对象于普通对象

// b) 蓝图的索引用调用，让新建的对象继承蓝图中的属性

## // 让对象拥有继承函数/属性成为可能的prototype( js的基本继承思路 )

// 原因( js中除原语外，万物皆对象 ):

// 0. 每一个建立的Object对象/函数都拥有prototype属性,所以我们才能在对象中调用一些函数和方法

// 1. 同理如果我们自定义的函数也加入到prototype中，在对象中同样可以调用我们自定义的函数

// 2. 对象在调用函数时，首先会在自身中搜索函数，如果没用则会去prototype中查找函数并调用，如果prototype中也没用则返回undefine

// 用于:

// 0. 常用于，与蓝图配合增加函数

// 1. 当然也可以增加属性，但是并不常用

## /\*\*\*\*\*\*\*\*\*\* 蓝图 - 对象继承使用方法,以及prototype使用方法

### // 创建蓝图,命名开头大写,易于于函数区分

var Parsen = function( name, age, address ){

this.name = name; // 注意加this.

this.age = age;

this.address = address;

}

#### // 使用蓝图创建对象

// 原理:

// a) new 运算符的作用是: 创建一个空对象

// b) 而蓝图为函数，加入this的变量被new操作为对象属性.

var zhao = new Parsen('zhao','21','大学嘉园');

var teng = new Parsen('teng','21','赵楼');

console.log( zhao,teng );

### // 使用prototype给蓝图对象添加自定义函数

#### // 可用于继承的原因:

// 0. prototype在每一个对象或者函数默认拥有的属性( 核心 )

// 1. prototype中包含官方自带的函数，所以我们才能在对象中使用

// 2. 同理如果将我们创建的自定义函数加入prototype属性中，那么也就可以在对象中调用我们的自定义函数

#### // 常用于:

// 0. 给对象增加函数,如蓝图

// 1. 也可以添加属性，但是并不常用

Parsen.prototype.cale = function(){

console.log( 100 - this.age );

}

zhao.cale(); // 调用函数

teng.cale();

\*/

## /\*\*\*\*\*\*\*\*\*\* 原型链细节补充，以及证明js皆对象

// 0. 在chrome中可以清楚的看到，js万物皆对象，除“ int,float,null,undefine,true/false ”数据类型为源语

// 1. 他们都有原型链 \_\_proto\_\_ 目的是方便继承，调用不同的函数或者方法。

// 2. 如创建的对象与蓝图中的原型链:

// a) zhao.\_\_proto\_\_ === Parsen.prototype // true 成立

// 3. 因为有\_\_proto\_\_(原型链)的存在，才可实现不同数据间对函数的灵活调用，且他默认包含官方的默认函数在\_\_proto\_\_中的\_\_proto\_\_可找到，利用谷歌控制器

console.log( zhao.\_\_proto\_\_ === Parsen.prototype ); // true

\*/

## /\*\*\*\*\*\*\*\*\*\* 在chrome控制器调试小技巧-查看原型链对象构成

### // .hasOwnProperty()可判断对象中是否有指定属性

zhao.hasOwnProperty( name );

### // 判断对象蓝图归属

zhao instanceof Parsen // true

// 查看数据的对象构成细节，如数组的对象细节

sss = [1,2];

console.info(sss);

// 如下图:

\*/

## /\*\*\*\*\*\*\*\*\*\* 使用Object.creat()打造继承-模板对象蓝图

// 前面我们演示了，用函数来缔造蓝图，这次用对象来缔造

// 使用Object.creat()，缔造对象模板蓝图

// 缺点: 比起使用函数来缔造对象蓝图，Object.creat其实只搞些普通的属性确实很麻烦

// 优点: 对于函数类型的属性，能够更加容易引用且可读性更高

// 推荐使用方法2

### // 方法1( 其实这个方法比较灵活 ):

//创建蓝图

var personProto = {

cale: function(){

console.log( 2019 - age );

}

}

// 继承蓝图

var zhao = Object.create( personProto );

zhao.name = 'zhao'; // 继承蓝图后在增加属性

zhao.age = '21';

### // 方法2( 这个写法是按照js底层的工作方式来写的value代表属性的内容 ):

var teng = Object.create( personProto, {

// 这里涉及的js的底层工作方式，在书中有提到,本人已解

name: { value: 'teng' },

age: { value: '21' }

} );

\*/

## /\*\*\*\*\*\*\*\*\*\* 关于js的赋值 \*/

### // 0. 普通变量间的赋值

// a) 不同与C，在JS中普通变量间的赋值是真正的复制

// b) 指定的不是内存的同一个位置，但内容相同

var a = 20,b;

b = a;

a = 18;

console.log(a,b); // 18,20

### // 1. 对象之间的赋值

// a) 二者之间并非真正的复制，而是指向同一个内存位置

var test1 = {

a : 'abc',

b : '2000'

}

var test2 = test1;

test1.a = '233';

console.log( test1.a,test2.a ); // 233,233

# 2019.8.5

## /\*\*\*\*\*\*\*\*\*\* 函数，接受函数 - 极为强大方便

// js的函数可以接受函数

### // 原因:

// 0. 函数可以当作变量进行传递

// 如: function a(){}; b = a;

// 那么b()具有a的同样功能

### // 目的:

// 0. 演示接受函数的便捷性

// 1. 实战时减少冗余代码，增加一个骚操作

// 2. js确实灵活

### // 示例

var year = [ 1998,2005,1995 ];

function cale( year,fuc ){ // 接受函数

var result = [];

for( var i = 0; i < year.length; i++ ){

result[i] = fuc( year[i] ); // 注意这里，经过接受的函数处理后在加入数组

}

return result;

}

// 功能1: 计算年龄

function age( year ){

return 2019 - year;

}

// 功能2: 判断是否未成年

function ifOld( year ){

var old = age(year);

return ( old >= 18 ? true : false );

}

console.log( cale(year,age) ); // 输出计算后的年龄

console.log( cale(year,ifOld) ); // 输出是否未成年

\*/

## /\*\*\*\*\*\*\*\*\*\* 函数，返回函数 - 极为强大

### // 目的:

// 可以根据条件灵活的变更函数，以处理不同的数据

### // 返回函数的原理:

// 其实这种返回函数的方法基于“函数表达式”构成的

// var a = function(x);

// 然后可以使用 a(x);

function interView( job ){

if( job == 'running' ){

return function( name ){

console.log( '你的一名运动员?' + name );

}

}

else if( job == 'code' ){

return function( name ){

console.log( name + '哈哈和我一样是玩代码的');

}

}

else{

return function( name ){

console.log( '你到底是做什么工作的?' + name );

}

}

}

#### // 普通调用

var qa = interView( 'game' );

qa('\_\_OO7\_\_');

#### // 便捷调用:

// 原因:

// 0. interView('code')('ZTaer');的调用相当于匿名函数function('ZTaer'){}函数被调用;

// 1. js是从"左到右"来处理代码的

interView( 'code' )( 'ZTaer' );

interView( 'running' )( 'OO7' );

\*/

## /\*\*\*\*\*\*\*\*\*\* IIFF函数表达式，专门创建隐私安全的变量

### // IIFF使用目的:

// 0. 创建极据安全的隐私变量，不影响全局变量，又能在全局中使用，也不需要用函数去命名

// 1. 使用IIFF创建的变量，是一个极具安全的变量

// 2. 可以对一些敏感数据变量进行化函数处理

// 3. 使我们的代码不受其它代码的影响,如变量名称相同时。因为有时一个js项目有多个人合作,可能难免因为变量名称的问题互相产生影响,造成程序混乱.

### // IIFF原理:

// 0. 在函数中外部无法访问函数内的变量

// 1. ()括号内为表达式状态将直接被执行, 其实就是欺骗编译器的一种手段

// 2. 如果无括号()那么只是对函数的声明，并不会被执行

### // IIFF原型:

// ( function(x){} )(x);

( function( money ){

console.log( money + 999 );

})( 1 ); // 直接被执行输出

\*/

## /\*\*\*\*\*\*\*\*\*\* 闭包 - 理解闭包,将能成为一个非常非常厉害的javascript的开发人员,而现在我已经掌握

### // 原因:

// 0. 函数在调用后，正常情况下,局部变量应在内存中清空

// 1. 但是在某些情况，如函数返回函数时，前函数"未闭包"他的局部变量依然在内存中，可以访问.

// 2. 我们利用这一特性，从而创造出更加便利的函数来使用

// 3. 至于"闭包"这不用担心，js将自动闭包

### // 目的:

// 1. 就是在返回函数的过程种调用前函数的变量来借用,以达到类似于“返回函数”便利的效果

// 2. 闭包特性利用的目的在于便捷性

### // 如下案列:

function iv( name ){

return function( job ){ // iv返回函数后，我们依然可以调用局部变量name值，此时就是未闭包

if( job == 'designer' ){

console.log( name + ' explain what UX design is ?' );

}

else if( job == 'teacher' ){

console.log( 'what sbject do you teach ' + name + ' ?' );

}

else{

console.log('hello ' + name + ' what do you do?');

}

}

}

iv('ZTaer')('designer'); // 其实这里利用时，函数返回函数的调用方式( js中函数是可以当一个变量来赋值的 )

iv('ZHAO')('code');

\*/

## /\*\*\*\*\*\*\*\*\* 函数的预设参数实验( 可行 )

function a( name = 'zhao', age = '19' ){

console.log(name,age);

}

a();

a( 'li','21' );

\*/

## /\*\*\*\*\*\*\*\*\*\* .call()方法借用,函数替换

### // 原理:

// 创建一个副本来进行操作

### // 0. 方法引用

// a) a.cale.call(b,x,y...); 模板

// b) call前为借用的方法，call的第一个参数为目标变量，其它为方法需要的参数

### // 1. 函数替换( 感觉并不常用 )

// a) test1.call(test2,x...); 模板

// b) 借用test1函数,相当于test1(x);

### // 0. 方法引用

var john = {

name:'john',

age:12,

cale: function( size ){

console.log( this.name + ' - ' + this.age + ' - ' + size );

}

}

var zhao = {

name:'zhao',

age:23

}

john.cale( 'young' );

john.cale.call(zhao,'old'); // 二对象之间引用方法，此时zhao对象引用john的方法

### // 1. 函数替换

function test1( x ){

console.log( x-100 );

}

function test2( x ){

console.log( x+100 );

}

test1(50);

test2.call( test1,50 ); // 借用test2函数

\*/

## /\*\*\*\*\*\*\*\*\*\*\* .apply()方法借用，功能于call()相同，只是传输参数的方法不同

john.cale.apply( zhao,['old'] ); // 用数组传输参数,其它功能于call一样

\*/

## /\*\*\*\*\*\*\*\*\*\*\* .bind()方法借用，预设参数，返回函数 \*/

### // 注意:

// bind()返回的是一个函数( 复制一个副本 )

### // 功能:

// 借用方法: var test = john.cale.bind( zhao,'big' ); // 返回的为一个函数

// 预设参数: test('code') // 因为有一个变量为预设参数，所以这里传输一个即可

// 返回一个复制的副本函数,以便后续使用

var john = {

name:'john',

age:12,

cale: function( size,job ){

console.log( this.name + ' - ' + this.age + ' - ' + size + ' ' + job );

}

}

var zhao = {

name:'zhao',

age:23

}

var bindTest = john.cale.bind( zhao,'big' ); // bind()具有借用方法功能，预设参数功能，返回的是一个函数

bindTest('code');

#### // 或者

john.cale.bind( zhao,'big' )('code');

## 对2019.8.5面向对象实战总结

1. 老师把所用到的函数都通过prototype的方式加入到了对象中，以方便调用
2. IIFF的目的使，我们的代码不受其它代码的影响,如变量名称相同时。因为有时一个js项目有多个人合作,可能难免因为变量名称的问题互相产生影响,造成程序混乱.

# 2019.8.9

## IIFF+模块化+闭包特性，小项目安全骨架参考：

请参考2019.8.8 – 小应用安全骨架参考 – app.js

// Init()函数，放置初始化函数

// 监听有自己函数

// 抓取前端有自己的函数

// 大模块 + API思维方式

# 2019.8.11

## 预算项目实战笔记

### // 监听全局"按键"并返回对象给event

/\*\*

\* 0. 监听返回一个对象，对象中包含一些参数可以在浏览器控制台中查看

\* 1. 故知Enter键按下返回值为event.keyCode:13;

\* 2. event.which:13;为老旧的浏览器准备，同样存在与返回的对象中

\*/

document.addEventListener('keypress',function(event){

if( event.keyCode == 13 || event.which == 13 ){

control\_add\_items();

}

});

## 抓取表单select参数

<select class="add\_\_type">

<option value="inc" selected>+</option>

<option value="exp">-</option>

</select>

#### Select如下图:



#### 抓取方式:

iType: document.querySelector( .add\_\_type ).value;

iType = inc; // 如果用户选择+则返回’inc’相反则返回’exp’

# 2019.8.12

### Math.round( x )四舍五入

Math.round( 2.5 ) // 为3

# 2019.8.14

## Js变量保存HTML代码

#### // 引用HTML给变量时，不能有空格间隙,并要以字符串的形式引用

html = '<div class="item clearfix" id="income-%id%"></div>';

### 替换字符串内容.replace();

newsHtml = html.replace('%id%',test); // 将html的字符串”%id%”替换为test内容

## 插入HTML代码，.insertAdjacentHTML( position, text );

1. 关于position参数
   1. Beforebegin // 在选中元素之前
   2. Afterbegin // 在选中标签内容中,上部插入
   3. Beforeend // 在选中标签内容中,下部插入
   4. Afterend // 在选中标签内容外部的底部插入内容



1. 实列:

// 插入HTML到前端

newsHtml = ‘<div class=”py-5”></div>’

document.querySelector(choose).insertAdjacentHTML('beforeend',newsHtml);

## // .querySelectorAll( )选中所有元素，返回NodeList对象

/\*\*

### \* 0. 选中指定所有元素，

并返回一个NodeList对象,对象中包括标签中所有被选中的元素( NodeList包含所有的CSS样式 )

### \* 1. 小技巧:

\* a) .querySelectorAll('p').length 可以查看HTML选中元素个数

\* b) 因为返回的是一个对象，对象中包括所有被选中的元素，所以需要迭代出所有元素进行操作

\* c) 但是如果想精确的选中元素进化操作，则需要转换为数组进行操作，因为objct类型数据为无序，数组为有序

### \* 2. 转换数组:

\* a) 借用数组的slice函数: Array.prototype.slice

\* b) 使用call()借用方法: Array.prototype.slice.call(example)即可返回一个数组;

\*/

// 此时为多选

var input = document.querySelectorAll(DOM\_strings.addDescroption + ',' + DOM\_strings.addValue);

input = Array.prototype.slice.call(input);

## // .forEach()方法迭代数组元素，并将元素传递给回调函数。

/\*\*

\* 0. forEach()默认会返回给函数3个元素

\* 1. array.forEach(function(currentValue, index, array), thisValue)

\* a) currentValue: 当前元素

\* b) index: 当前元素索引值

\* c) array: 当前所属数组对象

\* 2. 在函数中操控元素

\*/

input.forEach( function( item,index,array ){

item.value = ""; // 改变每一个迭代出的值

} );

## // .focus()成为焦点元素，表单选中状态

input[0].focus();

# 2019.8.16

## // 以下参数可以在prototype控制台中查看

### // 精确选中获取标签

// .target: 获取当前标签

// .parentNode: 获取父类标签

// .id 获取标签id

var itemsID = event.target.parentNode.parentNode.parentNode.parentNode.id;

## // .split('x')指定符号,切割字符串并返回数组

splitID = ‘inc-1’;

splitID = itemsID.split('-'); // 返回数组[ ‘inc’,1 ]

itype = splitID[0];

ID = parseInt( splitID[1] );

## //array.map() 遍历数组元素，并进化函数处理，处理完则返回一个新的数组

/\*\*

\* 0.默认会返回3个参数，数组元素为必须，其它2个为可选

\* array.map( function( 数组元素,索引值,当前数组 ){} );

\* 1. 遍历完数组元素后，进行函数处理，处理完则返回一个新的数组，

\* 2. 排序保持不变,且不会改变原数组

\* \*/

// data.allItems[ itype ] 为一个数组

ids = data.allItems[ itype ].map( function( current ){ // 通过map函数获取数组的id值

return current.id;

} ); // ids = [1,2,3]

indexID = ids.indexOf(ID) // 利用map返回数组排序不变，配合indexOf判断目标索引值

## //利用splice()修改数组( 注意split()与splice()区别 )

// array.splice()添加/删除数组( 注意，会直接影响原数组 )

/\*\*\*

\* 0. array.splice( 目标索引值，删除数量，添加的元素0,添加的元素1.... );

\* a) 注意splice将直接改变原数组内容，无需赋值操作

\* b) 将返回删除的数组

\* 1. 删除:

\* 返回删除的数组，且原数组也将改变

\* var fruits = ["Banana", "Orange", "Apple", "Mango"];

\* fruits.splice(2,2); // Banana,Orange

\* 2. 添加( 指定插入数组内容 ):

\* var fruits = ["Banana", "Orange", "Apple", "Mango"];

\* fruits.splice(2,0,"Lemon","Kiwi"); // Banana,Orange,Lemon,Kiwi,Apple,Mango

\* 3. 替换添加:

\* fruits.splice(2,1,"Lemon","Kiwi"); // Banana,Orange,Lemon,Kiwi,Mango

\*

\*/

## // 监听标签内DIV内的所有活动，并针对某活动进行函数处理

<div class=”container” ></div>

// 监听删除按钮

document.querySelector(getDOM.container).addEventListener( 'click', control\_delete\_items );

// 在配合其它函数获取特定的动态，达到指定特征监听活动效果



## //.removeChild( ) 删除父类中所有子类标签内容

// 目标父类.REMOVECHILD( 目标子类 ) 删除子标签

// a) 因为js，.removechild()只能通过父类来删除子类，进行删除目标( 目前没用方法直接删除目标标签 )

target = document.getElementById( targetID ); // 删除目标

target.parentNode.removeChild( target ); // 目标的父类.removeChild( 目标 )