**JaveScript 笔记一**

============================================================

**拓展：**

**JavaScript**是运行在客户端的嵌入式脚本语言。它诞生于1995年,是由Netscape公

司的布兰登**·**艾奇发明， 最初的目的是为了处理表单验证操作的。

**JavaScript由三部分组成(实现)：**

核 心（ECMAScript）：是脚本程序设计语言，又被称之为jscript

文档对象模型（DOM）：Document Object Model

浏览器对象模型（BOM）：Browser Object Model

**JavaScript的库：**（JavaScript在国内企业的应用）

淘宝网 YUI Kissy SeaJS 腾讯网 JEI Q+( WebQQ ) 百 度 Tangram 百度地图

=============================================================

* **语法结构**

=============================================================

**[一]、链接JavaScript的语法格式（script标签）：**

位 置： 脚本可位于HTML页面<body>和<head>里（head、body里都可以）

执行顺序：由上到下

①. 内嵌：<script></script>

属性(可选)： **type="text/javascript"**

**slanguage="javascript"**

②. 外部文件：<script src="路径/文件名.js" ></script>

属性：src属性是必选项，链接路径；

defer="defer" 延时 ----> 只对外部文件生效，对内嵌不生效。

**[二]、基本语法：**

1、JavaScript是区分大小写的语言

2、注释： Html <!-- - ->

CSS /\* \*/

JS 单行注释 // --> 快捷键Ctrl + /

多行注释 /\* \*/ --> 快捷键Ctrl + Shift + /

3、结束以 ；分号

**[三]、输出的三种方式：**

alert( ) 弹出框(警告框)

console.log( ) 控制台输出显示信息

document.write( ) 页面中输出 (文档内)

注： 1. 常量加双引号，变量不加双引号

2. 换行：在网页中实现将文本换行的是（<br/>） 如：document.write( "<br/>")

alert方法中实现将文本换行的是（\n） 如：alert("\n")

3. 加号（+）的共有两个功能：

①. 加法； 如：alert( 1+1 ) //值为2

②. 字符串连接，拼接。 如：alert( 1+"1" ) //值为11

**[四]、输入框**： prompt( text信息，value默认值 ) //value默认值可写可不写

prompt( "请输入信息","这里是默认值" )

**拓展：** JS中三种弹出式消息提醒的命令是什么?

①.确 认 框： confirm( ) 当点击确定是为true；当点击取消是则为false

②. 弹出(警告)框：alert( )

③. 信息输入框：prompt( )

**[五]、标识符：**所谓标识符，是指变量、函数、属性的名字，

**1.** 标识符命名规范：以字母、数字、下划线(\_)、或美元符号($)

首字符（第一个字符）必须是：字母、下划线(**\_**)、或美元符号($)；

关键字和保留字不能用来做标识符 。

**2.** 变量命名规范：以字母 数字 下划线命名 。

首字符（第一个字符）只能是：字母 和 下划线(**\_**)。

变量是使用关键字 var来声明的。

var定义的变量叫显式声明，没有用var的变量叫隐式声明。

定义变量的过程： var 变量名=赋值;

单个变量： ①. var num1=2334; ②. var num1; num1=2334;

多个变量之间用逗号隔开：

①. var n1=2334, n2=2334, n3=2334;

②. var n1,n2, 3; n1 = 2334; n2 = 1235; n3 = 9564;

注：变量后面赋的值覆盖前面赋的值（变量后赋值覆盖先赋值）

=============================================================

* **数据类型**

=============================================================

**[一]、判断数据类型的函数**：typeof

**语法：** typeof 变量名 或 typeof(变量名)

**[二]、变量的数据类型**：５个基本型，１个复杂型（对象类型 object）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **基本型** | | | | | **复杂型** |
| 未定义类型 | 空类型 | 布尔类型 | 数值型 | 字符串类型 | 对象类型 |
| Undefined | Null | Boolean | Number | String | Object |

**(1). 未定义类型 Undefined**（undefined值只有一个）

① 这个变量定义了，没有赋值，那么值为 undefined .类型也是 undefined

如： var a; console.log(a); // a=undefined

② 变量从未定义过 ，他的类型为undefined ，值报错。

**(2).空类型 Null**

Null类型是第二个只有一个值的类型，这个特殊值是null。

null值表示一个空对象指针。空类型null是一种特殊的对象类型：object

**(3).布尔类型 Boolean**

boolean只有两个值： true真 false 假。

类型转化 ：Boolean(变量名)

数 字： 非0数字为true 数字0为false

字符串： 非空字符串为true "" 空字符串为false

未定义和null类型都为false

**(4).数值类型 Number**

数字： 整型 45 浮点型 4.522

范围：Number.Min\_value : 5e-324

Number.Max\_ value : 1.7976931348623157e+308

-Infinity(负无穷) -3/0 +Infinity(正无穷) 3/0

NaN not a Number 不是一个数字 特殊的数值类型

isNaN(变量) 判断是不是数值的函数 (先往数值转化，然后在判断是不是数字)。

返回布尔值，若是数值就为false,若不是数值就为true (不是一个数字为真，是数字为假)

**类型转化 ：**

(1). 转化为数字：Number(变量名)

字符串 ： "12333" 12333

"adfk434" NaN

"" 0

"adsfdsf" NaN

布尔值： true 1 ; false 0

未定义类型： NaN

空类型null ： 0

(2).字符串转化为数字的函数：数值类型Number( )

**parseInt( )** 从第一位起取到整数部分

如： parseInt("4578abvcd") //4578

parseInt("ab123.5678k") //NaN

**parseFloat( )** 从第一位起取到小数部分

如：parseFloat("45.7962abvcd") //45.7962

parseInt("ab123.5678k") //NaN

注：parseInt（n）或parseFloat（n）中的n必须以数字开头

**(5).字符串类型 String** （双引号"" 或者 单引号''）

String 数据类型包含一些特殊的字符字面量,也叫转义序列,用于表示非打印字符。

\n 换行 \t 制表 \b 空格 \r 回车 \f 进纸

属性**length** 求字符串长度

**转化为字符串2种方法：**

1.可以转化任意类型 ：String( ) 如：console.log( String() )

String(123) "123"

String(3>4) "false"

String(3<4) "true"

String( n ) "undefined"

String(null) "null"

2.不能转化空类型和未定义类型： 变量名.toString()

如：console.log( user. toString() )

**(6).对象类型 Object**

对象：其实就是一组数据(特征)和功能的集合。

语法：(1). 直接定义

例： var \_news={

name : "张三",

sex : "男",

say:function(){

alert("你好")

}

}

console.log( \_news.sex ) //男

\_news.say( ) //你好

(2). 定义一个新的对象，在一个一个定义

例： var car=new Object();

car.name="奥迪";

car.color="黑色";

car.game=function(){

alert("能进行比赛")

}

console.log( car.color ) //黑色

car.game( ) //能进行比赛

=============================================================

* **操作符**

=============================================================

操作符： 将同类型的数据(如常量、变量、函数等)，用运算符号按一定的规则连接起来的、

有意义的式子称为表达式。

**[一]、一元运算符：**

只能操作一个值得操作符叫做一元操作符。

一元操作符有两个：递增操作符 ++ 递减操作符 --

前置与后置区别：前置：先增加值在运算；后置：先运算在增加值

例如：var num1,num2,num3;

num1=10;

num2=8;

num3=(++num1)+num2-- ; //num2--不参与num3本身运算

num4=(num1++)+(--)num2; //num1++不参与num4本身运算

console.log(num3); //num3=(1+10)+8=19 num1=11 num2=7

console.log(num4); //num4=11+(7-1)=17 num1=12 num2=6

**[二]、算数运算符：**加性运算符： 加法+ 减法-

乘性运算符：乘法\* 除法/ 求模%

优先级：乘性操作符优高于加性操作符，结合性：从左往右。

**[三]、关系运算符(比较运算符)：**

小于号**<** 大于号**>** 等于号**=** 小于等于号**<=** 大于等于号**>=**

**关系运算符**返回值为布尔值 true ，false

(1). 如果两个操作数都是数值，则执行数值比较。

(2). 如果两个操作数都是字符串,则比较两个字符串对应的字符编码值。按位比较。

(3). 如果一个操作数是数值，则将另一个操作数转换为一个数值，然后执行数值比较。

**注：**如果有一个转化为数字后为NaN，则返回值为false

**[四]、相等运算符：**

相等== 不等于!= 全等=== 不全等于!==

**相等运算符**返回值为布尔值 true false

相等和不等运算符：先转化类型在比较。

3==3或3=="3" true

3=="abc" false

3 != 3 false

3 != "abc" true

全等和不全等运算符：不转化，直接比较。

3===3 3==="3" 3!=="3"

**[五]、逻辑运算符(布尔操作符)：**

逻辑与 **&&** 逻辑或 **||** 逻辑非 **!**

**逻辑运算符**返回值大部分情况下是布尔值

**(1). 逻辑与 &&：**多个条件同时满足，才为真。

true true true

true false false

false true false

false false false

如果有一个操作数不是布尔值的时候：

①. 如果第一个操作数为true，则返回第二个操作数。

②. 如果第一个操作数为false，则不管后面是什么，都返回false。

**注：** 逻辑与操作属于短路操作，即如果第一个操作数能够决定结果，那么就不会再

对第二个操作数求值。

对于逻辑与操作而言,如果第一个操作数是false ,则无论第二个操作数是什么

值,结果都不再可能是true

**(2).逻辑或 ||：**只要一个条件满足，才为真。

true true true

true false true

false true true

false false false

**注：** 如果第一个操作数为true,则返回true。

如果第一个操作数为false,则返回第二个操作数。

**(3).逻辑非 !：**取反，返回结果是布尔值

**[六]、赋值运算符 ：**

简单的赋值操作符：

由等于号 （ = ） 表示,其作用就是把右侧的值赋给左侧的变量.

复合赋值操作9符：

乘/赋值（ \*= ） 除/赋值（ /= ）；

模/赋值（ %= ） 加/赋值（ += ）；

减/赋值（ -= ） 左移/赋值（ <<= ）；

有符号右移/赋值（ >>= ） 无符号右移/赋值（ >>>= ）；

**[七]、逗号操作符：**分隔多个变量

逗号操作符还可以用于赋值。用于赋值时，逗号操作符总会返回表达式中的最后一项。

**[八]、条件操作符：即**三元操作符

条件操作符应该算是ECMAScript中最灵活的一种操作符。

var iable = boolean\_expression ? true\_value : false\_value;

条件表达式 ? 为真的结果 : 为假的结果

例：var max = a>b ? a : b

**操作符优先级:**

赋值操作符 < 逻辑运算符 < 关系运算符 (相等运算符）< 算数运算符

=============================================================

* **语句**

=============================================================

流程控制语句：条件语句 和 循环语句

**[一]、条件语句：**

**1、if的语句的三种结构:**

(1). if( 条件表达式){ }

(2). if( 条件表达式 ){ }else{ }

(3). if( 条件表达式1 ){

}else if( 条件表达式2 ){

}

...

}else if( 条件表达式n ){

}else{ }

**2、swicth语句的结构：**

(1). 条件分支语句：

switch( 变量名 ){

case 值1：语句1; break;

case 值2：语句2; break;

case 值3：语句2; break;

...

default: 语句N;

}

(2). 并语句:  switch(变量名){

case 值1：

case 值2：

语句2; break;

...

default: 语句N;

}

switch： 在switch 语句中使用任何数据类型，无论是字符串，还是对象都没有问题。

每个 case 的值不一定是常量，可以是变量，甚至是表达式。

default：default关键字则用于在表达式不匹配前面任何一种情形的时候，执行机动代码。

**[二]、循环语句**

**1、前测试循环**

①. for**循环:**

for( 初始化变量 ; 条件表达式；增量）{

循环体；

}

注：初始化变量 -->初始化表达式

条件表达式 -->控制表达式

增量 -->循环后表达式

②. **for in 循环：**

是一种精准的迭代语句，可以用来枚举对象的属性

for( var 变量i in 数组名){

数组名[i]

}

③. **while循环:**

变量赋初值

while( 条件 ){

循环体;

增量

}

**2、后测试循环：**do- while 循环（至少执行一次循环体）

do{

循环体;

增量；

}while( 条件 )

**退出语句：**break 和 continue 语句用于在循环中精确地控制代码的执行。

其中, break 语句会立即退出循环，强制继续执行循环后面的语句。continue语

句虽然也是立即退出本次循环，但退出循环后会从循环的顶部继续执行。

=============================================================

* **函数**

=============================================================

**[一] 、1.函数的定义：** 实现某一功能的多条代码或语句。

可以在任何时候、任何地方调用 。（随时随地）

**2.函数没有重载**，函数不能重名，后写的函数覆盖先写的函数。

**[二]、声明函数关键字：**function

函数声明：function 函数名(形式参数1，形式参数2.....){ 函数体；}

如：function sum(num1,num2,……){ }

**[三]、函数表达式：**先定义再调用

var 函数名=function( 形式参数1，形式参数2...){ }

如：var sum= function sum(num1,num2,……){ }

**[四]、调用函数的三种方法：**

**1.直接调用：**在js中

函数名(实际参数1，实际参数2....)

**2.鼠标事件调用** : html网页中

<input type="button" value="计算" onclick="函数名(实际参数1，实际参数2)">

**3.写在js里不带参数:** DOM0事件处理程序

btn.onclick=function(){ }

**[五]、函数的返回值（return 返回）**

语法： return 变量名;

return之后的语句不执行。return 返回值是undefined.

**[六]、arguments对象：**JavaScript的一个内置对象

arguments指函数的参数对象(指实际传入的参数)。利用arguments对象可以实

现重载。在函数体内，arguments像数组，但并不是真的数组，它是一个Arguments

对象，可以来访问这个参数数组，从而获取传递给函数的每一个参数

arguments.length 指函数的参数对象的长度(获取函数的参数个数)

arguments[ idx ] 指第idx个参数的值（第一个为0）

arguments[0] arguments[1] ...... arguments[arguments.length-1]

**[七]、函数的作用域** （执行环境）

全局变量：可在全局环境中使用，可被任何函数使用。

局部变量：在局部环境中使用，不能被其他函数使用。

=============================================================

* **BOM**

=============================================================

**BOM:** 浏览器对象模型(Browser Object Model),它赋予了JS操作浏览器的能力,即window操作.

**[一]、window对象：**指当前的浏览器窗口, 是BOM的核心

**1. 计时器：**

(**1**)**. 间歇调用：**定时器，根据指定时间间隔执行一次代码

①.（设置）间歇调用时间间隔： setInterval( code , delay )

注：code指的是函数名或函数；delay指的是时间间隔，单位毫秒

如：Id名 = setInterval ( 函数名 / function(){函数体} , 毫秒）

②. 清除（间歇调用）时间间隔： clearInterval( Id名 )

**例：** timer = setInterval( auto , 1000 ） // 1秒=1000毫秒

clearInterval( timer )

(**2**)**. 超时调用：**延时器，只能调用一次

①.（设置）超时调用时间间隔：setTimeout( code, delay)

注：code指的是函数名或函数；delay指的是时间间隔，单位毫秒

如：Id名= setTimeout ( 函数名 / function(){函数体} , 毫秒）

②. 清除（超时调用）时间间隔： clearTimeout ( Id名 )

**例：** timer = setTimeout ( auto , 1000 );

clearTimeout ( timer ) ;

**2. 系统对话框：**

window. alert( ) 　　 弹出框（警告框）

window.prompt(text , value) 输入框

window.confirm( ) 确认框 返回值为布尔值：true false

简写为： alert( )

prompt(text , value)

confirm( )

**3. 窗口操作：**

(**1**)**.打开窗口：**window.open() 方法可用于打开新窗口

语法：window.open("url","窗口名字","窗口特性","replace")

**注：" url "：**设置打开窗口的路径，是必选项

**"窗口名字"：**指，是可选项。

**"窗口特性"：**设置窗口参数，多个参数可用逗号分隔，是可选项。

如：width=150,height=100,fullscreen=yes

**"replace"：**值为true或false，是可选项。

true时URL**替换**浏览历史中的**当前条目**；

false时URL在浏览历史中**创建新的条目。**

(**2**)**.关闭窗口：**window.close()方法可用于关闭当前窗口

语法：window.close() （ 简写为：wd.close() ）

**例：** wd=window.open("url","窗口名字","width=150,height=120, ")

wd.window.close()

**4.窗口的位置：**

(**1**)**.** IE、Chrome、Opera 和Safari兼容，Firxfox( FF )不兼容 :

window.screenLeft 与屏幕左侧的距离

window.screenTop 与屏幕顶部的距离

(**2**)**.** FF、Safari、Opera 和 Chrome兼容，IE不兼容:

window.screenX 与屏幕左侧的距离

window.screenY 与屏幕顶部的距离

**5. 滚动条滚动的高度**

document.documentElement.scrollTop　　//滚动条滚动的高度

**6.窗口的移动：**

(**1**)**.** moveTo(x,y) 接收的是新位置的 x 和 y 坐标值。

如：moveTo(100,200) （从整体而言）

(**2**)**.** moveBy(x,y) 接收的是在水平和垂直方向上移动的像素数。

如：moveBy(200,100) （是相对于原来窗口）

**7.窗口的大小：**

(**1**)**.** 浏览器窗口大小：

window.outerHeight window.outerWidth

(**2**)**.** 可视区大小：（视口大小）

window.innerHeight window.innerWidth

**8. 可视区的宽度和高度：**

html对象.clientWidth | html对象.clientHeight //浏览器的可视宽(高)度

html对象.scrollHeight　 //内容的高度( Html的高度)

**注：**html对象指：document.documentElement

document.body.clientWidth //内容的宽度( body的宽度)

document.body.clientHeight //内容的高度( body的高度)

**[二]、location对象：**提供加载的文档有关信息以及控制页面跳转

window.location是最有用的 BOM对象之一，它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，简单说就是可以给赋值，像使用open一样。

**语法: location.[ 属性 | 方法 ]**

**例：**获取地址栏信息(超链接跳转)： location.href="http://www.baidu.com"

**location对象属性：**

location.href　　 设置或返回完整的 URL（获取地址栏信息）

location.search　　设置或返回从问号(?)开始的 URL（查询部分）

location.pathname　　设置或返回当前 URL 的路径部分。

location.hash　　 设置或返回从井号(#)开始的 URL（锚）。

location.host　　设置或返回主机名和当前 URL 的端口号

location.hostname　　设置或返回当前 URL 的主机名，不包含端口

location.port　　设置或返回当前 URL 的端口号 （80 或 443）

location.protocol　　返回所使用的 web 协议（http:// 或 https://）

**[三]、navigator对象：**对象存储浏览器名称及版本信息

window.navigator对象现在已经成为识别客户端浏览器的事实标准。其中最常用

的属性是navigator.userAgent，是识别浏览器的关键，可用于获取当前浏览器版本信

息和判断浏览器的类型。返回一个字符串。

userAgent 是 navigator 对象的一个只读属性，译为"用户代理"，简称"UA"。

**<script>** //判断是 PC端-->电脑 或 移动端-->(手机)

var userAgent = navigator.userAgent.toLowerCase(),

systemVeersion = /windows/.test( userAgent ) ? '电脑' : '手机';

console.log( systemVeersion );

**</script>**

**[四]、history对象：**保存用户上网记录

window.history对象记录了用户曾经浏览过的页面(URL)，并可以实现浏览器前进与后退导航的功能。从窗口被打开的那一刻开始记录， 每个浏览器窗口、每个标签页乃至每个框架，都有自己的history对象与特定的window对象关联。在使用时可以不加window前缀。

**语法: window.history.[ 属性 | 方法 ]** 或 **history.[ 属性 | 方法 ]**

**(1)、History对象属性：**历史记录不唯一，即该对象具有length属性，可返回浏览器历

史列表中的URL数量。

**(2)、History对象方法：**

①. history.go(num) 根据当前所处的页面,加载 history 列表中的某个具体的页面

参数说明：0为当前页面 ; 1为前一个 页面; -1下一个页面。

②. history.forward( ) 前进 ( 加载 history 列表中的前一个URL，就是返回前一个页面 )

该方法等同于：history.go(1);

③. history.back( ) 后退 ( 加载 history 列表中的下一个URL，即返回下一个页面 )

该方法等同于history.go(-1)

**[五]、screen对象：**显示屏幕相关信息

screen 对象基本上只用来表明客户端的能力.其中包括浏览器窗口外部的显示器的

信息，如像素宽度和高度等。

screen.width 屏幕分辨率宽度

screen.height 屏幕分辨率高度

=============================================================

* **DOM**

=============================================================

**DOM:** 文档对象模型( Document Object Model ),它则用于创建删除节点,操作HTML文档

**[一]、节点**

**1. 节点之间的关系：**

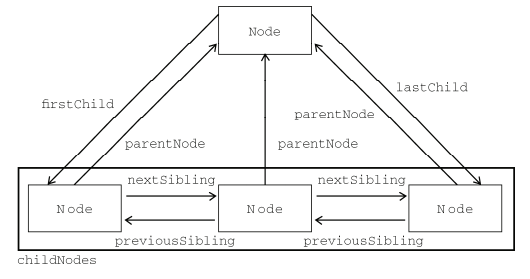
**父节点：**  parentNode

**子节点：** 第一个子节点 firstChild 最后一个子节点 lastChild

查找所有子节点 childNodes **如:** childNodes[0] childNodes[1]

获取子节点长度 childNodes.length

**兄弟节点：** 前(上)一个兄弟节点 previousSibling 下一个兄弟节点 nextSibling



**2.节点的类型：**总共有12种节点类型

element元素节点 1 Attr属性节点 2 文本节点 3

document文档节点 9 注释节点 8

**3.节点属性：**

|  |  |  |
| --- | --- | --- |
| 节点的类型  nodeType | 节点的名字  nodeName | 节点的值  nodeValue |
| 元素节点 1 | head div h1(元素名) | null |
| 文本节点 3 | #text | 文本本身 |
| 属性节点 2 | class href id（属性名） | 属性值 |

判断是否是文本节点：①. nodeName=="#text" ②. nodeType==3

**4. 获取属性节点：**

1.获取属性节点： getAttributeNode("属性名")

如：class="bg" 节点 //get 获取 Attribute 属性 Node 节点

2.获取属性节点的值：getAttribute("属性名") "bg" 值

3.设置属性节点的值：setAttribute("属性名","属性值"）

**5. 节点的操作方法：**

(1).节点的查找方法：

①. Id名=document.getElementById("id名") 查找到的是一个值

②. \_name=document.getElementsByName("name名")

查找到的是一组数据 如： \_name.length

③. 元素名=document.getElementsByTagName("标签名")

查找到的是一组数据 length 个数

(2). 创建(插入)节点：

①. 创建文本节点：

var text=document.createTextNode("文字信息")

②. 创建元素节点：

var 元素名=document.createElement("元素名")

(3). 添加(追加)节点：

末 端：

父节点.appendChild(子节点) 子节点作为父节点最后一个元素存在

指定位置前：

父节点.insertBefore(新子节点，插入到哪个节点之前)

(4). 删除节点 ：

父节点.removeChild(子节点)

(5). 替换节点 ：

父节点.replaceChild(newNode，oldNode)

即：父节点.replaceChild(新节点，要替换的节点)

(6). 复制(克隆)节点：

要复制的节点.cloneNode( [false /true] )

**注：**false (可不写)只复制结构,不复制内容； true 既复制结构也复制内容。

**[二]、DOM可以改变样式、内容**

(1). 获取html网页信息: innerHTML

如：document.getElementById("id名").innerHTML=""

(2).识别的文字信息： innerText ff不识别

textContent 都识别

(3).改变样式：

box.style.属性名="属性值"

font-size fontSize 驼峰规则

box.className="样式名"

box.className=""

(4).文档输入：输出流写入到网页的能力

document.write()

document.writeln() 空格的距离

open（）

close()

=============================================================

* **事件**

=============================================================

**[一]、事 件：**用户在浏览器窗口或文档中发生的一些特定的交互瞬间。

**[二]、事件流：**页面接收事件的顺序。

1. **IE(事件冒泡)事件：**即事件开始时由最具体的元素(文档中嵌套层次最深的那个节点)

接收，然后逐级向上传播到较为不具体的节点（文档）。

2. **事件捕获：**事件捕获的思想是不太具体的节点应该更早接收到事件，而最具体的节点

应该最后接收到事件。

事件捕获的用意在于在事件到达预定目标之前捕获它。

3. **DOM事件流：**"DOM2级事件"规定的事件流包括三个阶段：

事件捕获阶段、处于目标阶段和事件冒泡阶段。

**[三]、事件处理程序( 或事件侦听器 ):**

事件就是用户或浏览器自身执行的某种动作。如: click 、mouseover。而响应某个

事件的函数就叫做事件处理程序（或事件侦听器）。事件处理程序的名字以"on" 开头。

(1).HTML事件处理程序：

<input type="button" onclick="btn( )">

(2).DOM0级事件处理程序：

btn.onclick=function(){ }; //添加事件处理程序

btn.onclick=null; //删除事件处理程序

(3).DOM2级事件处理程序：

①.非 IE事件处理程序

**添加：**addEventListener( ) **删除：**removeEventListener( )

语法：addEventListener("事件名", "函数", "布尔值");

说明：布尔值若为true，表示在捕获阶段调用事件处理程序；

若为false，表示在冒泡阶段调用事件处理程序

**例：**addEventListener("click", "function()", "true");

**注：**addEventListener( )与removeEventListener( )一样

②. IE事件处理程序(IE只支持事件冒泡)

**添加：**attachEvent( ) **删除：**detachEvent( )

语法：attachEvent ("事件名", "函数");

**例：**attachEvent ("onclick", "function()", "true");

**注：**attachEvent ( )与detachEvent ( )一样

**[四]、鼠标事件：**

onclick 单击 **如：**btn.onclick=function(){}

ondbclick 双击 onmousemove 鼠标移动

onmouseup 鼠标释放 onmousedown 鼠标按下

onmouseover 鼠标滑过（进入） onmouseout 鼠标离开

onmouseenter 鼠标进入 (不会冒泡) onmouseleave 鼠标离开 (不会冒泡)

**[五]、键盘事件：**

keyup 释放 keydown 任意键 keypress 字符键

**[六]、Window事件：**

load 加载 unload 卸载 error 错误

select 文本框选中 resize 重调窗口大小 scroll 滚动条

**[七]、焦点事件：**  focus 获取焦点 blur 失去焦点

**拓展：**实现的功能

**[1]、**表单的值的获取与赋值：(按照id名获取元素)

document.getElementById("id名").value

document.getElementById("id名").value=""

**[2]、**DIV盒子的值的获取与赋值：

document.getElementById("id名").innerHTML

document.getElementById("id名").innerHTML=""

**[3]、**获取单选框：(再给表单去同样的名字name的情况下)

\_name = document.getElementsByName("name字")

**[4]、**获取标签名：

spans=document.getElementsByTagName("标签名")

=============================================================

* **数组（Array类型）**-------引用类型

=============================================================

**[一]、数组的特点：**

数组指的是：数据的有序列表(一组有序的数据序列)。

数组的大小可以动态调整。

数组的每一项可以保存任何数据类型(比如：数组的第一个位置保存字符串，第二 个位置保存数值，第三个位置保存对象)。

**[二]、创建数组的基本方式有两种：**

(1).使用Array构造函数：var colors=new Array();

简写为：var colors= Array();

(2).使用数组字面量表示法：

如：var names=[ ] //创建一个空数组，长度为0

var colors= ["red","blue", "green" ]; //创建一个包含3个字符串的数组

**[三]、数组的长度：**

数组名.length 数组下标的索引是：length-1

**如：**var arr = ["red","blue", "green" ];

console.log( arr.length ) //数组长度为3,索引下标分别为arr[0],arr[1],arr[2]

**[四]、数组的赋值：** arr[0]="a", arr[1]="b";

**[五]、数组的方法：**

**(1).转换方法：**

①. toString( ) 或 toLocaleString( ) 返回字符串

②. valueOf( ) 返回数组

③. join("分隔符" ) 用不同的分隔符连接数组的每一项。转化为字符串

如：var arr=["a","b","c"];

console.log( arr.toString( ) ); //值为a,b,c

console.log( arr.valueOf( ) ); //值为["a","b","c"]

console.log( arr.join("") ); //值为abc

console.log( arr. join("/") ); //值为a/b/c

**(2).栈方法：**

①. 追加(逐个添加到数组末尾)：**数组名.push(值)** 返回值是修改后数组的长度

②. 删除(从数组末尾移除最后一项)：**数组名.pop( )** 返回值是移除后的元素

**例：** var arr = [1,7,3], \_array = [2,3,4]

arr.push(5); // arr.push(5) 长度为4， arr值为 [1, 7, 3, 5]

\_array.pop(); // \_array.pop(5) 删除4，\_array值为 [2,3]

栈数据结构访问规则是一种LIFO( last- in-first-out后进先出 )。

**(3).队列方法：**

①. 追加到最前面：**数组名.unshift(值)** 返回的是新数组的长度

②. 删除元素：**数组名.shift( )** 返回值是删除的元素

**例：** var arr = [1,7,3], \_array = [2,3,4]

arr.unshift(5); // arr.unshift(5) 长度为4， arr值为 [5, 1, 7, 3]

\_array.shift(); // \_array.shift(5) 删除2， \_array值为 [3, 4]

队列数据结构访问规则是一种FIFO( first - in-first-out先进先出 )

**(4).操作方法：**

①. **slice( )方法：**返回新数组（生成一个新数组，不会改变数组本身）

截取数组**：**数组名.slice(开始下标,结束下标) 包含开始下标,不包含结束下标[1,3)

数组名.slice(开始下标) 从开始下标到数组的最后一个元素

**例：** var arr = [1,7,3,2,5,4]; console.log( arr.slice(0,2) ) // 值为[1, 7]

console.log( arr.slice(1) ) // 值为[7, 3, 2, 5, 4]

②.**concat( ) 方法：**不生成新数组，改变的是数组本身

数组名1.concat(数组名2，数组名3......) 连接数组的方法

③.**splice( )方法：**生成一个新数组，不会改变数组本身

删除子数组**：** 数组名.splice(开始下标，长度) 返回值是要删除的子数组

数组名.splice(开始下标） 删除开始下标之后的所有数组对象

添加子数组**：**数组名.splice(开始下标, 0, 值1, 值2....)

替换子数组**：**数组名.splice(开始下标, 长度, 值1, 值2....)

**(5).重排序方法：**返回值是经过排序之后的数组

**arr.reverse()** 反转数组项的顺序(颠倒前后)

**arr.sort()** 升序排列数组项(按从小到大的顺序排)

**总结：**

**数组元素的添加：**返回数组新长度

数组名.push(值)

数组名.unshift(值)

数组名.splice(开始下标, 0, 值1, 值2....)

**数组元素的删除：**

数组名.pop( )

数组名.shift( )

数组名.splice(开始下标，长度)

**数组的合并：**

数组名1.concat(数组名2，数组名3......)

**数组的截取：**

数组名.slice(开始下标,结束下标)

数组名.substr(开始下标,长度)

数组名. substring(开始下标,结束下标)

**数组的拷贝:** 返回数组的拷贝数组，即返回新数组

数组名.slice(0) 或 数组名.concat()

**例：**var arr = [1,7,3,2,5,4];

console.log( arr.slice(0) )

console.log( arr.concat() )

=============================================================

* **String类型**

=============================================================

**String类型：**是字符串的对象包装类型。

**[一]、创建字符串的基本方式有两种：**

(1).使用String构造函数：如：var str=new String("info here");

(2).直接创建函数：如：var str= "info here";

字符串属性：length属性(表示字符串中包含多个字符)

**[二]、字符串的方法：**

**trim()方法：清空字符串前后空格的方法**

语法：函数名.trim( ) (删除前置及后缀的所有空格，返回字符串的副本)

如：var string= " hello world ";

var \_string = string.trim( );

console.log( string ); //值为" hello world "

console.log ( \_string ); //值为"hello world"

**[三]、字符方法：**

charAt( index ) (以单字符串的形式)返回字符串中指定索引位置的字符

charCodeAt( index ) 返回字符串中指定索引位置字符的ascii码值

如：var strV= "Hello World";

console.log ( strV.charAt (1) ); //"e" 位置1处的字符是"e"

console.log ( strV.charCodeAt (1) ); //"101" 小写字母e的ascii码值

**[四]、字符串的位置方法：(查找子字符串)**

indexOf( ) (从前往后查)子字符串在父字符串的位置，返回索引

lastIndexOf( ) 从后往前查找第一次出现的位置，返回其所索引

语法：数组名.indexOf( "子字符串" ) 或 数组名.lastIndexOf( "子字符串" )

说明：从一个字符串中搜索给定的子字符串，然后返回子字符串的索引位置。

若存在，返回索引位置； 若不存在，则返回-1。

**例：** var str = "abc123bc";

console.log( str.indexOf("bc") ) //索引位置为1

console.log( str.lastIndexOf("bc") ) //索引位置为6

**[五]、字符串的大小写转化方法：**

**(1). 转化为大写的：**

toUpperCase() 通用的方法

toLocaleUpperCase() 针对特定地区的

**(2). 转化为小写的：**

oLowerCase() 通用的方法

toLocaleLowerCase() 针对特定地区的

**如：** var stringV= "Hello World";

console.log( stringV . toUpperCase() ); //值为"HELLO WORLD"

console.log ( stringV . toLowerCase() ); //值为"hello world"

**[六]、字符串操作方法：**

**(1).合并字符串：**

数组名1.concat(数组名2，数组名3......) 连接多个字符串

**(2).分割(**切割**)字符串：**

split("") 一个或多个字符的分割为数组

split("分隔符" ) 以分隔符分割。

**(3).截取字符串：**查找子字符串的方法

①. slice( 开始下标 ) 开始下标到最后一个元素，下标可以是负数

slice( 开始下标, 结束下标 ) 开始下标到结束下标(包含开始,不包含结束 )

②. substr(开始下标) 开始下标到最后一个元素

substr( 开始下标, 长度 ) 开始下标起按长度查找

开始下标可以是负值，长度如果是负值会转化为0

③. substring( 开始下标, 结束下标 ) 开始下标到结束下标前一位

下标不能是负数，如果是负数，自动转化为0

**例：**var str= "abcdefghijklmn";

console.log( str.slice(2) ) //值为cdefghijklmn

console.log( str.slice(2,5) ) //值为cde

console.log( str.substr(2) ) //值为cdefghijklmn

console.log( str.substr (2,5) ) //值为cdefg

console.log( str. substring(2,5) ) //值为cde

=======================================================

* **数学对象( Math对象 )**

=======================================================

**Math对象的属性：**

|  |  |
| --- | --- |
| 属性 | 说明 |
| Math.E | 自然对数的底数，即常量e的值 |
| Math.LN10 | 10的自然对数 |
| Math.LN2 | 2的自然对数 |
| Math.LOG2E | 以2为底e的对数 |
| Math.LOG10E | 以10为底e的对数 |
| Math.PI | π的值 |
| Math.SQRT1\_2 | 1/2的平方根(即2的平方根的倒数) |
| Math.SQRT2 | 2的平方根 |

**Math对象的方法：**

**1. 最小值Math.min( )**和**最大值Math.max( )**的方法：

如: var mix= Math.min( 1, 24, 6) //最小值是1

var max=Math.max(51,9,-7) //最大值是51

**2. 舍入方法：**

①. Math.ceil( ) 向上舍入 **例:** Math.ceil(25.9) | Math.ceil(25.1) //值为26

②. Math.floor( ) 向下舍入 **例:** Math.ceil(25.9) | Math.ceil(25.1) //值为25

③. Math.round( ) 四舍五入 **例:** Math.ceil(25.5) //值26 Math.ceil(25.1) //值25

**3. random随机数的方法:**

①. Math.random()方法 返回的是[0 ,1)的一个随机数

**公式：**随机整数 = Math.round( Math.random( )\*(大值-小值) )+小值

**如：** 返回一个[50,100]的随机整数

Math.round( Math.random()\*(100-50) )+50

②. 绝对值方法： Math.abs( num ) num的绝对值

幂的方法： Math.pow(底数,指数)

平方根方法： Math.sqrt(100) 100的平方根10

=======================================================

* **日期对象(Date 类型)**

=======================================================

Date类型使用自 UTC（Coordinated Universal Time，国际协调时间）1970年 1月 1日午夜（零时）开始经过的毫秒数来保存日期。

Date 类型保存的日期能够精确到 1970年 1月 1日之前或之后的 285 616年。

**创建日期的方法：**

使用new操作符Date构造函数：var colors=new Date( );

如：var now=new Date();

// 若不传递参数，则新创建的对象将自动获得当前日期和时间

var now=new Date(1996,3,21);

// 根据特定的日期和时间创建对象，则必须传入表示该日期的毫秒数

（即从 UTC时间 1970年 1月 1日午夜起至该日期止经过的毫秒数）

**日期的方法：**

**1. 继承的方法：**

toLocaleString()方法会按照与浏览器设置的地区相适应的格式返回日期和时间，

但不会包含时区信息

toString()方法则通常返回带有时区信息的日期和时间，其中时间一般以军用时间

( 即小时的范围是 0 到 23 )表示

valueOf()方法，返回的不是字符串，而是返回日期的毫秒表示。可以方便使用比

较操作符（小于或大于）来比较日期值

**2. 获取日期/时间的方法：**

①. **年：**getFullYear( ) 取得4位数的年份（如2007而非仅07）

如： var year=now.getFullYear() ;

②. **月：**getMonth( ) 返回日期中的月份(其中0表示一月，11表示十二月)

如： var month=now.getMonth()+1;

③. **日(天)：**getDate( ) 返回日期月份中的天数（1到31）

如： var day=now.getDate();

④. **星期几：**getDay( ) 返回日期中的星期几(其中0表示星期日，6表示星期六)

如： var weekday=now.getDay();

⑤.**小时：**getHours( ) 返回日期中的小时数（0到23）

如： var hours=now.getHours();

⑥.**分钟：**getMinutes( ) 返回日期中的分钟数（0到59）

如： var minutes=now.getMinutes();

⑦.**秒：** getSeconds( ) 返回日期中的秒数（0到59）

如： var seconds=now.getSeconds();

⑧.**毫秒：** getMilliseconds( ) 或 getTime( ) 返回日期中的毫秒数

如：var minsd=now.getMilliseconds()

var now=new Date(); console.log( now.getTime() )

**3. 设置日期/时间的方法：**

①. **年：**setFullYear(年) 设置日期的年份

传入的年份值必须是4位数字（如2007而非仅07）

如： var year=now.setFullYear() ;

②. **月：**setMonth(月) 设置日期的月份(其中0表示一月，11表示十二月)

传入的月份值必须大于0，超过11则增加年份返回日期中的月份

如： var month=now.setMonth()+1;

③. **日(天)：**setDate(日) 设置日期月份中的天数（1到31）

如果传入的值超过了该月中应有的天数，则增加月份

如： var day=now.getDate();

④.**小时：**setHours(时) 设置日期中的小时数（0到23）

传入的值超过了23则增加月份中的天数

如： var hours=now.setHours();

⑤. **分钟：**setMinutes(分) 设置日期中的分钟数（0到59）

传入的值超过59则增加小时数

如： var minutes=now.setMinutes();

⑥. **秒：** setSeconds(秒) 设置日期中的秒数（0到59）

传入的值超过了59会增加分钟数

如： var seconds=now.setSeconds();

⑦. **毫秒：** getMilliseconds(毫秒) 设置日期中的毫秒数

如：var minsd=now.setMilliseconds()

**JaveScript 笔记二**

============================================================

* **节点**

============================================================

**[一]、节点的类型：**总共有12种节点类型

element元素节点 1 Attr属性节点 2 文本节点 3

document文档节点 9 注释节点 8

**[二]、节点属性：**

|  |  |  |
| --- | --- | --- |
| 节点的类型  nodeType | 节点的名字  nodeName | 节点的值  nodeValue |
| 元素节点 1 | head div h1(元素名) | null |
| 文本节点 3 | #text | 文本本身 |
| 属性节点 2 | class href id（属性名） | 属性值 |

判断是否是文本节点：①. nodeName=="#text" ②. nodeType==3

**[三]、查找节点：**

**(1). 查找Id名：** Id名 = document.getElementById("id名") 得到的是一个值

**(2). 查找name名：\_name =** document.getElementsByName("name名")

得到的是一组数据，长度为**\_name.length**

注：适用于获取表单 (给表单取同样name名字的情况下)

**(3). 查找元素 (标签)名：**标签名 = document.getElementsByTagName("标签名")

查找到的是一组数据，长度为 **标签名.length**

**如：** <div id="divBox">

<input type="radio" name="\_name ">

<input type="radio" name="\_name ">

<div>

<script>

var \_divBox = document.getElementById("divBox ");

var \_names = document.getElementsByName("\_name ");

var \_inputs = document.getElementsByTagName ("input");

</script>

**(4). className**（浏览器不兼容，IE9以下不支持）

**(5). 特殊集合：forms**  --->（form表单域）

①. 语法：document.forms( )

功能：获取页面中的所有form元素

说明：得到的是nodeList(节点列表)，使用时以数组的方式

②. 语法： forms.elements( )

功能：获取页面中的所有表单元素

说明：得到的是nodeList(节点列表)，使用时以数组的方式

**(6). CSS选择器**（浏览器不兼容，IE9以下不支持）

querySelector( ) 或 querySelectorAll( )

语法： document. querySelector("标签名")

document. querySelectorAll("标签名")

功能： querySelector( )获取通过CSS选择器只能查找到第一个节点

说明： querySelectorAll( )以数组的形式获取所有的节点

**[四]、节点操作方法：**

**1. 创建节点：**

①. 文本节点：var text = document.createTextNode("text")

②. 元素节点：var 元素名 = document.createElement("元素名")

**2. 添加(追加)节点：**

末 端： 父节点**.**appendChild(子节点) 子节点作为父节点最后一个元素存在

指定位置前： 父节点**.**insertBefore(新子节点，插入到哪个节点之前)

**3. 删除节点：**

父节点**.**removeChild(子节点)

**4. 替换节点：**

父节点**.**replaceChild(newNode，oldNode)

即：父节点.replaceChild(新节点，要替换的节点)

**5. 复制(克隆)节点：**

要复制的节点**.**cloneNode( [false /true] )

注：false (可不写)只复制结构，不复制内容；

true 既复制结构也复制内容。

**[五]、DOM中的HTML中的属性（Html特性）操作：**

**1.标准属性**：id、class、type、name、title、src 、href

(1).获取(得到)属性名： **语法：**对象**.**属性名 ---> 得到的是一个属性值

(2).设置(更改) 属性名： **语法：**对象.属性名=值;

**例：**<h2 class="sure" title="确定"></h2>

console.log( sure.title ) //打印结果为: 确定

console.log(sure.title="取消") //打印结果为: 取消

**2. 自定义属性**（也适用于标准属性）

data-\*提供自定义的属性，即data-\*实际上就是data-前缀加上自定义

的属性名，使用这样的结构可以进行数据存放。

(1).获取(得到)属性值： **语法：**对象.getAttribute("属性名");

(2).设置(更改)属性值： **语法：**对象.setAttribute("属性名",值);

(3).删除(删除)属性值： **语法：**对象.removeAttribute("属性名");

**例：**<h2 class="sure" data-title="确定"></h2>

sure. getAttribute("data-title")

sure. setAttribute("data-title","取消")

sure. removeAttribute("data-title")

**[六]、遍历节点(通过节点关系来找节点)：**

**父节点**： parentNode

**子节点：** 查找所有子节点 childNodes 获取子节点长度 childNodes.length

第一个子节点 firstChild 最后一个子节点 lastChild

**兄弟节点：**前一个兄弟节点 previousSibling

下一个兄弟节点 nextSibling

**拓展：**

1、居中 第一种方法：left=(父元素的宽-当前元素的宽)/2

第二种方法：left=50%，margin-left: -(当前元素的宽/2)

如： left=50%; margin-left: -100px;

top =50%; margin-top: -100px;

2、找html：document.documentElement

找body：document.body

=============================================================

* **表格**

=============================================================

**[一]、表格的基本结构：**

表格：<table></table>

头部：<thead></thead>

主体：<tbody></tbody> //可有多个主体

表头：<th></th> //相当于标题

行标签：<tr></tr>

列标签(单元格标签)：<td></td>

底部：<tfoot></tfoot>

**[二]、表格标记的属性：**

①. width、height属性：设置表格的宽度和高度

②. border属性：设置表格的外边框宽度，值为数字

③. cellspacing间距属性：设置单元格之间的距离,默认2像素

④. cellpadding边距属性： 设置单元格边框与内容之间的距离

⑤. bgcolor背景颜色属性:设置整个表格的背景颜色

⑥. background背景图像属性：设置表格的背景图

1. **表格行标记是<tr></tr>**

①. **align**属性：设置行内容的**水平对齐方式**，属性值有left、center、right

②. **valign**属性：设置行内容的**垂直对齐方式**，属性值有top、middle、bottom

2. **单元格标记是<td></td>**

①.. colspan属性： 跨列合并,即设置水平合并的单元格数

②. rowspan属性：跨行合并,即设置垂直合并的单元格数

**[三]、表格查找：** 在函数使用的是全局变量的最终值

查找 thead 语法：table. tHead

查找 tfoot 语法：table. tFoot

查找 tbody 语法：table. tBodies 得到的是nodeList(节点列表)

查找行tr 语法：父元素**.rows** 得到的是nodeList(节点列表)

查找列td 语法：父元素**.cells** 得到的是nodeList(节点列表)

**[四]、表格操作：**

1. 添加表格元素： 注：index索引

行：insertRow( index ) 在指定位置处插入行

列：insertCell( index ) 在指定位置处插入行

2. 删除表格元素：

行：deleteRow( index ) 删除指定位置处的**行**

列：deleteCell( index ) 删除指定位置处的**列**

**注：**若只删除一个，则按照顺序或倒序的顺序来删除都可以；

若一次性删除多个，则只能按照倒序的顺序来删除。

3. 说明 : 以上几个方法都会返回一个对象

**拓展：** 1、提交按钮：

通用提交按钮 <input type="submit" value="Submit Form">

自定义提交按钮 <button type="submit">Submit Form</button>

图像按钮 <input type="image" src="graphic.gif">

2、在函数中使用的是全局变量的最终值

3、placeholder与value的区别：

①. placeholder=” ” 提示文本(提示文字)--->占位符

②. value="" 默认值 =============================================================

* **样 式**

=============================================================

**拓展：**

在html中css样式分为四种：行内样式 内嵌样式 链接样式 导入样式

在JS中样式分为两种：①. 行内样式；②. 非行内样式，即内嵌样式和链接样式。

**[一]、 行内样式操作：**

**获取：** 语法1: 对象.style.css属性

语法2: 对象.style["css属性"] //[ ]中的属性可存入到变量或参数中

注：获取的属性值返回的都是字符串

**设置：** 语法1: 对象.style.css属性=值

语法2: 对象.style["css属性"] =值 //[ ]中的属性可存入到变量或参数中

**[二]、 非行内样式操作：**

**获取：** 1、WEB标准浏览器支持：getComputedStyle

语法1: getComputedStyle( obj , null )**.**css属性名

语法2: getComputedStyle( obj , null )["css属性名"]

2、IE浏览器支持：currentStyle属性

语法1: obj. currentStyle.css属性 //obj指对象

语法2: obj. currentStyle["css属性"]

**[三]、元素大小**

**1. 偏移量：**

①. 元素的位置：

offsetLeft：元素左外边框至父元素左内边框的距离

offsetTop：元素上外边框至父元素上内边框的距离

**注：**offsetTop = marginTop+top

top是指绝对定位时对上面的高度 ; marginTop与top可以使用其中的一个

②. 元素的尺寸：

offsetWidth 元素的宽(元素在垂直方向上占用的空间大小)

offsetHeight 元素的高(元素在水平方向上占用的空间大小)

**注：** offsetHeight=height+padding+border 获取元素的实际高度

**2、客户端大小**(浏览器的大小)指的是 : 元素内容及其内边距所占据的空间大小

①. html对象.clientWidth 视口宽度(浏览器窗口宽度)

②. html对象**.**clientHeight 视口高度(浏览器窗口宽度)

**注：** html对象指：document.documentElement

**拓展：**pageX | pageY 或 offsetX | offsetY --->**（整个页面）**

screenX | screenY --->**（屏幕上的坐标）**

clientX | clientY --->**（可视区域）**

**3、滚动大小**

①.尺 寸： scrollHeight：在没有滚动条的情况下，元素内容的总高度

scrollWidth：在没有滚动条的情况下，元素内容的总宽度

②.滚动距离：scrollLeft：水平滚动的距离

scrollTop：垂直滚动的距离，即页面向上滚动或顶部隐藏的距离

=============================================================

* **事 件对象**

=============================================================

**[一]、事件对象：**指的是只要有事件，就有事件对象，保存了与事件相关的信息，比如：时间

目标，事件类型等。事件对象可用event、even、e中的任何一种表示。

**[二]、访问事件对象：**

标准浏览器：做事件函数的第一个参数**,**用event、even、e中的任何一种表示

IE浏览器：用window.event表示

解决兼容问题：event = event || window.event

**[三]、事件对象的方法：**

**1、阻止事件冒泡：**

①. WEB标准浏览器支持：stopPropagation( );

**语法：**event.stopPropagation( );

②. IE浏览器支持：cancelBubble属性

**语法：**event.cancelBubble =true;

**注：**若event.cancelBubble = true;为真，则是阻止事件冒泡

**2、取消默认行为(动作)：**

①. WEB标准浏览器支持：preventDefault( );

**语法：**event.preventDefault( );

②. IE浏览器支持：retutn false 或 retutnValue = false

**语法：**retutn false 或 retutnValue = false

**3、事件目标( 事件源 )：**指event对象

①. 标准浏览器支持：target 获取目标时,实际触发该事件的元素(只获取值不能更改值)

②. IE浏览器支持：srcElement

解决兼容问题：var event = event || window.event, //访问事件对象

target = event.target || event.srcElement; //触发源对象

**[四]、跨浏览器事件处理程序：**

**添加：**addHandler( ) **删除：**removeHandler()

**注：**以上两个方法是EventUtil对象,接受3个参数: 要操作的元素、事件名称、事件处理程序函数

**例：**EventUtil.addHandler( btn,"onclick",function( ){} ); //btn通过Id名或标签名查找

**[五]、事件委托：**

**原理：**利用冒泡的原理，把事件加到父级上，触发执行效果

**作用：**提高性能；针对新创建的元素，直接可以拥有事件

**使用情景：** 为DOM中的很多元素绑定相同事件；

　　 为DOM中尚不存在的元素绑定事件；

**例：**document.querySelectorAll('ul').addEventListener( 'onclick', 'li', false );

**注：** false 事件捕获(use capture)； true 事件冒泡

**[六]、事件类型type**

**1. UI事件类型：**指的是不一定与用户操作有关的事件

load 加载 unload 卸载 error 错误

select 文本框选中 scroll 滚动条

resize 重调窗口大小(浏览器窗口大小发生改变时触发该事件)

**2. 焦点事件：**(主要用于表单)

focus 获取焦点时触发该事件 blur 失去焦点时触发该事件

**拓展：**获取表单元素：

①. elements获取所有的表单元素

②. document.formName.eleName

说明：formName表单域的Name，eleName表单元素的Name

**3. 键盘事件：**

任意键keydown 按下任意键时触发该事件，如果不松开会重复触发

字符键 keypress 按下字符键时触发该事件，如果不松开会重复触发

释 放 keyup 当用户释放按键是时触发该事件

键 码 keyCode 获取按键的键值(Ascall码)

**拓展：** 键值属性keyCode 语法：event. keyCode

键码：A ~ Z ==> 65~90 a ~ z ==> 97~122

**4. 鼠标事件：**

onclick 单击 ondbclick 双击 onmousemove 鼠标移动

onmouseover 鼠标滑过 onmouseout 鼠标离开

onmousedown 鼠标按下 onmouseup 鼠标释放

onmouseenter 鼠标进入 (不冒泡) onmouseleave 鼠标离开(不冒泡)

=============================================================

* **正则表达式**

=============================================================

**[一]、正则表达式概念**

正则表达式：是由字母、数字和一些特殊符号构成的描述字符模式的对象。

功能：模式的匹配，文本的检索 及 替换功能。

**[二]、创建正则表达式**

**1.显示创建：** 通过 new RegExp（）构造函数来创建

**语法：**var reg = new RegExp("字符模式"，"修饰符")【实例对象】

**注：**变量名reg是正则表达式，数据类型是object对象

**2.隐示创建** 通过/ /字面量方式来创建，常用

**语法：**var reg = /字符模式/修饰符 【单个对象】

**[三]、正则表达式方法**

**(1). test( )方法**

**语 法：**reg**.**test( 被检测的内容 )

**功 能：**用于检测内容是否与正则相匹配

**返回值：**为布尔值，匹配返回true，不匹配返回false。

**(2). exec( )方法**

**语 法：**reg**.**exec( 被检测的内容 )

**功 能：**用于检测内容是否与正则相匹配

**返回值：**返回第一个匹配的字符串内容，匹配返回数组，不匹配返回null。

**说 明：**不支持全文匹配（不支持g模式）,支持index及input属性。

**注：**只可以用于检测正则表达式的方法是exec( )、test( )

**例：**var reg = new RegExp("abc"),

str = "3abc4,5abc6",

string = "3ab4,5abc6";

console.log( reg.test( str ) ); //值为true

console.log( reg.test( string ) ); //值为true

console.log( reg.exec( str ) ); //值为["abc", index: 1, input: "3abc4,5abc6"]

console.log( reg.exec(string ) ); //值为["abc", index: 6, input: "3abc4,5abc6"]

**[四]、修饰符**

①. **i** 模式 忽略大小写

②. g模式 全文匹配

③. m模式 多行匹配

**[五]、特殊符号**（元字符、直接量）：具有特定功能的符号。

①. [……] 任意字符，匹配括号内的任意一个字符

②. [^……] 匹配未包含(除了括号内以外)的任意字符

③. \w 匹配字母数字下划线。等价于[A-Za-z0-9\_]

④. \W 匹配非字母数字下划线。等价于[^A-Za-z0-9\_ ]

⑤. \d 匹配一个数字字符。等价于[0-9]

⑥. \D 匹配一个非数字字符。等价于[^0-9]

⑦. \s 匹配任何空白字符(空格、制表符、换页符)。等价于[\f\n\r\t\v]

⑧. \S 匹配任何非空白字符。等价于[^\f\n\r\t\v]

⑨. \f 换页符 \n 换行符

⑩. \r 回车符 \t 制表符

⑪. 点**·** 代表任意字符，除了换行符

**例：** '[abc]' 可以匹配 "plain" 中的 'a'；

'[^abc]' 可以匹配 "plain" 中的'p'、'l'、'i'、'n'。

**[六]、重复设置匹配的次数**

①. **{n}** 表示匹配前一项 n 次

②. **{n,m}** 表示匹配前一项 n 到 m 次（最少n次,最大m次）

③. **{n,}** 表示匹配前一项，最少 n 次，最大不限

④. **?** 表示匹配前一项 0 - 1次

⑤. **\*** 表示匹配前一项 0到多次

⑥. **+** 表示匹配前一项 1到多次

**[七]、限定符**

①. ^XX 表示以XX开头

②. XX$ 表示以XX结束

**[八]、转义符：**将特殊符号恢复本身字面意思

需要转义的字符有：**？ + \* [] {} () . \ ^ $** 【在特殊符号前加反斜杠 \ 】

**[九]、选择：** **" | "**指的意思是：**或者**。 即x | y 匹配 x 或 y

**例：** 'z|food' 能匹配 "z" 或 "food"

'(z|f)ood' 则匹配 "zood" 或 "food"

**[十]、字符串支持正则的方法**

**(1). search()方法**

**语 法：**str.search( reg ) **注：**str指字符串对象，reg指正则表达式

**功 能：**可按正则表达式查找字符串

**返回值：**找到返回匹配元素的位置索引，找不到返回-1

**说 明：**当查找的内容不为正则表达式时，会自动换换成正则去进行查找。

不支持 g 模式(不全文匹配)

**(2). replace()方法**

**语 法：**str.replace( reg/substr，替换的内容 ) **注：**substr指字符串

**功 能：**用于查找并替换的操作

**返回值：**该方法不会直接替换原字符串，而是返回一个新(的替换后的)字符串。

**说 明：**支持g 模式(全文匹配) 。查找的内容：可以直接是字符，也可以是正则

**注 意：**若reg具有全局标志g，那么 replace()方法将替换所有匹配的子字符串；

否则，它只替换第一个匹配子字符串。

**(3). split()方法**

**语 法：**str.split( 分隔符/正则 )

**功 能：**按分隔符将字符串转换为数组。

**(4). match()方法**

**语 法：**str.match( reg )

**功 能：**验证字符串是否与正则表达式匹配，即验证匹配。等同于exec( )

**返回值：**匹配返回数组，不匹配返回 null

**说 明：**支持g 模式(全文匹配)，返回的数组中包括字符串中的所有匹配结果

**[十一]、分组：**

小括号( ) 将多个正则组合成一个整体，既分组又记忆

(?**:**正则表达式) 只分组不记忆

**拓 展：** 匹配中文字(汉字)符： [\u4e00-u9fa5]

匹配双字节字符(包括汉字在内)**：** [^x00-xff]

=============================================================

* **函 数**

=============================================================

**[一]、函数**

(1).函数的定义： 实现某一功能的多条代码或语句。

可以在任何时候、任何地方调用（随时随地）。

(2). 函数没有重载，函数不能重名，后写的函数覆盖先写的函数。

**[二]、函数声明关键字：**function

**语法：**function 函数名( [参数] ){ 语句体 }

**例：**function sum( num1,num2,…… ){

函数体；

}

**特点：**声明提升，在执行脚本时，首先执行函数声明语句，再去执行其他语句，调用函

数放在声明函数前后都可以。

**[三]、函数表达式：**先定义再调用

**语法：**var 函数名 = fn； **注：** fn可以是匿名函数，也可以是有名函数

**例：**var sum= function sum(num1,num2,……){

函数体；

}

**注：**函数表达式中调用函数一定要放在函数表达式之后。

**[四]、调用函数的三种方法：**

①. 直接调用（在js中）： 函数名(实际参数1，实际参数2....){ 语句体 }

②. 鼠标事件调用（html网页中）:

<input type="button" value="计算" onclick="函数名(实参1，实参2)">

③. 写在js里不带参数 ( DOM0事件处理程序 )： btn.onclick=function( ){ }

**[五]、函数的返回值**（return 返回）

用 法：return 变量名;

说 明：return true;：返回正确的处理结果。 return;：将把握权返回给页面。

return false;：返回错误的处理结果以及组织代码继续向下执行。

返回值：return 返回值是undefined。

**[六]、函数的作用域** （执行环境）

全局变量：可在全局环境中使用，可被任何函数使用。

局部变量：在局部环境中使用，不能被其他函数使用。

**拓展：**

**(1). 递归函数**

定义：通过函数名称调用自身的情况。以函数表达式实现递归最安全。

arguments.callee指向正在执行的函数，在严格模式下不支持。

语法：Jc = function fn(n){

if( n==1 ){

return 1;

}else{

return n\*fn(n-1);

}

}

**(2).闭包函数**

定义：能够读取其他函数内部变量的函数。

语法：function out( ){

var a = 10;

function in( ){

alert( a )

}

return in;

}

用途：可以读取函数内部的变量。使变量始终保持在内存中。

优点：不增加额外的全局变量。执行过程中所有变量都在函数内部。

缺点： 由于闭包使的函数中的变量都保持在内存中，内存消耗很大，所以不能

滥用闭 包。否则会造成网页性能问题。

在IE中可能导致内存泄露。**解决方案：** 在退出函数前，将不用的局部变

量删除闭包会在父级函数外部改变函数内部变量的值，这是一定要小心，不 用随便改变父函数内部变量的值。

=============================================================

* **闭 包**

=============================================================

**[一]、闭包**

**定 义：**闭包即能够读取其它函数内部变量的函数，也可以把闭包简单的理解成 "定义在

一个函数内部的函数"。

**用 途：**①.可以读取函数内部的变量； ②. 是让变量始终保持在内存中。

**优 点**：①.不增加额外的全局变量； ②. 执行过程中所有变量都在匿名函数内部。

**缺 点**：①.闭包会在父函数外部，改变父函数内部变量的值，一定要小心

②.闭包会使得内存消耗有可能会很大，可能导致内存泄露

**[二]、使用闭包注意点**

①. 由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭

包，否则会造成网页的性能问题,在IE中可能导致内存泄露。解决方法是，在退出

函数前，将不使用的局部变量全部删除。

②. 闭包会在父函数外部改变父函数内部的变量的值。所以，如果你把父函数当作对象

( objec ) 使用，把闭包当作它的公用方法（Public Mthod），这时一定要小心，

不要随便改变父函数内部变量的值。

**拓展：**用if语句判断正负值

if( iTarget >speed ){

正值

}else{

负值

}

**例：** -134 0 正值 0 -134 负值

=============================================================

* **对 象**

=============================================================

**[一]、对象：**由属性和方法构成。

属 性：属性即变量，用于存取数据。

方 法：方法即函数，用于完成特定功能。

**[二]、对象分类：**全局(Global)对象、内置对象以及自定义对象

内置对象包括：String 字符串对象

Math 数学对象

Array 数组对象

Date 日期对象

RegExp 正则表达式对象

全局(Global)对象：即window对象

所有在全局作用域中定义的属性和函数，都是全局(Global)对象的属性

全局(Global)对象的函数(方法)： isNaN( ) isFinite( )

paseInt( ) paseFloat( ) setInterval( ) setTimeout( )

alert( ) confirm( ) clearInterval( ) clearTimeout( )

**[三]、创建对象：**

**1. 创建自定义对象方法**：通过Object()构造函数 或 字面量方式创建

**(1). 显式创建：**通过Object()构造函数来创建

**语 法：**var objName = new Object( ); **注：**objName指对象名

**添加属性和方法：**

添加属性：objName.attrName = value; 对象名.属性名=属性值

添加方法：objName.methodName = fn; 对象名.方法名=函数

**(2). 隐式创建：**字面量创建方式

**语 法：**var objName={ };

**添加属性和方法：**

添加属性：attrName : value 属性名 : 属性值

添加方法：methodName : fn 方法名 : 函数

**说 明：** attrName：自定义属性名； methodName：自定义方法名;

value：属性值； fn：函数（有名函数或匿名函数）

属性和属性和方法之间都用逗号；最后结束不加符号

**注：** Object构造函数或对象字面量适用来创建单个对象(单体对象)

this：在自定义对象中this即该对象

**缺 点：**使用同一接口创建很多对象，会产生大量代码。

**解决方法：**使用工厂模式。

**2. 工厂模式：**

**语 法：**function createObject( [参数] ){

var person{ //通过字面量创建一个对象

person.name=value，

person.fangfa=function(){

alert(this.name)

}

}

return person

}

**缺 点：**没有解决对象识别的问题（即怎么知道一个对象的类型）

**解决方法：**构造函数模式

**3. 构造函数模式：**在构造函数中this指的是对象的实例

**语 法：**function CreateObject(name,age){

this.name = name;

this.age = age;

this.job = function( ){

alert( this.name )

}

}

**与工厂模式的区别：** 1. 没有显示的创建对象；

2. 将属性和方法都作为this对象的属性和方法；

3. 没有return语句；

4. 构造函数相当于其他语言中的类。

**说 明：**构造函数，即函数，必须通过new运算符来调用

用来创建对象时，构造函数的首字母要大写

**注：** 构造函数创建的实例，构造函数中的属性和方法都是每个实例独有的属

性和方法，这些属性和方法不共享

**缺 点：**就是每个方法在每个实例对象上重复创建

**解决方法：**原型模式

**拓展：**实例与this对象

**实 例：**

定 义： 通过构造函数创建出来的对象叫实例（或实例对象）。

属 性：constructor（构造函数）属性

识别对象类型：

①. 实例.constructor

②. instanceof运算符

语法：实例 instanceof 构造函数

功能：检测该实例对象是否是指定的构造函数的实例

返回值：布尔值，即true代表是，false代表不是

**this对象：**

①. 在事件中this指的是绑定事件的元素

obj.onclick=function(){

this //this指绑定事件的元素obj

}

②. 在全局作用域中this指的是window对象

setInterval( function( ){

this //this指window对象

} ,2000)

③. 在自定义对象中this即该对象

var newObj={

this //this指对象本身

}

④. 在构造函数中this指的是对象的实例

function CreateObject( ){

this.x = 1; //this指实例

}

var newsObj = new CreateObject( );

console.log( newsObj.x );

**注：**构造函数，生成一个新的对象，通过new运算符来调用。

**4. 原型模式：**

原型： 每个函数都有一个prototype(原型)属性；这个属性指向一个原型对象；

原型这个对象的用途是包含实例共享的属性和方法

访问原型：函数名. prototype

原型模式：function 构造函数名( ){ }

原型.属性名 = 值;

原型.方法名= function( ){ };

**语 法：** function CreateObject(name,age){

CreateObject.prototype.name=value;

CreateObject.prototype.age=value;

CreateObject.prototype.job=function( ){

alert(this.name);

}

}

**注：** 当实例和原型中有相同的属性和方法时，优先使用实例中的属性和方法，原

型中的属性和方法就被忽略。

**缺 点：**原型中所有的属性是被很多实例共享的。

**5. 组合模式：**将构造函数和原型组合起来的模式

原则：将独立的属性作为构造函数模式来创建，

将共享的属性和方法作为原型模式来创建。

**总结：javaScript对象的几种创建方式：**

工厂模式 构造函数模式 原型模式 组合模式(混合构造函数和原型组)

动态原型模式 寄生构造函数模式 稳妥构造函数模式

**[四]、for in语句**

遍历对象获取里面的属性和属性值

语法：for(初始值 in 对象名){ }

案例：for(var i=0 in fruit){

i代表属性名

获取属性值：对象[属性名]

}

**知识拓展：**

1. 已知c1,c2是由Car构造函数创建的对象， 则用isPrototypeOf( )用来检测c1,c2的

原型是否是 Car.prototype,若是返回真，否则返回假。

2. hasOwnProperty()方法用于检测某个属性是否属于某个实例自己的属性，而不是原

型中的属性；属于某个实例的属性，而不属于原型的就返回真，否则返回假；格式

为对象实例.hasOwnProperty("属性名")。

3. hasOwnProperty()和in操作符理解正确hasOwnProperty()返回只属于某个对象实

例的属性，而不是原有型中的；in判断无论属于对象实例，还是原型中的属性，都

返回true。

4. 判断某个实例是否属于某个构造函数的方法instanceof。

5. 确定原型和实例之间的关系isPrototypeOf()和instanceof。

6. prototype.constructor含义指向某个构造函数。

=============================================================

* **面向对象**(Object-Oriented,OO)----继承

=============================================================

**[一]、概论**

继承：用于重置对象。

OO语言支持两种继承方式：接口继承和实现继承【JS支持】

**[二]、继承方式**

**(1). 原型链继承：**利用原型让一个应用类型继承另一个引用类型的属性和方法。

语法：subType.prototype=new SuperType()

缺点：①. 最主要的问题来自包含引用类型值的原型，在通过原型来实现继承时，

原型实际上会变成另一个类型的实例。

②. 创建子类型的实例时，不能向超类（父类）的构造函数中传递参数。

案例：function SuperType( ){

this.car="别克";

this.house="别墅";

}

SubType.prototype.job = function( ){

alert("我是一个老板");

}

function SubType( ){ };

SubType.prototype = new SuperType( );

var son = new SubType( );

**(2). 借用构造函数：** 在子类构造函数的内部调用超类构造函数，通过使用apply( )、

call( )方法可以在新创建的对象上执行构造函数。

语 法：function SubType( ){

SuperType.call(this)

}

优 点：在子类中向父类传参。

缺 点：方法都在构造函数中定义，因此函数复用就无从谈起。

**(3). 组合继承：**将原型链和借用构造函数组合起来使用。

语法：function SubType(car，house){

SuperType.call(this，car，house)

}

SubType.prototype=new SuperType();

**(4). 原型式继承：**必须有一个对象可以作为另一个对象的基础。

语法：function object(o){

function F(){}

F.prototype=o;

return new F();

}

**总结：javaScript继承的6种方法：**

原型链继承 借用构造函数继承 组合继承(原型+借用构造)

原型式继承 寄生式继承 寄生组合式继承