# JavaScript简介

## JavaScript由来

JavaScript由Netscape 在1995年发明。早期的主要目的是处理一些用户的输入验证操作。而在JavaScript语言出现之前客户端的页面时需要提交到服务器端，由服务器去检测的。在刚刚普及的电话线调制解调器时代，对用户是一种考验，著名的Netscape Navigator（早期浏览器） 通过引入JavaScript来解决该问题

随着互联网的流行，网页已经不断变得更大和复杂，如果用户想要注册表单，需要直接将表单提交到服务器进行验证，需要和服务器进行多次的往返交互，例如，用户注册一个表单，点击提交按钮，等待30秒服务器返回处理后，返回的是用户名不符合规则。这种用户体验是很不友好的。此时的Netscape（网景）开始着手解决这个问题。

Netscape在 1995 年发行的 Netscape Navigator 2.0 开发一个称之为 LiveScript 的脚本语言，当时的目的是在浏览器和服务器（本来要叫它 LiveWire）端使用它。后来Netscape和Sun公司合作，所以最后Netscape 与 Sun 及时完成 LiveScript 实现。就在 Netscape Navigator 2.0 即将正式发布前，Netscape 将其更名为 JavaScript，目的是为了利用 Java 这个因特网时髦词汇，此后JavaScript 从此变成了因特网的必备组件。

三足鼎立

微软进军 微软决定进军浏览器，发布了 IE 3.0 并搭载了一个 JavaScript 的克隆版，叫做 JScript（这样命名是为了避免与 Netscape 潜在的许可纠纷）。

在微软进入后，有 3 种不同的 JavaScript 版本同时存在：Netscape Navigator 3.0 中的 JavaScript、IE 中的 JScript 以及 CEnvi 中的 ScriptEase。JavaScript 并没有一个标准来统一其语法或特性，而这 3 种不同的版本恰恰突出了这个问题。随着业界担心的增加，这个语言的标准化显然已经势在必行。

标准化

1997 年，JavaScript 1.1 作为一个草案提交给欧洲计算机制造商协会（ECMA）。第 39 技术委员会（TC39）被委派来“标准化一个通用、跨平台、中立于厂商的脚本语言的语法和语义” 。由来自 Netscape、Sun、微软、Borland 和其他一些对脚本编程感兴趣的公司的程序员组成的 TC39 锤炼出了 ECMA-262，该标准定义了名为 ECMAScript 的全新脚本语言。随后，国际标准化组织及国际电工委员会（ISO/IEC）也采纳 ECMAScript 作为标准。

从此，Web 浏览器就开始努力将 ECMAScript 作为 JavaScript 实现的基础。

JavaScript 是属于网络的脚本语言！JavaScript 被数百万计的网页用来改进设计、验证表单、检测浏览器、创建cookies，以及更多的应用。是因特网上最流行的脚本语言。

注：

javascript运行必须依赖于宿主环境语言，即页面语言HTML。

是解释型的语言，解释型：不需编译，解释器程序会每读取一条语句就执行。运行速度慢，浏览器中默认内置了javascript的解释器程序。

常见的脚本语言：

perl, Python, PHP, tcl, guile, ruby。

UNIX 上的各种 shells, Windows 上的 visual basic

ECMASCRIPT主要进行所有脚本语言的标准制定。

JavaScript

JScript

VBScript

ActionScript

JavaScript是基于对象和事件的脚本语言。

★用这些语言写的程序, 执行方式都以解译 (interpreting) 为主, 执行效率也许不如 C, C++, Java, Pascal, 等以编译 (compiling) 为主的程序, 但在研发速度上往往比后者快上数倍。★

## JavaScript特点

1. 安全性（不允许直接访问本地硬盘），可做的是信息的动态交互。

不被允许访问本地的硬盘，且不能将数据存入服务器，不允许对网络文档进行修改和删除，只能通过浏览器实现信息浏览或动态交互。从而有效地防止数据的丢失或对系统的非法访问。

1. 跨平台性。（只要是可以解释Js的浏览器都可执行，和平台无关。）

JavaScript依赖于浏览器本身，与操作环境无关，只要计算机能运行浏览器，并支持JavaScript的浏览器，就可正确执行，“编写一次，走遍天下”。

## JavaScript与Java不同

1. JS是Netscape公司的产品，Java是Sun公司的产品
2. JS是基于对象，Java是面向对象。

JS也可被看作是一种面向对象的语言，意味着JS能运用其已经创建的对象。因此，许多功能可来自于脚本环境中对象的方法与脚本的相互作用。

1. JS只需解释就可以执行，Java需要先编译成字节码文件，再执行。

JS提供了方便的开发过程。JS的语法基本结构形式与C、C++、Java十分类似。但在使用前，不像这些语言需要先编译，而是在程序运行过程中被逐行地解释。JavaScript与HTML标识结合在一起，从而方便用户的使用操作。

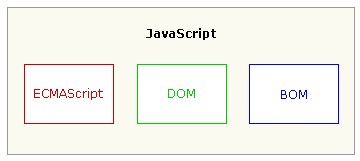
1. JS是弱类型，Java是强类型。

JS简单性体现在：

首先，JS是一种基于Java基本语句和控制流之上的简单而紧凑的设计，从而对于使用者学习Java或其他C语系的编程语言是一种非常好的过渡，而对于具有C语系编程功底的程序员来说，JavaScript上手也非常容易；

其次，其变量类型是采用弱类型，并未使用严格的数据类型。

## JavaScript内容



JavaScript语法基础

使用JS操作网页（DOM）

使用JS操作浏览器（BOM）

# JavaScript基础

## 语法

### 引入方式与引入位置

1. 向HTML页面插入JavaScript:

(1)直接在HTML页面中嵌入JavaScript代码:

JS代码存放在标签对<script>．．．</script>中。

(2)包含外部的JavaScript文件:

使用script标签的src属性引入一个js文件。（方便后期维护，扩展）

例：<script src=”test.js” type=”text/javascript”></script>

type属性必须

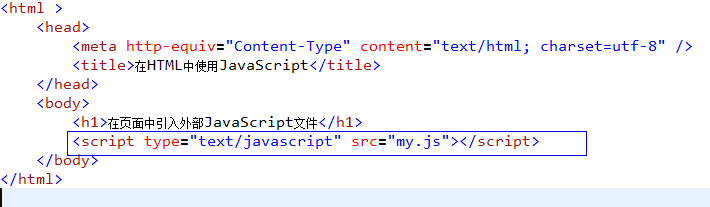
2.例子

(1)内部

|  |
| --- |
| <html >  <head>  <meta http-equiv**=**"Content-Type" content**=**"text/html; charset=utf-8" />  <title>在HTML中使用JavaScript</title>  </head>  <body>  <h1>在页面中嵌入JavaScript</h1>  <script type**=**"text/javascript">  window.document.write("hello,world");  </script>  </body>  </html> |

(2)外部

html文件



JavaScript文件



JavaScript代码



1. <script>标签

应该放在页面的<head>元素中。

(1)页面上可有多个

(2)按顺序执行

(3)可出现在任意的页面位置

(4)一定要写</script>关闭。否则没有任何错误信息，但是没有运行结果。

### 区分大小写

与 Java 一样，变量、函数名、运算符以及其他一切东西都是区分大小写的。

### 注释

|  |
| --- |
| Java： // /\* \*/ /\*\* \*/  HTML： <!-- -->  CSS： /\* \*/  JavaScript：// /\* \*/ **与 Java语言的注释相同** |

## 常用函数

window.alert() 或写为 alert() ：显示一个提示框显示内容。

window.document.write() ：在网页的当前位置处写内容。

## 变量

1. **弱类型的**

ECMAScript 中变量无特定类型，定义变量时只用 var 运算符，可将它初始化为任意值。

**例子**

|  |
| --- |
| var color = "red";  var num = 25;  var visible = true; |

|  |
| --- |
| var message;  var message ="hi"; |

1. 同时，可随时改变变量所存数据的类型（尽量避免这样做）。

|  |
| --- |
| <script type**=**"text/javascript" >  //定义变量  **var** message = "hello";  message = 100;  alert(message);  </script> |

3. 细节：

(1)var关键字在定义变量的时候可省略不写

(2)变量名可以重复，后面的将覆盖前面的变量

(3)变量类型取决于值类型

(4)一条语句定义多个变量。中间用逗号隔开即可，可初始化，也可不初始化。

|  |
| --- |
| <script type**=**"text/javascript" >  **var** name = "jack", age = 28, gender = "男";  </script> |

4.JS代码块及其代码块变量作用域

(1)例1

|  |
| --- |
| <script type="text/javascript" >  //代码块 (Java)对代码进行封装 隔离变量的作用域  {  var a = 100;  document.write(a + "</br>"); //100  } {  //JS不会隔离变量作用域  document.write(a + "</br>"); //100    }  </script> |

(2)例2

|  |
| --- |
| <script type="text/javascript" >  /\*function test(){  for(var i=0;i<3;i++){  }  alert(i); //结果：3  //可见，在块外，块中定义的变量i仍可访问。  //也就是说，JS并不支持块级作用域，它只支持函数作用域，且在一个函数中的任何位//置定义的变量在该函数中的任何地方都是可见的。  };  test(); \*/  function test(){  (function (){ //闭包 相当于给for循环的外层添加了一个函数作用域  for(var i=0;i<4;i++){  }  })();  alert(i); // i is not defined  }  test();  </script> |

## 数据类型

### typeof操作符

1. 4种基本数据类型: Undefined, Boolean,Number和String
2. Typeof:可获取变量类型

|  |
| --- |
| <script type**=**"text/javascript" >  **var** a = 100;  **var** b = 3.14;  **var** c = true;  **var** d = 'a';  **var** e = "hello";  **var** f;  document.write(a + "<br/>"); //100  document.write(b + "<br/>"); //3.14  document.write(c + "<br/>"); //true  document.write(d + "<br/>"); //a  document.write(e + "<br/>"); //hello  document.write(f + "<br/>"); //undefined  document.write("<hr/>");  document.write( typeof a + "<br/>"); //number  document.write( typeof b + "<br/>"); // number  document.write( typeof c + "<br/>"); // boolean  document.write( typeof d + "<br/>"); // string  document.write( typeof e + "<br/>"); // string  document.write( typeof f + "<br/>"); // undefined  </script> |

3.类型总结：

数值都是number型

字符和字符串都是string类型

布尔是boolean类型

若变量没有初始化值，其类型为undefined。表示未定义。

4.JavaScript中数值的表示形式：

|  |
| --- |
| <script type**=**"text/javascript" >  **var** a = 12;  **var** b = 012; //八进制 10  **var** c = 0x12; //十六进制 18  document.write(a + "<br/>");  document.write(b + "<br/>");  document.write(c + "<br/>");  </script> |

细节：

(1)八进制中如果数据超过0~7，默认解析为十进制数据。

|  |
| --- |
| <script type**=**"text/javascript" >  **var** a=09; //无效的八进制 解析为十进制数据9  document.write(a+"<br/>");  </script> |

(2)浮点数：

定义浮点值，必须包括小数点和小数点后的一位数字（例如，用 1.0 而不是 1）。

|  |
| --- |
| <script type**=**"text/javascript" >  **var** a = 3.14;//浮点数  **var** b = 1. //合法，小数点后面没有数字，解析为1  **var** c = 10.0 //整数，解析为10。  document.write(a + "<br/>");  document.write(b + "<br/>");  document.write(c + "<br/>");  </script> |

由于保存浮点数需要的内存空间是保存正数值的两倍，因此JS会将浮点数值转换为整数值。如果小数点后没有跟任何数字和浮点数本身就是一个整数就会被转换为整数。

### 转换成数字

1. 把非数字的原始值转换成数字: parseInt() 和 parseFloat()。
2. 对 String 类型调用这些方法，才能正确运行；对其他类型返回的都是 NaN。
3. 在判断字符串是否是数值:
4. parseInt()

首先查看位置 0 处字符，判断它是否是有效数字；如不是，该方法将返回 NaN，不再继续执行其他操作。如该字符是有效数字，该方法将查看位置 1 处的字符，进行同样的测试。这一过程将持续到发现非有效数字的字符为止，此时 parseInt() 将把该字符之前的字符串转换成数字。

1. 例如，如要把"12345red" 转换成整数， parseInt() 将返回 12345，因当它检查到字符 r 时，就会停止检测过程。

|  |
| --- |
| <script type**=**"text/javascript" >  **var** num = parseInt("1234abc");  document.write(num); //1234  </script> |

2)字符串中包含的数字字面量会被正确转换为数字，比如 "0xA" 会被正确转换为数字 10。3)不过，字符串 "22.5" 将被转换成 22，因为对于整数来说，小数点是无效字符。

|  |
| --- |
| <script type**=**"text/javascript" >  **var** num1 = parseInt("12345red");//返回 12345  **var** num2 = parseInt("0xA"); //返回 10  **var** num3 = parseInt("56.9"); //返回 56  **var** num4 = parseInt("red"); //返回 NaN  document.write(num1 + "<br/>");  document.write(num2 + "<br/>");  document.write(num3 + "<br/>");  document.write(num4 + "<br/>");  </script> |

(2) parseFloat()

|  |
| --- |
| <script type="text/javascript" >  document.write(parseFloat("123") + "<br/>");// 123  document.write(parseFloat("123.53") + "<br/>");// 123.53  document.write(parseFloat("012") + "<br/>");// 12  document.write(parseFloat("000012") + "<br/>");// 12  document.write(parseFloat("0x20") + "<br/>");// 0  document.write(parseFloat("-59") + "<br/>");// -59  document.write(parseFloat("abc123") + "<br/>");// NaN (Not a Number)，不是一个数字。  document.write(parseFloat("123ab2c") + "<br/>");//123 会解析前面正确的数字，  //直到遇到不正确的数字。  </script> |

(3)判断是否是有效的数值.

isNaN( 表达式 )，返回的结果是否为NaN（非数值）

|  |
| --- |
| <script type**=**"text/javascript" >  document.write(isNaN("abc")+"<br/>"); //true  document.write(isNaN("123")+"<br/>"); //false  </script> |

## 运算符(operator)

### 算术

1. 加法运算符: （+）

某个运算数是 NaN，结果为 NaN。

|  |
| --- |
| var result = 5 + 5; //两个数字  alert(result); //输出 "10" |

1. 字符串拼接

(1)如两个运算数都是字符串，把第二个字符串连接到第一个上。

(2)如只有一个运算数是字符串，把另一个运算数转换成字符串，再连接成字符串。

|  |
| --- |
| var result2 = 5 + "5";//一个数字和一个字符串  alert(result); //输出 "55" |

1. 注意：变量是布尔类型的时候，true=1, false=0;

|  |
| --- |
| **var** a = 5;  **var** b = true;  alert(a + b); //6 |

|  |
| --- |
| **var** a = 5;  **var** b = false;  alert(a + b);//5 |

3. 乘法运算符: （\*）

与 Java语言中的相同：

注释：如果运算数是数字，那么执行常规的乘法运算，即两个正数或两个负数为正数，两个运算数符号不同，结果为负数。

1. 除法运算符: （/）

|  |
| --- |
| var a = 10;  var b = 3;  document.write((a / b)+"<br/>"); //3.3333333333333335 |

### 比较

比较运算符:小于、大于、小于等于和大于等于

1.

|  |
| --- |
| var bResult1 = 2 > 1 //true  var bResult2 = 2 < 1 //false |

1. 比较数字和字符串

(1) 两个字符串比较, 比较的是两个字符串对应的字符顺序.

|  |
| --- |
| var bResult = "25" < "3";  alert(bResult); //输出 "true" |

1. 字符串与数字比较，会把字符串转成数字再比较。

|  |
| --- |
| var bResult = "25" < 3;  alert(bResult); //输出 "false" |

### 2.5.3逻辑运算符

|  |
| --- |
| && 与  || 或  ! 非 |

1. &&: 逻辑 AND 运算符

(1)短路运算

|  |
| --- |
| var bTrue = true;  var bFalse = false;  var bResult = bFalse && bTrue; // false |

(2)

|  |
| --- |
| var bTrue = true;  var bResult = (bTrue && bUnknown); //发生错误 bUnknown is not defined  alert(bResult); //这一行不会执行 |

在进行&&时将引发错误，因变量 bUnknown 未定义。变量 bTrue 的值为 true，因为逻辑 AND 运算将继续计算变量 bUnknown。这样做就会引发错误，因为 bUnknown 的值是 undefined，不能用于逻辑 AND 运算。

1. OR : （||）

(1)与 Java 中相同

|  |
| --- |
| var bTrue = true;  var bFalse = false;  var bResult = bTrue || bFalse;  下面的真值表描述了逻辑 OR 运算符的行为：  运算数 1 运算数 2 结果  true true true  true false true  false true true  false false false |

(2)简便运算。如果第一个运算数值为 true，就不再计算第二个运算数。

1. NOT: （!）
2. 与Java 中相同。常用于控制循环

|  |
| --- |
| var bFound = false;  var i = 0;  while (!bFound) {  if (aValue[i] == vSearchValues) {  bFound = true;  } else {  i++;  }  } |

### 2.5.4赋值运算符

1. 简单的赋值运算符：（=）

例如：var iNum = 10;

2.复合赋值运算：由乘性运算符、加性运算符或位移运算符加等号（=）实现的。是下列这些常见情况的缩写形式：

|  |
| --- |
| var iNum = 10;  iNum = iNum + 10； |

复合赋值运算符改写第二行代码：

|  |
| --- |
| var iNum = 10;  iNum += 10； |

乘法/赋值（\*=）

除法/赋值（/=）

取模/赋值（%=）

加法/赋值（+=）

减法/赋值（-=）

### 2.5.5三目

形式与 Java 中相同

|  |
| --- |
| variable = boolean\_expression ? true\_value : false\_value; |

例如：var iMax = (iNum1 > iNum2) ? iNum1 : iNum2;

## 流程控制语句

### 判断

if 语句：if (condition) statement1 else statement2

注意：

判断的条件会发生自动类型转换：

|  |
| --- |
| number：如果非0为true，0为false  string：如果非null或非空为true，否则为false  undefined：false  NaN: false  对象类型：非null为true，否则为false。 |

### 选择

1. switch 语句：

|  |
| --- |
| switch (expression)  case value: statement;  break;  case value: statement;  break;  case value: statement;  break;  case value: statement;  break;  ...  case value: statement;  break;  default: statement; |

1. JS 和 Java 中的 switch 语句有两点不同

2.1 java中

(1)switch(A), A取值只能是整型或可转换为整型的数值类型(byte、short、int、char),还有枚举； 不能为long和String类型

(2)case B：C；

case是常量表达式，即B的取值只能是常量（需定义一个final型的常量）或者int、byte、short、char

如需要在此处写一个表达式或者变量，就要加上单引号；

2.2 在JS 中，switch 语句可以用于字符串，而且能用不是常量的值说明情况：

(1)基本数据类型都可传递给switch case语句。

(2)case语句可以是表达式

|  |
| --- |
| function test6() {  var color = "xx";  var value1 = "red", value2 = "green";  switch(color) {//witch 语句用于字符串 color  case value1:  alert("红色");  break;  case value2:  alert("绿色");  break;  default: //结果:"执行默认default"  alert("执行默认default");  }  } |

练习：switch中传递boolean类型值？

|  |
| --- |
| var num = 20;  switch(true) {  case num >= 0 && num <= 10:  alert("大于0小于等于10");  break;  case num>10&&num<=20: //(2)case语句可以是表达式  alert("大于10小于等于20"); //结果:"大于10小于等于20"  break;  } |

### 循环

1. while 语句：同java(略)
2. do-while 语句：同java(略)

do-while 语句是后测试循环，即退出条件在执行循环内部的代码之后计算。

语法：do {statement} while (expression);

|  |
| --- |
| var i = 0;  do {i += 2;} while (i < 10);  alert(i); //10 |

1. for 语句：同java(略) 作用域规则不相同

前测试循环

语法：for (initialization; expression; post-loop-expression) statement

例子：

|  |
| --- |
| function test9() {  iCount = 6;  for (var i = 0; i < iCount; i++) {  alert("循环内i=" + i);//0 1 2 3 4 5  }  alert("循环外i=" + i);//循环外i=6  }  test9(); |

for循环中不会有局部变量，都是全局的变量。

### for in语句

for 语句是严格的迭代语句，用于枚举对象的属性或者遍历一个数组的元素。

语法：for (*property* in *expression*) *statement*

例子：

|  |
| --- |
| for (sProp in window) { // 此处sProp可命名为任意变量  alert(sProp);//close stop focus blur open……  } |

这里，for-in 语句用于显示 window 对象的所有属性。

### With语句

1. 作用:存取对象属性和方法时就不用重复指定参考对象。
2. 格式 ：

with(obj){

操作obj的属性语句;

}

|  |
| --- |
| <script type="text/javascript">  with(document){  write("您好 !");  write("<br>这个文档的标题是 : \"" + title + "\".");// "在HTML中使用JavaScript".  write("<br>这个文档的 URL 是: " + URL);  write("<br>现在您不用每次都写出 document 对象的前缀了 !");  }  </script> |

### 练习

1，显示“\*”为正方形，5行5列。

2，显示“\*”为三角型，5行，第1行有1个“\*”，第2行有2个“\*”……

3，显示乘法口诀。

1\*1=1

1\*2=2 2\*2=4

....

1\*9=9 2\*9=18 ...

## JavaScript函数基础

函数:封装具体的功能代码。

1. 声明：

注意：函数的形参是不用声明使用的。

语法：

|  |
| --- |
| function functionName(arg0, arg1, ... argN) {  statements  } |

例如：

|  |
| --- |
| function sayHi(sName, sMessage) {  alert("Hello " + sName + sMessage);  } |

1. 调用函数

sayHi("David", " Nice to meet you!")

1. 函数的返回值

(1)没有返回值类型，如一个函数需要返回值直接使用return即可.

只需要使用 return, 后跟要返回的值即可。

function sum(iNum1, iNum2) {

return iNum1 + iNum2;

}

(2)注释:如函数无明确的返回值,或调用无参的return语句，其真正返回的值是 undefined。

-----------------------------------------------------------------

|  |
| --- |
| 函数格式  Function 函数名(参数列表){  代码;  return 返回值; //return 不是必须的.  } |

示例:

|  |
| --- |
| function foo() {  alert("test");  }  function foo2( a ){  alert(a);  return; // 没有返回值，表示方法执行完了，后面如果有代码就不会执行了  }  function foo3( a, b ){  alert(a + b);  }  function foo4( a, b ){  return a + b;  } |

(3)注意:

1.不需要类型，只写形参名即可。

2声明时不需要指定返回值类型；return可有，也可没有。

3.JS中的函数没有重载的形式，后定义的函数会直接覆盖前面 的函数。

|  |
| --- |
| 没有重载:当同一个名称的函数定义多次时，最终使用的是最后一个的定义。没有重载！  例：  function sum(){ alert("11"); }  function sum(a,b){ alert("22"); }\*  调用时写sum(1,2)，结果还是显示“22”。 |

4.一个函数可以接收任意个 参数。

1)JS的函数都是类似于Java中可变参数的。

2)函数中可直接使用arguments变量，arguments是一个数组，表示所有传递过来的参数。

3)声明函数时，定义的形参就对应arguments数组中相应位置的参数值，写形参只是用着方便一点。

示例

|  |
| --- |
| 定义：  function foo( a, b, c ){  alert(a+b+c);  } |

|  |
| --- |
| 调用：  foo();//NAN  foo( 1, 2 );//NAN  foo( 1, 2, 3 );//6  foo( 1, 3, 5, 7, 9 );//9  foo( "aa", null, "cc" );//aanullcc |

总结：

特点：

1、使用function关键字定义函数。

2、没有任何的返回值类型。

3、函数的定义优先执行，与顺序无关。

4、可以通过函数名赋值函数，函数变量的类型是function对象。

5、参数列表不能有var关键字。

6、函数的参数列表在具体调用的时候实参个数可以和形参个数不同。

7、JavaScript函数天然的支持可变参数。

8、同名的函数最后声明会覆盖前面以声明。Javascript中没有函数重载。

案例:体验JavaScript

定义一个输入框输入的月份，然后输出本月对应的天数。

|  |
| --- |
| <html>  <head>  <title>学习JavaScript</title>  </head>  <body>  月份：  <input type="text" id="month">  <input type="button" value="显示天数" onclick="foo()">  <!--  31天：1,3,5,7,8,10,12  30天：4,6,9,11  28天：2  -->  <script type="text/javascript">  function foo(){  var monthValue = document.getElementById("month").value;  // 如果是非数字，则不再继续执行  if( isNaN(monthValue) ){  alert("请输入正确的月份！");  return;  }  // 转为数字  monthValue = parseInt(monthValue);  // 判断范围  if( monthValue < 1 || monthValue > 12){  alert("请输入正确的月份（1--12）！");  return;  }  // 根据月份显示天数  switch(monthValue){ // 这时不会自动转换类型，所以先转型  case 2:  alert("28天");  break;  case 4:  case 6:  case 9:  case 11:  alert("30天");  break;  default:  alert("31天");  break;  }  }  </script>  </body>  </html> |

# 面向对象

JavaScript面向对象的脚本语言，开发者在开发的时候需要找对象，默认提供了内置的对象。也可根据开发者的需求自己定义对象。

## 基本数据类型包装类

为了便于操作基本类型值，JS 提供了3个特殊的引用类型：Boolean,Number,String。

当读取基本数据类型时，后台就会创建一个对应的基本包装类对象，在操作基本数据类型时，可直接调用一些方法。

### String

|  |
| --- |
| //通过构造函数创建String 对象  **var** b = new String("java");  **var** c = new String("java");  document.write(b + "," + c);// java,java  document.write(b == c); //false  //比较字符串的值是否相等  document.write(b.valueOf() == c.valueOf());//true |

|  |
| --- |
| //属性  **var** a = "helloworld";  document.write(a.length); //10 |

2.常见方法：

anchor（） 生产锚点

charAt() 返回指定索引位置处的字符。

charCodeAt() 回一个整数，代表指定位置上字符的 Unicode 编码。

concat() 回字符串值，该值包含了两个或更多个提供的字符串的连接。

fontcolor() 把带COLOR 属性的 HTML <FONT> 标记放置在 String 对象中的文本两端

indexOf() 返回 String 对象内第一次出现子字符串的字符位置

italics() 把 HTML <I> 标记放置在 String 对象中的文本两端。

link() 把一个有 HREF 属性的 HTML 锚点放置在 String 对象中的文本两端。

replace() 返回根据正则表达式进行文字替换后的字符串的复制

split() 切割

substr() 截取子串

toUpperCase() 转大写

[toLowerCase](mk:@MSITStore:E:\文档\JavaScript2.chm::/html/jsmthtolowercase.htm)  转小写

|  |
| --- |
| document.write("第五章".anchor("ch1")+"<br/>");//第五章  document.write("第五章".blink()+"<br/>"); //第五章  //返回在指定位置的字符。  document.write("第五章".charAt(0)+"<br/>"); //第  //回一个整数，代表指定位置上字符的 Unicode 编码。  document.write("a".charCodeAt(0)+"<br/>"); //97    //回字符串值，该值包含了两个或更多个提供的字符串的连接  document.write("第五章".concat("abc")+"<br/>"); //第五章abc  //把带COLOR 属性的 HTML <FONT> 标记放置在 String 对象中的文本两端  document.write("第五章".fontcolor("#ff0000")+"<br/>");//第五章    //返回 String 对象内第一次出现子字符串的字符位置  document.write("第五章".indexOf("五")+"<br/>"); //1  //把 HTML <I> 标记放置在 String 对象中的文本两端  document.write("第五章".italics()+"<br/>"); //*第五章*    //把一个有 HREF 属性的 HTML 锚点放置在 String 对象中的文本两端。  document.write("第五章".link()+"<br/>"); //第五章    //返回根据正则表达式进行文字替换后的字符串的复制  document.write("helloworld".replace(/l/g, "L")+"<br/>");//heLLoworLd  document.write("<br/>");  //切割  var names = "jack-lili-lucy".split("-");  for (var temp in names) {  document.write(names[temp] + "<br/>");//jack lili lucy  }  document.write("<br/>");    //截取子串  document.write("helloworld".substr(1, 2)+"<br/>"); //el  document.write("helloworld".substring(1, 2)+"<br/>");//e  //转大写  document.write("helloworld".toUpperCase()+"<br/>"); //HELLOWORLD  document.write(new String("java").toString() == new String("java").toString()+"<br/>");//false |

### Number

1. 构造函数

|  |
| --- |
| var a = 12;  var b = new Number(12);  document.write((a + ":" + b )+"<br/>");//12:12  //比较数值  document.write((a == b) +"<br/>");//true  //没有转型:比较数值和类型  document.write((a === b) +"<br/>");//false  //false  document.write((new Number(12) == new Number(12)) +"<br/>");//false  //false  document.write((new Number(12) === new Number(12))+"<br/>");//false  // false  document.write((new Number(12).valueOf() == new Number(12).valueOf())+"<br/>");//true  //true  document.write((new Number(12).valueOf() === new Number(12).valueOf())+"<br/>");//true |

1. 属性:

|  |
| --- |
| document.write("最大值" + Number.MAX\_VALUE + "</br>");//最大1.7976931348623157e+308  document.write("最小值" + Number.MIN\_VALUE + "</br>");//最小值5e-324 |

1. 方法

|  |
| --- |
| document.write(new Number(12).toString(10) + "<br/>");//12  document.write(new Number(12).toString(2) + "<br/>"); //1100 --转2进制  document.write(new Number(12).toString(8) + "<br/>"); //14  document.write(new Number(12).toString(16) + "<br/>"); //c  document.write(new Number(12.12345) + "<br/>"); //12.12345  document.write(new Number(12.12345).toFixed() + "<br/>"); //12  // toFixed()是指定小数位数。  document.write(new Number(12.12345).toFixed(2) + "<br/>");//12.12  document.write(new Number(12.12345).toFixed(3) + "<br/>");//12.123 |

### Boolean

|  |
| --- |
| var a = new Boolean(false);  //var a = new Boolean(0); //效果一样  //var a = new Boolean(''); //效果一样  //function Boolean() { [native code] }  document.write(a.constructor + "<br/>");  document.write(a.toString() + "<br/>");//false  document.write(a.valueOf() + "<br/>");//false  if (a.valueOf()) {  document.write("条件成立<br/>");  } else {//条件不成立  document.write("条件不成立<br/>");  } |

使用细节:构造函数中可传递任意值,实参在内部会进行自动类型转换,转为true或者false

## 3.2 Math类

Math工具类中的方法

max()

min()

random()

...更多请看文档

|  |
| --- |
| //返回大于等于n的最小整数。  document.write(Math.ceil(12.34) + "<br/>");//13  // n的整数部分(返回小于等于n的最大整数)。  document.write(Math.floor(12.34) + "<br/>");//12  // num四舍五入后的整数  document.write(Math.round(12.54) + "<br/>");//13  document.write(Math.round(12.35) + "<br/>");//12  //结果为0-1间的一个随机数(包括0,不包括1)  document.write(Math.random() + "<br/>");//0~1  document.write(Math.max(12, 100, -1, 200, 320, 500) + "</br>");// 500  document.write(Math.min(12, 100, -1, 200, 320, 500) + "</br>");//-1 |

## Date对象

1. 练习: 标题栏显示时间,每一秒变化一次, 格式为: xxxx年xx月xx日 xx:xx:xx

提示:需要使用

setInterval(代码, 毫秒数) 每经过指定毫秒值后执行指定的代码。

setTimeout(代码, 毫秒数) 经过指定毫秒值后执行指定的代码（只执行一次）。

其中一个方法

|  |
| --- |
| **function** showDate**()** {  //获取系统时间  **var** date = new Date();  //获取年 月 日时分秒  **var** str = date.getFullYear() + "年" + (date.getMonth() + 1) + "月" + date.getDate() + "日" +  "星期" + date.getDay() + " " + date.getHours() + "时" + date.getMinutes() + "分" +  date.getMinutes() + "分" + date.getSeconds() + "秒"  //将格式化好的时间显示在叶面的span标签体中  **var** span = document.getElementById("mydate");  //给span标签设置标签体内容  span.innerHTML = str.fontcolor("red");  //调用定时函数  window.setTimeout("showDate()", 1000);  }  //调用函数  showDate(); |

## 数组

3.4.1 创建数组的方式有2种.

1. 第一种是使用Array的构造函数

(1)不指定长度 var arrs=new Array();

(2)指定长度 var arrs=new Array(20);

该数组有20个元素,每一项的默认值都是undefined

(3)指定元素 var arrs=new Array("hello"); 会创建一个元素的数组.

1. 第二种方式

(1)指定元素

创建一个包含3个字符串的数组

var arrs=["aa","bb","cc"];

创建一个空数组

var arrs=[];

var values=[1,2];

(2)数组初始化

var arrs=["aa","bb","cc"];

color[2]="xx";//修改第3项

color[3]="zz";//新增第4项

(3)数组遍历：

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

alert(arr[i]);

}

或

for( var i in arr ){

// i 是遍历出的数组中的索引，从0开始，一直到最大索引。

/ 如果有其他属性，也会遍历出来，其实索引就是数组对象的属性。

alert(arr[i]);

}

3.4.2 注意:JS数组可以动态增长：

|  |
| --- |
| //数组的声明和初始化,注意:数组的长度取决于整数下标的个数  **var** arr = new Array();  arr[0] = "aa"; //2  arr[1] = "bb";  arr[6] = 'xx';  document.write(arr + "<br/>");//aa,bb,,,,,xx  //遍历数组  **for** (**var** i = 0; i < arr.length; i++) {  document.write(arr[i]+"<br/>");  }  //下标越界时：  **var** arr = ["aa", "bb"];  document.write(arr[200]);// 显示undefined |

3.4.3 常见方法

Array中方法：

sort()

对当前数组进行排序，并返回已经进行了排序的此Array对象。

不会创建新对象，是在指定数组中直接进行排序。

reverse()

对当前数组进行反转排序，返回一个元素顺序被反转的 Array 对象。

不会创建新对象，是在指定数组中直接进行排序。

push( [item1 [item2 [. . . [itemN ]]]] )

将以新元素出现的顺序添加这些元素。

如果参数之一为数组，那么该数组将作为单个元素添加到数组中。

pop()

移除数组中的最后一个元素并返回该元素。

如果该数组为空，那么将返回 undefined。

shift()

移除数组中的第一个元素并返回该元素。

如果该数组为空，那么将返回 undefined。

unshift([item1[, item2 [, . . . [, itemN]]]])

将指定的元素插入数组开始位置并返回该数组。

splice(start, deleteCount, [item1[, item2[, . . . [,itemN]]]])

移除从 start 位置开始的指定个数的元素，并插入新元素，从而修改 concat([item1[, item2[, . . . [, itemN]]]])

返回一个新数组，这个新数组是由两个或更多数组组合而成的。原数组不变。

要加的项目（item1 … itemN）会按照从左到右的顺序添加到数组。

如果某一项为数组，那么添加其内容到 array1 的末尾。

如果该项目不是数组，就将其作为单个的数组元素添加到数组的末尾。

slice(start, [end])

返回一个包含了指定的部分元素的数组。

将返回索引大于等于start且小于end的一段。

原数组不变。

join(separator)

把所有元素由指定的分隔符分隔开来组成一个字符串，返回这个字符串值。

如果数组中有元素没有定义或者为 null，将其作为空字符串处理。

|  |
| --- |
| **var** a = [1, 2, 3];  //连接两个或更多的数组，并返回结果。  document.write(a.concat([100, 30]) + "<br/>"); // 1,2,3,100,30  document.write(a.concat([100, 30], [60, 90]) + "<br/>");// 1,2,3,100,30,60,90  //把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。  **var** b = a.join("$");  document.write(b + "<br/>"); // 1$2$3  document.write( typeof b + "<br/>"); //string  //删除并返回数组的最后一个元素  document.write(a.pop() + "<br/>"); //3  //向数组的末尾添加一个或更多元素，并返回新的长度。  document.write(a.push(99, 78) + "<br/>");//4  //把数组转换为字符串，并返回结果。  document.write(a.toString() + "<br/>"); //1,2,99,78  //返回数组对象的原始值  **var** c = a.valueOf();  //当数组作为操作数并使用typeof操作符时，其结果会返回"object",因为数组也是对象  document.write( typeof c + "<br/>"); //object |

排序：

|  |
| --- |
| var a = [100, 9, 20, 3, 7, 96];  document.write(a + "<br/>");//100,9,20,3,7,96    //如果调用该方法时没有使用参数，将按字母顺序(符编码的顺序)对数组中的元素进行排序  a.sort();  document.write(a + "<br/>"); //100,20,3,7,9,96  //指定比较方式  a.sort(compareTo);  document.write(a + "<br/>"); //3,7,9,20,96,100  function compareTo(a, b) {//比较数值大小-排序函数  //若 a 小于 b，在排序后的数组中 a 应该出现在 b 之前，则返回一个小于 0 的值。  //若 a 等于 b，则返回 0。  //若 a 大于 b，则返回一个大于 0 的值。  return a - b;  } |

## 自定义对象

### 构造函数方式创建对象

**3.5.1.1 无参构造函数**

|  |
| --- |
| //自定义对象  function Person() {  window.document.write("constructor()<br>");  }  var person = new Person();  //定义属性  person.name = "jack";  person.age = 23;  //定义方法  person.work = function() {  window.document.write("person working...<br/>");  };  //获取构造函数  //function Person() { window.document.write("constructor()"); }  window.document.write(person.constructor + "<br/>");  //jack,23  window.document.write(person.name + "," + person.age + "<br/>");  //person working...  person.work(); |

**3.5.1.2有参数构造函数**

|  |
| --- |
| function Person(name, age) {  this.name = name;  this.age = age;  this.work = function() {  alert(this.name + ":" + this.age);  }  }  var person = new Person("jack", 28);  person.work(); //jack:28 |

**3.5.1.3 Object方式**

|  |
| --- |
| /\*  \* 自定义对象Object方式  \*/  **var** person = new Object();  person.name = "jack";  person.age = 28;  person.sayHello = **function()** {  document.write(this.name + ":" + this.age);  }  person.sayHello(); |

该例子创建了一个名为person的对象,并添加了2个属性(name age)和一个方法(sayHello)

总结:

1.使用构造函数模式方式创建对象：

(1)先自定义类，其实就是定义一个和类名相同的函数。

(2)再使用new关键字调用定义好的类。

(3)（任何函数，只要通过new操作符来调用，那它就可以作为构造函数）

2.说明：

要将属性与方法赋给this,这样才是表示是对象的属性，如果写var str则是方法中的局部变量。

不要写new Object()。

不要写return语句。

1. JS在new Person()会做如下事：

(1)创建一个新对象。

(2)this指向新对象。

(3)执行构造方法中的代码。

(4)返回新对象。

### 字面量方式创建对象

|  |
| --- |
| var person={  name:"李四",  age:88,  eat:function(){  alert("eating...");  }  };  alert(person.name+person.age);  person.eat(); |

练习：自定义一个数组工具对象，该对象有查找指定的元素功能，获取最大、最小值功能

|  |
| --- |
| var tool = new MyArrayTool(); //创建了数组工具对象  function MyArrayTool(arr){  //定义查找元素方法。  this.get = function(arr,element){  for(var i = 0 ; i<arr.length ; i++){  if(arr[i]==element){  return i;  }  }  return -1;  }    //定义最大值的方法  this.max = function(arr){  var max = arr[0];  for(var i = 1; i<arr.length ; i++){  if(max<arr[i]){  max = arr[i];  }  }  return max;  }    //定义最小值的方法  this.min = function(arr){  var min = arr[0];  for(var i = 1; i<arr.length ; i++){  if(min>arr[i]){  min = arr[i];  }  }  return min;  }  } |

### [prototype](http://www.nowamagic.net/librarys/veda/detail/587)

1. 字面翻译是“原型”。

prototype是JS中的函数(function)的一个保留属性，值是一个对象（可称"prototype对象"）。

注意细节：

是函数的必备属性(书面说法是"保留属性")(只要是function,就一定有一个prototype属性)

prototype的值是一个对象

可任意修改函数的prototype属性的值。

一个对象会自动拥有这个对象的构造函数的prototype的成员属性和方法。

1. 例子

prototype 属性使您有能力向对象添加属性和方法。

语法object.prototype.name=value

|  |
| --- |
| function employee(name,job,born) {  this.name=name;  this.job=job;  this.born=born;  }  var bill=new employee("Bill Gates","Engineer",1985);  employee.prototype.salary=null;  bill.salary=20000;  document.write(bill.salary); //20000 |

# 练习

需求：1.自定义一个数组的工具类。

|  |
| --- |
| function arrTool(){}  var ArrayTool = new arrTool();  ArrayTool.getMax = function(arr)  {  var max = 0;    for(var x=1; x<arr.length; x++)  {  if(arr[x]>arr[max])  max = x;  }  return arr[max];  };  ArrayTool.getMin = function(arr)  {  var min = arr[0];    for(var x=1; x<arr.length; x++)  {  if(arr[x]<min)  min = arr[x];  }  return min;  }  /\*  ArrayTool.getMax = getMax;  //获取最值。  function getMax(arr)  {  var max = 0;    for(var x=1; x<arr.length; x++)  {  if(arr[x]>arr[max])  max = x;  }  return arr[max];  }  \*/  ArrayTool.binarySearch = function(arr,key)  {  var min,max,mid;    min = 0;  max = arr.length-1;    while(min<=max)  {  mid = (min+max)>>1;    if(key>arr[mid])  min = mid + 1;  else if(key<arr[mid])  max = mid - 1;  else  return mid;  }  return -1;  }  /\*  ArrayTool.binarySearch = halfSearch;  //折半查找，必须是对有序的数组。  function halfSearch(arr,key)  {  var min,max,mid;    min = 0;  max = arr.length-1;    while(min<=max)  {  mid = (min+max)>>1;    if(key>arr[mid])  min = mid + 1;  else if(key<arr[mid])  max = mid - 1;  else  return mid;  }  return -1;  } |

1. 数组的工具方法添加到javascript内置对象Array中。

|  |
| --- |
| Array.prototype.getIndex = function(element)  {  for(var x=0; x<this.length; x++)  {  if(this[x]==element)  return x;  }  return -1;  }  Array.prototype.getMin = function()  {  var min = this[0];  for(var x=1; x<this.length;x++)  {  if(this[x]<min)  min = this[x];  }  return min;  } |

1. 给字符串添加新的方法

|  |
| --- |
| // JavaScript Document  //给字符串string对象添加新功能。  //去除字符串两端的空格。  String.prototype.trim = function()  {  var start,end;  start = 0;  end = this.length-1;    while(start<=end && this.charAt(start)==" ")  start++;    while(start<=end && this.charAt(end)==' ')  end--;    return this.substring(start,end+1);  }  //将字符串变成数组。  String.prototype.toCharArray = function()  {  var arr = new Array(this.length);    for(var x=0; x<this.length; x++)  {  arr[x] = this.charAt(x);  }    return arr;  }  //将字符串进行反转。  String.prototype.reverseString = function()  {  var arr = this.toCharArray();    return arr.reverse().join("");  } |