JavaScript是什么？JavaScript简介

[JavaScript](http://c.biancheng.net/js/)（简称“JS”）是当前最流行、应用最广泛的客户端脚本语言，用来在网页中添加一些动态效果与交互功能，在 Web 开发领域有着举足轻重的地位。
[Java](http://c.biancheng.net/java/)Script 与 HTML 和 CSS 共同构成了我们所看到的网页，其中：

* HTML 用来定义网页的内容，例如标题、正文、图像等；
* CSS 用来控制网页的外观，例如颜色、字体、背景等；
* JavaScript 用来实时更新网页中的内容，例如从服务器获取数据并更新到网页中，修改某些标签的样式或其中的内容等，可以让网页更加生动。

## JavaScript 的历史

JavaScript 最初被称为 LiveScript，由 Netscape（Netscape Communications Corporation，网景通信公司）公司的布兰登·艾奇（Brendan Eich）在 1995 年开发。在 Netscape 与 Sun（一家互联网公司，全称为“Sun Microsystems”，现已被甲骨文公司收购）合作之后将其更名为了 JavaScript。

## 如何运行 JavaScript

作为一种脚本语言，JavaScript 代码不能独立运行，通常情况下我们需要借助浏览器来运行 JavaScript 代码，所有 Web 浏览器都支持 JavaScript。

## JavaScript 的特点

JavaScript 解释型脚本语言

JavaScript 是一种解释型脚本语言，与ASP PHP等语言需要先编译再运行不同，使用 JavaScript 编写的代码不需要编译，可以直接运行。

## 在 HTML 文档中嵌入 JavaScript 代码

在 HTML 页面中嵌入 JavaScript 脚本需要使用 <script> 标签，用户可以在 <script> 标签中直接编写 JavaScript 代码，具体步骤如下。
第1步，新建 HTML 文档，保存为 test.html。
第2步，在 <head> 标签内插入一个 <script> 标签。
第3步，为 <script> 标签设置type="text/javascript"属性。

现代浏览器默认 <script> 标签的脚本类型为 JavaScript，因此可以省略 type 属性；如果考虑到兼容早期版本浏览器，则需要设置 type 属性。

语法：在 <script> 标签内输入 JavaScript 代码document.write("<h1>C语言中文网：c.biancheng.net</h1>");。

在 JavaScript 脚本中，document 表示网页文档对象；document.write() 表示调用 Document 对象的 write()  方法，在当前网页源代码中写入 HTML 字符串"<h1>C语言中文网：c.biancheng.net</h1>"。

## 在脚本文件中编写 JavaScript 代码

JavaScript 程序不仅可以直接放在 HTML 文档中，也可以放在 JavaScript 脚本文件中。JavaScript 脚本文件是文本文件，扩展名为.js，使用任何文本编辑器都可以编辑。
常用的文本编辑器有 Windows 系统中的记事本、Linux 系统中的 Vim、Sublime Text、Notepad++ 等。对于初学者来说，建议先使用文本编辑器来编写 JavaScript 代码，这样有助于我们对 JavaScript 语法、关键字、函数等内容的记忆。等到了实际开发阶段，则可以选择一些更加专业的代码编辑器，例如 Visual Studio Code（简称“VS Code”）、WebStorm（收费）、Atom 等，这样可以提高开发效率。
新建 JavaScript 文件的步骤如下。
第1步，新建文本文件，保存为 test.js。注意，扩展名为.js，它表示该文本文件是 JavaScript 类型的文件。
第2步，打开 test.js 文件，在其中编写如下 JavaScript 代码。

alert("C语言中文网：c.biancheng.net");

在上面代码中，alert() 表示 Window 对象的方法，调用该方法将弹出一个提示对话框，显示参数字符串 "Hi, JavaScript!"。
第3步，保存 JavaScript 文件。在此建议把 JavaScript 文件和网页文件放在同一个目录下。
JavaScript 文件不能够独立运行，需要导入到网页中，通过浏览器来执行。使用 <script> 标签可以导入 JavaScript 文件。
第4步，新建 HTML 文档，保存为 test.html。
第5步，在 <head> 标签内插入一个 <script> 标签。定义 src 属性，设置属性值为指向外部 JavaScript 文件的 URL 字符串。代码如下：

1. **<script** type="text/javascript" src="test.js"**></script>**

## 注意：使用＜script＞标签包含外部 JavaScript 文件时，默认文件类型为 Javascript。因此，不管加载的文件扩展名是不是 .js，浏览器都会按 JavaScript 脚本来解析。

定义 src 属性的 <script> 标签不应再包含 JavaScript 代码。如果嵌入了代码，则只会下载并执行外部 JavaScript 文件，嵌入代码将被忽略。

## JavaScript 代码执行顺序

浏览器在解析 HTML 文档时，将根据文档流从上到下逐行解析和显示。JavaScript 代码也是 HTML 文档的组成部分，因此 JavaScript 脚本的执行顺序也是根据 <script> 标签的位置来确定的。

#### 示例

使用浏览器测试下面示例，会看到 JavaScript 代码从上到下逐步被解析的过程。

# JavaScript中的几个重要概念

## 1. 标识符

所谓标识符（Identifier），就是名字。JavaScript 中的标识符包括变量名、函数名、参数名、属性名、类名等。
合法的标识符应该注意以下强制规则：

* 第一个字符必须是字母、下划线（\_）或美元符号（$）。
* 除了第一个字符外，其他位置可以使用 Unicode 字符。一般建议仅使用 ASCII 编码的字母，不建议使用双字节的字符。
* 不能与 JavaScript 关键字、保留字重名。
* 可以使用 Unicode 转义序列。例如，字符 a 可以使用“\u0061”表示。

#### 示例

在下面示例中，str 就是变量的名字：。

1. **var** str = "C语言中文网：c.biancheng.net";
2. document.write(str);

第1行代码定义了一个变量，名字为 str，第2行通过 str 这个名字使用了变量。

## 2. 关键字

关键字（Keyword）就是 JavaScript 语言内部使用的一组名字（或称为命令）。这些名字具有特定的用途，用户不能自定义同名的标识符，具体说明如表所示。

|  |
| --- |
| JavaScript 关键字 |
| break | delete  | if | this  | while |
| case | do | in | throw | with |
| catch  | else  | instanceof  | try |   |
| continue  | finally  | new  | typeof |   |
| debugger（ECMAScript 5 新增） | for  | return  | var |   |
| default  | function  | switch  | void |   |

## 3. 保留字

保留字就是 JavaScript 语言内部预备使用的一组名字（或称为命令）。这些名字目前还没有具体的用途，是为 JavaScript 升级版本预留备用的，建议用户不要使用。具体说明如表所示。

|  |
| --- |
| JavaScript 保留字 |
| abstract  | double  | goto  | native  | static |
| boolean  | enum  | implements  | package  | super |
| byte  | export  | import  | private  | synchronized |
| char  | extends  | int  | protected  | throws |
| class  | final  | interface  | public  | transient |
| const  | float  | long  | short  | volatile |

* 多全局变量和函数，用户也应该避免使用它们，具体说明如表所示。

|  |
| --- |
| JavaScript 预定义全局变量和函数 |
| arguments  | encodeURL  | Infinity  | Number  | RegExp |
| Array  | encodeURLComponent  | isFinite  | Object  | String |
| Boolean  | Error  | isNaN  | parseFloat  | SyntaxError |
| Date  | eval  | JSON  | parseInt  | TypeError |
| decodeURL  | EvalError  | Math  | RangeError  | undefined |
| decodeURLComponent  | Function  | NaN  | ReferenceError  | URLError |

不同的 JavaScript 运行环境都会预定义一些全局变量和函数，上表列出的仅针对 Web 浏览器运行环境。

无论是在严格模式下还是在非严格模式下，都不要在定义变量名、函数名或者属性名时使用上面列举出的保留字，以免同学们入坑。

## 4. 区分大小写

JavaScript 严格区分大小写，所以 Hello 和 hello 是两个不同的标识符。
为了避免输入混乱和语法错误，建议采用小写字符编写代码，在以下特殊情况下可以使用大写形式：
1) 构造函数的首字母建议大写。构造函数不同于普通函数。

#### 示例

下面示例调用预定义的构造函数 Date()，创建一个时间对象，然后把时间对象转换为字符串显示出来。

1. d = **new** Date(); //获取当前日期和时间
2. document.write(d.toString()); // 显示日期
2) 如果标识符由多个单词组成，可以考虑使用骆驼命名法——除首个单词外，后面单词的首字母大写。例如：
3. typeOf();
4. printEmployeePaychecks();

#### 提示：

上述都是约定俗成的一般习惯，不构成强制性要求，用户可以根据个人习惯进行命名。

## 5. 直接量

字面量（Literal）也叫直接量，就是具体的值，即能够直接参与运算或显示的值，如字符串、数值、布尔值、正则表达式、对象直接量、数组直接量、函数直接量等。

#### 示例

下面示例分别定义不同类型的直接量：字符串、数值、布尔值、正则表达式、特殊值、对象、数组和函数。

[纯文本复制](http://c.biancheng.net/view/5359.html)

1. //空字符串直接量
2. 1 //数值直接量
3. **true** //布尔值直接量
4. /a/g //正则表达式直接量
5. **null** //特殊值直接量
6. {} //空对象直接量
7. [] //空数组直接量
8. **function**(){} //空函数直接量，也就是函数表达式

# JS注释（多行注释+单行注释）

1. 单行注释以双斜杠//开头，//之后的所有内容都会看作是注释的内容，对//之前的内容则不会产生影响
2. 多行注释以/\*开头，并以\*/结尾，出现在/\*和\*/之间的所有内容都会看作是注释的内容
3. 使用注释除了可以为代码添加解释说明或描述外，在调试代码的时候，我们也可以将不需要运行的代码注释起来，这样在运行程序时浏览器就会忽略注释中的代码
4. JavaScript 还能够识别 HTML 注释的开始符<!--，并将其看作单行注释，与//效果相同。至于 HTML 注释的结束符-->JavaScript 则不能识别，因此若要使用 HTML 注释来注释 JavaScript 代码，应该使用 JavaScript 的单行注释将 HTML 注释的结束符-->注释掉，例如//-->。

# JS变量定义和赋值

## 1. 变量的命名规则

在 JavaScript 中，变量名称并不能随便定义，需要遵循标识符的命名规则，如下所示：

* 变量名中可以包含数字、字母、下划线\_、美元符号$；
* 变量名中不能出现汉字；
* 变量名中不能包含空格；
* 变量名不能是 JavaScript 中的关键字、保留字；
* 变量名不能以数字开头，即第一个字符不能为数字。
在定义变量时，变量名要尽量有意义，让自己或者他人能轻易看懂，例如可以使用 name 来定义一个存储姓名的变量、使用 dataArr 来定义一个数组类型的变量。
当变量名中包含多个英文单词时，推荐使用驼峰命名法（大驼峰：每个单词首字母大写，例如 FileType、DataArr；小驼峰：第一个单词首字母小写后面的单词首字母大写，例如 fileType、dataArr）。

## 2. 定义变量

在 JavaScript 中，定义变量需要使用var关键字，语法格式如下：

var 变量名;

举几个例子：

1. **var** str; //用来存储字符串
2. **var** age; //用来存储年龄
3. **var** prePage; //用来存储上一页
定义变量时，可以一次定义一个或多个变量，若定义多个变量，则需要在变量名之间使用逗号,分隔开，如下例所示：

var a, b, c;    // 同时声明多个变量

变量定义后，如果没有为变量赋值，那么这些变量会被赋予一个初始值——undefined（未定义）。

## 3. 为变量赋值

变量定义后，可以使用等于号=来为变量赋值，等号左边的为变量的名称，等号右边为要赋予变量的值，如下例所示：

1. **var** num; // 定义一个变量 num
2. num = 1; // 将变量 num 赋值为 1

此外，也可以在定义变量的同时为变量赋值，如下例所示：

1. **var** num = 1; // 定义一个变量 num 并将其赋值为 1
2. **var** a = 2, b = 3, c = 4; // 同时定义 a、b、c 三个变量并分别赋值为 2、3、4
3. // var a = 2, // 为了让代码看起来更工整，上一行代码也可以写成这样
4. // b = 3,
5. // c = 4;

## 4. 变量提升

JavaScript 在预编译期会先预处理声明的变量，但是变量的赋值操作发生在 JavaScript 执行期，而不是预编译期。

1. document.write(str); //显示undefined
2. str = http://c.biancheng.net/js/;
3. document.write(str); //显示 http://c.biancheng.net/js/
4. **var** str;

# 在上面示例中，声明变量放在最后，赋值操作放在前面。由于 JavaScript 在预编译期已经对变量声明语句进行了预解析，所以第1行代码读取变量值时不会抛出异常，而是返回未初始化的值 undefined。第3行代码是在赋值操作之后读取，故显示为数字 1。JS数据类型（基本数据类型+引用类型）

数据类型指的是可以在程序中存储和操作的值的类型，每种编程语言都有其支持的数据类型，不同的数据类型用来存储不同的数据，例如文本、数值、图像等。
JavaScript 是一种动态类型的语言，在定义变量时不需要提前指定变量的类型，变量的类型是在程序运行过程中由 JavaScript 引擎动态决定的，另外，您可以使用同一个变量来存储不同类型的数据，例如：

1. **var** a; // 此时 a 为 Undefined
2. a = "http://c.biancheng.net/"; // 此时 a 为 String 类型
3. a = 123; // 此时 a 为 Number 类型

JavaScript 中的数据类型可以分为两种类型：

* 基本数据类型（值类型）：字符串（String）、数字（Number）、布尔（Boolean）、空（Null）、未定义（Undefined）、Symbol；
* 引用数据类型：对象（Object）、数组（Array）、函数（Function）。

#### typeof 操作符

在开始介绍各种数据类型之前，先来了解一下 typeof 操作符，使用 typeof 操作符可以返回变量的数据类型。

typeof 操作符有带括号和不带括号两种用法，如下例所示：

[纯文本复制](http://c.biancheng.net/view/9340.html)

1. **typeof** x; // 获取变量 x 的数据类型
2. **typeof**(x); // 获取变量 x 的数据类型

## 1. JS 基本数据类型

#### 1) String 类型

字符串（String）类型是一段以单引号''或双引号""包裹起来的文本，例如 '123'、"abc"。需要注意的是，单引号和双引号是定义字符串的不同方式，并不是字符串的一部分。
定义字符串时，如果字符串中包含引号，可以使用反斜杠\来转义字符串中的引号，或者选择与字符串中不同的引号来定义字符串，如下例所示：

1. **var** str = "Let's have a cup of coffee."; // 双引号中包含单引号
2. **var** str = 'He said "Hello" and left.'; // 单引号中包含双引号
3. **var** str = 'We\'ll never give up.'; // 使用反斜杠转义字符串中的单引号

#### 2) Number 类型

数值（Number）类型用来定义数值，JavaScript 中不区分整数和小数（浮点数），统一使用 Number 类型表示，如下例所示：

1. **var** num1 = 123; // 整数
2. **var** num2 = 3.14; // 浮点数

另外，Number 类型中还有一些比较特殊的值NaN

* NaN：即非数值（Not a Number 的缩写），用来表示无效或未定义的数学运算结构，例如 0 除以 0。

#### 3) Boolean 类型

布尔（Boolean）类型只有两个值，true（真）或者 false（假），在做条件判断时使用的比较多，您除了可以直接使用 true 或 false 来定义布尔类型的变量外，还可以通过一些表达式来得到布尔类型的值，例如：

1. **var** a = **true**; // 定义一个布尔值 true
2. **var** b = **false**; // 定义一个布尔值 false
3. **var** c = 2 > 1; // 表达式 2 > 1 成立，其结果为“真（true）”，所以 c 的值为布尔类型的 true
4. **var** d = 2 < 1; // 表达式 2 < 1 不成立，其结果为“假（false）”，所以 c 的值为布尔类型的 false

#### 4) Null 类型

Null 是一个只有一个值的特殊数据类型，表示一个“空”值，即不存在任何值，什么都没有，用来定义空对象指针。

使用 typeof 操作符来查看 Null 的类型，会发现 Null 的类型为 Object，说明 Null 其实使用属于 Object（对象）的一个特殊值。因此通过将变量赋值为 Null 我们可以创建一个空的对象。

#### 5) Undefined 类型

Undefined 也是一个只有一个值的特殊数据类型，表示未定义。当我们声明一个变量但未给变量赋值时，这个变量的默认值就是 Undefined。例如：

1. **var** num;
2. console.log(num); // 输出 undefined

在使用 typeof 操作符查看未赋值的变量类型时，会发现它们的类型也是 undefined。对于未声明的变量，使用 typeof 操作符查看其类型会发现，未声明的变量也是 undefined，示例代码如下：

1. **var** message;
2. console.log(**typeof** message); // 输出 undefined
3. console.log(**typeof** name); // 输出 undefined

## 2. JS 引用数据类型

#### 1) Object 类型

JavaScript 中的对象（Object）类型是一组由键、值组成的无序集合，定义对象类型需要使用花括号{ }，语法格式如下：

{name1: value1, name2: value2, name3: value3, ..., nameN: valueN}

其中 name1、name2、name3、...、nameN 为对象中的键，value1、value2、value3、...、valueN 为对应的值。

在 JavaScript 中，对象类型的键都是字符串类型的，值则可以是任意数据类型。要获取对象中的某个值，可以使用对象名.键的形式，如下例所示：

1. **var** person = {
2. name: 'Bob',
3. age: 20,
4. tags: ['js', 'web', 'mobile'],
5. city: 'Beijing',
6. hasCar: **true**,
7. zipcode: **null**
8. };
9. console.log(person.name); // 输出 Bob
10. console.log(person.age); // 输出 20

#### 2) Array 类型

数组（Array）是一组按顺序排列的数据的集合，数组中的每个值都称为元素，而且数组中可以包含任意类型的数据。在 JavaScript 中定义数组需要使用方括号[ ]，数组中的每个元素使用逗号进行分隔，例如：

1. [1, 2, 3, 'hello', **true**, **null**]

另外，也可以使用 Array() 函数来创建数组，如下例所示：

1. **var** arr = **new** Array(1, 2, 3, 4);
2. console.log(arr); // 输出 [1, 2, 3, 4]

数组中的元素可以通过索引来访问。数组中的索引从 0 开始，并依次递增，也就是说数组第一个元素的索引为 0，第二个元素的索引为 1，第三个元素的索引为 2，以此类推。如下例所示：

1. **var** arr = [1, 2, 3.14, 'Hello', **null**, **true**];
2. console.log(arr[0]); // 输出索引为 0 的元素，即 1
3. console.log(arr[5]); // 输出索引为 5 的元素，即 true
4. console.log(arr[6]); // 索引超出了范围，返回 undefined

#### 3) Function 类型

函数（Function）是一段具有特定功能的代码块，函数并不会自动运行，需要通过函数名调用才能运行，如下例所示：

1. **function** sayHello(name){
2. **return** "Hello, " + name;
3. }
4. **var** res = sayHello("Peter");
5. console.log(res); // 输出 Hello, Peter

此外，函数还可以存储在变量、对象、数组中，而且函数还可以作为参数传递给其它函数，或则从其它函数返回，如下例所示：

[纯文本复制](http://c.biancheng.net/view/9340.html)

1. **var** fun = **function**(){
2. console.log("http://c.biancheng.net/js/");
3. }
4. **function** createGreeting(name){
5. **return** "Hello, " + name;
6. }
7. **function** displayGreeting(greetingFunction, userName){
8. **return** greetingFunction(userName);
9. }
10. **var** result = displayGreeting(createGreeting, "Peter");
11. console.log(result); // 输出 Hello, Peter

# JS运算符汇总

运算符是用来告诉 JavaScript 引擎执行某种操作的符号，例如加号（+）表示执行加法运算，减号（-）表示执行减法运算等，本节我们就来介绍一下 JavaScript 中不同的运算符。

## 算术运算符

算数运算符用来执行常见的数学运算，例如加法、减法、乘法、除法等，下表中列举了 JavaScript 中支持的算术运算符：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **示例** |
| + | 加法运算符 | x + y 表示计算 x 加 y 的和 |
| - | 减法运算符 | x - y 表示计算 x 减 y 的差 |
| \* | 乘法运算符 | x \* y 表示计算 x 乘 y 的积 |
| / | 除法运算符 | x / y 表示计算 x 除以 y 的商 |
| % | 取模（取余）运算符 | x % y 表示计算 x 除以 y 的余数 |

示例代码如下：

1. **var** x = 10,
2. y = 4;
3. console.log("x + y =", x + y); // 输出：x + y = 14
4. console.log("x - y =", x - y); // 输出：x - y = 6
5. console.log("x \* y =", x \* y); // 输出：x \* y = 40
6. console.log("x / y =", x / y); // 输出：x / y = 2.5
7. console.log("x % y =", x % y); // 输出：x % y = 2

上述代码中，双引号中的内容是一个字符串，所以其中的运算符会原样输出，并不参与运算。

## 赋值运算符

赋值运算符用来为变量赋值，下表中列举了 JavaScript 中支持的赋值运算符：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **示例** |
| = | 最简单的赋值运算符，将运算符右侧的值赋值给运算符左侧的变量 | x = 10 表示将变量 x 赋值为 10 |
| += | 先进行加法运算，再将结果赋值给运算符左侧的变量 | x += y 等同于 x = x + y |
| -= | 先进行减法运算，再将结果赋值给运算符左侧的变量 | x -= y 等同于 x = x - y |
| \*= | 先进行乘法运算，再将结果赋值给运算符左侧的变量 | x \*= y 等同于 x = x \* y |
| /= | 先进行除法运算，再将结果赋值给运算符左侧的变量 | x /= y 等同于 x = x / y |
| %= | 先进行取模运算，再将结果赋值给运算符左侧的变量 | x %= y 等同于 x = x % y |

示例代码如下：

1. **var** x = 10;
2. x += 20;
3. console.log(x); // 输出：30
4. **var** x = 12,
5. y = 7;
6. x -= y;
7. console.log(x); // 输出：5
8. x = 5;
9. x \*= 25;
10. console.log(x); // 输出：125
11. x = 50;
12. x /= 10;
13. console.log(x); // 输出：5
14. x = 100;
15. x %= 15;
16. console.log(x); // 输出：10

## 字符串运算符

JavaScript 中的+和+=运算符除了可以进行数学运算外，还可以用来拼接字符串，其中：

* +运算符表示将运算符左右两侧的字符串拼接到一起；
* +=运算符表示先将字符串进行拼接，然后再将结果赋值给运算符左侧的变量。
示例代码如下：
1. **var** x = "Hello ";
2. **var** y = "World!";
3. **var** z = x + y;
4. console.log(z); // 输出：Hello World!
5. x += y;
6. console.log(x); // 输出：Hello World!

## 自增、自减运算符

自增、自减运算符用来对变量的值进行自增（+1）、自减（-1）操作，下表中列举了 JavaScript 中支持的自增、自减运算符：

|  |  |  |
| --- | --- | --- |
| **运算符** | **名称** | **影响** |
| ++x | 自增运算符 | 将 x 加 1，然后返回 x 的值 |
| x++ | 自增运算符 | 返回 x 的值，然后再将 x 加 1 |
| --x | 自减运算符 | 将 x 减 1，然后返回 x 的值 |
| x-- | 自减运算符 | 返回 x 的值，然后将 x 减 1 |

示例代码如下：

1. **var** x;
2. x = 10;
3. console.log(++x); // 输出：11
4. console.log(x); // 输出：11
5. x = 10;
6. console.log(x++); // 输出：10
7. console.log(x); // 输出：11
8. x = 10;
9. console.log(--x); // 输出：9
10. console.log(x); // 输出：9
11. x = 10;
12. console.log(x--); // 输出：10
13. console.log(x); // 输出：9

## 比较运算符

比较运算符用来比较运算符左右两侧的表达式，比较运算符的运算结果是一个布尔值，结果只有两种，不是 true 就是 false。下表中列举了 JavaScript 中支持的比较运算符：

|  |  |  |
| --- | --- | --- |
| **运算符** | **名称** | **示例** |
| == | 等于 | x == y 表示如果 x 等于 y，则为真 |
| === | 全等 | x === y 表示如果 x 等于 y，并且 x 和 y 的类型也相同，则为真 |
| != | 不相等 | x != y 表示如果 x 不等于 y，则为真 |
| !== | 不全等 | x !== y 表示如果 x 不等于 y，或者 x 和 y 的类型不同，则为真 |
| < | 小于 | x < y 表示如果 x 小于 y，则为真 |
| > | 大于 | x > y 表示如果 x 大于 y，则为真 |
| >= | 大于或等于 | x >= y 表示如果 x 大于或等于 y，则为真 |
| <= | 小于或等于 | x <= y 表示如果 x 小于或等于 y，则为真 |

示例代码如下：

1. **var** x = 25;
2. **var** y = 35;
3. **var** z = "25";
4. console.log(x == z); // 输出： true
5. console.log(x === z); // 输出： false
6. console.log(x != y); // 输出： true
7. console.log(x !== z); // 输出： true
8. console.log(x < y); // 输出： true
9. console.log(x > y); // 输出： false
10. console.log(x <= y); // 输出： true
11. console.log(x >= y); // 输出： false

## 逻辑运算符

逻辑运算符通常用来组合多个表达式，逻辑运算符的运算结果是一个布尔值，只能有两种结果，不是 true 就是 false。下表中列举了 JavaScript 中支持的逻辑运算符：

|  |  |  |
| --- | --- | --- |
| **运算符** | **名称** | **示例** |
| && | 逻辑与 | x && y 表示如果 x 和 y 都为真，则为真 |
| || | 逻辑或 | x || y 表示如果 x 或 y 有一个为真，则为真 |
| ! | 逻辑非 | !x 表示如果 x 不为真，则为真 |

示例代码如下：

1. **var** year = 2021;
2. // 闰年可以被 400 整除，也可以被 4 整除，但不能被 100 整除
3. **if**((year % 400 == 0) || ((year % 100 != 0) && (year % 4 == 0))){
4. console.log(year + " 年是闰年。");
5. } **else**{
6. console.log(year + " 年是平年。");
7. }

## 三元运算符

三元运算符（也被称为条件运算符），由一个问号和一个冒号组成，语法格式如下：

条件表达式 ? 表达式1 : 表达式2 ;

如果“条件表达式”的结果为真（true），则执行“表达式1”中的代码，否则就执行“表达式2”中的代码。
示例代码如下：

1. **var** x = 11,
2. y = 20;
3. x > y ? console.log("x 大于 y") : console.log("x 小于 y"); // 输出：x 小于 y

# JS输出语句汇总（5种）

JavaScript 中为我们提供了多种不同的输出语句来向浏览器中输出内容：

1. 使用 alert() 函数来弹出提示框；
2. 使用 confirm() 函数来弹出一个对话框；
3. 使用 document.write() 方法将内容写入到 HTML 文档中；
4. 使用 innerHTML 将内容写入到 HTML 标签中；
5. 使用 console.log() 在浏览器的控制台输出内容。

## 1. alert() 函数

使用 JS alert() 函数可以在浏览器中弹出一个提示框，在提示框中我们可以定义要输出的内容，语法格式如下：

alert(message);

其中 message 为要在提示框中输出的内容，需要注意的是，alert() 中只能输出文本内容。
alert() 函数是 window 对象下的一个函数，所以有时为了代码更严谨，我们也可以使用 window.alert() 的形式来调用 alert() 函数。

## 2. confirm() 函数

JS confirm() 函数与 alert() 函数相似，它们都是 window 对象下的函数，同样可以在浏览器窗口弹出一个提示框，不同的是，使用 confirm() 函数创建的提示框中，除了包含一个“确定”按钮外，还有一个“取消”按钮。如果点击“确定”按钮，那么 confirm() 函数会返回一个布尔值 true，如果点击“取消”按钮，那么 confirm() 函数会返回一个布尔值 false。
示例代码如下：

[纯文本复制](http://c.biancheng.net/view/9342.html)

1. <!DOCTYPE html>
2. **<html** lang="en"**>**
3. **<head>**
4. **<meta** charset="UTF-8"**>**
5. **<title>**JavaScript**</title>**
6. **</head>**
7. **<body>**
8. **<script** type="text/javascript"**>**
9. var res = window.confirm("这里是要显示的内容");
10. if(res == true){
11. alert("你点击了“确定”按钮");
12. }else{
13. alert("你点击了“取消”按钮");
14. }
15. **</script>**
16. **</body>**
17. **</html>**

## 3. console.log()

使用 JS console.log() 可以在浏览器的控制台输出信息，我们通常使用 console.log() 来调试程序，其语法格式如下：

console.log(message);

其中 message 为要输出的内容，可以是字符串或者对象类型。与 window.alert() 和 window.confirm() 可以分别简写成 alert() 和 confirm() 不同，console.log() 不能简写。
要看到 console.log() 的输出内容需要先打开浏览器的控制台。以 Chrome 浏览器为例，要打开控制台您只需要在浏览器窗口按 F12 快捷键，或者点击鼠标右键，并在弹出的菜单中选择“检查”选项即可。最后，在打开的控制台中选择“Console”选项

## 4. document.write()

使用 JS document.write() 可以向 HTML 文档中写入 HTML 或者 JavaScript 代码，语法格式如下：

document.write(exp1, exp2, exp3, ...);

其中 exp1、exp2、exp3 为要向文档中写入的内容，document.write() 可以接收多个参数，即我们可以一次向文档中写入多个内容，内容之间使用逗号进行分隔。

示例代码如下：

[纯文本复制](http://c.biancheng.net/view/9342.html)

1. <!DOCTYPE html>
2. **<html** lang="en"**>**
3. **<head>**
4. **<meta** charset="UTF-8"**>**
5. **<title>**JavaScript**</title>**
6. **</head>**
7. **<body>**
8. **<script** type="text/javascript"**>**
9. document.write("**<p>**现在的时间是：**</p>**");
10. document.write(Date());
11. **</script>**
12. **</body>**
13. **</html>**

## 5. innerHTML

与前面介绍的几个函数不同，innerHTML 是一个属性而不是一个函数，通过它可以设置或者获取指定 HTML 标签中的内容

# JS if else语句：条件判断语句

[Java](http://c.biancheng.net/java/)Script 中支持以下几种不同形式的条件判断语句：

* if 语句；
* if else 语句；
* if else if else 语句；
* switch case 语句

## if 语句

if 语句是 JavaScript 中最简单的条件判断语句，语法格式如下：

if(条件表达式){
    // 要执行的代码;
}

当条件表达式成立，即结果为布尔值 true 时，就会执行{ }中的代码。

## if else 语句

if else 语句是 if 语句的升级版，它不仅可以指定当表达式成立时要执行的代码，还可以指定当表达式不成立时要执行的代码，语法格式如下：

if(条件表达式){
    // 当表达式成立时要执行的代码
}else{
    // 当表达式不成立时要执行的代码
}

## if else if else 语句

if 和 if else 语句都只有一个条件表达式，而 if else if else 语句是它们更高级的形式，在 if else if else 语句中允许您定义多个条件表达式，并根据表达式的结果执行相应的代码，语法格式如下：

if (条件表达式 1) {
    // 条件表达式 1 为真时执行的代码
} else if (条件表达式 2) {
    // 条件表达式 2 为真时执行的代码
}
...
  else if (条件表达式N) {
    // 条件表达式 N 为真时执行的代码
} else {
    // 所有条件表达式都为假时要执行的代码
}

提示：if else if else 语句在执行过程中，当遇到成立的条件表达式时，会立即执行其后{ }中的代码，然后退出整个 if else if else 语句，若后续代码中还有成立的条件表达式，则不会执行。

## switch case 语句的语法：

* **switch** (表达式){
* **case** value1:
* statements1 // 当表达式的结果等于 value1 时，则执行该代码
* **break**;
* **case** value2:
* statements2 // 当表达式的结果等于 value2 时，则执行该代码
* **break**;
* ......
* **case** valueN:
* statementsN // 当表达式的结果等于 valueN 时，则执行该代码
* **break**;
* **default** :
* statements // 如果没有与表达式相同的值，则执行该代码
* }

switch 语句根据表达式的值，依次与 case 子句中的值进行比较：

* 如果两者相等，则执行其后的语句段，当遇到 break 关键字时则跳出整个 switch 语句。
* 如果不相等，则继续匹配下一个 case。
* switch 语句包含一个可选的 default 关键字，如果在前面的 case 中没有找到相等的条件，则执行 default 后面的语句段。
* switch 语句的执行流程（工作原理）如下图所示：
* 

#### break 关键字

switch 语句是逐行执行的，当 switch 语句找到一个与之匹配的 case 子句时，不仅会执行该子句对应的代码，还会继续向后执行，直至 switch 语句结束。为了防止这种情况产生，需要在每个 case 子句的末尾使用 break 来跳出 switch 语句。

break 除了可以用来跳出 switch 语句外，还可以用来跳出循环语句（for、for in、while、do while 等）

1. **var** id = 1;
2. **switch** (id) {
3. **case** 1 :
4. console.log("普通会员");
5. **break**; //停止执行，跳出switch
6. **case** 2 :
7. console.log("VIP会员");
8. **break**; //停止执行，跳出switch
9. **case** 3 :
10. console.log("管理员");
11. **break**; //停止执行，跳出switch
12. **default** : //上述条件都不满足时，默认执行的代码
13. console.log("游客");
14. }

## case 子句

case 子句可以省略语句，这样当匹配时，不管下一个 case 条件是否满足，都会继续执行下一个 case 子句的语句。

在 switch 语句中，case 子句只是指明了执行起点，但是没有指明执行的终点，如果在 case 子句中没有 break 语句，就会发生连续执行的情况，从而忽略后面 case 子句的条件限制，这样就容易破坏 switch 结构的逻辑。
如果在函数中使用 switch 语句，可以使用 return 语句终止 switch 语句，防止代码继续执行

## default语句

default 是 switch 子句，可以位于 switch 内任意位置，不会影响其它 case 子句的正常执行。

下面示例使用 switch 语句设计一个四则运算函数。在 switch 结构内，先使用 case 枚举 4 种可预知的算术运算，当然还可以继续扩展 case 子句，枚举所有可能的操作，但是无法枚举所有不测，因此最后使用 default 处理意外情况。

1. **function** oper (a,b,opr) {
2. **switch** (opr) {
3. **case** "+" : //正常枚举
4. **return** a + b;
5. **case** "-" : //正常枚举
6. **return** a - b;
7. **case** "\*" : //正常枚举
8. **return** a \* b;
9. **case** "/" : //正常枚举
10. **return** a / b;
11. **default** : //异常处理
12. **return** "非预期的 opr 值";
13. }
14. }
15. console.log(oper (2,5,"\*")); //返回10

# JS while循环语句的用法

## JS while 循环语法

JS while 循环的语法格式如下：

while (条件表达式) {
    // 要执行的代码
}

while 循环在每次循环之前，会先对条件表达式进行求值，如果条件表达式的结果为 true，则执行{ }中的代码，如果条件表达式的结果为 false，则退出 while 循环，执行 while 循环之后的代码。

while 循环的执行流程如下图所示：


图1：JS while 循环的执行流程

示例代码如下：

1. **var** i = 1;
2. **while**( i <= 5) {
3. document.write(i+", ");
4. i++;
5. }

运行结果：
1, 2, 3, 4, 5,

#### 注意事项

在编写循环语句时，一定要确保条件表达式的结果能够为假（即布尔值 false），因为只要表达式的结果为 true，循环会一直持续下去，不会自动停止，对于这种无法自动停止的循环，我们通常将其称为“无限循环”或“死循环”。

## JS while 循环示例

使用 while 循环计算 1~100 之间所有整数的和：

1. **var** i=1;
2. **var** sum=0;
3. **while** (i <= 100){
4. sum += i;
5. i++;
6. }
7. document.write("1 + 2 + 3 + ... + 98 + 99 + 100 = " + sum)

运行结果：
1 + 2 + 3 + ... + 98 + 99 + 100 = 5050

找出 1900 年到 2020 年之间所有的闰年，并以每行 6 个进行输出：

1. **var** i = 1900;
2. **var** count = 0; //计数闰年的个数
3. **while** (i <= 2020) {
4. //判断是否是闰年
5. **if** (i % 4 == 0 && i % 100 != 0 || i % 400 == 0) {
6. document.write(i + "&nbsp;&nbsp;");
7. count++;
8. **if** (count % 6 == 0) {
9. document.write("<br/>");
10. }
11. }
12. i++;
13. }

运行结果：

1904 1908 1912 1916 1920 1924

1928 1932 1936 1940 1944 1948

1952 1956 1960 1964 1968 1972

1976 1980 1984 1988 1992 1996

2000 2004 2008 2012 2016 2020

#### do while 循环语法

JS do while 循环的语法格式如下：

do {
    // 需要执行的代码
} while (条件表达式);

提示：do while 循环与 while 循环还有一点不同，那就是 do while 循环的末尾需要使用分号;进行结尾，而 while 循环则不需要。

do while 循环的执行流程如下图所示：


图1：do while 循环执行流程
示例代码如下：

1. **var** i = 1;
2. **do**{
3. document.write(i + " ");
4. i++;
5. }**while** (i > 5);

运行结果：
1

#### JS do while 循环示例

使用 do while 循环计算 1~100 之间所有整数的和：

1. **var** i=1;
2. **var** sum=0;
3. **do** {
4. sum += i;
5. i++;
6. } **while** (i <= 100);
7. document.write("1 + 2 + 3 + ... + 98 + 99 + 100 = " + sum)

运行结果：
1 + 2 + 3 + ... + 98 + 99 + 100 = 5050

#### for 循环语法

JS for 循环适合在已知循环次数时使用，语法格式如下：

for(initialization; condition; increment) {
    // 要执行的代码
}

for 循环中包含三个可选的表达式 initialization、condition 和 increment，其中：

* initialization：为一个表达式或者变量声明，我们通常将该步骤称为“初始化计数器变量”，在循环过程中只会执行一次；
* condition：为一个条件表达式，与 while 循环中的条件表达式功能相同，通常用来与计数器的值进行比较，以确定是否进行循环，通过该表达式可以设置循环的次数；
* increment：为一个表达式，用来在每次循环结束后更新（递增或递减）计数器的值。

示例代码如下：

1. **for** (**var** i = 1; i <= 10; i++) {
2. document.write(i + " ");
3. }

运行结果：
1 2 3 4 5 6 7 8 9 10

#### for 循环中的三个表达式

JS for 循环中括号中的三个表达式是可以省略的，但是用于分隔三个表达式的分号不能省略，如下例所示：

1. // 省略第一个表达式
2. **var** i = 0;
3. **for** (; i < 5; i++) {
4. // 要执行的代码
5. }
6. // 省略第二个表达式
7. **for** (**var** y = 0; ; y++) {
8. **if**(y > 5){
9. **break**;
10. }
11. // 要执行的代码
12. }
13. // 省略第一个和第三个表达式
14. **var** j = 0;
15. **for** (; j < 5;) {
16. // 要执行的代码
17. j++;
18. }
19. // 省略所有表达式
20. **var** z = 0;
21. **for** (;;) {
22. **if**(z > 5){
23. **break**;
24. }
25. // 要执行的代码
26. z++;
27. }

#### JS for 循环嵌套

无论是哪种循环，都可以嵌套使用（即在一个循环中再定义一个或多个循环），下面就以 for 循环为例，来演示一下循环的嵌套使用：

1. **for** (**var** i = 1; i <= 9; i++) {
2. **for** (**var** j = 1; j <= i; j++) {
3. document.write(j + " x " + i + " = " + (i \* j) + "&emsp;");
4. }
5. document.write("<br>");
6. }

运行结果：

1 x 1 = 1

1 x 2 = 2 2 x 2 = 4

1 x 3 = 3 2 x 3 = 6 3 x 3 = 9

1 x 4 = 4 2 x 4 = 8 3 x 4 = 12 4 x 4 = 16

1 x 5 = 5 2 x 5 = 10 3 x 5 = 15 4 x 5 = 20 5 x 5 = 25

1 x 6 = 6 2 x 6 = 12 3 x 6 = 18 4 x 6 = 24 5 x 6 = 30 6 x 6 = 36

1 x 7 = 7 2 x 7 = 14 3 x 7 = 21 4 x 7 = 28 5 x 7 = 35 6 x 7 = 42 7 x 7 = 49

1 x 8 = 8 2 x 8 = 16 3 x 8 = 24 4 x 8 = 32 5 x 8 = 40 6 x 8 = 48 7 x 8 = 56 8 x 8 = 64

1 x 9 = 9 2 x 9 = 18 3 x 9 = 27 4 x 9 = 36 5 x 9 = 45 6 x 9 = 54 7 x 9 = 63 8 x 9 = 72 9 x 9 = 81

## 定义函数

JS 函数声明需要以 function 关键字开头，之后为要创建的函数名称，function 关键字与函数名称之间使用空格分开，函数名之后为一个括号( )，括号中用来定义函数中要使用的参数（多个参数之间使用逗号,分隔开），一个函数最多可以有 255 个参数，最后为一个花括号{ }，花括号中用来定义函数的函数体（即实现函数的代码），如下所示：

function functionName(parameter\_list) {
    // 函数中的代码
}

示例代码如下：

1. **function** sayHello(name){
2. document.write("Hello " + name);
3. }

上面示例中定义了一个函数 sayHello()，该函数需要接收一个参数 name，调用该函数会在页面中输出“Hello ...”。