JavaScript 的语法

如果你学过 C，C++，或者 Java 的话，就可发现，javascript 的语法和它们是一样的。javascript 也有一些保留字，与 C 不同，javascript 有一些预定义的内建函数，完成对浏览器和页面的基本操作。

特殊符号和操作符

Javascript 的操作符主要用于算术运算、逻辑运算和字符串运算。大部分操作符都有两个操作对象，分别叫左操作数和右操作数。Javascript 的操作符和 C、C++的操作符几乎完全一样。

基本的算术操作符（运算符）是 + (加)，- (减或一元负号)，* (乘)，/ (除)，% (模，或者叫取余数)以及 ++ (整数自身加 一), -- (整数自身减一)。以及位操作， || (OR), && (AND)，! (各位取反)，^ (XOR 异 或)，<< (左移位)，<<< (左移位，并用 0 填充左边的位)，>> (右移 位) 和 >>> (右移位，用 0 填充右边的位)。除了++，--和一元负（-）之外，所有这些算术、逻辑操作符都允许和 =连用， 表示左操作数和右操作数运算的结果赋值给左操作数本身。 例如：

```
variable = variable * 42;
// 这句可写成 variable *= 42;
```

这些操作符符合一般的优先法则，可以用括号改变运算的次序。如果对运算的优先法则搞不清楚，也可以用括号保证运算的正确顺序。

```
zzz = (xxx * 32) / (yyy + xxx);
```

javascript 的关系运算符包括：<(小于)，> (大于)，== (等于)，!= (不等于)，<= (不大于)，& gt;= (不小于)。请注意区分<赋值符（=）和 关系运算符等于（==）。另一个经常出现的错误是将不等于(!=)写作<>。

每一个完整的语句之间用分号隔开。如下所示：

```
variable *= 42; zzz = (xxx * 32) / (yyy + xxx);
```

+号同时被用来表示两个字符串相连接。例如：

```
yourname = "Dave"
line_of_text = "What are you doing " + yourname + "?";
```

执行了上述语句后，line_of_text 的值应该是，"What are you doing Dave?"

字符串常量要用引号（双引号或者单引号都可）括起来。两者之间的区别在于：用双引号括起来的字符串允许包括一些控制符。为了表示这些控制符，必须用以反斜杠 (\)开始的转义符，例如以 \n 表示换行。由于\用于引导转义符，所以在字符串中用到反斜杠符时必须用\\表示。

括号被用于指定数组的元素。例如要指定数组 myarrary 的第一个元素，要写

成 myarrary[0]。请注意，数组的元素是从 0 开始编号的。

大括号({ 和 })，被用来组织语句块（某些地方也称复合语句，及几条语句被从逻辑上看作是一条语句）。这些语句块经常和控制语句（比如分支 if...then...else 、循环 while ）等一起出现。注意，函数也是由一个语句块构成的。

多行或单行注释由 /* 和*/括起来，单行的注释也可以用//开始，直到行尾。

最后要介绍的是一个三元操作符， 条件表达式 ? 结果表达式 1 : 结果表达式 2 。这个运算符的意思是：如果条件表达式成立，就取结果表达式 1 的值，反之，就取条件表达式 2 的值。任何时候，这个操作符都可以用 if 和 else 来模拟。例如：

```
if (notMine == true)
{
   someoneElses = true;
}
else
{
   someoneElses = false;
}
```

和下式等价：

```
someoneElses = (notMine == true) ? true : false;
```

上面介绍了所有的操作符。接下来，介绍 javascript 的数据类型。

数据类型和变量

JavaScript 有四种基本的数据类型：对象(object，能用于任何对象)、数值型 (number，浮点数或整数)、字符串型 (string)、布尔型(boolean)。变量通过变量名来区分。变量名区分大小写，也就是说大写与小写是不同的。组成变量名的符号包括字母、数字和 下划线(_)，并且必须以字母开始。例如：

```
myvariable = "A line of text";
//这是字符串型变量
count = 0;
//数值型
Super_Long_VariableNamethatishardtoread = "";
//字符串，空串



WorldIsRound = true;
//布尔型
```

另外，数组和函数指针也是常见的数值类型。

语句

语句是程序的可执行部分。结构及控制语句决定了程序的结构，结构及控制语句通常使用了保留字。下面列出了所有的当前支持的保留字。

```
break           comment          continue
for             for...in         function
if...else       return           var
while           with
```

if...else 结构

if..then...else 结构是最容易理解的控制语句。如果满足条件，就执行特定的语句序列，反之条件不满足，就执行另外的语句序列。请看下例：

```
if (variable > 20)
{
    //如果成立,执行这一段...
    ...
}
else
{
    //不成立时,执行这段...
}
```

if...else... 允许嵌套使用。if...else if...，这样的形式可以很好地表达多重分支结构。

while 循环

while 循环结构提供了一种手段，使得在条件满足的情况下重复一段程序。如果不小心，使得条件永远不能打破的话，程序将陷入死循环。下例重复 10 次：

```
count = 0;
while (count < 10)
{
    write("Hello.\n");
    count++;
}
```

for 循环

for 循环是另一种选择。与上例相同的功能可以用 for 循环来实现。

```
for (count = 0; count < 10; count++)
{
    write("Hello.\n");
}
```

for 循环的控制部分可分为三个部分，各部分用分号隔开。初始部分是 count = 0;判断部分是 count <10; 最 后，count++是条件变动部分。循环的初始部分仅仅在循环开始时被执行一次，然后每一次循环都检查判断部分的条件是否满足，如不满足则就跳出循环，否 则，执行循环体（即大括号括起来的语句块），执行条件变动部分，检查判断条件，周而周而复始，直到循环条件被破坏为止。

for..in 结构

for..in 结构对集合里的每一个元素执行相同的操作。下面的例子显示 myArray 数组中的每一个元素的内容。

```
for count in myArray
{
    write(myArray[count]);
}
```

break, continue

break 和 continue 这两个保留字用于在循环体内改变语句的流程。break 强制打断循环，continue 跳过循环体中余下的语句，直接回到循环体的起始。请看下例，当 count 的值等于 5 时，循环被打断，因此，最后一次的输出的值是 4...：

```
count = 0;
while (count < 10)
{
    if (count == 5)
    break;

    write(count + "...");
      count++;
}
```

下面的例子说明了 continue 的作用：

```
for iteration in myArray
{
    if (Math.odd(iteration))
      continue;
    writeln(myArray[iteration]);
}
```

if 语句中的判断使得当 iteration 为奇数时，被跳过。

函数

保留字 function 用来定义一个函数。函数是一个子程序，可通过函数名在任何地方被调用。说明一个带两个参数(parameter1, parameter2)的函数（函数名

myfunction）的格式如下所示：

```
function myfunction(parameter1, parameter2)
{
    // 由若干条语句组成的函数体
}
```

function 后面紧接着函数名（可以在别处通过函数名而调用函数）。函数名和变量名的命名方法是一样的。函数名后面紧接着参数表。在参数表中不 需要说明参数的数据类型，这点与 BASIC、Foxbase 等解释型语言一样。在函数体中定义具体的操作。在任何函数中都可以用保留字 return 返回 一个结果给调用者。下面的示例定义了一个函数：

```
function Average(value1, value2)
{
    average = (value1 + value2) / 2;
    return average;
}
```

下面的程序调用了上面定义的函数：

```
xxx = 23;
yyy = 14;
averaged_value = Average(xxx, yyy);
```

执行了上述程序后，averaged_value 的值为(23 + 14)/2，等于 18。