第1章 JavaScript 基础

变量（variables）

变量的类型

JavaScript 的数据类型可分为两类——基本数据类型（primitive data type）和对象数据类型（object data type），基本数据类型如数值类型（number）的 1,2,3,4,……,n、字符串类型（string）的 "string type"、"Hello!" 等和 true、false 的布尔类型（boolean）。JavaScript 的变量及变量数据区分大小写，这和 C、C++ 与 Java 语言是一样的，例如变量 myVar、MyVar 和变量数据的 "JavaScript"、"javascript" 和 "JAVASCRIPT" 是不一样的，它们分别表示不同的变量与变量值。而对象数据类型除了可以包含对象属性的变量数据外并可定义对象的方法（函数），例如 Number 对象的 MAX_VALUE、MIN_VALUE 属性和 toString()函数，另外，如 String 对象的长度属性 length 和 toLowerCase()、toUpperCase() 函数等等。

变量的声明

在 JavaScript 里声明变量时不需指定变量的类型，这一点和 C、C++ 与 Java 语言不一样，例如在 Java 语言中声明变量时必须用下面的方式声明：

```
int num;
String language="Java Language";
```

而在 JavaScript 中，不论变量的数据类型为何，都是用 var 保留字来声明，例如：

```
var num=1;
var language="JavaScript Language";
```

也就是说，变量的实际类型要视变量数据的内容而定，像上面的 num 是 number 类型，而 language 是 string 类型。另外，变量的类型也可以随时被改变，只要指定不同类型的数据，变量的类型就会跟着改变：

```
var num=1; //number类型
var language="JavaScript Language"; //string类型
......
number="1"; //string类型
```

```
language=new String("JavaScript Language"); //String对象类型
```

利用 var 声明一个变量时可以不指定初始值，此时变量的内容会是一个预设的数据类型——undefined，它的意义是没有定义任何初始值：

```
var myVar; // myVar 的值是 undefined，类型亦是 undefined
```

在 JavaScript 中并没有强制一定要用 var 来定义变量，也可以不用。不过当没有用 var 来声明变量时，JavaScript 的编译器会先检查变量名称是否曾经被声明过，如果没有，就会自动用所指定的变量名称帮你声明一个变量，但是有一点必须注意的是，这时候你一定要设定变量的初始值，否则将会产生编译错误：

```
newValue=0; //不可以只写newValue;
```

除了 0 之外，也可以是任何类型的数据或是不具任何值的 null、NaN、undefined (Navigator)、void(0)等等。

虽然可以不以 var 来声明变量，但是要注意变量名称是否已经被声明过，否则只会将原来定义的变量值改变而已，并不是产生一个新的变量，所以好的习惯是当变量第一次使用时都用 var 来声明。

变量的参考

变量的参考可分为两种，一种是值的参考，另一种是内存地址的参考，这两种有什么差别呢？先来看看下面的程序：

```
var valueA=123;
var valueB=valueA;
valueA=456;
alert(valueB); //123
```

当 valueA 赋给 valueB 时，JavaScript 会复制一份 123 的值给变量 valueB，此时 valueA 和 valueB 会被配置在不同的内存区块，大小都为 8 个 byte，所以当 valueA 的内存内容被改为 456 而 valueB 的内容仍为 123，并不会因 valueA 的值被改变而同时改变 valueB 的内容。像这种具有固定大小内存位置的变量类型都是这样的情形，例如 number、string、boolean、Number、String 和 Boolean 等数据类型。接下来，再看看不固定大小内容的变量类型：

```
var valueA=new Array(1,2,3);
var valueB=valueA;
valueA[0]=4; //内存地址的参考
alert(valueB); //4,2,3
```

因为在 JavaScript 中的 Array 内容不限制为同一数据类型，因此，当声明一个 Array 时，虽然一开始时所占的内存大小是一样的，可是一旦 Array 的内容被改变时，整个 Array 的内存大小也会随着改变，像参考这种具有不固定大小数据类型的变量时，即属于内存地址的参考。以上面的例子来说，valueA 和 valueB 会被配置在同一个内存地址，所以当 valueA 其中一个内容被改变时，valueB 的内容也会被改变，因为实际上它们是参考同一个内存地址。另外，像对象（Object）的参考也是一样，例如

```
function obj(){
   this.a=1;
   this.b=2;
}
objA=new obj;
objB=objA;
objA.a=3;
alert(objB.a);//3
```

关于对象的说明请参考 Object 的章节。

变量的参考范围

当使用一个未经声明的变量时，应特别注意它的参考范围，例如下面范例在 getAttrInc 函数里使用了一个未在 function 内定义的变量 attr，此时 JavaScript 编译器会检查 function 外面是否定义 attr 变量，由于声明了一个值为 1 的 attr 变量，所以第一次调用 getAttrInc 函数时会传回 2。但如果 attr 并没有被声明过，那么 attr = attr + 1 这个式子会产生错误，因为等号右边的 attr+1 会先被计算，而由于它没被声明过，所以就错了。

```
var attr=1;

function getAttrInc(){
  attr = attr + 1;
  return attr;
}
```

再者，因为getAttrInc函数内的attr属性始终会参考到外部的attr变量，所以每次一调用getAttrInc函数时，这个attr变量值都会被加1，也许你只是要取得attr加1的值，而不希望attr变量的值被改，此时应该把它改成下面的方式：

```
var attr=1;

function getAttrInc(){
  var tmp = attr + 1;
  return tmp;
}
```

全局变量（global variables）与局部变量（local variables）

在JavaScript中，只要是在函数外声明的变量就称为全局变量，全局变量一经声明后在每一个地方都可以参考到它，所以，如果要定义一个共用的变量，应该将它声明为全局变量，这个变量和它的值会一直存在着，除非网页被重新载入。至于定义在函数内的变量则称为局部变量，如果变量只是在函数内被用到，就应该将它用var声明，这样除了避免更改到已经声明过的变量值外，这个局部变量的内存内容也会随着函数的执行完毕而被回收，否则如果没有用var来定义一个不曾被声明过的变量，那么在函数执行完成后它会变成一个全局的变量，举个例子说明这种情形：

```
<script language="JavaScript">
  function testFunc(){
    attr=3;
    var c=4;
  }

  testFunc();

  alert(++attr);
    //attr参考到testFunc声明的attr全局变量
  alert(c)
    //错误，无法参考到testFunc内用var声明的变量
</script>
```

你可能觉得奇怪，为什么会这样呢？其实是在编译的过程中，JavaScript 将 testFunc 内的 attr=3 编译成 window.attr=3，因为 window 是预设的参考对象，也就是 window 被增加了一个 attr 的属性而导致它成了全局的变量！

保留字（Reserved Words）

JavaScript 内定有一些保留字是不允许用来当变量名称、函数名称或是标签名称，因为这些保留字在 JavaScript 的语法上有其个别意义，这些保留字如下：

break	case	continue	default	delete
do	else	export	false	for
function	if	import	in	new
null	return	switch	this	true
typeof	var	void	while	with

除了语法上的保留字外，一些内定对象的名称、属性名称和函数名称也要避免，例如 window 对象和它的所有属性和函数名称，如 document、history、location、name、alert、confirm、prompt、open、opener、blur、focus、status、defaultStatus 等等，内定的函数名称 Object、Math、String、Number、Boolean、Date、Array、RegExp 等等。特别要注意的是 window 的所有属性，因为 window 是预设的参考对象，如果取了一个和 window 对象属性名称一模一样的，很可能会不小心改变了它的预设值。

基本数据类型（primitive types）

JavaScript 提供三种基本数据类型 boolean、number 和 string：

boolean

boolean 类型只有两个可能的值 true 和 false，这两个值都是小写，boolean 值是比较运算式的结果，例如下面的式子：

```
b = n > 3;
```

当 n 大于 3 时 n>3 会传回 true，b 等于 true，而如果 n 小于或等于 3 时会传回 false，所以 b 会是 false。boolean 值也可能是逻辑运算式的结果，例如：

```
b = ( n > 3 ) && ( n < 6 );
```

当n的值是4或5时会传回true，否则会传回false。逻辑运算式的结果在JavaScript中不一定是boolean类型的true或false，为什么呢？后面在介绍逻辑运算符时会详细说明。

number

number是最基本的数据类型，在JavaScript中是以64位，也就是8个bytes的双精准度浮点数值（double precision floating point number）来表示，正数的范围约为（5e-324 ~ 1.797693e+308），负数的范围约为（-1.797693e+308 ~ -5e-324），此处的e+n是以10为底的n次方，e-n是以10为底的-n次方。Number有4种表示法：指数（exponential）、十进制（decimal）、八进制（octal）和十六进制（hexadecimal）表示法，例如：

5e+3 -4e-3 1.3e-3	exponential
1234	decimal，非0开头的数值
0123 0777 045	octal，以0开头的数值，且每个数字的范围是0~7
0xFF 0xaf 0x93C	hexadecimal，以0x开头的数值，且每个数字的范围是0~9和A~F，A~F可以是小写，分别代表十进制的10~15

String

string是指包含在两个单引号（''）或双引号（""）中的字符，不过也可以是没有任何字符的空字符串，例如"JavaScript"、'Hello'和""或''空字符串等。有一些特殊字符或特殊功能的字符是不能直接包含在字符串里的，例如单引号（'）、双引号（"）和换行字符等等，这些字符称为Escape Characters，如果要显示这些字符则都要以反斜线(\)开头：

Escape Characters	说 明
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab

Escape Characters	说 明
\'	Single quote
\"	Double quote
\\	Backslash
\XXX	ASCII 字符，每一个大 X 是一个八进制数值，范围为 0~377。例如： "\101" 是 "A"
\xXX	ASCII 字符，每一个大 X 是一个十六进制数值。例如："\x65 是 "e"
\uXXXX	Unicode 字符，每一个大 X 是一个十六进制数值。例如："\u005C" 是 Backslash，和 "\\" 结果一样，"\u000A" 是 new l ine，和 "\n" 结果是一样的

null 和 undefined

null 的类型是一个对象，用来表示一个变量没有任何数值，而 undefined 是指变量没有定义任何值：

```
var empty=null;
var value;
```

一个经过 var 声明的变量但是没有设定初始值时，它是 undefined，undefined 代表的也是一个值——就是它本身，你可以用 typeof 查看它的类型，是一个 undefined 字符串。undefined 的意义和 null 是一样的，当你用 == 比较这两个值时会发现它们是相等的，就以上面声明的 empty 和 value 为例，下面的比较式子结果是 true：

```
alert(empty==value)
```

你也可以不用 var 来声明一个变量，但是至少必须定义一个初始值给它，否则会发生语法错误。在 Navigator 里，可以直接将 undefined 设定给一个没有经过 var 声明的变量，但是 IE 不行，这一点要特别注意：

```
a=undefined;//Navigator 可以，但 IE 会发生错误
b=void(0);//IE和Navigator的结果都是undefined
```

NaN 和 Infinity

NaN 和 Inifinity 的数据类型都是 number，NaN 的意思是“不是一个数值”或是“没

有意义的算数运算”或是“无法转成数值类型”，例如：

```
var n1=0/0;
var n2=new Number( "12n" );
var n1Type=typeof n1;//n1Type的值是 "NaN"
var n2Type=typeof n2;// n2Type的值是 "NaN"
```

在程序执行时，有几种方式可以用来判断变量是否为一数值，以上面的n1为例：

```
n1==NaN //传回true
```

NaN是一个类型为nunber的保留字，你可以用比较运算符的==或是!=来检验变量或是某算数运算的结果是否为NaN。再者，可以利用typeof运算符来判断n1的类型字符串是否为"NaN"：

```
(typeof n1)=="NaN" //传回true
```

或是用isNaN函数来判断n1是否为NaN，如果是会传回true，否则传回false：

```
isNaN(n1); //传回true
```

Infinity的意思是无限大，也就是无法表示出来的时候，会传回number类型的Infinity。例如，下面的式子都会传回Inifinity：

```
1/0
5e+8000
```

任何数除以0会是Inifinity，而5e+8000已经超过最大的数值范围，所以也产生Infinity的结果。另外，还有一个isInfinte函数可以方便用来检查数值是否为Infinity，例如：

```
isFinite(2/0); //传回false，是Infinity
```

如果isFinite的传入参数是Infinite，则传回false，否则传回true。另外，Number对象也有提供POSITIVE_INFINITY和NEGATIVE_INFINITY属性，分别是Infinity和-Infinity，也可以利用它们来判断数值是否是Infinity或-Infinity，例如：

```
alert( (1/0)== Number.POSITIVE_INFINITY );//true
alert( (-1/0)== Number.NEGATIVE_INFINITY );//true
```

对象数据类型（object types）

基本数据类型（primitive types）所包含的只是一个单一的数据值，如前面介绍的字符串 string、数字 number 和布尔 boolean，而对象（Object）是一群属性（properties，可以是基本数据类型或对象）与方法（methods，即函数 function）的集合。这些属性与方法是用来描述有关对象的特性与行为，JavaScript 本身就是建构在对象的基础上，例如 window 对象有它的名称属性 name、所包含的文件属性 document、开启新视窗的方法 open( )和关闭视窗的方法 close( )等等。

JavaScript 提供许多对象的建构函数：Array,Boolean,Date,Function,Math,Number,RegExp,String,Image 和 Option 等。

你可以利用 new 运算符来建立它们的对象，new 运算符在本章后面会介绍，至于对象则在 Object 一章会有详细的说明。

逻辑运算符（Logical Operators）

JavaScript 提供三个逻辑运算符 &&（AND）、||（OR）和!(NOT)，逻辑运算符是使用在逻辑运算的式子里，逻辑运算式子的结果通常是 boolean 类型的 true 或 false，它的基本形式是：

```
操作数A  逻辑运算符  操作数B
```

这是当逻辑运算符是 && 和 || 时，它们必须有两个操作数。例如：

```
[illegible] 和 [illegible]
```

而!的操作数只有一个，紧接在!后面：

```
逻辑运算符（!）操作数
```

例如：

```
[illegible]
```

操作数可以是另一个逻辑运算式子的结果，例如：

```
(true && false) ||  true
```

下图分别用 &&、|| 和!说明不同的操作数将产生不同的结果：

AND

&&	true	false
true	true	false
false	false	false

OR

||	true	false
true	true	true
false	true	false

NOT

! true	false
! false	true

图 1-1

AND 和 OR 的右下 4 个方格是逻辑运算的结果，第一列下两个可以想成是操作数 A，而第一行右边两个是操作数 B，以 AND 为例：true&&false 的结果是 false，而 false||true 的结果是 true。&& 中文的意思是“且”，也就是当操作数 A 和操作数 B 都为 true 时，&& 的结果才是 true，我们用串联的两个开关来说明 AND 的情形：

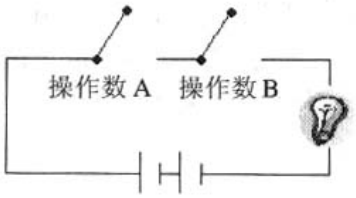

图 1-2

操作数A&&操作数B

由上面的线路图可以看出，除非两个开关（操作数 A 和操作数 B）都按下（true），电灯泡才会亮（true），若只按其中一个开关（操作数 A 或操作数 B）电灯泡是不会亮（false）的。

|| 中文的意思是“或”，除非两个操作数都是 false 结果才会是 false，否则只要任意一个操作数为 true，结果就会是 true。同样的，我们用并联的两个开关来说明 OR 的情形：

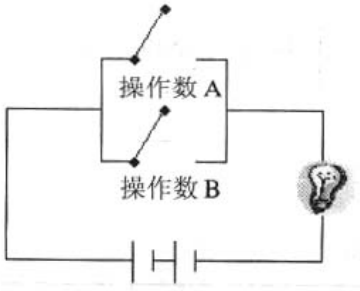

图 1-3

```
操作数A || 操作数B
```

只要操作数 A 或操作数 B 其中一个开关按下（true），电灯泡就会亮，除非两个操作数开关都是放开的，这样电灯泡才不会亮了（false）。

NOT!比较容易理解，它会把 true 转为 false 而把 false 变成 true。

接着我们用几个范例来说明 &&、|| 和!:

—&&—

```
true && true (=true)
true && false (=false)
true && (true && false)(=false)
(1==2) && true (=false)
```

除了 true 和 false 的 boolean 类型外，其他类型的值也可以当作逻辑运算式的操作数，其中特别要注意的是 0、null、""（空字符串）和 undefined 的意义也是 false。前面在解释 AND-&& 时曾说明过，只要有一个操作数是 true，结果就会是 true，在 JavaScript 中是这样定义的——如果 AND 的第一个操作数是 true 或非 false，结果就会是第二个操作数的值。例如

```
"black" && "white"  (="white")
true && "white"  (="white")
"Cat" && false  (=false)
true && 0  (=0)
(new Number(0)) && true  (=true)
alert( new String('') && false)  (=false)
```

new Number(0)的值虽然是 0，不过它的类型是 Number 对象（非 false），new String ("")意思也是一样。

再者，如果 AND 的第一个操作数是 false 或非 true，结果就是第一个操作数的值。例如：

```
null && true  (=null)
"" && false  (="")
false && "white"  (=false)
```

—||—

```
false || false  (=false)
true || false  (=true)
true || (true && false)  (=true)
(1==2) || true  (=true)
```

除非第一个和第二个操作数都是 false，否则结果将会是 true。和 && 一样，JavaScript 中是这样定义 OR —|| 的：如果 OR 的第一个操作数是 true 或非 false，结果就会是第一个操作数的值。例如：

```
"black" || "white"  (="black")
true || "white"  (="true")
"Cat" || false  (="Cat")
true || 0  (=true)
(new Number(0)) || true  (=0,非false)
alert( new String("") || false)  (="",非false)
```

虽然(new Number(0)) || true 的结果是 0，不过类型是 Number 对象，并非 false。再者，如果 OR 的第一个操作数是 false 或非 true，结果就会是第二个操作数的值。例如：

```
null || true  (=true)
"" || false  (=false)
"" || 0  (=0)
false && "white"  (="white")
```

—! —

not 的结果不是 true 就是 false。例如：

```
! false  (=true)
! ""  (=true)
! 0  (=true)
! "123"  (=false)
```

比较运算符（Comparison Operators）

JavaScript 提供八个比较运算符，比较运算符用在比较运算的式子里，比较运算式子的结果都是 boolean 类型的 true 或 false，它的形式是：

操作数A 比较运算符 操作数B

算数运算符	说 明	例 子
==	比较两个操作数的值是否相等，相等时传回 true，否则传回 false。如果两个操作数的类型不一样的话，JavaScript 会试着将他们转换成相同的类型然后再作比较。	1 == 1(true) 2 == "2"(true) '3' == "3"(true)
!=	如果两个操作数的值不相等，则传回 true，否则传回 false。如果两个操作数的类型不一样的话，JavaScript 会试着将他们转换成相同的类型，然后再作比较。	1 != "1"(false) '2' != "2"(false) 3 != 4(true)
>	如果第一个操作数的值大于第二个操作数的值，则传回 true，否则传回 false。如果两个操作数的类型不一样的话，JavaScript 会试着将他们转换成相同的类型，然后再作比较。	2 > 0(true) "2" > 3(false) 3 > '2'(true)
<	如果第一个操作数的值小于第二个操作数的值，则传回 true，否则传回 false。如果两个操作数的类型不一样的话，JavaScript 会试着将他们转换成相同的类型，然后再作比较。	2 < 0(false) "2" < 3(true) 3 < '2'(false)
>=	如果第一个操作数的值大于或等于第二个操作数的值，则传回 true，否则传回 false。如果两个操作数的类型不一样的话，JavaScript 会试着将他们转换成相同的类型，然后再作比较。	2 >= 0(false) "2" >= 2(true) 3 >= '2'(true)
<=	如果第一个操作数的值小于或等于第二个操作数的值，则传回 true，否则传回 false。如果两个操作数的类型不一样的话，JavaScript 会试着将他们转换成相同的类型，然后再作比较。	2 <= 0(false) "2" <= 2(true) 3 <= '2'(false)
===	如果两个操作数的类型和值都相等的话，则传回 true，否则传回 false。JavaScript 对操作数不会做类型的转换。	2 === 2(true) 3 === "3"(false)
!==	如果两个操作数的类型或值不相等时，则传回 true，否则传回 false。JavaScript 对操作数不会做类型的转换。	2 != 3(true) 2 != '2'(true)

===和！==比较运算符会先检查两个操作数的类型是否一样，如果一样的话，再比较值是否相同，不过这只限于基本数值（number）与字符串（string）类型的比较。对象的比较就不同了，即使是两个值相同的字符串对象(Number object)或数值对象（String object），他们仍然是不等的，除非是指向同一个对象。例如：

```
var a=new Number(2);
var b=new Number(2);
alert(a===b); //结果会是false
alert(a===a); //结果会是true
```

算数运算符（Arithmetic Operators）

算数运算符	说 明	例 子
∠	两个操作数的加法	3+5=8
⌒	两个操作数的减法	9-2=7
*	两个操作数的乘法	2*3=6
/	两个操作数的除法	3/2=1.5
++	单一操作数加一	将操作数的值加1，例如： var x=3; ++x或x++后x=4
--	单一操作数减一	将操作数的值减1，例如： var x=3; --x或x--后x=2
%	求两个操作数相除的余数	10%3=1 12%5=2
-	操作数的相反数	var x=3 x=-3

除法（/）的结果是浮点数，不同于C或Java的结果是整数。

增量（++）：

++运算符的操作数只有一个，用来将操作数的值加1，例如：

```
var x=3; ++x或x++后x=4
```

如果是用在算式中，则会依照操作数摆放位置而定，例如：

```
var x=3;
var y=x++;
```

会先将 x 等于 3 的值设给 y，然后将 x 值加一，所以 y 值会是 3 而 x 值是 4，再者如：

```
var x=3;
var y=++x;
```

由于 ++ 摆在操作数 x 的前面，所以会先将 x 值加 1，此时 x=4，然后再将 x 等于 4 的值设给 y，所以 x 和 y 值都会是 4。

减量（－－）

如同 ++ 运算符，－－运算符的操作数也只有一个，用来将操作数的值减 1，例如：

var x=3; －－x 或 x －－后 x=2，如果是用在算式中，则会依照操作数摆放的位置而定，例如：

```
var x=3;
var y=x--;
```

会先将 x 等于 3 的值设给 y，然后将 x 值减一，所以 y 值会是 3 而 x 值是 2，再者如：

```
var x=3;
var y=--x;
```

先将 x 值减 1，此时 x=2，然后将 x 等于 2 的值设给 y，所以 x 和 y 值都会是 2。

位运算符(Bitwise Operators)

位运算符是用来对操作数的每个位作运算，而操作数是以 32 位二进制表示法表示，最左边的位是符号位（sign-bit，正为 0、负为 1），例如：

```
(12)10=(00000000 00000000 00000000 00001100)2
(-12)10=(11111111 11111111 11111111 11110100)2
```

负数是用 2 的补数表示法，例如上面－12 的 32 位二进制值是将 12 的 32 位二进制值的每个位作 NOT 运算（0 变 1、1 变 0）然后再加 1。

虽然运算的对象是位，不过传回的值仍是 10 进制数值的表示法，例如某位运算的结果是：