第 2 章　JavaScript 语法

JavaScript 脚本语言作为一门功能强大、使用范围较广的程序语言，其语言基础包括数据类型、变量、运算符、函数及核心语句等内容。本章主要介绍 JavaScript 脚本语言的基础知识，带领读者初步领会 JavaScript 脚本语言的精妙之处，并为后续章节的深入学习打下坚实的基础。

- JavaScript 的基本数据类型
- JavaScript 的表达式和常用运算符
- JavaScript 的语句构成
- 函数的使用及其属性和方法

2.1　JavaScript 语法基础

2.1.1　变量

变量（Variable）是相对于常量而言的，常量通常是一个不会改变的固定值，而变量是对应到某个值的一个符号，这个符号中的值可能会随着程序的执行而改变，因此称为“变量”。JavaScript 脚本语言和其他程序设计语言一样也引入了变量，其主要作用是存取数据及提供存放信息的容器。在实际脚本开发过程中，变量为开发者与脚本程序交互的主要工具。

JavaScript 中的变量命名同其他语言非常相似，这里要注意以下几点：

（1）第一个字符必须是字母（大、小写均可）、下划线（_）或者美元符号（$）。

（2）后续的字符可以是字母、数字、下划线或者美元符号。

（3）变量名称不能是关键字或保留字。

（4）不允许出现中文变量名，且大小写敏感。

在 JavaScript 脚本语言中，声明变量的过程相当简单，JavaScript 脚本语言使用关键字 var 作为其唯一的变量标识符，其用法为在关键字 var 后面加上变量名。

例如，通过下面的代码声明名为 age 的变量：

```
var age;
```

JavaScript 脚本语言允许开发者不用声明变量就能直接使用，而在变量赋值时自动声明该变量。一般来说，为培养良好的编程习惯，同时为了使程序结构更加清晰易懂，建议在使用变量前对变量进行声明。

变量赋值和变量声明可以同时进行。例如，下面的代码声明名为 age 的变量，同时给该变量赋初值 25：

```
var age = 25;
```

当然，可在一句 JavaScript 脚本代码中同时声明两个以上的变量，例如：

```
var age , name;
```

同时初始化两个以上的变量也是允许的，例如：

```
var age = 35 , name = "tom";
```

JavaScript 中的变量可以根据其有效范围分为全局变量和局部变量两种。其中，全局（Globe）变量从定义开始，到整个 JavaScript 代码结束为止，都可以使用；而局部（Local）变量只有在函数内部才有效。如果不写 var，而是直接对变量进行赋值，那么 JavaScript 将自动把这个变量声明为全局变量。

2.1.2 关键字与保留字

ECMA-262 定义了 JavaScript 支持的一套关键字（Keyword）。根据规定，关键字不能用作变量名或函数名。表 2-1 是关键字的完整列表。

表 2-1 JavaScript 关键字

break	case	catch	continue	default
delete	do	else	finally	for
function	if	in	instanceof	new
return	switch	this	throw	try
typeof	var	void	while	with

JavaScript 还定义了一套保留字（Reserved Word）。保留字在某种意义上是为将来的关键字而保留的单词。因此，保留字也不能被用作变量名或函数名。保留字的完整列表如表 2-2 所示。

表 2-2 JavaScript 保留字

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

2.1.3 原始值与引用值

在 JavaScript 中，变量可以存放两种类型的值，即原始值和引用值。原始值指的就是代表原始数据类型（基本数据类型）的值，即 Undefined、Null、Number、String、Boolean 类型所表示的值。引用值指的就是复合数据类型的值，即 Object、Function、Array 及自定义对象等。

原始值是存储在栈中的简单数据段，也就是说，它们的值直接存储在变量访问的位置。堆是存放数据的基于散列算法的数据结构，在 JavaScript 中，引用值是存放在堆中的。引用值是存储在堆中的对象，也就是说，存储在变量处的值（即指向对象的变量，存储在栈中）是一个指针，指向存储对象的内存处。

为变量赋值时，JavaScript 的解释程序必须判断该值是原始类型的还是引用类型的。要实现这一点，解释程序需要尝试判断该值是否为 JavaScript 的原始类型之一，即 Undefined、Null、Boolean、String 类型。由于这些原始类型占据的空间是固定的，所以可以将它们存储在较小的内存区域——栈中。这样便于迅速查询变量的值。

如果一个值是引用类型的，那么它的存储空间将从堆中分配。由于引用值的大小会改变，所以不能把它放在栈中，否则会降低变量查询的速度。相反，放在变量的栈空间中的值是该对象存储在堆中的地址。地址的大小是固定的，所以把它存储在栈中对变量性能无任何负面影响。

2.2　JavaScript 数据类型

2.2.1　基础数据类型

变量包含多种类型，JavaScript 脚本语言支持的基本数据类型包括 Number 型、String 型、Boolean 型、Undefined 型和 Null 型，分别对应于不同的存储空间，如表 2-3 所示。

表 2-3　基本数据类型

类型	举例	简要说明
Number	45 , -34 , 32.13 , 3.7E-2	数值型数据
String	"name" , 'Tom'	字符型数据，需加双引号或单引号
Boolean	true , flase	布尔型数据，不加引号，表示逻辑真或假
Undefined		表示未定义
Null	null	表示空值

1．Number 型

Number 型数据即为数值型数据，包括整数型和浮点型，整数型数制可以使用十进制、八进制及十六进制标识，而浮点型为包含小数点的实数，且可用科学记数法来表示。一般来说，Number 型数据为不在括号内的数字，例如：

```
var myDataA=8;
var myDataB=6.3;
```

上述代码分别定义值为整数 8 的 Number 型变量 myDataA 和值为浮点数 6.3 的 Number 型变量 myDataB。

除了常用的数字之外，JavaScript 还支持以下两个特殊的数值：

（1）Infinity。当在 JavaScript 中使用的数字大于 JavaScript 所能表示的最大值时，JavaScript 就会将其输出为 Infinity，即无穷大的意思。当然，如果 JavaScript 中使用的数字小于 JavaScript 所能表示的最小值，JavaScript 也会输出-Infinity。

（2）NaN。JavaScript 中的 NaN 是“not a number”（不是数字）的意思。通常是在进行数

字运算时产生了未知的结果或错误，JavaScript 就会返回 NaN，这代表着数字运算的结果是一个非数字的特殊情况。如用 0 来除以 0，JavaScript 就会返回 NaN。NaN 是一个很特殊的数字，不会与任何数字相等，包括 NaN。在 JavaScript 中只能使用 isNaN()函数来判断运算结果是不是 NaN。

2. String 型

String 型数据表示字符型数据。JavaScript 不区分单个字符和字符串，任何字符或字符串都可以用双引号或单引号括起来。例如，下列语句中定义的 String 型变量 nameA 和 nameB 包含相同的内容：

```
var nameA = "Tom";
var nameB = 'Tom';
```

如果字符串本身含有双引号，则应使用单引号将字符串括起来；若字符串本身含有单引号，则应使用双引号将字符串括起来。一般来说，在编写脚本过程中，双引号或单引号的选择在整个 JavaScript 脚本代码中应尽量保持一致，以养成好的编程习惯。

3. Boolean 型

Boolean 型数据表示的是布尔型数据，取值为 true 或 false，分别表示逻辑真和逻辑假，且任何时刻都只能使用两种状态中的一种，不能同时出现。例如，下列语句分别定义 Boolean 变量 bChooseA 和 bChooseB，并分别赋予初值 true 和 false：

```
var bChooseA = true;
var bChooseB = false;
```

值得注意的是，给 Boolean 型变量赋值时，不能在 true 或 false 外面加引号，例如：

```
var happyA = true;
var happyB = "true";
```

上述语句分别定义初始值为 true 的 Boolean 型变量 happyA 和初始值为字符串"true"的 String 型变量 happyB。

4. Undefined 型

Undefined 型即为未定义类型，用于声明了变量但未对其初始化时赋予该变量的值。例如，下列语句定义变量 name 为 Undefined 型：

```
var name;
```

Undefined 类型只有一个值，即 undefined。当声明的变量未初始化时，该变量的默认值是 undefined。定义 Undefined 型变量后，可在后续的脚本代码中对其进行赋值操作，从而自动获得由其值决定的数据类型。

5. Null 型

Null 型数据表示空值，它只有一个专值 null，null 用来表示尚未存在的对象。如果函数或方法要返回的是对象，找不到该对象时，返回的通常是 null。

2.2.2 数据类型转换

JavaScript 是一种无类型的语言，这种“无类型”并不是指 JavaScript 没有数据类型，而是指 JavaScript 是一种松散类型、动态类型的语言。因此，在 JavaScript 中定义一个变量时，不需要制定变量的数据类型，这就使得 JavaScript 可以很方便、灵活地进行隐式类型转换。所谓隐式类型转换，就是不需要程序员定义，JavaScript 会自动将某一个类型的数据转换成另一个类型的数据。JavaScript 隐式类型转换的规则是：将类型转换到环境中应该使用的类型。

JavaScript 中除了可以隐式转换数据类型之外，还可以显式转换数据类型。显式转换数据类型可以增强代码的可读性。常用的类型转换方法有以下几种：

1. 转换成字符串

JavaScript 中三种主要的原始值布尔值、数字、字符串及其他对象都有 toString()方法，可以把它们的值转换成字符串。代码如下：

```
var myNum = 100;
console.log(myNum.toString());    //输出“100”
var bFound = false;
console.log(bFound.toString());     //输出“false”
```

各种类型向字符串型转换的结果如下：

（1）undefined 值：转换成“undefined”。

（2）null 值：转换成“null”。

（3）布尔值：值为 true，则转换成“true”；值为 false，则转换成“false”。

（4）数字型值：NaN 或数字型变量的完整字符串。

（5）其他对象：如果该对象的 toString()方法存在，则返回 toString 方法的返回值，否则返回 undefined。

2. 转换成数字

ECMAScript 提供了两种把非数字的原始值转换成数字的方法，即 parseInt()和 parseFloat()。只有对 String 类型调用这些方法，它们才能正确运行，对其他类型返回的都是 NaN。

（1）提取整数的 parseInt()方法。

parseInt()方法用于将字符串转换为整数，其格式为：

```
parseInt(numString,[radix])
```

需要说明的是：

1）第一个参数为必选项，用来指定要转化为整数的字符串。当使用仅包括第一个参数的 parseInt()方法时，表示将字符串转换为整数。其转换过程为：从字符串第一个字符开始读取数字（跳过前导空格），直至遇到非数字字符时停止读取，将已经读取的数字字符串转换为整数，并返回该整数值。如果字符串的开始位置不是数字，而是其他字符（空格除外），那么 parseInt()方法返回 NaN，表示所传递的参数不能转换为一个整数。例如：

```
parseInt("437abc45");        //返回值为 437
```

2）第二个参数是可选项。使用该参数的 parseInt()方法能够完成八进制、十六进制等数据的转换。其中[radix]表示要将 numString 作为几进制数进行转化，[radix]的值在 2～36 之间。当省略第二个参数时，默认将第一个参数按十进制转换。但如果字符以 0x 或 0X 开头，那么按十六进制转换。不管指定按哪一种进制转换，方法 parseInt()总是以十进制值返回结果。例如：

```
parseInt("100abc",8);
```

表示将“100abc”按八进制数进行转换，由于“abc”不是数字，所以实际是将八进制数 100 转换为十进制数，转换的结果为十进制数 64。

（2）提取浮点数的 parseFloat()方法。

parseFloat()方法用于将字符串转换为浮点数，其格式为：

```
parseFloat(numString)
```

parseFloat()方法与 parseInt()方法很相似。不同之处在于 parseFloat()方法能够转换浮点数。参数 numString 即为要转换的字符串，如果字符串不是以数字开始，则 parseFloat()方法返回

NaN，表示所传递的参数不能转换为一个浮点数。例如：

```
parseFloat(19.3abc);        //转化的结果为 19.3
```

3．基本数据类型转换

在 JavaScript 中可以使用下面 3 个函数将数据转换成数字型、布尔型和字符串型，下面看一下它的几个强制转换的函数：

（1）Boolean(value)：把值转换成 Boolean 类型。

（2）Number(value)：把值转换成数字（整型数或浮点数）。

（3）String(value)：把值转换成字符串。

首先来看 Boolean()。如果要转换的值为“至少有一字符的字符串”、“非 0 的数字”或“对象”，那么 Boolean()将返回 true；如果要转换的值为“空字符串”、“数字 0”、“undefined”、“null”，那么 Boolean()会返回 false。例如：

```
var t1 = Boolean("");                          //返回 false，空字符串
var t2 = Boolean("s");                         //返回 true，非空字符串
var t3 = Boolean(0);                           //返回 false，数字 0
var t4 = Boolean(1), t5 = Boolean(-1);         //返回 true，非 0 数字
var t6 = Boolean(null), t7 = Boolean(undefined);  //返回 false
var t8 = Boolean(new Object());                //返回 true，对象
```

再来看看 Number()。Number()与 parseInt()和 parseFloat()类似，它们区别在于 Number()转换的是整个值，而 parseInt()和 parseFloat()则可以只转换开头的数字部分。例如，Number("1.2.3")、Number("123abc")会返回 NaN，而 parseInt("1.2.3")返回 1、parseInt("123abc")返回 123、parseFloat("1.2.3")返回 1.2、parseFloat("123abc")返回 123。Number()会先判断要转换的值能否被完整地转换，然后再判断是调用 parseInt()还是 parseFloat()。表 2-4 列出了一些值调用 Number()之后的结果。

表 2-4　调用 Number()方法的结果

用法	结果
Number(false)	0
Number(true)	1
Number(undefined)	NaN
Number(null)	0
Number("1.2")	1.2
Number("12")	12
Number("1.2.3")	NaN
Number(new Object())	NaN
Number(123)	123

最后是 String()。这个就比较简单了，它可以把所有类型的数据转换成字符串，例如：

```
String(false);     //返回"false"
String(1);         //返回"1"
```

String()和 toString()方法有些不同，区别在于对 null 或 undefined 值用 String()进行强制类型转换可以生成字符串而不引发错误，代码如下：

```
var t1 = null;
var t2 = String(t1);       //t2 的值 "null"
var t3 = t1.toString();    //这里会报错
var t4;
var t5 = String(t4);       //t5 的值 "undefined"
var t6 = t4.toString();    //这里会报错
```

2.2.3　引用类型

除了基本的数据类型之外，JavaScript 还支持引用类型，引用类型包括对象和数组两种。本节将简要介绍上述引用类型的基本概念及其用法，在本书后续章节将进行专门论述。

1. 对象

JavaScript 中的对象是一个属性的集合，其中的每一个都包含一个基本值。对象中的数据是已命名的数据，通常作为对象的属性来引用，这些属性可以访问值。保存在属性中的每个值都可以是一个值或是一个对象，甚至是一个函数。对象使用花括号创建。例如，下面的代码创建了一个名为 myObject 的空对象：

```
var myObject = {};
```

这里有一个带有几个属性的对象：

```
var dvdCatalog = {
    "identifier": "1",
    "name": "Coho Vineyard"
};
```

这段示例代码创建了一个名为 dvdCatalog 的对象，它有两个属性，一个叫做 identifier，另一个叫做 name。两个属性中包含的值分别是 1 和“Coho Vineyard”。可以使用以下方法访问 dvdCatalog 对象的 name 属性：

```
dvdCatalog.name
```

JavaScript 中的对象除了拥有属性之外，还可以拥有方法。例如，一个窗口（Window）对象有一个名为 alert 的方法，可以通过以下方法来引用：

```
window.alert(message)
```

2. 数组

数组和对象一样，也是一些数据的集合，这些数据也可以是字符串类型、数字类型、布尔类型，或者是引用类型。例如，下面的定义：

```
var score = [56,34,23,76,45];
```

上述语句创建数组 score，中括号“[]”内的成员为数组元素。由于 JavaScript 是弱类型语言，因此不要求目标数组中各元素的数据类型均相同，例如：

```
var score = [56,34, "23",76, "45"];
```

在数组中为每个数据都编了一个号，这个号称为数组的下标。在 JavaScript 中数组的下标是从 0 开始的，通过使用数组名加下标的方法可以获取数组中的某个数据。例如，下列语句声明变量 m 返回数组 score 中第四个元素：

```
var m = score [3];
```

2.3　JavaScript 运算符

编写 JavaScript 脚本代码过程中，对目标数据进行运算操作需用到运算符。运算符用于

将一个或者几个值变成结果值，使用运算符的值称为操作数，运算符即操作数的组合称为表达式。JavaScript 脚本语言支持很多种运算符，下面分别予以介绍。

2.3.1 算术运算符

算术运算符是最简单、最常用的运算符，可以使用它们进行通用的数学计算，如表 2-5 所示：

表 2-5 算术运算符

运算符	表达式	说明	示例
+	x+y	返回 x 加 y 的值	x=4+2，结果为 6
-	x-y	返回 x 减 y 的值	x=8-6，结果为 2
*	x*y	返回 x 乘以 y 的值	x=3*5，结果为 15
/	x/y	返回 x 除以 y 的值	x=6/3，结果为 2
%	x%y	返回 x 与 y 的模（x 除以 y 的余数）	x=8%3，结果为 2
++	x++、++x	返回数值递增、递增并返回数值	
--	x--、--x	返回数值递减、递减并返回数值	

这里需要注意的是：自加和自减运算符放置在操作数的前面和后面其含义不同。运算符写在变量名前面，则返回值为自加或自减前的值；而写在后面，则返回值为自加或自减后的值。如下面代码所示：

```
var x = 5, y = 0;
y = x++;            //先执行 y=x; 后执行 x=x+1;
```

上述代码执行后，x 的值为 6，y 的值为 5。如果将代码改成下面的前置形式：

```
var x = 5, y = 0;
y = ++ x;           //先执行 x=x+1; 后执行 y=x;
```

修改后的代码执行后，x 的值为 6，y 的值也为 6。

由上面的代码示例可以看出：

（1）若自加（或自减）运算符放置在操作数之后，执行该自加（或自减）操作时，先将操作数的值赋值给运算符前面的变量，然后操作数自加（或自减）。

（2）若自加（或自减）运算符放置在操作数之前，执行该自加（或自减）操作时，操作数先进行自加（或自减），然后将操作数的值赋值给运算符前面的变量。

JavaScript 脚本语言的运算符在参与数值运算时，其右侧的变量将保持不变。从本质上讲，运算符右侧的变量作为运算的参数而存在，脚本解释器执行指定的操作后，将运算结果作为返回值赋予运算符左侧的变量。赋值运算符（=）是编写 JavaScript 脚本代码时最为常用的操作，其作用是给一个变量赋值，即将某个数值制定给某个变量。有些赋值运算符可以和其他运算符组合使用，对变量中包含的值进行计算，然后用新值更新变量，表 2-6 中列出了一些常用的赋值运算符。

表 2-6 赋值运算符

运算符	说明	示例
=	将运算符右边变量的值赋给左边变量	m=n
+=	将运算符两侧变量的值相加并将结果赋给左边变量	m+=n

续表

运算符	说明	示例
-=	将运算符两侧变量的值相减并将结果赋给左边变量	m-=n
*=	将运算符两侧变量的值相乘并将结果赋给左边变量	m*=n
/=	将运算符两侧变量的值相除并将整除的结果赋给左边变量	m/=n
%=	将运算符两侧变量的值相除并将余数赋给左边变量	m%=n

2.3.2　逻辑运算符

逻辑运算符通常用于执行布尔运算，JavaScript 脚本语言的逻辑运算符包括“&&”、“||”和“!”等，用于两个逻辑型数据之间的操作，返回值的数据类型为布尔型。表 2-7 列出了 JavaScript 的逻辑运算符。

表 2-7　逻辑运算符

运算符	表达式	说明	示例
&&	表达式 1&&表达式 2	若两边表达式的值都为 true，则返回 true；任意一个值为 false，则返回 false	5>3&&5<6 返回 true 5>3&&5>6 返回 false
||	表达式 1||表达式 2	只有表达式的值都为 false 时，才返回 false，否则返回 true	5>3||5>6 返回 true 5>7||5>6 返回 false
!	! 表达式	求反。若表达式的值为 true，则返回 false，否则返回 true	!(5>3) 返回 false !(5>6) 返回 true

2.3.3　关系运算符

JavaScript 脚本语言中用于比较两个数据的运算符称为关系运算符，包括“==”、“!=”、“>”、“<”、“<=”和“>=”等。关系运算符用于比较两个操作数的大小，其比较的结果是一个布尔型的值。当两个操作数满足关系运算符指定的关系时，表达式的值为 true，否则为 false。其具体作用见表 2-8。

表 2-8　关系运算符

运算符	说明	示例
==	相等，若两数据相等，则返回布尔值 true，否则返回 false	num==8
!=	不相等，若两数据不相等，则返回布尔值 true，否则返回 false	num!=8
>	大于，若左边数据大于右边数据，则返回布尔值 true，否则返回 false	num>8
<	小于，若左边数据小于右边数据，则返回布尔值 true，否则返回 false	num<8
>=	大于或等于，若左边数据大于或等于右边数据，则返回布尔值 true，否则返回 false	num>=8
<=	小于或等于，若左边数据小于或等于右边数据，则返回布尔值 true，否则返回 false	num<=8

2.3.4　位运算符

位运算符是对操作数按其在计算机内表示的二进制数逐位地进行逻辑运算或移位运算。位运算符是对其操作数（要求是整型的操作数）按二进制形式逐位进行运算，运算完毕后，将

结果转换成十进制数值。位操作运算符如表 2-9 所示。

表 2-9 位运算符

运算符	说明	示例
&	按位与，若两数据对应位都是 1，则该位为 1，否则为 0	9&4
^	按位异或，若两数据对应位相反，则该位为 1，否则为 0	9^4
|	按位或，若两数据对应位都是 0，则该位为 0，否则为 1	9|4
~	按位非，若数据对应位为 0，则该位为 1，否则为 0	～4
>>	算术右移，将左侧数据的二进制值向左移动由右侧数值表示的位数，右边空位补 0	9>>2
<<	算术左移，将左侧数据的二进制值向右移动由右侧数值表示的位数，忽略被移出的位	9<<2
>>>	逻辑右移，将左边数据表示的二进制值向右移动由右边数值表示的位数，忽略被移出的位，左侧空位补 0	9>>>2

例如，有下面的例子。

```
var a = 6;                      //二进制值 0000 0110b
var b = 36;                     //二进制值 0010 0100b
var result = 0;
result = a&b;                   //结果为二进制 0000 0100b，对应的十进制结果为 4
result = a^b;                   //结果为二进制 0010 0010b，对应的十进制结果为 34
result = a|b;                   //结果为二进制 0010 0110b，对应的十进制结果为 38
result = ~a;                    //结果为二进制 1000 0111b，对应的十进制结果为 -7
var targetValue = 189;          //目标数据二进制值 1011 1101b
var iPos = 2;                   //目标数据移动的位数
result = targetValue>>iPos;     //结果为二进制 0010 1111b，对应的十进制结果为 47
result = targetValue<<iPos;     //结果为二进制 10 1111 0100b，对应的十进制结果为 756
result = targetValue>>>iPos;    //结果为二进制 0010 1111b，对应的十进制结果为 47
```

2.4 JavaScript 语句

表达式的作用只是生成并返回一个值，在 JavaScript 中还有很多种语句，通过这些语句可以控制程序代码的执行次序，从而可以完成比较复杂的程序操作。

2.4.1 选择语句

选择语句是 JavaScript 中的基本控制语句之一，其作用是让 JavaScript 根据条件选择执行哪些语句或不执行哪些语句。在 JavaScript 中的选择语句可以分为 if 语句和 switch 语句两种。

1．if 语句

if 条件假设语句是比较简单的一种选择结构语句，若给定的逻辑条件表达式为真，则执行一组给定的语句。其基本结构如下：

```
if(conditions)
{
    statements;
}
```

逻辑条件表达式 conditions 必须放在小括号里，且仅当该表达式为真时，执行花括号内包

含的语句，否则将跳过该条件语句而执行其下的语句。花括号内的语句可为一个或多个，当仅有一个语句时，花括号可以省略。但一般而言，为养成良好的编程习惯，同时增强程序代码的结构化和可读性，建议使用花括号将指定执行的语句括起来。

if 后面可增加 else 进行扩展，即组成 if...else 语句，其基本结构如下：

```
if(conditions)
{
    statement1;
}
else
{
    statement2;
}
```

当逻辑条件表达式 conditions 运算结果为真时，执行 statement1 语句（或语句块），否则执行 statement2 语句（或语句块）。

当需要提供多重选择时，可以使用 if...else if...else 语句。其语法格式如下：

```
if (条件 1)
{
  条件 1 成立时执行代码
}
else if (条件 2)
{
  条件 2 成立时执行代码
}
else
{
  条件 1 和条件 2 均不成立时执行代码
}
```

if（或 if...else）结构可以嵌套使用来表示所示条件的一种层次结构关系。值得注意的是，嵌套时应重点考虑各逻辑条件表达式所表示的范围。

【例 2-1】求一元二次方程 $ax^2+bx+c=0$ 的根。

```
<html>
  <head>
    <title>if...else...示例</title>
    <script type="text/javascript">
      var a,b,c,x1,x2;
      a=1;
      b=3;
      c=2;
      if(a==0)
      {
        x1=-c/b;
        x2=x1;
        var str="方程的解为：x="+x1;
        console.log(str);
      }
      else if(b*b-4*a*c>=0)
```

```
            {
              x1=(-b+Math.sqrt(b*b-4*a*c))/(2*a);
              x2=(-b-Math.sqrt(b*b-4*a*c))/(2*a);
              var str="方程的解为：x1= "+x1+", x2= "+x2;
              console.log(str);
            }
            else
            {
              console.log("该方程无解！ ");
            }
          </script>
        </head>
        <body>
        </body>
      </html>
```

其中用到了 Math.sqrt()方法来求平方根，程序输出结果为：

```
方程的解为：x1= -1, x2= -2
```

在 if…else 语句中可以添加任意多个 else if 子句提供多种选择，但是使用多个 else if 子句经常会使代码变得非常繁琐。在多个条件中进行选择的更好方法是使用 switch…case 语句。

2. switch 语句

switch…case 语句提供了 if…else 语句的一个变通形式，可以从多个语句块中选择其中一个执行。switch…case 语句提供的功能与 if…else 语句类似，但是可以使代码更加简练易读。switch…case 语句在其开始处使用一个简单的测试表达式，表达式的结果将与结构中每个 case 子句的值进行比较。如果匹配，则执行与该 case 关联的语句块。其基本语法结构如下：

```
switch (a)
{
  case a1:
      statement 1;
      [break;]
  case a2:
      statement 2;
      [break];
  …
  default:
      [statement n;]
}
```

其中 a 是数值型或字符型数据，将 a 的值与 a1、a2、…、an 相比较，若 a 与其中某个值相等时，执行相应数据后面的语句，且当遇到关键字 break 时，程序跳出 statement n 语句，并重新进行比较；若找不到与 a 相等的值，则执行关键字 default 下面的语句。

【例 2-2】使用 switch…case 语句对学生分数进行分级。

```
<html>
  <head>
    <title>switch...case...示例</title>
    <script type="text/javascript">
      var score,flag;
```

```
            score=85;
            flag=(score-score%10)/10;
            switch(flag)
            {
               case 10:
               case 9:
                  console.log("成绩为优（90～100）");
                  break;
               case 8:
                  console.log("成绩为良（80～89）");
                  break;
               case 7:
                  console.log("成绩为一般（70～79）");
                  break;
               case 6:
                  console.log("成绩为及格（60～69）");
                  break;
               default:
                  console.log("成绩不及格");
            }
         </script>
      </head>
      <body>
      </body>
   </html>
```

程序输出结果为：

```
成绩为良（80～89）
```

需要注意 switch…case 语句只计算一次开始处的一个表达式，而 if…else 语句计算每个 else if 子句的表达式，这些表达式可以各不相同。仅当每个 else if 子句计算的表达式都相同时，才可以使用 switch…case 语句代替 if…else 语句。

3．?...:运算符

在 JavaScript 脚本语言中，“?...:”运算符用于创建条件分支。在动作较为简单的情况下，较之 if…else 语句更加简便，其语法结构如下：

```
(condition)?statementA:statementB;
```

载入上述语句后，首先判断条件 condition，若结果为真则执行语句 statementA，否则执行语句 statementB。值得注意的是，由于 JavaScript 脚本解释器将分号“;”作为语句的结束符，statementA 和 statementB 语句均必须为单个脚本代码，若使用多个语句会报错。例如，下列代码浏览器解释执行时得不到正确的结果：

```
(condition)?statementA:statementB;ststementC;
```

例如：var flag = (x>y) ? 1 : 0;

如果 x 的值大于 y 的值，则表达式的值为 1；否则，如果 x 的值小于或等于 y 值，则表达式的值为 0。

可以看出，使用“?...:”运算符进行简单的条件分支，语法简单明了，但若要实现较为复杂的条件分支，推荐使用 if…else 语句或者 switch 语句。

2.4.2 循环语句

在编写程序的过程中，有时需要重复执行某个语句块，这时就用到了循环语句。JavaScript 中的循环语句包括 while 语句、do…while 语句、for 语句和 for…in 语句 4 种。

1. while 语句

while 语句属于基本循环语句，用于在指定条件为真时重复执行一组语句。while 语句的语法结构如下：

```
while(condition)
{
    statements;
}
```

参数 condition 表示一个条件表达式，statements 表示当条件为 true 时所要反复执行的语句模块。while 循环语句是在逻辑条件表达式为真的情况下，反复执行循环体内包含的语句（或语句块）。

【例 2-3】依次打印输出 10 以内的偶数。

```
<html>
  <head>
    <title>while 循环</title>
    <script type="text/javascript">
      var i=0;
      while(i<10)
      {
        console.log(i);
        i+=2;
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

代码运行结果为：

```
0  2  4  6  8
```

该例中的 while 循环体执行了 5 次。需要注意的是：while 语句的循环变量 i 的赋值语句在循环体前，循环变量 i 的更新则放在循环体内。

在某些情况下，while 循环花括号内的 statements 语句（或语句块）可能一次也不被执行，因为对逻辑条件表达式的运算在执行 statements 语句（或语句块）之前。若逻辑条件表达式运算结果为假，则程序直接跳过循环而一次也不执行 statements 语句（或语句块）。

2. do…while 语句

do…while 语句类似于 while 语句，不同的是 while 语句是先判断逻辑条件表达式的值是否为 true 之后再决定是否执行循环体中的语句，而 do…while 循环语句是先执行循环体中的语句之后，再判断逻辑条件表达式是否为 true，如果为 true 则重复执行循环体中的语句。do…while 语句的语法结构如下：

```
do {
```

```
    statements;
}while(condition);
```

do…while 语句中各参数定义与 while 语句相同。若希望至少执行一次 statements 语句（或语句块），就可用 do…while 语句。

下面将通过一个例子区别 do…while 语句和 while 语句的用法。

【例 2-4】使用 do…while 语句。

```
<html>
  <head>
    <title>do...while 语句</title>
    <script type="text/javascript">
      var i=1,j=1,m=0,n=0;
      while(i<1)
      {
        m=m+1;
        i++;
      }
      console.log("while 语句循环执行了"+m+"次");
      do{
        n=n+1;
        j++;
      }while(j<1);
      console.log("do...while 语句循环执行了"+n+"次")
    </script>
  </head>
  <body>
  </body>
</html>
```

代码运行结果为：

```
while 语句循环执行了 0 次
do...while 语句循环执行了 1 次
```

在这个例子中变量 i、j 的初始值都为 1，do...while 语句与 while 语句的循环条件都是小于 1，但是由于 do...while 语句是先执行循环体再进行条件判断，因此即使条件判断为 false，但是循环体还是执行了一次。

3．for 语句

for 循环语句也类似于 while 语句，使用起来更为方便。for 语句按照指定的循环次数，循环执行循环体内语句（或语句块），它提供的是一种常用的循环模式，即初始化变量、判断逻辑条件表达式和改变变量值。for 语句的语法结构如下：

```
for(initialization; condition; loop-update)
{
    statements;
}
```

循环控制代码（即小括号内代码）内各参数的含义如下：

（1）initialization 表示循环变量初始化语句。

（2）condition 为控制循环结束与否的条件表达式，程序每执行完一次循环体内语句（或

语句块），均要计算该表达式是否为真，若结果为真，则继续运行下一次循环体内语句（或语句块）；若结果为假，则跳出循环体。

（3）loop-update 指循环变量更新的语句，程序每执行完一次循环体内语句（或语句块），均需要更新循环变量。

上述循环控制参数之间使用分号“;”间隔开来。初始化语句、条件语句和更新语句都可以选择，也可以省略，但是分号“;”不可以省略。

【例 2-5】使用 for 语句求一个数的阶乘。

```
<html>
  <head>
    <title>Untitled Document</title>
    <script type="text/javascript">
      var i=1,n=5,sum=1;
      for(i=1;i<=n;i++)
      {
        sum*=i;
      }
      console.log(n+"的阶乘是"+sum);
    </script>
  </head>
  <body>
  </body>
</html>
```

代码运行结果为：

```
5 的阶乘是 120
```

这个例子中 for 循环的执行过程如下：

1）执行“i=1;”初始化变量。

2）判断表达式“i<=n”是否为 true，如果返回 true 就执行步骤 3）；如果返回 false 则结束 for 循环语句。

3）执行“i++”语句，更新循环变量。

4）执行循环体中的语句。

5）重复执行步骤 2）。

4. for…in 语句

使用 for…in 循环语句可以遍历数组或者对指定对象的属性和方法进行遍历，其语法结构如下：

```
for (变量名 in 对象名)
{
    statements;
}
```

下面给出一个使用 for…in 语句的具体示例，输出了数组中的所有元素。

【例 2-6】使用 for…in 语句遍历数组。

```
<html>
  <head>
    <title>for...in 语句</title>
    <script type="text/javascript">
```

```
            var mycars=["Audi","Volvo","BMW"];
            for(var k in mycars)
            {
              console.log(mycars[k]);
            }
          </script>
        </head>
        <body>
        </body>
      </html>
```

代码运行结果为：

```
Audi
Volvo
BMW
```

2.4.3　跳转语句

所谓跳转语句，就是在循环控制语句的循环体中的指定位置或是满足一定条件的情况下直接退出循环。JavaScript 跳转语句分为 break 语句和 continue 语句。

1．break 语句

使用 break 语句可以无条件地从当前执行的循环结构或者 switch 结构的语句块中中断并退出，其语法格式如下：

```
break;
```

由于它是用来退出循环或者 switch 语句，所以只有当它出现在这些语句时，这种形式的 break 语句才是合法的。

【例 2-7】使用 break 语句。

```
<html>
  <head>
    <title>Untitled Document</title>
    <script type="text/javascript">
      for(var i=1;i<=5;i++)
      {
        if(i==3)   break;
        console.log(i);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

代码运行结果为：

```
1    2
```

上述代码中 for 语句在变量 i 为 1 和 2 时执行，当 i 为 3 时，if 语句条件为真，执行 break 语句，终止 for 循环。这时程序将跳出 for 循环而不再执行下面的循环。如果未使用 break 语句，程序将执行 for 循环语句中的循环体，直到变量 i 的值不满足条件 i<=5。

注意：在嵌套的循环语句中使用 break 语句时，break 语句只能跳出最近的一层循环，而不是跳出所有的循环。

2. continue 语句

continue 语句的工作方式与 break 语句有点类似，但是其作用不同。continue 语句是只跳出本次循环而立即进入到下一次循环；break 语句则是跳出循环后结束整个循环。

下面将例 2-7 中的 break 语句换成 continue 语句，看看输出结果有什么不同。

【例 2-8】使用 continue 语句。

```
<html>
  <head>
    <title>continue 语句</title>
    <script type="text/javascript">
      for(var i=1;i<=5;i++)
      {
        if(i==3)   continue;
        console.log(i);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

代码运行结果为：

```
1   2   4   5
```

在修改后的代码执行中，for 循环在 i 等于 1、2、3、4、5 时都执行了，但是输出结果中却没有 3。当 i 为 3 时，if 语句条件为真，执行 continue 语句，跳过循环体后面的语句，继续执行下一次循环，所以 3 没有被输出。

2.4.4 异常处理语句

在代码的运行过程中一般会发生两种错误：一是程序内部的逻辑错误或者语法错误；二是运行环境或者用户输入中不可预知的数据造成的错误。JavaScript 可以捕获异常并进行相应的处理，通常用到的异常处理语句包括 throw 和 try-catch-finally 语句两种。

1. throw 语句

throw（抛出）语句的作用是抛出一个异常。所谓抛出异常，就是用信号通知发生了异常情况或错误。throw 语句的预防代码如下：

```
throw   表达式;
```

以上代码中的表达式，可以是任何类型的表达式。该表达式通常是一个 Error 对象或 Error 对象的某个实例。可以通过 new Error(message)来创建这个对象，异常的描述被作为 error 对象的一个属性 message，可以由构造函数传入，也可以以后赋值。通过这个异常描述，可以让程序获取异常的详细信息，从而自动处理。

2. try-catch-finally 语句

try-catch-finally 语句是 JavaScript 中的用于处理异常的语句，该语句与 throw 语句不同。throw 语句只是抛出一个异常，但对该异常并不进行处理，而 try-catch-finally 语句可以处理所

抛出的异常。其语法形式如下：

```
try{
  //语句块 1：要执行的代码
}catch(e){
  //语句块 2：处理异常的代码
}finally{
  //语句块 3：无论异常发生与否，都会执行的代码
}
```

说明如下：

（1）语句块 1 是有可能要抛出异常的语句块。

（2）catch(e)中的 e 是一个变量，该变量为从 try 语句块中抛出的 Error 对象或其他值。

（3）语句块 2 是处理异常的语句块；如果在语句块 1 中没有抛出异常，则不执行该语句块中的代码。

（4）无论在语句块 1 中是否抛出异常，JavaScript 都会执行语句块 3 中的代码；但是语句块 3 中的语句与 finally 关键字可以一起省略。

【例 2-9】计算两个数据相除的异常处理。

```
<html>
  <head>
    <title>异常处理</title>
    <script type="text/javascript">
      function myFun(x,y)
      {
        var z;
        try{
          if(y==0)
          {
            throw new Error("除数不能为 0");
          }
          z=x/y;
        }
        catch(e)
        {
          z=e.message;
        }
        return z;
      }
      console.log(myFun(1,0));
    </script>
  </head>
  <body>
  </body>
</html>
```

代码运行结果为：

```
除数不能为 0
```

在这个例子中，创建了一个名为 myFun 的函数，该函数的作用是将两个参数相除，并返

回结果，如果在相除时产生异常，则返回错误信息。当用 1 除以 0 时抛出异常，catch 语句接收到由 throw 语句抛出的异常，并进行处理。

2.5 JavaScript 函数

JavaScript 脚本语言允许开发者通过编写函数的方式组合一些可重复使用的脚本代码块，增加了脚本代码的结构化和模块化。函数是通过参数接口进行数据传递，以实现特定的功能。

2.5.1 函数的创建与调用

函数由函数定义和函数调用两部分组成，应首先定义函数，然后再进行调用，以养成良好的编程习惯。

函数的定义应使用关键字 function，其语法规则如下：

```
function funcName ([parameters])
{
    statements;
    [return 表达式;]
}
```

函数的各部分含义如下：

（1）funcName 为函数名，函数名可由开发者自行定义，与变量的命名规则基本相同。

（2）parameters 为函数的参数，在调用目标函数时，需将实际数据传递给参数列表以完成函数特定的功能。参数列表中可定义一个或多个参数，各参数之间用逗号“,”分隔开来，当然，参数列表也可为空。

（3）statements 是函数体，规定了函数的功能，本质上相当于一个脚本程序。

（4）return 指定函数的返回值，为可选参数。

自定义函数一般放置在 HTML 文档的 head 标记之间。除了自定义函数外，JavaScript 脚本语言提供大量的内建函数，无需开发者定义即可直接调用。例如，window 对象的 alert()方法即为 JavaScript 脚本语言支持的内建函数。

函数定义过程结束后，可在文档中任意位置调用该函数。引用目标函数时，只需在函数名后加上小括号即可。若目标函数需引入参数，则需在小括号内添加传递参数。如果函数有返回值，可将最终结果赋值给一个自定义的变量并用关键字 return 返回。

【例 2-10】函数调用实例。

```
<html>
  <head>
    <title>函数调用实例</title>
    <script type="text/javascript">
      function test()
      {
        console.log("无返回值的函数调用！");
        var str="123456";
      }
      function add(x,y)
      {
```

```
            console.log("有返回值的函数调用！");
            var z=x+y;
            return z;
          }
          test();
          var a=10,b=20;
          var c=add(a,b);
          console.log(a+"+"+b+"="+c);
        </script>
      </head>
      <body>
      </body>
    </html>
```

代码运行结果为：

```
无返回值的函数调用！
有返回值的函数调用！
10+20=30
```

在本例中，定义了两个函数，一个是没有参数也没有返回值的函数 test，另一个是带两个参数并且有返回值的函数 add。调用时 test 函数直接用函数名加上括号成为调用语句；而 add 函数需要将函数的返回值赋给变量 c，再输出结果。

2.5.2　函数的参数

与其他程序设计语言不同，JavaScript 不会验证传递给函数的参数个数是否等于函数定义的参数个数。如果传递的参数个数与函数定义的参数不同，则函数执行起来往往会有可能产生一些意想不到的错误。开发者定义的函数都可以接收任意个参数（根据 Netscape 的文档，最多能接收 25 个），而不会引发错误，任何遗漏的参数都会以 undefined 传递给函数，多余的参数将忽略。为了避免产生错误，一个程序员应该会让传递的函数参数个数与函数定义的参数个数相同。

在 JavaScript 中提供了一个 arguments 对象，该对象可以获取从 JavaScript 代码中传递过来的参数，并将这些参数存放在 arguments[]数组中，因此也可以通过 arguments 对象来判断传递过来的参数个数，引用属性 arguments.length 即可。arguments 为数组，因此通过 arguments[i]可以获得实际传递的参数值。

【例 2-11】判断函数的参数传递个数。

```
<html>
  <head>
    <title>Untitled Document</title>
    <script type="text/javascript">
      function add(x,y)
      {
        if(arguments.length!=2)
        {
          var str="传递的参数个数有误，一共传递了"+
            arguments.length+"个参数，分别为：\n";
          for(var i=0;i<arguments.length;i++)
```

```
            {
              str+="第"+(i+1)+"个参数的值为："+arguments[i]+"\n";
            }
            return str;
          }
          else
          {
            var z=x+y;
            return z;
          }
        }
        console.log("add(2,4,6): "+add(2,4,6)+"\n");          //①
        console.log("add(2): "+add(2)+"\n");                  //②
        console.log("add(2,4): "+add(2,4)+"\n");              //③
      </script>
    </head>
    <body>
    </body>
  </html>
```

代码运行结果为：

```
add(2,4,6): 传递的参数个数有误，一共传递了 3 个参数，分别为：
第 1 个参数的值为：2
第 2 个参数的值为：4
第 3 个参数的值为：6

add(2): 传递的参数个数有误，一共传递了 1 个参数，分别为：
第 1 个参数的值为：2

add(2,4): 6
```

在本例中调用语句①传递了 3 个参数，此时 add 函数会将参数 x 的值赋为 2，将参数 y 的值赋为 4，并将传递过来的第 3 个参数值 6 忽略掉。但是在 add 函数中的 arguments 对象可以完全接收传递过来的 3 个参数，因此 arguments.length 的值为 3，arguments[0]的值为 2，arguments[1]的值为 4，arguments[2]的值为 6。程序进入错误处理代码，并输出错误信息。

然后调用语句②只传递了一个参数，此时 add 函数会将参数 x 的值赋为 2，而参数 y 的值保持为初始值，即 undefined。arguments.length 的值为 1，arguments[0]的值为 2。程序进入错误处理代码，并输出错误信息。

最后调用语句③传递了两个参数，此时 add 函数会将参数 x 的值赋为 2，将参数 y 的值赋为 4。而 arguments.length 的值为 2，arguments[0]的值为 2，arguments[1]的值为 4，程序不会进入错误处理代码，而会直接返回结果 6。

由于 JavaScript 是一种无类型的语言，因此在定义函数时，不需要为函数的参数指定数据类型。事实上，JavaScript 也不会去检测传递过来的参数类型是否符合函数的需要。如果一个函数对参数的要求很严格，那么可以在函数体内使用 typeof 运算符来检测传递过来的参数是否符合要求。

【例 2-12】判断函数的参数传递的类型。

```
<html>
  <head>
    <title>判断函数的参数传递的类型</title>
    <script type="text/javascript">
      function myFun(a,b)
      {
        if(typeof(a)=="number"&&typeof(b)=="number")
        {
          var c=a*b;
          return c;
        }
        else
        {
          return "传递的参数不正确，请使用数字型的参数！";
        }
      }
      console.log(myFun(2,4));
      console.log(myFun(2,"s"));
    </script>
  </head>
  <body>
  </body>
</html>
```

代码运行结果为：

```
8
传递的参数不正确，请使用数字型的参数！
```

本例中使用 typeof 运算符判断传递过来的参数类型，如果都是数字型，则返回两个参数之积；否则返回错误信息。

2.5.3　函数的属性与方法

在 JavaScript 中，函数也是一个对象。既然函数是对象，那么函数也拥有自己的属性与方法。

1．length 属性

函数的 length 属性与 arguments 对象的 length 属性不一样，arguments 对象的 length 属性可以获得传递给函数的实际参数的个数，而函数的 length 属性可以获得函数定义的参数个数。同时 arguments 对象的 length 属性只能在函数体内使用，而函数的 length 属性可以在函数体之外使用。

【例 2-13】函数的 length 属性与 arguments 对象的 length 属性的区别。

```
<html>
  <head>
    <title>函数的 length 属性</title>
    <script type="text/javascript">
      function add(x,y)
      {
        if(add.length!=arguments.length)
        {
```

```
            return "传递过来的参数 个数与函数定义的参数个数不一致！";
        }
        else
        {
            var z=x+y;
            return z;
        }
    }
    console.log("函数 add 的 length 值为："+add.length);
    console.log("add(3,4):"+add(3,4));
    console.log("add(3,4,5):"+add(3,4,5));
  </script>
 </head>
 <body>
 </body>
</html>
```

代码运行结果为：

```
函数 add 的 length 值为：2
add(3,4):7
add(3,4,5):传递过来的参数 个数与函数定义的参数个数不一致！
```

本例中定义了一个名为 add 的函数，该函数的作用是返回两个参数的和。代码中有两处用到了函数 add 的 length 属性：一次是在 add 函数体内，在返回两个参数之和之前，先判断传递过来的参数个数与函数定义的参数个数是否相同，如果不同则返回错误信息；另一次是在函数体之外使用直接输出函数 add 的 length 属性值。

2. call()和 apply()方法

在 JavaScript 中，每个函数都有 call()方法和 apply()方法，使用这两个方法可以像调用其他对象的方法一样来调用某个函数，它们的作用都是将函数绑定到另一个对象上去运行，两者仅在定义参数的方式上有所区别。

call()方法的使用语法格式如下：

```
函数名.call(对象名, 参数 1, 参数 2, …)
```

apply()方法的使用语法格式如下：

```
函数名.apply(对象名, 数组)
```

由上可以看出，两个方法的区别是，call()方法直接将参数列表放在对象名之后，而 apply()方法却是将列表放在数组里，并将数组放在对象名之后。

请看以下代码，在该代码的第一行定义了一个对象，第二行定义了一个数组，第三行中使用 call()方法来调用 myFun 函数，第四行中使用了 apply()方法来调用 myFun 函数。

```
var myObj = new Object();
var arr = [1,3,5];
myFun.call(myObj,1,2,3);
myFun.apply(myObj, arr);
```

其中 apply()方法要求第 2 个参数为数组，JavaScript 会自动将数组中的元素值作为参数列表传递给 myFun 函数，也可以将数组作为参数直接放在 apply()方法内，代码如下：

```
myFun.apply(myObj, [2,4,6]);
```

【例 2-14】函数的 call()方法和 apply()方法的使用。

```
<html>
  <head>
    <title>函数的 call()方法和 apply()方法的使用</title>
    <script type="text/javascript">
      function getSum()
      {
        var sum=0;
        for(var i=0;i<arguments.length;i++)
        {
          sum+=arguments[i];
        }
        return sum;
      }
      var myObj=new Object();
      var arr=[1,3,5];
      console.log("sum1="+getSum.call(myObj,2,4,6));
      console.log("sum2="+getSum.apply(myObj,arr));
    </script>
  </head>
  <body>
  </body>
</html>
```

代码运行结果为：

```
sum1=12
sum2=9
```

2.5.4　闭包

JavaScript 支持闭包（Closure）。所谓闭包，是指词法表示包括不必计算的变量的函数，也就是说，该函数能使用函数外定义的变量。在 JavaScript 中使用全局变量时一个简单的闭包实例如下面的代码：

```
var sMessage = "Hello World!";
function sayHelloWorld()    {
  alert(sMessage);
}
sayHelloWorld();
```

在这段代码中，脚本被载入内存后，并未为函数 sayHelloWorld()计算变量 sMessage 的值，该函数捕获 sMessage 的值只是为以后使用，也就是说，解释程序知道在调用该函数时要检查 sMessage 的值。sMessage 将在函数调用 sayHelloWorld()时（最后一行）被赋值，显示消息“Hello World!”。

在一个函数中定义另一个函数会使闭包变得复杂，代码如下：

```
var iBaseNum = 10;
function addNumbers(iNum1, iNum2)   {
  function doAddition()    {
    return iNum1 + iNum2 + iBaseNum;
  }
```

```
        return doAddition();
    }
```

这里，函数 addNumbers()包括函数 doAddition()（闭包）。内部函数是个闭包，因为它将获取外部函数的参数 iNum1 和 iNum2 及全局变量 iBaseNum 的值。addNumbers()的最后一步调用了内部函数，把两个参数和全局变量相加，并返回它们的和。这里要掌握的重要概念是 doAddition()函数根本不接收参数，它使用的值是从执行环境中获取的。

可以看到，闭包是 JavaScript 中功能非常强大的一部分，可以用于执行复杂的计算。就像使用任何高级函数一样，在使用闭包时要当心，因为它们可能会变得非常复杂。

本章小结

JavaScript 与其他语言一样，也支持常量与变量，不过 JavaScript 中的变量是无类型的，即可以存储任何一种类型的数据。JavaScript 中的基本数据类型有数字型、字符串型和布尔型，此外，JavaScript 还支持对象、数组、Null 和 Undefined 数据类型。各种不同的数据类型直接可以通过显式或隐式方式进行转换。

本章主要介绍了 JavaScript 中的表达式、操作数与运算符。JavaScript 的所有功能都是通过语句来实现的，本章对 JavaScript 中的表达式语句、语句块、选择语句、循环语句、跳转语句、异常处理语句和其他语句进行了详细介绍，熟练掌握这些语句是学习 JavaScript 必不可少的基础。

本章还介绍了函数的定义与使用方法。函数在 JavaScript 中是一个很重要的部分，JavaScript 有很多内置函数，程序员可以直接使用这些内置函数，也可以自定义函数以供程序使用。

习　　题

2-1　JavaScript 中的变量是使用什么关键字来进行声明的？

2-2　声明 3 个变量，一个数字型变量和两个字符串变量。数字型变量的值是 120，字符串变量的值分别为"2150"和"Two Hundred"。将创建的两个字符串类型变量转换成数字型变量，它们能否转换成功？如果不行，为什么？

2-3　创建一个带有 3 个数字的数组。

2-4　简述 for 循环、while 循环和 do…while 循环的区别。

2-5　throw 语句的作用是什么？

2-6　通过什么方法获取函数中传递的参数个数？

综合实训

一、目标

定义一个函数，该函数的作用是使用冒泡法将传递过来的数字从小到大进行排序，并输出排序的结果。

二、准备工作

在进行本实训前，必须掌握 JavaScript 的基本语法、条件和循环控制语句，函数的定义和使用函数的参数。

由于排序的数字个数不定，因此，在定义该函数时并没有定义参数，只有在调用该函数时才使用 arguments 对象来获取实际的传递参数值。获取实际传递的参数后，再通过冒泡法对参数值进行排序，最后通过循环输出排序后的结果。

三、实训预估时间：45 分钟

按升序排序的冒泡算法的基本思路是将要排序的数字放在一个数组中，并将数组中相邻的两个元素值进行比较，将数值小的数字放在数组的前面，具体操作方法如下：

（1）假设数组 a 中有 n 个数字，在初始状态下，a[0]～a[n-1]的值为无序数字。

（2）第一次扫描，从数组最后一个元素开始比较相邻两个元素的值，大的放在数组后面，小的放在数组前面。即依次比较 a[n-1]与 a[n-2]、a[n-2]与 a[n-3]、…、a[2]与 a[1]、a[1]与 a[0]的值，小的放前面，大的放后面。例如，a[1]的值小于 a[2]的值，就将这两个元素的值交换。一次扫描完毕后，最小的数字就会存放在 a[0]元素上。

（3）第二次扫描，第二小的数字就会存放在 a[1]元素上。

（4）依此类推，直到循环结束。