第5章

JavaScript 基础

JavaScript 是一种被广泛用于 Web 开发的编程语言，常用来为网页增添交互性和动态性，为用户提供更流畅、绚丽的浏览效果。JavaScript 具有简单易学、高度可扩展性以及跨平台兼容性等特点，现已成为最流行的编程语言之一。除在 Web 开发中使用外，也被广泛用于构建桌面应用程序、构建移动应用程序、游戏开发、机器学习和人工智能等领域。

5.1 JavaScript 简介

JavaScript 是一种高级的、多范式、解释型的编程语言，支持面向对象编程、命令式编程以及函数式编程。

5.1.1 基本语法

1. 区分大小写

JavaScript 对大小写敏感，即区分大小写。例如，变量名 Car 和变量名 car 是两个不同的变量。

2. 标识符

JavaScript 的标识符包括变量名、函数名、参数名和属性名等，也包括某些循环语句中跳转位置的标记，在 JavaScript 中，所有可自主命名的名称都可以称为标识符。

合法的标识符应该满足以下规则。

（1）标识符可以是一个或多个字符。

（2）第一个字符必须是字母、下画线“_”或符号“$”，其后字符可以是字母、数字、下画线或符号“$”。

（3）自定义标识符不能与 JavaScript 关键字、保留字重名。

（4）可使用 Unicode 转义序列。例如，字符 a 可使用“\u0061”表示，因此，下列标识符都是合法的：a、_a、$a_ab_23ab、a23、\u0061。

3. 关键字

JavaScript 的关键字不可以用作变量、标签或者函数名，主要关键字如表 5-1

所示。

表 5-1　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

4. 保留字

ECMA-262 规定保留字是 JavaScript 语言内部预备使用的一组标识符，建议不要使用，主要保留字如表 5-2 所示。

表 5-2　保留字

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

5. 常量

常量表示固定不变的数据，与变量一样，均是用于存储数据的容器，只不过常量的值在程序的运行过程中不发生改变。在 JavaScript 语言下一代标准 ES6(ECMAScript 6.0，已在 2015 年 6 月正式发布)之前，并没有声明常量的方法，在 ES6 中新增加了 const 来定义常量，例如：

```
const a = 1                          //当常量 a 被创建时，再次给 a 赋值时，a 仍为 1
console.log(a) ;
a = 10;
console.log(a)                       //报错
```

常量分为以下四类。

(1) 整型常量，也就是整数，包括正整数、负整数和 0。

(2) 实型常量，也称为浮点常量，即日常生活中的小数。

(3) 字符串常量，指使用单引号或者双引号括起来的内容。

(4) 布尔常量，即真或假，其取值为 true 或者 false。

6. 变量

变量表示可以变化的数据，是用于存储数据的容器，在程序的运行过程中，可发生变化或者被再次赋值。

JavaScript 中定义变量有两种方式：var 与 let，其中，let 是 ES6 新增的语法。

(1) var 定义变量名称。

```
var a;                                  //定义一个变量
a = 1;                                  //往变量中存储数据
console.log(a);                         //从变量中取出存储的数据
```

(2) let 定义变量名称。

```
let num;                                //定义一个变量
num = 2;                                //给变量初始化
console.log(num);                       //取出存储的数据
```

两种定义方式的区别如下：

(1) var 是函数作用域，let 是块级作用域。var 声明的变量，只有函数能限制其作用域，而任何大括号都可以限制 let 声明变量的作用域。

(2) var 存在变量提升，let 不存在变量提升。var 可以先使用再声明，在变量没有被声明时，其值为 undefined；但在使用 let 关键字声明变量时，必须先声明再使用，否则会报错。

(3) var 变量可以重复声明，let 变量不可以重复声明。多人团队开发项目时，使用 let 声明变量，可以避免变量名的覆盖。

(4) var 声明的全局变量，会成为全局对象 window 的属性；let 声明的变量不会。

根据变量作用域的不同，可分为块级变量、局部/函数变量和全局变量。

块级变量是在块中声明的变量，只在块中有效。

局部变量声明在函数之中，其作用域是局部的，只能在函数内部访问，因此能够在不同函数中使用同名变量。函数执行时，会创建局部变量，函数执行完毕后，会销毁局部变量，例如：

```
//此处的代码不能使用 carName 变量
function myFunction() {
    var carName = "porsche";
    //此处的代码能使用 carName 变量
}
```

全局变量声明在函数之外，其作用域是全局的，即网页的所有脚本和函数都能够访问它，例如：

```
var carName = " porsche";
//此处的代码能够使用 carName 变量
function myFunction() {
    //此处的代码也能够使用 carName 变量
}
```

注意：如果为尚未声明的变量赋值，此变量会自动成为全局变量。

7. 注释

单行注释：以“//”开头，任何位于“//”与行末之间的文本都会被 JavaScript 忽略。

多行注释：以“/*”开头，以“*/”结尾，任何位于“/*”和“*/”之间的文本都会被 JavaScript 忽略。

5.1.2　引入方式

JavaScript 的引入方式有三种，分别为行内式、内联式以及外链式。

1. 行内式

行内式是指将 JavaScript 代码作为 HTML 标签的属性值来使用，例如：

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>JavaScript 引入方式</title>
    </head>
    <body>
        <!-- 行内式引入 JS 代码  -->
        <input type="button" value="点我" onclick="alert('hello!')">
    </body>
</html>
```

2. 内联式

内联式，也称内嵌式，或嵌入式，是指将 JavaScript 代码包裹在＜script＞标签之中，直接编写在 HTML 文件里，其中，type＝"text/javascript"用于告知浏览器，此代码语言是 JavaScript。在 HTML5 中，其属于默认值，可省略不写。

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>JavaScript 引入方式</title>
    <!-- 内联式引入 JS 代码  -->
        <script type="text/javascript">
            alert("hello!");
        </script>
    </head>
    <body>
        网页内容
    </body>
</html>
```

3. 外链式

外链式是指将 JavaScript 代码保存到一个单独的扩展名为.js 的文件中，然后使用<script>标签的 src 属性引入该 js 文件。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>JS 外链式</title>
    <!-- 外链式引入 JS 代码   -->
    <script src="JavaScript.js"></script>
</head>
<body>
    网页内容
</body>
</html>
```

相比内联式和行内式，推荐使用外链式来编写和引入 JavaScript 代码。

在 Web 开发中，提倡结构、样式、代码分离，即 HTML、CSS、JavaScript 三部分代码，需避免直接在 HTML 中编写，可分为 xx.html、xx.css、xx.js 三个文件编写，以便于后期维护。

5.2 基本数据类型

JavaScript 中，主要有数值 number、字符串 string、布尔值 boolean、null 以及 undefined 五种基本数据类型，可通过 typeof 运算符返回变量或表达式的类型。

5.2.1 数值

JavaScript 不区分整数和浮点数，所有数都以 64 位浮点数形式进行存储，当运算需要用到整数时，JavaScript 会自动把 64 位浮点数转换为整数，再进行整数运算。

1. 数值精度

JavaScript 中，根据 IEEE-754 标准，浮点数 64 位二进制组成方式如下：

第 1 位：符号位 sign，0 代表正数，1 则代表负数，用于确定一个数的正负。

第 2 位到第 12 位(共 11 位)：代表指数部分 exponent，用于确定该数的大小。

第 13 位到第 64 位(共 52 位)：代表小数部分 mantissa，用于确定该数的精度。

计算公式如下：

$$number=(-1)^{sign}\times(1.mantissa)\times 2^{exponent-1023}$$

2. 数值范围

64 位浮点数指数部分的长度是 11 位，可知指数的最大值是 2047，即 $2^{11}-1$，其中，一半用来表示负数，那么 JavaScript 指数范围在 2^{-1023} 到 2^{1024} 之间。如果指数范围等于或超过最

大正值 1024，JavaScript 会返回 Infinity，即“正向溢出”；如果等于或超过最小负值，JavaScript 会把该数转为 0，即“负向溢出”。例如：

```
console.log(Math.pow(2,1024))                              //Infinity
console.log(Math.pow(2,-1074)==Number.MIN_VALUE)           //true
console.log(Math.pow(2,-1075))                             //0
```

3. 数值表示法

数值的表示可使用字面形式直接表示，例如十进制、八进制等，也可使用科学记数法表示。例如，1110000 可以表示为 111e4，0.111 可以表示为 111e－3。

注意：e 和 E 一样，在它们的后面需要跟一个整数，该整数表示当前数值的指数部分。数值一般直接采用字面形式表示，但是，以下两种情况，JavaScript 会自动将数值转换为科学记数法表示。

（1）小数点前的数字多于 21 位。

（2）小数点后的 0 多于 5 个，例如，0.00000003 被转换为 3e－8。

4. 进制

JavaScript 有以下四种进制。

（1）十进制：正常书写，取值数字 0～9，前面不加 0。

（2）二进制：取值数字 0 和 1，前缀 0b 或者 0B。

（3）八进制：取值数字 0～7，前缀 0o 或者 0O。

（4）十六进制：取值数字 0～9 和 a～f，前缀 0x 或者 0X。

JavaScript 会自动将八进制、十六进制、二进制转换为十进制。

5. 特殊数值

任何一个数都有正负，0 也一样。JavaScript 中存在两个 0，即＋0 与－0，它们是等价的，唯独当＋0 或－0 作为分母时，返回值不相等，例如：

```
(1 / +0) === (1 / -0)          //返回值:false
```

原因是左边除以＋0 得到的是＋Infinity，右边除以－0 得到的是－Infinity，这两个值并不相等。Infinity 表示无穷，当正数值过大或负数值过小无法表示，或者以非零数字除以 0，都会得到 Infinity。

6. NaN

NaN(Not a Number)是 JavaScript 特殊值，表示非数值，属于 JavaScript 保留词，指示某个数不是合法数。使用一个非数值字符串进行除法运算，会得到 NaN。另外，0/0 也会得到 NaN。

NaN 是一个特殊值，其类型依然是 number，其运算规则如下：

（1）NaN 不等于任何值，包括其本身。

(2) NaN 的布尔值为 false。

(3) NaN 与任何数的运算,得到的都是 NaN。

5.2.2 字符串

字符串 string 类型用于表示由 16 位 Unicode 字符组成的字符序列,且字符序列由双引号或单引号引起来,下面两种字符串的书写方式均有效。

```
var firstName = "ZhiHao";
var lastName = 'Wang';
```

若需要将一个值转换为一个字符串,可以使用 toString()方法,例如:

```
var age = 10;
var ageAsString = age.toString();          //字符串"10"
var found = true;
var foundAsString = found.toString();      //字符串"true"
```

5.2.3 布尔值

布尔 boolean 类型只有两个值:true 和 false。

注意: boolean 类型的字面值 true 和 false 区分大小写。也就是说,True 和 False 不是 boolean 值,只是标识符。JavaScript 中,所有类型的值都可使用转型函数 Boolean()转换为 boolean 值,例如:

```
var message = "Hello world!";
var messageAsBoolean = Boolean(message);
```

5.2.4 null 和 undefined 类型

undefined 类型只有一个值,为 undefined,表示已声明但未对其初始化的变量,例如:

```
var message;
alert(message == undefined);               //true
```

null 类型是第二个只有一个值的数据类型,值为 null,表示尚未存在的对象。例如,如果函数或方法返回的是对象,而找不到该对象时,通常返回 null,也可通过设置值为 null 来清空对象。

```
var person = null;                         //值是 null,但是类型仍然是对象
alert(typeof(person));                     //输出 object
```

5.2.5 类型转换

类型转换是将一种数据类型转换为另一种数据类型,对于任何数据类型,无论是原始类

型还是对象，都可以进行类型转换。

1. 强制类型转换

强制类型转换，也称为显式类型转换，通过 JavaScript 内置 API 将一种类型人为地转换为另一种类型，这些 API 包括 Boolean()、Number()、String()、parseInt()、parseFloat()等，例如：

```
String(2 - true);                    //'1'
'56' === String(56);                 //true
Number('2350e-2');                   //'23.5'
Number('23') + 7;                    //30
Boolean('');                         //false
Boolean(2) === true;                 //true
```

2. 隐式类型转换

隐式类型转换，也称为自动类型转换，通过 JavaScript 编译器自动根据运算符进行类型转换，以使运算符或函数正常工作，例如：

```
'25' + 15;                           //'2515'
23 * '2';                            //46
23 - true;                           //22
true - null;                         //1
false + undefined;                   //NaN
parseFloat('10.81');                 //10.81
parseInt('10.20');                   //10
```

注意：下面这些常见的操作会触发隐式类型转换。

(1) 与运算相关的操作符：+、-、+=、++、*、/、%、<<、& 等。

(2) 与数据比较相关的操作符：>、<、==、<=、>=、===。

(3) 与逻辑判断相关的操作符：&&、!、||、三目运算符。

5.3　运算符和流程控制

运算符是专门用于程序执行特定运算或逻辑操作的符号，根据其作用，可分为七类：算术运算符、字符串运算符、赋值运算符、比较运算符、逻辑运算符、三元运算符以及位运算符。流程控制用于控制程序的执行流程，包括顺序结构、选择结构以及循环结构。

5.3.1　运算符

1. 算术运算符

算术运算符用于对数值类型的变量及常量进行算术运算，常用运算符及其示例如表 5-3 所示。

表 5-3　常用算术运算符及其示例

运　算　符	运　　算	示　　例	结　　果
＋	加	5＋5	10
－	减	6－4	2
*	乘	3 * 4	12
/	除	3/2	1.5
%	求余	5%7	5
**	幂运算	3**4	81
＋＋	自增(前置)	a＝2,b＝＋＋a	a＝3;b＝3
＋＋	自增(后置)	a＝2,b＝a＋＋	a＝3;b＝2
－ －	自减(前置)	a＝2,b＝－ －a	a＝1;b＝1
－ －	自减(后置)	a＝2,b＝a－ －	a＝1;b＝2

算术运算符在实际应用中,需注意以下几点。

(1) 四则混合运算时,运算顺序需遵循“先乘除后加减”原则。

(2) 取模运算时,运算结果的正负取决于被模数的符号,与模数的符号无关。例如,(－8) % 7 ＝ －1,而 8 % (－7) ＝ 1。

(3) “＋＋”或“－ －”放在操作数前面时,先进行自增或自减运算,再进行其他运算。若放在操作数之后,则先进行其他运算,再进行自增或自减运算。

2. 字符串运算符

JavaScript 中,“＋”操作的两个数据中,只要有一个是字符型,则“＋”表示字符串运算符,用于返回两个数据拼接后的字符串,例如:

```
var tel = 110 + '120100';
console.log (tel);                    //输出结果为:'110120100'
console.log(typeof tel);              //输出结果:string
```

从上述示例可知,当变量或值通过运算符“＋”与字符串进行运算时,变量或值就会被自动转换为字符型,再与指定的字符串进行拼接。

3. 赋值运算符

赋值运算符将运算符右边的值赋给左边的变量,“＝”是最基本的赋值运算符,而非数学意义上的相等关系,常用赋值运算符及其示例如表 5-4 所示。

表 5-4　常用赋值运算符及其示例

运算符	运　　算	示　　例	结　　果
＝	赋值	a＝3,b＝2	a＝3　　b＝2

续表

运算符	运　　算	示　　例	结　　果
＋＝	加并赋值	a＝3,b＝2;a＋＝b	a＝5　　b＝2
－＝	减并赋值	a＝3,b＝2;a－＝b	a＝1　　b＝2
＊＝	乘并赋值	a＝3,b＝2;a＊＝b	a＝6　　b＝2
/＝	除并赋值	a＝3,b＝2;a/＝b	a＝1.5　　b＝2
%＝	模并赋值	a＝3,b＝2;a%＝b	a＝1　　b＝2
＋＝	连接并赋值	a＝ "abc ";a＋＝"def"	a＝ "abcdef "
**＝	幂运算并赋值	a＝2;a**＝ 5	a＝32
＜＜＝	左移位赋值	a＝9,b＝2;a ＜＜＝ b	a＝36　　b＝2
＞＞＝	右移位赋值	a＝－9,b＝2;a ＞＞＝ b	a＝－3　　b＝2
＞＞＞＝	无符号右移位赋值	a＝－9,b＝2;a ＞＞＞＝ b	a＝ 1073741821　　b＝2
&＝	按位与赋值	a＝3,b＝9;a &＝ b	a＝1　　b＝9
^＝	按位异或赋值	a＝3,b＝9;a ^＝ b	a＝10　　b＝9
|＝	按位或赋值	a＝3,b＝9;a |＝ b	a＝11　　b＝9

（1）同时赋值。

赋值运算符不仅可以为指定变量赋值，还可以利用一条赋值语句，同时为多个变量进行赋值，例如：

```
var a = b = c = 8;          //为三个变量同时赋值
```

在上述代码中，一条赋值语句可同时为变量 a、b、c 赋值，这是由于赋值运算符的结合性为“从右向左”，即先将 8 赋值给变量 c，然后再把变量 c 的值赋值给变量 b，最后把变量 b 的值赋值变量 a，表达式赋值完成。

（2）“＋＝”运算符。

“＋”运算符在 JavaScript 中既可以表示加运算、正数运算，也可表示字符串运算。因此，“＋＝”运算符在使用时，若其操作数都是非字符型数据，则表示相加后赋值，否则用于拼接字符串，例如：

```
var  num1= 1,num2 = "2";
num1 += 3;
num2 += 3;
console.log(num1, num2);              //输出结果为:4  "23"
```

4. 比较运算符

比较运算符用于对两个数值或变量进行比较，其结果是一个布尔值，常用比较运算符及其示例如表 5-5 所示。