第3章

JavaScript 数据类型与运算符

本章主要内容是 JavaScript 数据类型与运算符，主要包括基本数据类型、对象、类型转换、运算符的用法。

本章学习目标：

- 掌握 JavaScript 的基本数据类型与对象类型；
- 掌握 JavaScript 类型转换方法；
- 掌握 JavaScript 运算符的使用。

3.1 JavaScript 基本数据类型

JavaScript 有 5 种原始类型，分别是 Number（数字）、Boolean（布尔值）、String（字符串）、Null（空值）和 Undefined（未定义）。

JavaScript 提供了 typeof 方法用于检测变量的数据类型，该方法会根据变量本身的数据类型给出对应名称的返回值。其语法格式如下。

```
typeof 变量名称
```

对于指定的变量使用 typeof 方法，其返回值是提示数据类型的文本内容，常见 5 种情况，如表 3-1 所示。

表 3-1　typeof 方法的常见返回值一览表

返　回　值	示　　例	解　　释
undefined	var x; alert(x);	该变量未赋值
boolean	var x = true; alert(x);	该变量为布尔值
string	var x = "Hello"; alert(x);	该变量为字符串
number	var x = 3.14; alert(x);	该变量为数值
object	var x = null; alert(x);	该变量为空值或对象

3.1.1　Undefined 类型

所有 Undefined 类型的输出值都是 undefined。若需要输出的变量从未声明过，或者使

用关键字 var 声明过但是从未进行赋值，此时会显示 undefined 字样。例如：

```
alert(y); //返回值为 undefined，因为变量 y 之前从未使用关键字 var 进行声明
或
var x;
alert(x); //返回值也是 undefined，因为未给变量 x 赋值
```

【例 3-1】 JavaScript 基础数据类型 Undefined 的简单应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 变量之 Undefined 类型</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 变量之 Undefined 类型</h3>
9.          <hr/>
10.         <script>
11.             //声明变量 msg
12.             var msg;
13.             //在 alert()方法中使用变量 msg
14.             alert(msg);
15.         </script>
16.     </body>
17. </html>
```

运行效果如图 3-1 所示。

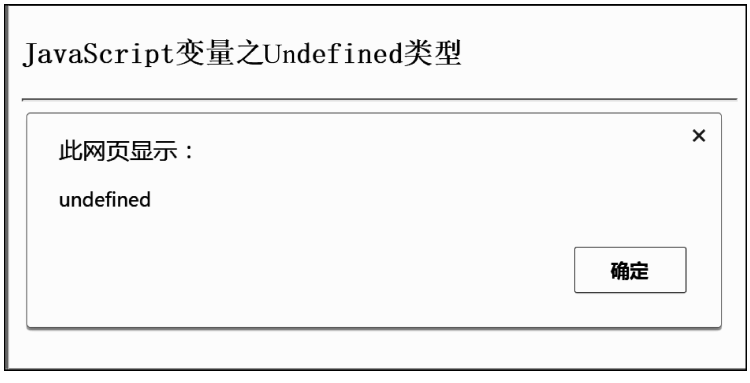

图 3-1　JavaScript 基础数据类型 Undefined 的简单应用效果

【代码说明】

本例使用关键字 var 声明了变量 msg，但未对其进行初始赋值就直接使用 alert(msg)方法要求在对话框中显示该变量内容。由图 3-1 可见，此时显示出来的结果为 undefined。

3.1.2 Null 类型

null 值表示变量的内容为空，可用于初始化变量，或者清空已经赋值的变量。例如：

```
var x=99;
x=null;
alert(x); //此时返回的值是 null 而不是 99
```

【例 3-2】 JavaScript 基础数据类型 Null 的简单应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 变量之 Null 类型</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 变量之 Null 类型</h3>
9.          <hr/>
10.         <script>
11.             //声明变量 msg
12.             var msg=99;
13.               //将变量 msg 赋值为 null
14.               msg=null;
15.             //在 alert()方法中使用变量 msg
16.             alert(msg);
17.         </script>
18.     </body>
19. </html>
```

运行效果如图 3-2 所示。

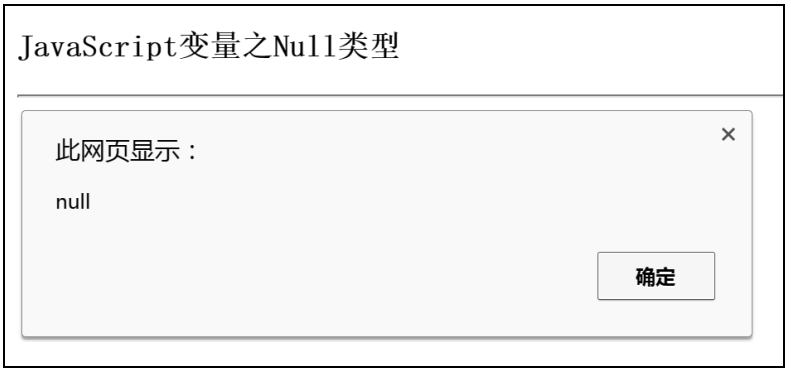

图 3-2 JavaScript 基础数据类型 Null 的简单应用效果

【代码说明】

本例使用关键字 var 声明了变量 msg 并将其赋值为 null，然后使用 alert(msg)方法要求

在对话框中显示该变量内容。由图 3-2 可见，此时显示出来的结果为 null。

3.1.3　String 类型

在 JavaScript 中 String 类型用于存储文本内容，又称为字符串类型。在为变量进行字符串赋值时需要使用引号（单引号或双引号均可）括住文本内容。例如：

```
var country='China';
或
var country="China";
```

与 JavaScript 不同的是，在 Java 中使用单引号声明单个字符、使用双引号声明字符串，而在 JavaScript 中没有区分单个字符和字符串，因此两种声明方式任选一种都是有效的。

如果字符串内容本身也需要带上引号，则用于包围字符串的引号不可以和文本内容中的引号相同。如果字符串本身带有双引号，则使用单引号包围字符串，反之亦然。例如：

```
var dialog='Today is a gift, that is why it is called "Present".';
或
var dialog="Today is a gift, that is why it is called 'Present'. ";
```

此时字符串内部的引号会默认保留字面的样式。

String 对象中包含了一系列方法，常用方法如表 3-2 所示。

表 3-2　JavaScript String 对象常见方法一览表

方　法　名	解　　释
charAt()	返回指定位置上的字符
charCodeAt()	返回指定位置上的字符的 Unicode 编码
concat()	连接字符串
indexOf()	正序检索字符串中指定内容的位置
lastIndexOf()	倒序检索字符串中指定内容的位置
match()	返回匹配正则表达式的所有字符串
replace()	替换字符串中匹配正则表达式的指定内容
search()	返回匹配正则表达式的索引值
slice()	根据指定位置节选字符串片段
split()	把字符串分割成字符串数组
substring()	根据指定位置节选字符串片段
toLowerCase()	将字符串中的所有字母都转换为小写
toUpperCase()	将字符串中的所有字母都转换为大写

1．字符串长度

在字符串中每一个字符都有固定的位置，其位置从左往右进行分配。这里以单词 HELLO 为例，其位置规则如图 3-3 所示。

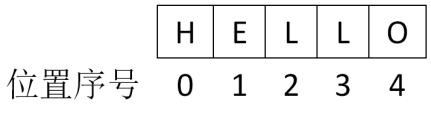

图 3-3　字符位置对照图

首字符 H 从位置 0 开始，第 2 个字符 L 是位置 1，以此类推，直到最后一个字符 O 的位置是字符串的总长度减 1。

用户可以使用 String 对象的属性 length 获取字符串的长度。例如：

```
var s="Hello";
var slen=s.length; //返回值是变量 s 的字符串长度，即 5
```

【例 3-3】 JavaScript 获取字符串长度的简单应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 获取字符串长度</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 获取字符串长度</h3>
9.          <hr/>
10.         <script>
11.             //声明变量 msg
12.             var msg="Hello JavaScript!";
13.             //获取字符串长度
14.               var len=msg.length;
15.             alert("Hello JavaScript!的字符串长度为："+len);
16.         </script>
17.     </body>
18. </html>
```

运行效果如图 3-4 所示。

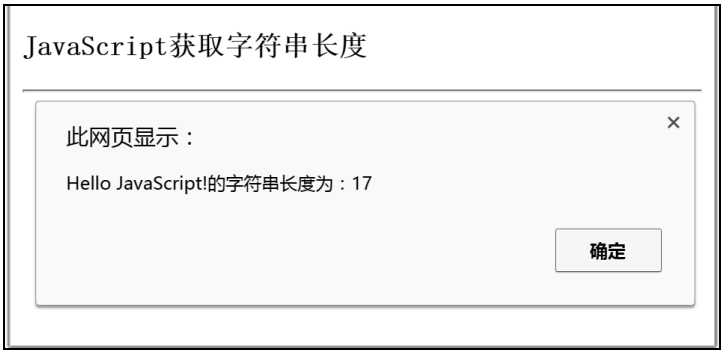

图 3-4 JavaScript 获取字符串长度的简单应用效果

【代码说明】

本例使用关键字 var 声明了变量 msg 并将其赋值为"Hello JavaScript!"，然后使用字符串类型的 length 属性获取其字符长度并使用 alert()方法显示出来。由图 3-4 可见，此时显示出来的结果为 17，因为空格和感叹号各算 1 个字符位置，所以总长度为 5+1+10+1=17。

2．获取字符串中的单个字符

在 JavaScript 中可以使用 charAt()方法获取字符串指定位置上的单个字符，其语法结构如下。

```
charAt(index)
```

其中，index 处填写需要获取的字符所在位置。

例如：

```
var msg="Hello JavaScript";
var x=msg.charAt(0); //表示获取 msg 中的第 1 个字符，返回值为 H
```

如果需要获取指定位置上单个字符的字符代码，可以使用 charCodeAt()方法，其语法结构如下。

```
charCodeAt(index)
```

其中，index 处填写需要获取的字符所在位置。

例如：

```
var msg="Hello JavaScript";
var x=msg.charCodeAt(0); //表示获取 msg 中的第 1 个字符的字符代码，返回值为 72
```

【例 3-4】 JavaScript 获取字符串中单个字符的应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 获取字符串中的单个字符</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 获取字符串中的单个字符</h3>
9.          <hr/>
10.         <script>
11.             //声明变量 msg
12.             var msg="Hello JavaScript";
13.             //获取字符串中的单个字符
14.             var letter=msg.charAt(10);
15.             //获取字符串中单个字符的代码
16.             var code=msg.charCodeAt(10);
17.             alert("Hello JavaScript 在第 10 位上的字符为："+letter+"\n 其字符代
                码为："+code);
18.         </script>
19.     </body>
20. </html>
```

运行效果如图 3-5 所示。

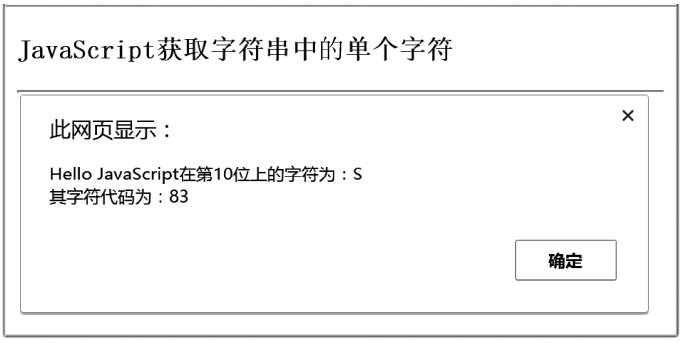

图 3-5　JavaScript 获取字符串中单个字符的应用效果

【代码说明】

本例使用关键字 var 声明了变量 msg 并将其赋值为"Hello JavaScript"，然后使用字符串类型的 charAt()方法获取其中第 10 位上的字符，并且用 charCodeAt()方法获取该字符的代码，最后用 alert()方法将结果显示出来。由图 3-5 可见，此时显示出来的结果为 S，其对应的字符代码为 83。alert()方法中的\n 为转义字符，表示换行。

3．连接字符串

在 JavaScript 中可以使用 concat()方法将新的字符串内容连接到原始字符串上，其语法结构如下。

```
concat(string1, string2,...,stringN);
```

该方法允许带有一个或多个参数，表示按照从左往右的顺序依次连接这些字符串。

例如：

```
var msg="Hello";
var newMsg=msg.concat(" JavaScript");
alert(newMsg); //返回值为"Hello JavaScript"
```

用户也可以直接使用加号（+）进行字符串的连接，其效果相同，上述代码可改为如下。

```
var msg="Hello";
var newMsg=msg+" JavaScript";
alert(newMsg); //返回值为"Hello JavaScript"
```

【例 3-5】 JavaScript 连接字符串的简单应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 连接字符串</title>
6.      </head>
7.      <body>
```

```
8.          <h3>JavaScript 连接字符串</h3>
9.          <hr/>
10.         <script>
11.             //声明变量 s1、s2、s3
12.             var s1="Hello";
13.             var s2="Java";
14.             var s3="Script";
15.             //连接字符串
16.             var msg=s1.concat(s2, s3);
17.             alert(msg);
18.         </script>
19.     </body>
20. </html>
```

运行效果如图 3-6 所示。

图 3-6　JavaScript 连接字符串的简单应用效果

【代码说明】

本例在 JavaScript 中首先声明了 3 个字符串变量，即 s1、s2 和 s3，然后对 s1 使用 concat()方法连接 s2 和 s3 形成新的变量 msg，最后使用 alert()语句测试输出变量 msg 的效果。由图 3-6 可见，变量 msg 为变量 s1、s2 和 s3 的连接。

本例也可以直接使用加号(+)连接 3 个变量实现同样的效果，写成 var msg = s1+s2+s3。注意使用 concat()方法只会连接形成新的返回值，不会影响变量 s1 的初始内容。

4．查找字符串是否存在

使用 indexOf()和 lastIndexOf()方法可以查找原始字符串中是否包含指定的字符串内容，其语法格式如下。

```
indexOf(searchString, startIndex)
或
lastIndexOf(searchString, startIndex)
```

其中，searchString 参数位置填入需要用于对比查找的字符串片段，startIndex 参数用于指定搜索的起始字符，该参数内容如果省略则按照默认顺序搜索全文。

indexOf()和lastIndexOf()方法都可以用于查找指定内容是否存在，如果存在，其返回值为指定内容在原始字符串中的位置序号；如果不存在，则直接返回−1。它们的区别在于indexOf()是从序号 0 的位置开始正序检索字符串内容，而 lastIndexOf()是从序号最大值的位置开始倒序检索字符串内容。

【例 3-6】 JavaScript 检测字符串是否存在的简单应用

分别使用 indexOf()与 lastIndexOf()方法查找字符串中是否包含指定的字母。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>JavaScript 检测字符串是否存在</title>
    </head>
    <body>
        <h3>JavaScript 检测字符串是否存在</h3>
        <hr/>
        <p>查找字母 y 在字符串"Happy Birthday"中的位置</p>
        <script>
            //声明变量 msg
            var msg="Happy Birthday";
            //检测字符 y 存在的位置（正序）
            var firstY=msg.indexOf("y");
            //检测字符 y 存在的位置（倒序）
            var lastY=msg.lastIndexOf("y");
            alert('indexOf("y"): '+firstY+'\nlastIndexOf("y"): '+lastY);
        </script>
    </body>
</html>
```

运行效果如图 3-7 所示。

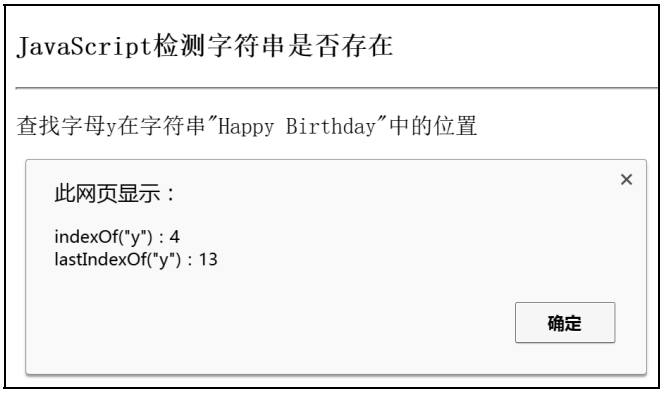

图 3-7　JavaScript 检测字符串是否存在的简单应用效果

【代码说明】

本例在 JavaScript 中声明了变量 msg 作为测试样例，并检测其中字母 y 存在的位置，分别使用 indexOf()和 lastIndexOf()方法进行正序和倒序检测并获取返回值，最后使用 alert()方法输出返回结果。

由图 3-7 可见，对于同一个字母 y 使用 indexOf()和 lastIndexOf()方法获取位置的结果不同，正序查找的结果为 4，倒序查找的结果为 13。原因是原字符串 msg 中包含了不止一个字母 y，而这两个方法会返回在字符串中查找到的第 1 个符合条件的字符位置，因此结果不同。需要注意的是，JavaScript 是大小写敏感的脚本语言，因此如果本例查找大写字母 Y 会获取返回值−1，表示该字符不存在。

5．查找与替换字符串

在 JavaScript 中使用 match()、search()方法可以查找匹配正则表达式的字符串内容。

match()方法的语法格式如下。

```
match(regExp)
```

在参数 regExp 的位置需要填入一个正则表达式，例如 match(/a/g)表示全局查找字母 a，后面的小写字母 g 是英文单词 gobal 的首字母简写，表示全局查找，其返回值为符合条件的所有字符串片段。关于正则表达式的更多用法，读者可查阅本章 3.2 节中的相关内容。

search()方法的语法格式如下。

```
search(regExp)
```

在参数 regExp 的位置同样需要填入一个正则表达式，不同之处在于 search()方法的返回值是符合匹配条件的字符串索引值。

在 JavaScript 中使用 replace()方法可以替换匹配正则表达式的字符串内容。

replace()方法的语法格式如下。

```
replace(regExp, replaceText)
```

在参数 regExp 的位置需要填入一个正则表达式，在参数 replaceText 的位置填入需要替换的新的文本内容。例如 replace(/a/g,"A")表示把所有的小写字母 a 都替换为大写形式。该方法的返回值是已经替换完毕的新字符串内容。

【例 3-7】 JavaScript 查找与替换字符串

分别使用 match()、search()与 replace()方法查找与替换字符串。

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 查找与替换字符串</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 查找与替换字符串</h3>
9.          <hr/>
```

```
10.         <p>用于查找和替换的字符串为"Happy New Year 2016"</p>
11.         <script>
12.             //声明变量 msg
13.             var msg="Happy New Year 2016";
14.             //检测字符 y 是否存在
15.             var result1=msg.search(/y/);
16.             //全局查找数字
17.             var result2=msg.match(/\d/g);
18.             //将小写字母 a 全部替换为大写字母 A
19.             var result3=msg.replace(/a/g,"A");
20.             alert('search(/y/g): '+result1+'\nmatch(/\\d/g): '+result2+
                '\nreplace(/a/g,"A"): '+result3);
21.         </script>
22.     </body>
23. </html>
```

运行效果如图 3-8 所示。

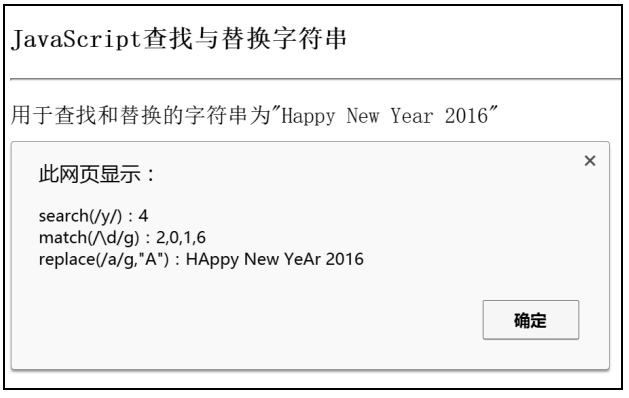

图 3-8　JavaScript 查找与替换字符串

【代码说明】

本例在 JavaScript 中声明了变量 msg 作为测试样例，分别使用 match()、search()与 replace()方法查找和替换字符串，最后使用 alert()方法输出全部的返回结果。

由图 3-8 可见，search()方法可获取指定内容的所在索引位置，而 match()方法是把符合条件的所有字符串以逗号隔开的形式全部展现出来。其中，\d 表示数字 0～9 的任意一个字符，/g 表示全局查找。replace()方法将原字符串中所有的小写字母 a 均替换成大写字母的形式。如果没有加全局字符 g，则只会替换其中第 1 个小写字母 a。

6．获取字符串片段

在 JavaScript 中可以对字符串类型的变量使用 slice()和 substring()方法截取其中的字符串片段，slice()方法用于去掉指定片段，substring()方法用于节选指定片段。

slice ()方法的语法格式如下。

```
slice(start, end)
```

其中，start 参数位置填写需要删除的字符串的第 1 个字符位置，end 参数位置填写需要删除字符串的结束位置（不包括该位置上的字符），如果 end 参数省略，则默认填入字符串长度。如果填入的属性值为负数，表示从字符串的最后一个位置开始计算，例如−1 表示倒数第 1 个字符。

substring()方法的语法格式如下。

```
substring(start, end)
```

与 slice()方法的语法结构类似，其中 start 参数位置填写需要节选的字符串的第 1 个字符位置，end 参数位置填写需要节选字符串的结束位置（不包括该位置上的字符）。同样，如果 end 参数省略，则默认填入字符串长度。

当参数均为非负数时，substring()与 slice()方法获取的结果完全一样，只有当参数值存在负数情况时这两个方法才会有所不同：substring()方法会忽略负数，直接将其当作 0 来处理；而 slice()方法会用字符串长度加上该负数值，计算出对应的位置。

例如：

```
var msg="happy";                    //该字符串长度为 5 位
var result1=msg.substring(1, -1);   //返回值为 h
var result2=slice(1, -1);           //返回值为 app
```

其中，substring(1, −1)会忽略负数，直接当作 0 来对待，因此实际运行的是 substring(1, 0)方法。由于此时结束位置比开始位置靠前，JavaScript 会自动对调位置转换成 substring(0, 1)方法，最终获得返回值 h；而 slice(1,−1)会将负数加上字符串长度换算成 slice(1,4)方法，因此最终获得返回值 app。

【例 3-8】 JavaScript 获取字符串片段的简单应用

分别使用 substring()与 slice()方法获取字符串片段。

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 获取字符串片段</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 获取字符串片段</h3>
9.          <hr/>
10.         <p>分别使用 substring()和 slice()方法节选字符串"Happy Birthday"</p>
11.         <script>
12.             //声明变量 msg
13.             var msg="Happy Birthday";
14.             //使用 substring 节选指定位置范围的字符串
15.             var result1=msg.substring(0,5);
16.             //使用 slice 节选指定位置范围的字符串
17.             var result2=msg.slice(0,-9);
```

```
18.            alert('substring(0,5): '+result1+'\nslice(0,-9): '+result2);
19.        </script>
20.    </body>
21. </html>
```

运行效果如图 3-9 所示。

图 3-9　JavaScript 获取字符串片段的简单应用效果

【代码说明】

本例在 JavaScript 中声明了变量 msg 作为字符串测试样例，其中字符串内容为"Happy Birthday"，共计 14 个字符位置，分别使用 substring(0,5)和 slice(0,−9)方法进行字符串节选，最后使用 alert()方法输出返回结果。

由图 3-9 可见，substring(0,5)方法获取了从第 0 位开始到第 5 位结束（不包括第 5 位本身）的所有字符。slice(0,−9)方法因为带有负数−9，表示倒数第 9 位字符，将其加上字符串长度换算后得到 slice(0,5)，在没有负数的情况下与 substring(0,5)效果完全相同，因此得到同样的结果。

7．字符串大小写转换

在 JavaScript 中可以对字符串类型的变量使用 toLowerCase()和 toUpperCase()方法转换其中存在的大小写字母，toLowerCase()表示将所有字母转换为小写，toUpperCase()表示将所有字母转换为大写。

【例 3-9】 JavaScript 字符串大小写转换

对字符串分别使用 toLowerCase()和 toUpperCase()方法进行大小写转换。

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 字符串大小写转换</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 字符串大小写转换</h3>
9.          <hr/>
```

```
10.         <p>将字符串"Merry Christmas"分别转换为全大写和全小写的形式。</p>
11.         <script>
12.             //声明变量 msg
13.             var msg="Merry Christmas";
14.             //将字符串转换为全大写形式
15.             var upper=msg.toUpperCase();
16.             //将字符串转换为全小写形式
17.             var lower=msg.toLowerCase();
18.             alert('全大写：'+upper+'\n 全小写：'+lower);
19.         </script>
20.     </body>
21. </html>
```

运行效果如图 3-10 所示。

图 3-10　JavaScript 字符串大小写转换的效果

8．转义字符

在前面的例题中可以看到 alert()方法中带有\n 符号，表示换行，这种符号称为转义字符。与 C 语言、Java 语言相似，在 JavaScript 中 String 类型也包含一系列转义字符，具体情况如表 3-3 所示。

表 3-3　JavaScript 常用转义字符对照表

转 义 字 符	含　　义
\n	换行
\t	空一个 Tab 格，相当于 4 个空格
\b	空格符
\r	回车符
\f	换页符
\\	反斜杠
\'	单引号
\"	双引号
\0nnn	八进制代码 nnn 表示的字符，n 是 0～7 中的一个八进制数字，范围是 000～377
\xnn	十进制代码 nn 表示的字符，n 是 0～F 中的一个十六进制数字，范围是 00～FF
\unnn	十六进制代码 nnn 表示的字符，n 是 0～F 中的一个十六进制数字

【例 3-10】 JavaScript 转义字符的简单应用

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>JavaScript 转义字符的简单应用</title>
    </head>
    <body>
        <h3>JavaScript 转义字符的简单应用</h3>
        <hr/>
        <script>
            alert("双引号 \"\n 单引号 \'");
        </script>
    </body>
</html>
```

运行效果如图 3-11 所示。

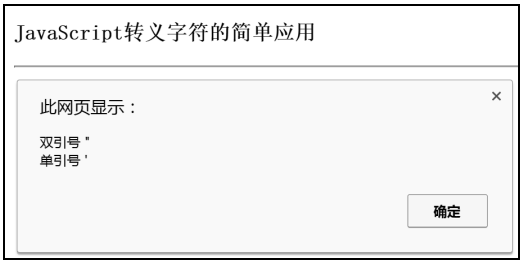

图 3-11 JavaScript 转义字符的简单应用效果

3.1.4 Number 类型

在 JavaScript 中使用 Number 类型表示数字，其数字可以是 32 位以内的整数或 64 位以内的浮点数。例如：

```
var x = 9;
var y = 3.14;
```

Number 类型还支持使用科学记数法、八进制和十六进制的表示方式。

1．科学记数法

对于极大或极小的数字也可以使用科学记数法表示，格式如下。

```
数值 e 倍数
```

上述格式表示数字后面跟指数 e 再紧跟乘以的倍数，其中数值可以是整数或浮点数，倍数允许为负数。例如：

```
var x1=3.14e8;
var x2=3.14e-8;
```

变量 x1 表示的数是 3.14 乘以 10 的 8 次方，即 314000000；变量 x2 表示的数是 3.14 乘以 10 的−8 次方，即 0.0000000314。

【例 3-11】 JavaScript 科学记数法的简单应用

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>JavaScript 科学记数法</title>
    </head>
    <body>
        <h3>JavaScript 科学记数法</h3>
        <hr/>
        <script>
            var x1=3e6;
            var x2=3e-6;
            alert('3e6='+x1+'\n3e-6='+x2);
        </script>
    </body>
</html>
```

运行效果如图 3-12 所示。

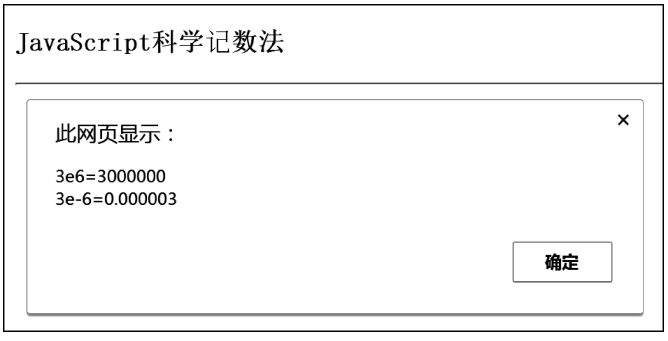

图 3-12　JavaScript 科学记数法的简单应用效果

2．八进制与十六进制数

在 JavaScript 中，Number 类型也可以用于表示八进制或十六进制的数。

八进制的数需要用数字 0 开头，后面跟的数字只能是 0～7（八进制字符）中的一个。例如：

```
var x=010; //这里相当于十进制的 8
```

十六进制的数需要用数字 0 和字母 x 开头，后面跟的字符只能是 0～9 或 A～F（十六进制字符）中的一个，大小写不限。例如：

```
var x=0xA; //这里相当于十进制的 10
或
```

```
var x=0xa; //等同于 0xA
```

虽然 Number 类型可以使用八进制或十六进制的赋值方式，但是在执行代码时仍然会将其转换为十进制结果。

【例 3-12】 JavaScript 八进制与十六进制数

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 八进制与十六进制数</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 八进制与十六进制数</h3>
9.          <hr/>
10.         <script>
11.             //八进制数
12.             var x1=020;
13.             //十六进制数
14.             var x2=0xAF;
15.             alert('八进制数 020='+x1+'\n 十六进制数 0xAF='+x2);
16.         </script>
17.     </body>
18. </html>
```

运行效果如图 3-13 所示。

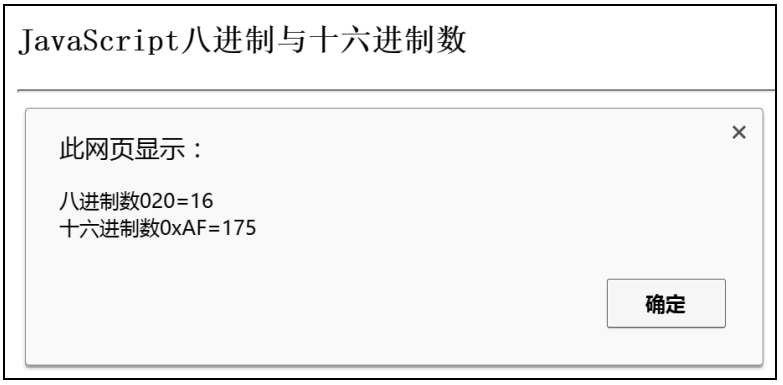

图 3-13　JavaScript 八进制与十六进制数的输出结果

【代码说明】

本例为变量 x1 赋值了 0 开头的数字代表八进制数，为变量 x2 赋值了 0x 开头的数字代表十六进制数，并使用 alert()语句将其显示在消息提示对话框中。由图 3-13 可见，最终显示结果会自动转换为十进制数。注意，如果需要正常表示十进制整数，则不要使用数字 0 开头，以免被误认为是八进制数。

3．浮点数

如果要定义浮点数，必须使用小数点，并且小数点后面至少跟一位数字。例如：

```
var x=3.14;
var y=5.0;
```

即使小数点后面的数字为 0 也被认为是浮点数类型。

如果浮点数类型的小数点前面的整数位为 0，可以省略。例如：

```
var x=.15; //等同于 0.15
```

浮点数可以使用 toFixed()方法规定小数点后保留几位数，其语法格式如下。

```
toFixed(digital)
```

其中，参数 digital 换成小数点后需要保留的位数即可。例如：

```
var x=3.1415926;
var result=x.toFixed(2); //返回值为 3.14
```

该方法遵循四舍五入的规律，即使进位后小数点后面只有 0 也会保留指定的位数。例如：

```
var x=0.9999;
var result=x.toFixed(2); //返回值为 1.00
```

需要注意的是，在 JavaScript 中使用浮点数进行计算有时会产生误差。例如：

```
var x=0.7 + 0.1;
alert(x);//返回值会变成 0.7999999999999999，而不是 0.8
```

这是由于表达式使用的是十进制数，但实际的计算是转换成二进制数进行计算再转回十进制结果的，在此过程中可能会损失精度。此时使用自定义函数将两个加数都乘以 10 进行计算后再除以 10 还原。

【例 3-13】 JavaScript 浮点数类型的简单应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 浮点数类型的简单应用</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 浮点数类型的简单应用</h3>
9.          <hr/>
10.        <p>
11.            浮点数的加法运算。
12.        </p>
13.        <script>
```

```
14.         //直接将两数相加
15.         var result1=0.7+0.1;
16.         //将浮点数转换成整数后相加再还原
17.         var result2=(0.7*10+0.1*10)/10;
18.         //输出结果
19.         document.write("0.7+0.1="+result1+"<br>(0.7*10+0.1*10)/10="+result2);
20.         </script>
21.     </body>
22. </html>
```

运行效果如图 3-14 所示。

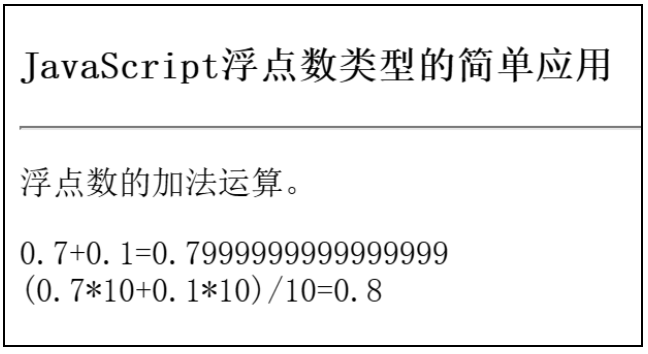

图 3-14　JavaScript 浮点数类型的简单应用效果

【代码说明】

本例用于测试两个浮点数在进行算术运行时导致的误差，并给出了解决办法。事实上，目前 JavaScript 尚不能解决该问题，必须手动将浮点数放大 10 的倍数成为整数再进行计算才能避免误差，未来也可以使用自定义函数处理此类问题。

4．特殊 Number 值

在 JavaScript 中，Number 类型还有一些特殊值，如表 3-4 所示。

表 3-4　JavaScript 中 Number 类型的特殊值

特　殊　值	解　　释
Infinity	正无穷大，在 JavaScript 中使用 Number.POSITIVE_INFINITY 表示
-Infinity	负无穷大，在 JavaScript 中使用 Number.NEGATIVE_INFINITY 表示
NaN	非数字，在 JavaScript 使用 Number.NaN 表示
Number.MAX_VALUE	数值范围允许的最大值，大约等于 1.8e308
Number.MIN_VALUE	数值范围允许的最小值，大约等于 5e-324

1）Infinity

Infinity 表示无穷大，有正、负之分。当数值超过了 JavaScript 允许的范围时就会显示为 Infinity（超过上限）或-Infinity（超过下限）。例如：

```
var x=9e30000;
alert(x); //因为该数字已经超出上限，返回值为 Infinity
```

在比较数字大小时，无论原数据值为多少，认为结果为 Infinity 的两个数相等，同样两个-Infinity 也是相等的。例如：

```
var x1=3e9000;
var x2=9e3000;
alert(x1==x2);  //判断变量 x1 与 x2 是否相等，返回值为 true
```

上述代码中变量 x1 与 x2 的实际数据值并不相等，但是由于它们均超出了 JavaScript 可以接受的数据范围，因此返回值均为 Infinity，从而在判断是否相等时会返回 true（真）。

在 JavaScript 中使用数字 0 作为除数不会报错，如果正数除以 0 返回值就是 Infinity，负数除以 0 返回值为-Infinity，特殊情况“0 除以 0”的返回值为 NaN（非数字）。例如：

```
var x1=5/0;      //返回值是 Infinity
var x2=-5/0;     //返回值是-Infinity
var x3=0/0;      //返回值是 NaN
```

Infinity 不可以与其他正常显示的数字进行数学计算，返回结果均是 NaN。例如：

```
var x=Number.POSITIVE_INFINITY;
var result=x+99;
alert(result);  //返回值为 NaN
```

【例 3-14】 JavaScript 特殊值 Infinity 的应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 特殊值 Infinity</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 特殊值 Infinity</h3>
9.          <hr/>
10.         <script>
11.             var x1=2e9000;
12.             var x2=-2e9000;
13.             var result=x1+x2;
14.             alert("x1(2e9000)="+x1+"\nx2(-2e9000)="+x2+"\nx1+x2="+result);
15.         </script>
16.     </body>
17. </html>
```

运行效果如图 3-15 所示。

2）NaN

NaN 表示的是非数字（Not a Number），该数值用于表示数据转换成 Number 类型失败的情况，从而无须抛出异常错误。例如将 String 类型转换为 Number 类型。NaN 因为不是

真正的数字，不能用于进行数学计算，并且即使两个数值均为 NaN，它们也不相等。

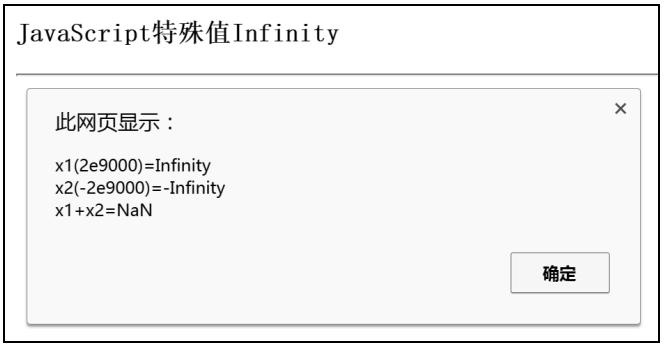

图 3-15　JavaScript 特殊值 Infinity 的输出结果

例如将英文单词转换为 Number 类型会导致转换结果为 NaN，具体代码如下。

```
var x="red";
var result = Number(x); //返回值为 NaN，因为没有对应的数值可以转换
```

JavaScript 还提供了用于判断数据类型是否为数值的方法 isNaN()，其返回值是布尔值。当检测的数据无法正确转换为 Number 类型时返回真（true），其他情况返回假（false）。

其语法规则如下。

```
isNaN(变量名称)
```

例如：

```
var x1="red";
var result1=isNaN(x1);  //返回值是真（true）

var x2="999";
var result2=isNaN(x2);  //返回值是假（false）
```

【例 3-15】 JavaScript 特殊值 NaN 的应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 特殊值 NaN</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 特殊值 NaN</h3>
9.          <hr/>
10.         <script>
11.             var x1="hello";
12.             var x2=999;
13.             var result1=Number(x1);
```

```
14.            alert('x1(hello)不是数字：'+isNaN(x1)+'\nx2(999)不是数字：'+isNaN(x2)
               +'\nx1 转换为数字的结果为：'+result1);
15.        </script>
16.    </body>
17. </html>
```

运行效果如图 3-16 所示。

图 3-16　JavaScript 特殊值 NaN 的输出结果

3.1.5　Boolean 类型

布尔值（Boolean）在很多程序语言中被用于进行条件判断，其值只有两种——true（真）和 false（假）。

布尔类型的值可以直接使用单词 true 或 false，也可以使用表达式。例如：

```
var answer=true;
var answer=false;
var answer=(1>2);
```

其中，1>2 的表达式不成立，因此返回结果为 false（假）。

【例 3-16】 JavaScript Boolean 类型的简单应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript Boolean 类型的简单应用</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript Boolean 类型的简单应用</h3>
9.          <hr/>
10.         <script>
11.             var x1=Boolean("hello");
12.             var x2=Boolean(999);
13.             var x3=Boolean(0);
```

```
14.            var x4=Boolean(null);
15.            var x5=Boolean(undefined);
16.            alert("hello:"+x1+"\n999:"+x2+"\n0:"+x3+"\nnull:"+x4+"
               \nundefined:"+x5);
17.        </script>
18.    </body>
19. </html>
```

运行效果如图 3-17 所示。

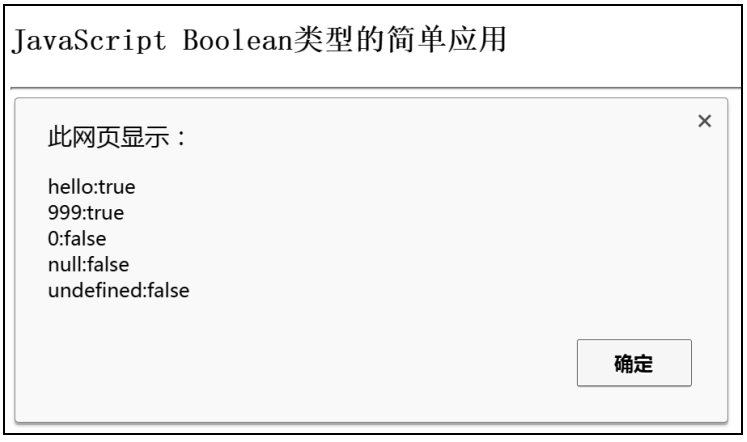

图 3-17　JavaScript Boolean 类型的简单应用效果

3.2 JavaScript 对象类型

在 JavaScript 中对象类型分为 3 种，即本地对象、内置对象和宿主对象。本地对象（native object）是 ECMAScript 定义的引用类型；内置对象（built-in object）指的是无须实例化可以直接使用的对象，其实也是特殊的本地对象；宿主对象（host object）指的是用户的机器环境，包括 DOM 和 BOM。

3.2.1　本地对象

1．Array 对象

在 JavaScript 中可以使用数组（Array）类型在单个变量中存储一系列的值。例如：

```
var mobile=new Array();
var mobile [0]="苹果";
var mobile [1]="三星";
var mobile [2]="华为";
```

数组是从 0 开始计数的，因此第 1 个元素的下标是[0]，后面每新增一个元素下标加 1。使用 Array 类型存储数组的特点是无须在一开始声明数组的具体元素数量，可以在后续代码中陆续新增数组元素。

如果一开始就可以确定数组的长度，即其中的元素不需要后续动态加入，可以直接写

成如下形式。

```
var mobile=new Array("苹果", "三星", "华为");
或
var mobile=["苹果", "三星", "华为"];
```

此时数组元素之间使用逗号隔开。

Array 对象还包含了 length 属性，可以用于获取当前数组的长度，即数组中元素的个数。如果当前数组中没有包含元素，则 length 值为 0。例如：

```
var mobile=["苹果", "三星", "华为"];
var x=mobile.length; //这里 x 值为 3
```

Array 对象还包含了一系列方法用于操作数组，常用方法如表 3-5 所示。

表 3-5　JavaScript Array 对象的常用方法

方 法 名 称	解　　释
concat(array1,array2, …, arrayN)	用于在数组末尾处连接一个或者多个新的数组或数组元素。其中参数 array1 为必填内容，后面的参数个数不限，均为可选内容。参数内容可以是数组或数组元素的值
join(separator)	把数组中的所有元素用指定的分隔符进行分割，并在同一个字符串中显示出来。其中 separator 参数表示指定的自定义分隔符。该参数为可选内容 0，在不填写的情况下默认用逗号分隔
pop()	删除数组中的最后一个元素，并返回该元素的值。如果数组内容是空的，则该方法返回 undefined，并且不进行操作
push(element1, element2,…,elementN)	在数组的结尾处插入一个或者多个元素，并返回最新数组长度。其中参数 element1 为必填内容，表示至少添加一个元素，后面的参数个数不限，均为可选内容
reverse()	用于将数组中的所有元素倒序重组。该方法会直接更改原始数组，而不是生成一个新的数组
shift()	删除数组中的第 1 个元素，并返回该元素的值。如果数组内容为空，则该方法返回 undefined，并且不进行操作
slice(start, end)	用于返回数组中指定了开始与结束范围的一系列元素。其中参数 start 为必填内容，表示从第几个元素开始选取；参数 end 为可选内容，表示选取到第几个元素结束，并且不包括该元素本身。如果没有指定 end 参数，则一直选取到数组的最后一个元素结束。如果这两个参数填写了负数，则表示从数组末尾开始计算个数。例如−1 表示最后一个元素，−2 表示倒数第 2 个元素，以此类推
toString()	用于把数组元素显示在同一个字符串中，并且用逗号分隔，相当于没有指定分隔符的 join()方法的使用
unshift(element1,…, elementN)	在数组的开头插入一个或多个元素，并返回最新数组长度。其中参数 element1 为必填内容，表示至少添加一个元素，后面的参数个数不限，均为可选内容

【例 3-17】 JavaScript Array 对象的简单应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
```

```
5.          <title>JavaScript Array 对象的简单应用</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript Array 对象的简单应用</h3>
9.          <hr/>
10.         <script>
11.         //使用 new Array()构建数组对象
12.         var students=new Array();
13.         students[0]="张三";
14.         students[1]="李四";
15.         students[2]="王五";
16.
17.         //直接声明数组对象
18.         var mobile=["Nokia","HTC","iPhone"];
19.         alert(students+"\n"+mobile);
20.         </script>
21.     </body>
22. </html>
```

运行效果如图 3-18 所示。

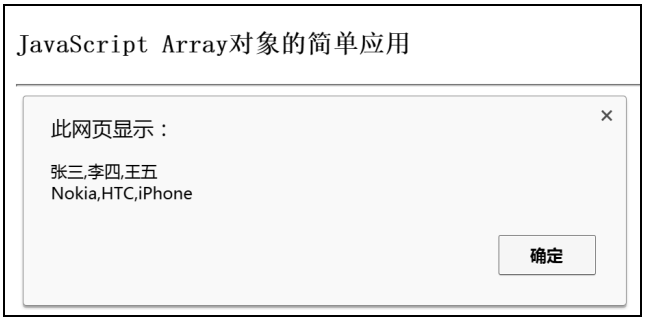

图 3-18　JavaScript Array 对象的简单应用效果

【代码说明】

本例分别使用了两种不同的声明方式创建数组，并在最后使用 alert()语句在弹出的提示框中显示这两个数组的内容。由图 3-18 可见，两种声明方式均可成功使用 Array 对象创建数组，并对数组元素进行赋值。

2．Date 对象

在 JavaScript 中使用 Date 对象处理与日期、时间有关的内容，有 4 种初始化方式，列举如下。

```
//表示获取当前的日期与时间
new Date();
//使用表示日期时间的字符串定义时间，例如填入 May 10, 2000 12:12:00
new Date(dateString);
//使用从 1970 年 1 月 1 日到指定日期的毫秒数定义时间，例如填入 1232345
```

```
new Date(milliseconds);
//自定义年、月、日、时、分、秒和毫秒，时、分、秒和毫秒参数省略时默认为 0
new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

用户可以通过 Date 对象的一系列方法分别获取指定的内容，Date 对象的常用方法如表 3-6 所示。

表 3-6　JavaScript Date 对象的常用方法

方 法 名 称	解　　释
Date()	获取当前的日期和时间
getDate()	获取 Date 对象处于一个月里面的哪一日（1～31）
getDay()	获取 Date 对象处于星期几（0～6），其中 0 表示星期日
getMonth()	获取 Date 对象处于几月份（0～11），其中 0 表示一月份
getFullYear()	获取 Date 对象的完整年份（4 位数）
getHours()	获取 Date 对象的小时（0～23）
getMinutes()	获取 Date 对象的分钟（0～59）
getSeconds()	获取 Date 对象的秒（0～59）
getTime()	返回 1970 年 1 月 1 日至今经历的毫秒数
setDate()	重新设置 Date 对象中的日期，精确到天
setFullYear()	重新设置 Date 对象中的年份，年份必须为 4 位数的完整表达
setHours()	重新设置 Date 对象中的小时（0～23）
setMinutes()	重新设置 Date 对象中的分钟（0～59）
setMonth()	重新设置 Date 对象中的月份（0～11）
setSeconds()	重新设置 Date 对象中的秒（0～59）
setTime()	重新以 1970 年 1 月 1 日至今经历的毫秒数设置 Date 对象
toDateString()	将 Date 对象的日期部分转换为字符串
toLocaleDateString()	根据本地时间格式将 Date 对象的日期部分转换为字符串
toLocaleTimeString()	根据本地时间格式将 Date 对象的时间部分转换为字符串
toLocaleString()	根据本地时间格式将 Date 对象的日期和时间转换为字符串
toUTCString()	根据世界时间格式将 Date 对象的日期和时间转换为字符串

【例 3-18】 JavaScript Date 对象的简单应用

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript Date 对象的简单应用</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript Date 对象的简单应用</h3>
9.          <hr/>
10.         <script>
11.         //获取当前日期时间对象
12.         var date=new Date();
```

```
13.         //获取年份
14.         var year=date.getFullYear();
15.         //获取月份
16.         var month=date.getMonth()+1;
17.         //获取天数
18.         var day=date.getDate();
19.         //获取星期
20.         var week=date.getDay();
21.         alert("当前是"+year+"年"+""+month+"月"+day+"日，星期"+week);
22.         </script>
23.     </body>
24. </html>
```

运行效果如图 3-19 所示。

图 3-19　JavaScript Date 对象的简单应用效果

【代码说明】

本例声明了当前的日期时间对象 date，然后分别获取其中的年、月、日与星期，并使用 alert()语句在弹出的提示框中显示组合内容。由于月份的取值范围是 0～11，因此如果要显示实际月份，需要将获取到的月份值再进行加 1 处理。

3．RegExp 对象

RegExp 对象表示正则表达式（regular expression），通常用于检索文本中是否包含指定的字符串，其语法格式如下。

```
new RegExp(pattern [, attributes])
```

参数解释如下。

- pattern：该参数为字符串形式，用于规定正则表达式的匹配规则或填入其他正则表达式。
- attributes：该参数为可选参数，可以包含属性值 g、i 或者 m，分别表示全局匹配、区分大小写匹配与多行匹配。

例如：

```
var pattern=new RegExp([0-9], g);
```

上述代码表示声明了一个用于全局检索文本中是否包含数字 0～9 的任意字符的正则

表达式。

它还有一种简写形式，格式如下。

```
/pattern/[attributes]
```

前面用于全局检索数字 0～9 的正则表达式声明可修改为如下内容。

```
var pattern=/[0-9]/g;
```

两种写法效果完全相同，需要注意参数 attributes 均仅适用于参数 pattern 为匹配规则字符串的情况。如果参数 pattern 填写的是其他正则表达式，则参数 attributes 必须省略不写。

JavaScript 中常用的正则表达式如表 3-7 所示。

表 3-7　JavaScript 中常用的正则表达式

括号表达式	解　　释
[0-9]	查找 0～9 的数字
[a-z]	查找从小写字母 a 到小写字母 z 之间的字符
[A-Z]	查找从大写字母 A 到大写字母 Z 之间的字符
[A-z]	查找从大写字母 A 到小写字母 z 之间的字符
[abc]	查找括号之间的任意一个字符
[^abc]	查找括号内字符之外的所有内容
(red|blue|green)	查找“|”符号间隔的任意选项内容
量词表达式	**解　　释**
n+	查找任何至少包含一个 n 的字符串
n*	查找任何包含 0 个或多个 n 的字符串
n?	查找任何包含 0～1 个 n 的字符串
n{X}	查找包含 X 个 n 的字符串
n{X,Y}	查找包含 X 或 Y 个 n 的字符串
n{X,}	查找至少包含 X 个 n 的字符串
n$	查找任何以 n 结束的字符串
^n	查找任何以 n 开头的字符串
?=n	查找任何后面紧跟字符 n 的字符串
?!n	查找任何后面没有紧跟字符 n 的字符串
元　字　符	**解　　释**
.	查找除了换行符与行结束符以外的单个字符
\w	查找单词字符。w 表示 word（单词）
\W	查找非单词字符
\d	查找数字字符。d 表示 digital（数字）
\D	查找非数字字符
\s	查找空格字符。s 表示 space（空格）
\S	查找非空格字符
\n	查找换行符
\f	查找换页符
\r	查找回车符
\t	查找制表符

续表

元 字 符	解 释
\xxx	查找八进制数字 xxx 对应的字符，如果没有找到则返回 null。例如\130 表示的是大写字母 X
\xdd	查找十六进制数字 dd 对应的字符，如果没有找到则返回 null。例如\x58 表示的是大写字母 X
\uxxxx	查找十六进制数字 xxxx 对应的 Unicode 字符，如果没有找到则返回 null。例如\u0058 表示的是大写字母 X

注：量词表达式中的 n 可以替换成其他任意字符。

在 RegExp 对象创建完毕后有两种方法用于检索文本，如表 3-8 所示。

表 3-8 JavaScript 中 RegExp 对象的方法

方 法 名	解 释
exec()	该方法适用于具有参数的情况，每次运行都按照顺序从文本中找到对应的字符串，直到全部找完，再次运行会返回 null 值
test()	用于检索文本中是否包含指定的字符串

1）exec()方法的应用

exec()方法用于检索文本中匹配正则表达式的字符串内容，其语法格式如下。

```
RegExpObject.exec(string)
```

该方法如果找到了匹配内容，其返回值为存放有检索结果的数组；如果未找到任何匹配内容，则返回 null 值。

例如：

```
var pattern=new RegExp("e");          //检索文本中是否包含小写字母 e 的正则表达式
var result1=pattern.exec("Hello ");//返回值为 e，因为字符串中包含小写字母 e
var result2=pattern.exec("Hello ");//返回值为 null，因为字符串后续内容中不包含小
                                      //写字母 e
```

如果查到的内容较多，可以使用 while 循环语句进行检索。例如：

```
var s="Hello everyone";               //初始字符串
var pattern=new RegExp("e");          //检索文本中是否包含小写字母 e 的正则表达式
var result;                           //用于获取每次的检索结果
//while 循环
while((result=pattern.exec(s))!=null){
    alert(result);                    //输出本次检索结果
}
```

关于 while 循环的更多内容可查阅第 4 章的 4.2 节。

【例 3-19】 JavaScript 正则表达式 exec()方法的简单应用

使用 exec()匹配符合正则表达式的字符串。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>JavaScript 正则表达式 exec()方法的应用</title>
    </head>
    <body>
        <h3>JavaScript 正则表达式 exec()方法的应用</h3>
        <hr/>
        <p>原始字符串为："Happy New Year 2016"</p>
        <script>
        //原始字符串
        var s="Happy New Year 2016";
        //定义正则表达式，用于全局检索字母 N 是否存在
        var pattern=/N/g;
        //第一次匹配结果
        var result1=pattern.exec(s);
        //第二次匹配结果
        var result2=pattern.exec(s);
        alert("第一次匹配结果:"+result1+"\n 第二次匹配结果:"+result2);
        </script>
    </body>
</html>
```

运行效果如图 3-20 所示。

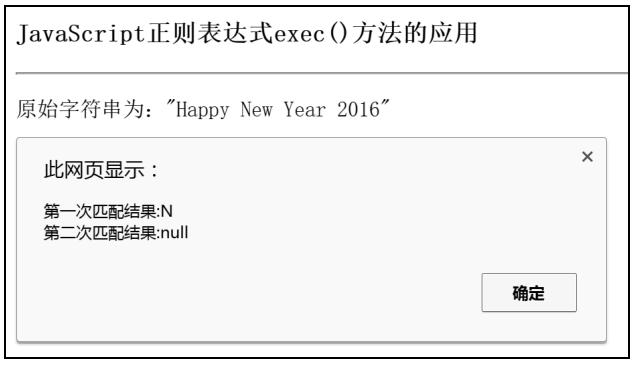

图 3-20　JavaScript 正则表达式 exec()方法的应用效果

2）test()方法的应用

test()方法用于检测文本中是否包含指定的正则表达式内容，返回值为布尔值，其语法格式如下。

```
RegExpObject.test(string)
```

其中，RegExpObject 指的是自定义的 RegExp 对象；参数 string 指的是需要被检索的文本内容。如果文本中包含该 RegExp 对象指定的内容，返回值为 true，否则返回值为 false。

该方法只用于无参数的情况，并且只检索一次，一旦检索到了就停止并提供返回值。

例如：

```
var pattern=new RegExp("e");        //检索文本中是否包含小写字母 e 的正则表达式
var result=pattern.test("Hello ");  //返回值为 true，因为字符串中包含小写字母 e
```

【例 3-20】 JavaScript 正则表达式 test()方法的应用

使用 test()匹配符合正则表达式的字符串。

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 正则表达式 test()方法的应用</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 正则表达式 test()方法的应用</h3>
9.          <hr/>
10.         <p>原始字符串为："Happy New Year 2016"</p>
11.         <script>
12.         //原始字符串
13.         var s="Happy New Year 2016";
14.         //定义正则表达式，用于全局检索字母 N 是否存在
15.         var pattern=/N/g;
16.         //匹配结果
17.         var result=pattern.test(s);
18.         alert("查找字母 N 的匹配结果:"+result);
19.         </script>
20.     </body>
21. </html>
```

运行效果如图 3-21 所示。

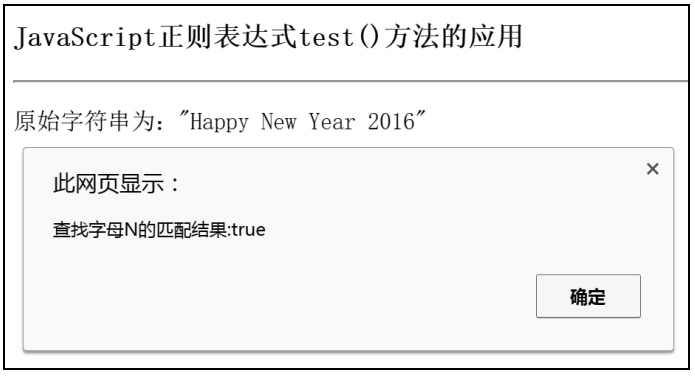

图 3-21　JavaScript 正则表达式 test()方法的应用效果

4．Object 对象

在 JavaScript 中所有类型都是对象，例如字符串、数字、数组等，这些可以带有属性和方法的变量称为对象。例如 String 对象包含了 length 属性用于获取字符串长度，也包含了 substring()、indexOf()等方法用于处理字符串。

属性是与对象相关的值，方法是对象可执行的动作。例如将学生作为现实中的对象，具有学号、姓名、班级、专业等属性值，也可以具有选课、学习和考试等行为动作。

在 JavaScript 中创建 student 对象的写法如下。

```
var student=new Object();
student.name="张三";              //姓名
student.id="2016010212";          //学号
student.major="计算机科学与技术"; //专业
//学习方法
student.study=function(){
    alert("开始学习");
};
```

上述代码为 student 对象添加了 name、id 和 major 属性以及 study()方法，分别用于表示学生的姓名、学号、专业属性和学习的行为动作。对象名称为自定义，其内部的属性和方法均可以根据实际需要自定义名称与数量。

获取对象中的指定属性有两种方法：一是对象变量名称后面加点（.）和属性名称（对象名.属性名）；二是对象变量名称后面使用中括号和引号包围属性名称（对象名["属性名"]）。

这里仍然以上面的 student 对象为例，获取其中学生姓名的写法如下。

```
var result=student.name;
或
var result=student["name"];
```

另外还可以用该方法直接修改对象中的属性值，例如将之前的学生姓名“张三”换成新内容“李四”。

```
student.name="李四";
alert(student.name); //此时的输出结果不再是张三，而是修改后的李四
```

【例 3-21】 JavaScript Object 对象的简单应用

创建自定义名称的 Object 对象。

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript Object 对象的简单应用</title>
6.      </head>
7.      <body>
```

```
8.          <h3>JavaScript Object 对象的简单应用</h3>
9.          <hr/>
10.         <p>
11.             自定义 ticket 对象表示电影票信息。
12.         </p>
13.         <script>
14.             //自定义 JavaScript 对象 ticket 表示电影票
15.             var ticket=new Object();
16.             //电影票主题
17.             ticket.topic="海底总动员";
18.             //电影票时间
19.             ticket.time="2016 年 10 月 1 日 14:30";
20.             //电影票价格
21.             ticket.price="25 元";
22.             //电影票座位号
23.             ticket.seat="8 排 6 号";
24.             alert("电影主题: " + ticket.topic+"\n 电影时间: " + ticket.time+"\n
                电影票价格: "+ticket.price+"\n 座位号: "+ticket.seat);
25.         </script>
26.     </body>
27. </html>
```

运行效果如图 3-22 所示。

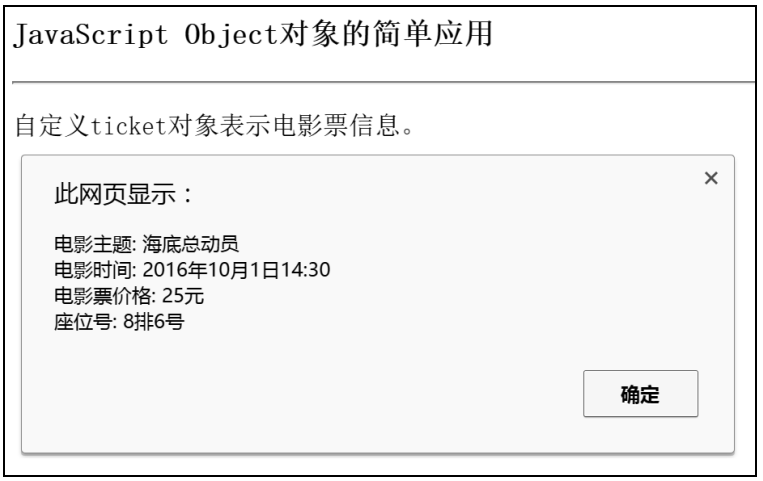

图 3-22　JavaScript Object 对象的简单应用效果

3.2.2　内置对象

1．Global 对象

在 JavaScript 中 Global 对象又称为全局对象，其中包含的属性和函数可以用于所有的本地 JavaScript 对象。Global 对象的全局属性和方法分别如表 3-9 和表 3-10 所示。

表 3-9　Global 对象的全局属性一览表

属 性 名 称	解　　释
Infinity	表示正无穷大的数值，在数值超过了 JavaScript 规定的范围时使用
java	表示引用的一个 java 包，是 Packages.java 的简写
NaN	表示非数值（Not a Number），通常在其他类型转换成 Number 类型时使用
Packages	表示 JavaPackage 对象，是所有 java 包的根
undefined	表示未声明或未赋值的变量值

表 3-10　Global 对象的全局方法一览表

属 性 名 称	解　　释
decodeURI()	解码 URI
decodeURIComponent()	解码 URI 组件
encodeURI()	把字符串编码为 URI
encodeURIComponent()	把字符串编码为 URI 组件
escape()	对字符串进行编码
eval()	将 JavaScript 字符串转换为脚本代码
getClass()	返回 Java 对象的类
isFinite()	判断某个值是否为无穷大
isNaN()	判断值是否为数字
Number()	把对象的值转换为数字类型
parseInt()	把字符串转换为整数
parseFloat()	把字符串转换为浮点数
String()	把对象的值转换为字符串类型
unescape()	对使用 escape()编码的字符串进行解码

2．Math 对象

在 JavaScript 中 Math 对象用于数学计算，无须初始化创建，可以直接使用关键字 Math 调用其所有的属性和方法。Math 对象的常用属性和常用方法分别如表 3-11 和表 3-12 所示。

表 3-11　Math 对象的常用属性

属 性 名 称	解　　释
E	返回算术常量 e（约为 2.718）
LN2	返回 log 以算术常量 e 为底的 2 的对数（约为 0.693）
LN10	返回 log 以算术常量 e 为底的 10 的对数（约为 2.302）
LOG2E	返回 log 以 2 为底的算术常量 e 的对数（约为 1.414）
LOG10E	返回 log 以 10 为底的算术常量 e 的对数（约为 0.434）
PI	返回圆周率 π 的值（约为 3.1415926）
SQRT1_2	返回数字 2 的平方根的倒数（约为 0.707）
SQRT2	返回数字 2 的平方根（约为 1.414）

表 3-12　Math 对象的常用方法

方法名称	解　　释	使 用 示 例	示例中 result 的值
abs(x)	返回数字的绝对值	var x =−100; var result = abs(x);	100

续表

方法名称	解　　释	使 用 示 例	示例中 result 的值
ceil(x)	使用进一法返回整数值，即舍去小数点和后面的所有内容，整数部分加 1	var x = 3.1415; var result = ceil(x);	4
cos(x)	返回数字的余弦值，x 指的是弧度值	var x = Math.PI/2; var result = cos(x);	0
floor(x)	使用去尾法返回整数值，即舍去小数点和后面的所有内容，整数部分不变	var x = 3.1415; var result = floor(x);	3
max(x,y)	返回两个数之间的最大值	var x = 2, y = 3; var result = max(x, y);	3
min(x,y)	返回两个数之间的最小值	var x = 2, y = 3; var result = min(x, y);	2
pow(x,y)	返回 x 的 y 次方	var x = 2, y = 3; var result = pow(x, y);	8
random()	返回[0,1)的随机数	var result = random();	0～1 的随机浮点数
round(x)	返回数字四舍五入后的整数	var x = 3.1415; var result = round(x);	3
sin(x)	返回数字的正弦值，x 指的是弧度值	var x = Math.PI/2; var result = sin(x);	1
sqrt(x)	返回数字的平方根	var x = 9; var result = sqrt(x);	3
tan(x)	返回数字的正切值，x 指的是弧度值	var x = Math.PI/4; var result = tan(x);	1

注：角度值 360°相当于弧度值 2π。

【例 3-22】 JavaScript Math 对象的简单应用

使用 Math 对象中的部分属性和方法计算球体的体积，公式为 $V=4/3\pi R^3$。

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript Math 对象的简单应用</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript Math 对象的简单应用</h3>
9.          <hr/>
10.         <p>
11.             已知球体半径为 100m，使用 Math 对象计算球体的体积。<br>
12.               公式：V=4/3 π R<sup>3</sup>
13.         </p>
14.         <script>
15.          //初始化球体半径
16.          var R=100;
17.          //计算球体的体积
```

```
18.          var V=4/3*Math.PI*Math.pow(R,3);
19.          //四舍五入后显示计算结果
20.          alert("半径为 100 的球体体积是：  "+Math.round(V)+"m³ ");
21.        </script>
22.    </body>
23. </html>
```

运行效果如图 3-23 所示。

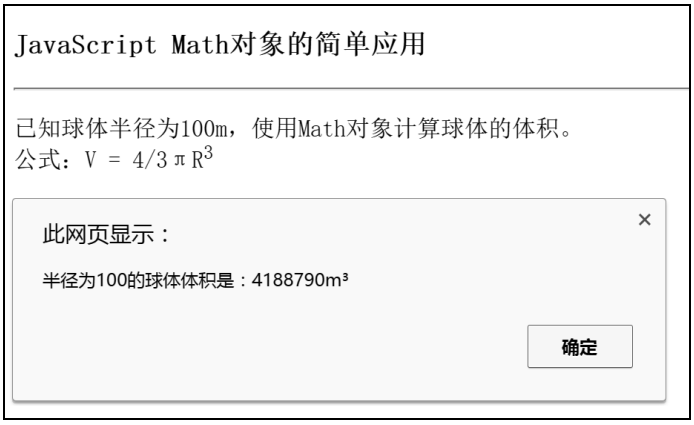

图 3-23　JavaScript Math 对象的简单应用效果

3.2.3　宿主对象

宿主对象包括 HTML DOM（文档对象模型）和 BOM（浏览器对象模型），具体内容和用法请参考第 6 章。

3.3　JavaScript 类型转换

3.3.1　转换成字符串

在 JavaScript 中，布尔值类型（Boolean）和数字类型（Number）均可使用 toString()方法把值转换为字符串形式。

布尔值类型（Boolean）的 toString()方法只能根据初始值返回 true 或者 false。例如：

```
var x=true;
var result=x.toString(); //返回"true"
```

数字类型（Number）使用 toString()方法有两种模式，即默认模式和基数模式。

在默认模式中，toString()不带参数直接使用，此时无论是整数、小数或者科学记数法表示的内容都会显示为十进制的数值。例如：

```
var x1=99;
var x2=99.90;
var x3=1.25e8;
```

```
var result1=x1.toString(); //返回值为"99"
var result2=x2.toString(); //返回值为"99.9"
var result3=x3.toString(); //返回值为"125000000"
```

如果小数点后面是以 0 结束，那么在转换成 String 类型时最末端的 0 都会被省略，本例中变量 x2 的返回值就是"99.9"，而不是原始的"99.90"；使用科学记数法表示的数值也会显示成计算后的十进制完整结果，本例中变量 x3 的返回值就是"125000000"，而不是"1.25e8"本身。

在基数模式下，需要在 toString()方法的括号内部填入一个指定的参数，根据参数指示把原始数据转换为二进制、八进制或十六进制数。其中，二进制对应基数 2，八进制对应基数 8，十六进制对应基数 16。例如：

```
var x=10;
var result1=x.toString(2);  //声明将原始数据转换成二进制数，返回值为"1010"
var result2=x.toString(8);  //声明将原始数据转换成八进制数，返回值为"12"
var result3=x.toString(16); //声明将原始数据转换成十六进制数，返回值为"A"
```

由此可见，对于同一个变量使用 toString()方法进行转换，如果填入的基数不同会导致返回完全不同的结果。

【例 3-23】 JavaScript 转换字符串类型的简单应用

使用 toString()方法将变量值转换为字符串类型。

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 转换字符串类型的简单应用</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 转换字符串类型的简单应用</h3>
9.          <hr/>
10.         <p>
11.             将数值 200 分别转换为二进制、八进制和十六进制数。
12.         </p>
13.         <script>
14.         //初始化变量
15.         var x=200;
16.         //转换为二进制
17.         var result1=x.toString(2);
18.         //转换为八进制
19.         var result2=x.toString(8);
20.         //转换为十六进制
21.         var result3=x.toString(16);
22.         //输出结果
```

```
23.         alert("toString(2):"+result1+"\ntoString(8):"+result2+"\ntoString(16):
            "+result3);
24.         </script>
25.     </body>
26. </html>
```

运行效果如图 3-24 所示。

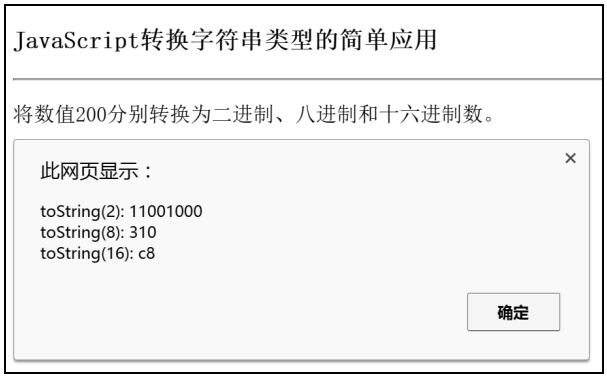

图 3-24　JavaScript 转换字符串类型的简单应用效果

3.3.2　转换成数字

JavaScript 提供了两种将 String 类型转换为 Number 类型的方法，即 parseInt()和 parseFloat()，其中 parseInt()用于将值转换为整数，parseFloat()用于将值转换为浮点数。这两种方法仅适用于对 String 类型的数字内容的转换，其他类型的返回值都是 NaN。

1．parseInt()方法

parseInt()方法转换的原理是从左往右依次检查每个位置上的字符，判断该位置上是否为有效数字，如果是则将有效数字转换为 Number 类型，直到发现不是数字的字符，停止后续的检查工作。例如：

```
var x="123hello";
var result=parseInt(x); //返回值是 123，因为 h 不是有效数字，停止检查
```

如果需要转换的字符串从第一个位置就不是有效数字，则直接返回 NaN。例如：

```
var x="hello";
var result=parseInt(x);//返回值是 NaN，因为第一个字符 h 就不是有效数字，直接停止检查
```

由于 parseInt()只能进行整数数字的转换，因此即使检测到某个字符位置上是小数点也会认为不是有效数字，从而终止检测和转换。例如：

```
var x="3.14";
var result=parseInt(x); //返回值是 3，因为小数点不是有效数字，停止检查
```

parseInt()方法还有一个参数二，可以用于声明需要转换的数字为二进制、八进制、十进制、十六进制等。例如：

```
var x="10";
var result1=parseInt(x, 2);     //表示原始数据为二进制，返回值为 2
var result2=parseInt(x, 8);     //表示原始数据为八进制，返回值为 8
var result3=parseInt(x, 10);    //表示原始数据为十进制，返回值为 10
var result4=parseInt(x, 16);    //表示原始数据为十六进制，返回值为 16
```

有一种特殊情况需要注意：如果原始数据为十进制数，但是前面以数字 0 开头，则最好使用参数二进行特别强调，否则会被默认转换为八进制数。例如：

```
var x="010";
var result1=parseInt(x);        //表示原始数据为八进制，返回值为 8
var result2=parseInt(x, 10);    //表示原始数据为十进制，返回值为 10
var result3=parseInt(x, 8);     //表示原始数据为八进制，返回值为 8
```

因此，如果是声明十进制数，尽量避免使用 0 作为开头的写法。

【例 3-24】 JavaScript 转换整数类型的简单应用

使用 parseInt()方法将变量值转换为整数类型。

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 转换整数类型的简单应用</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 转换整数类型的简单应用</h3>
9.          <hr/>
10.         <p>
11.             var x="3.99";
12.         </p>
13.         <script>
14.          //初始化变量 x
15.          var x="3.99";
16.          //直接使用+号
17.          var result1=x+1;
18.          //转换为整数后使用+号
19.          var result2=parseInt(x)+1;
20.          //输出结果
21.          alert("x+1="+result1+"\nparseInt(x)+1="+result2);
22.         </script>
23.     </body>
24. </html>
```

运行效果如图 3-25 所示。

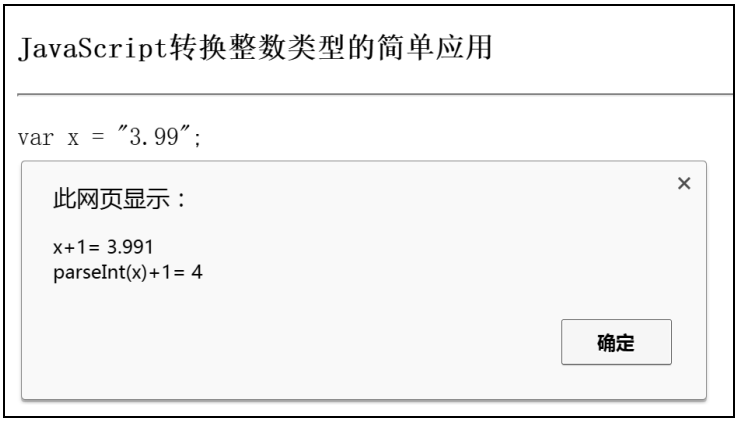

图 3-25　JavaScript 转换整数类型的简单应用效果

【代码说明】

本例声明了变量 x 并赋值为字符串"3.99"作为测试用例，变量 result1 的返回值为直接使用加号（+）连接变量 x 和数字 1 的结果；变量 result2 的返回值为使用 parseInt()方法对变量 x 进行类型转换后再与数字 1 相加的结果。

由图 3-25 可见，当直接使用加号（+）时会默认为是两个字符串的连接，并没有进行加法计算，只有先将变量 x 转换为整数类型后才进行了数学运算，其中字符串"3.99"转换后为数字 3。这是由于加号的特殊性，既可以做数学运算符号也可以做字符串的连接符号。

2．parseFloat()方法

parseFloat()方法的转换原理与 parseInt()方法类似，都是从左往右依次检查每个位置上的字符，判断该位置上是否为有效数字，如果是则将有效数字转换为 Number 类型，直到发现不是数字的字符，停止后续的检查工作。

与 parseInt()方法类似，如果需要转换的字符串从第一个位置就不是有效数字，则直接返回 NaN。例如：

```
var x="hello3.14";
var result=parseFloat(x);//返回值是 NaN，因为第一个字符 h 就不是有效数字，停止检查
```

与 parseInt()方法不同的是，小数点在 parseInt()方法中也被认为是无效字符，但是在 parseFloat()方法中首次出现的小数点被认为是有效的。例如：

```
var x="3.14hello";
var result=parseFloat(x); //返回值是 3.14，因为 h 不是有效数字，停止检查
```

如果同时出现多个小数点，只有第一个小数点是有效的。例如：

```
var x="3.14.15.926";
var result=parseFloat(x); //返回值是 3.14，因为第二个小数点不是有效数字，停止检查
```

parseFloat()和 parseInt()还有一个不同之处：parseFloat()方法只允许接受十进制的表示

方法，而 parseInt()方法允许转换二进制、八进制和十六进制数。

对于八进制数，如果是最前面带有数字 0 的形式，会直接忽略 0 转换为普通十进制数。例如：

```
var x="010";
var result1=parseInt(x);          //默认为八进制数，返回值为 8
var result2=parseFloat(x);        //默认为十进制数，返回值为 10
```

对于十六进制数，如果出现字母，则直接按照字面的意思认为是无效的字符串。例如：

```
var x="A";
var result1=parseInt(x, 16);      //parseInt()允许十六进制数，返回值为 10
var result2=parseFloat(x);        //parseFloat()不允许十六进制数，返回值为 NaN
```

【例 3-25】 JavaScript 转换浮点数类型的简单应用

使用 parseFloat()方法将变量值转换为浮点数类型。

```
1.  <!DOCTYPE html>
2.  <html>
3.      <head>
4.          <meta charset="utf-8">
5.          <title>JavaScript 转换浮点数类型的简单应用</title>
6.      </head>
7.      <body>
8.          <h3>JavaScript 转换浮点数类型的简单应用</h3>
9.          <hr/>
10.         <p>
11.             不同内容的字符串转换浮点数的结果。
12.         </p>
13.         <script>
14.         //纯字母的情况
15.         var result1=parseFloat("hello");
16.         //多个小数点的情况
17.         var result2=parseFloat("12.12.13");
18.         //既有数字又包含字母的情况
19.         var result3=parseFloat("3.14PI");
20.         //输出结果
21.         alert("hello=> "+result1+"\n12.12.13=> "+result2+"\n3.14PI=> "+
            result3);
22.         </script>
23.     </body>
24. </html>
```

运行效果如图 3-26 所示。

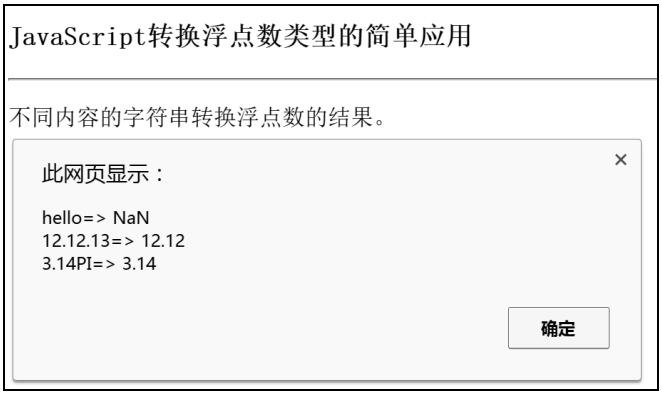

图 3-26　JavaScript 转换浮点数类型的简单应用效果

3.3.3　强制类型转换

一些特殊的值无法使用 toString()、parseInt()或 parseFloat()方法进行转换，例如 null、undefined 等，此时可以使用 JavaScript 中的强制转换（Type Casting）对其进行转换。

在 JavaScript 中有 3 种强制类型转换函数，解释如下。

- Boolean(value)：把指定的值强制转换为布尔值。
- Number(value)：把指定的值强制转换为数值（整数或浮点数）。
- String(value)：把指定的值强制转换为字符串。

1．Boolean()函数

JavaScript 中的所有其他类型都可以使用类型转换函数 Boolean()转换成布尔值，再进行后续计算。

当需要转换的值为非空字符串时，Boolean()函数的返回值为 true；当需要转换的值为空字符串时会返回 false。例如：

```
var result1=Boolean("hello");        //非空字符串的返回值为 true
var result2=Boolean("");             //空字符串的返回值为 false
```

当需要转换的值为数字时，整数 0 的返回值为 false，其余所有整数与浮点数的返回值为 true。例如：

```
var result1=Boolean(0);              //数字 0 的返回值为 false
var result2=Boolean(999);            //非 0 整数的返回值为 true
var result3=Boolean(3.14);           //浮点数的返回值为 true
```

当需要转换的值为 null 或 undefined 时，Boolean()函数的返回值均为 false。例如：

```
var result1=Boolean(null);           //返回值为 false
var result2=Boolean(undefined);      //返回值为 false
```

当需要转换的值本身就是布尔值时会转换成原本的值。例如：

```
var result1=Boolean(true);           //返回值为 true
var result2=Boolean(false);          //返回值为 false
```

2．Number()函数

在 JavaScript 中 Number()函数可以将任意类型的值强制转换为数字类型。当需要转换的内容为符合语法规范的整数或小数时，Number()将调用对应的 parseInt()或 parseFloat()方法进行转换。例如：

```
var x=Number("2");          //返回值为整数 2
var y=Number("2.9");        //返回值为浮点数 2.9
```

当需要转换的值为布尔值时，true 会转换为整数 1，false 会转换为整数 0。例如：

```
var x=Number(true);         //返回值为整数 1
var y=Number(false);        //返回值为整数 0
```

与直接使用 parseInt()和 parseFloat()方法进行数字类型转换不同的是，如果需要转换的值为数字后面跟随超过一个小数点或其他无效字符，Number()会返回 NaN。例如：

```
var x="2.12.13";
var result1=parseInt(x);    //返回值为整数 2
var result2=parseFloat(x);  //返回值为浮点数 2.12
var result3=Number(x);      //返回值为 NaN
```

当需要转换的值为 null 或 undefined 时，Number()函数分别返回 0 和 NaN。例如：

```
var x1=null;                //null 值
var x2;                     //undefined 值
var result1=Number(x1);     //返回整数 0
var result2=Number(x2);     //返回 NaN
```

当需要转换的值为其他自定义对象时，返回值均为 NaN。例如：

```
var student=new Object();
var result=Number(student); //返回 NaN
```

3．String()函数

在 JavaScript 中 String()函数可以将任意类型的值强制转换为字符串类型并保留字面内容，这与 toString()的转换方法类似。与 toString()方法的不同之处在于，String()函数还可以将 null、undefined 类型强制转换为字符串类型。例如：

```
var x=null;
var result1=String(x);      //返回值为字符串"null"
var result2=x.toString();   //发生错误，无返回值
```

3.4 JavaScript 运算符

3.4.1 赋值运算符

在 JavaScript 中，=运算符专门用来为变量赋值，因此也称为赋值运算符。在声明变量时可以使用赋值运算符对其进行初始化，例如：

```
var x1=9;              //为变量 x1 赋值整数 9
var x2="hello";        //为变量 x2 赋值字符串"hello"
```

用户也可以使用赋值运算符将已存在的变量值赋给新的变量，例如：

```
var x1=9;              //为变量 x1 赋值整数 9
var x2=x1;             //将变量 x1 的值赋给新声明的变量 x2
```

另外还可以使用赋值运算符为多个变量连续赋值，例如：

```
var x=y=z=99;          //此时变量 x、y、z 的赋值均为整数 99
```

赋值运算符的右边还可以接受表达式，例如：

```
var x=100+20;          //此时变量 x 将赋值为 120
```

这里使用了加法（+）运算符形成的表达式，在运行过程中会优先对表达式进行计算，然后再对变量 x 进行赋值。加法运算符属于算术运算符的一种，接下来将介绍常用的各类算术运算符。

3.4.2　算术运算符

在 JavaScript 中所有的基本算术均可以使用对应的算术运算符完成，包括加、减、乘、除、求余等。算术运算符的常见用法如表 3-13 所示。

表 3-13　算术运算符的常见用法

运算符	解　　释	示　　例	变量 result 的返回值
+	加号，将两端的数值相加求和	var x=3, y=2; var result = x + y;	5
–	减号，将两端的数值相减求差	var x=3, y=2; var result = x−y;	1
*	乘号，将两端的数值相乘求积	var x=3, y=2; var result = x * y;	6
/	除号，将两端的数值相除求商	var x=4, y=2; var result = x / y;	2
%	求余符号，将两端的数值相除求余数	var x=3, y=2; var result = x % y;	1

++	自增符号，数字自增 1	var x=3;
		x++;
		var result = x;
			4

--	自减符号，数字自减 1	var x=3;
		x--;
		var result = x;
			2

其中，加号还有一个特殊用法，即可用于连接文本内容或字符串变量。例如：

```
var s1="Hello";
var s2=" JavaScript";
var s3=s1+s2; //结果是 Hello JavaScript
```

如果将字符串和数字用加号相连，则会先将数字转换为字符串，再进行连接。例如：

```
var s="Hello";
var x=2016;
var result=s+x;//结果是 Hello2016
```

在上述代码中即使字符串本身也是数字内容，使用加号连接仍然不会进行数学运算。例如：

```
var s="2015";
var x=2016;
var result=s+x;//结果是 20152016，而不是两个数字相加的和
```

将赋值运算符（等号）和算术运算符（加、减、乘、除、求余数）结合使用可以达到简写的效果，具体用法如表 3-14 所示。

表 3-14　运算符组合一览表

运算符组合	格　式	解　释
+=	x += y	等同于 x = x + y
–=	x –= y	等同于 x = x – y
*=	x *= y	等同于 x = x * y
/=	x /= y	等同于 x = x / y
%=	x %= y	等同于 x = x % y

3.4.3　逻辑运算符

逻辑运算符有 3 种类型，即 NOT（逻辑非）、AND（逻辑与）和 OR（逻辑或）。逻辑运算符使用的符号与对应关系如表 3-15 所示。

表 3-15　逻辑运算符一览表

运 算 符	解　释
!	逻辑非，表示对布尔值结果再次反转。例如原先为 true，加上！符号后返回值就变为 false
&&	逻辑与，表示并列关系。注意，在&&符号前后的条件均为 true，返回值才为 true；只要有一个条件为 false，则返回值就为 false
‖	逻辑或，表示二选一的关系。在‖符号前后的条件只要有一个为 true，返回值就为 true；如果两个条件都为 false，则返回值才为 false

在进行逻辑运算之前，JavaScript 中自带的抽象操作 ToBoolean 会将运算条件转换为逻辑值，转换规则如表 3-16 所示。

表 3-16　ToBoolean 的转换规则

值	示　例	转 换 结 果
布尔值真（true）	var x = true;	维持原状，仍为 true
布尔值假（false）	var x = false;	维持原状，仍为 false
null	var x = null;	false
undefined	var x = undefined;	false

续表

值	示　例	转 换 结 果
非空字符串	var x = "Hello";	true
空字符串	var x = "";	false
数字 0	var x = 0;	false
NaN	var x = NaN;	false
其他数字（非 0 或 NaN）	var x = 99;	true
对象	var student = new Object();	true

1．逻辑非运算符

在 JavaScript 中，逻辑非运算符（NOT）与在 C 语言和 Java 语言中相同，使用感叹号（!）并放置在运算内容左边。逻辑非运算符的返回值只能是布尔值，即 true 或者 false。逻辑非的运算规则如表 3-17 所示。

表 3-17　逻辑非运算符的规则一览表

运算数类型	示　例	返　回　值
数字 0	var result = !0;	true
其他非 0 的数字	var result = !99;	false
对象	var student = new Object(); var result = !student;	false
空值 null	var x = null; var result = !x;	true
NaN	var x = NaN; var result = !x;	true
未赋值 undefined	var x; var result = !x;	true

2．逻辑与运算符

在 JavaScript 中，逻辑与运算符（AND）使用双和符号（&&）表示，用于连接符号前后的两个条件判断，表示并列关系。当两个条件均为布尔值时，逻辑与的运算结果也是布尔值（true 或者 false），判断结果如表 3-18 所示。

表 3-18　逻辑与（&&）的布尔值对照表

条件 1	条件 2	返　回　值
真（true）	真（true）	真（true）
真（true）	假（false）	假（false）
假（false）	真（true）	假（false）
假（false）	假（false）	假（false）

由表 3-18 可见，在条件 1 和条件 2 本身均为布尔值的前提下，只有当两个条件均为真时（true），逻辑与的返回值才为真（true），只要有一个条件为假（false），逻辑与的返回值就为假（false）。

另外还有一种特殊情况：当条件 1 为假（false）时，无论条件 2 是什么内容（例如 null 值、undefined、数字、对象等），最终返回值都是假（false）。原因是逻辑与有简便运算的

特性，即如果第一个条件为假（false），直接判断逻辑与的运行结果为假（false），不再执行第二个条件。例如：

```
var x1=false;
var result=x1&&x2;  //因为 x1 为 false，可以忽略 x2 直接判断最终结果
alert(result);      //该语句的执行结果为 false
```

由于条件 1 为 false，逻辑与会直接判定最终结果为 false，忽略条件 2。因此，即使本例中条件 2 的变量未声明也不影响代码的运行。

但是如果条件 1 为真（true），无法判断最终结果，此时仍然需要判断条件 2。例如上例中修改变量 x1 的值为真（true），代码如下。

```
var x1=true;
var result=x1&&x2;  //因为未声明变量 x2，所以执行时发生错误
alert(result);      //该语句不会被执行
```

此时由于逻辑与需要判断条件 2 的值，所以会发现变量 x2 从未被声明过，从而在执行时发生错误，导致后续语句不会被执行。

如果存在某个条件是数字类型，则先将其转换为布尔值再继续判断。其中，数字 0 对应的是假（false），其他非 0 数字对应的都是真（true）。例如：

```
var x1=0;           //对应的是 false
var x2=99;          //对应的是 true
var result=x1&&x2;  //结果是 false
```

逻辑与运算符的返回值不一定是布尔值，如果其中某个条件的返回值不是布尔值，有可能出现其他返回值。逻辑与的运算规则如表 3-19 所示。

表 3-19　逻辑与（&&）特殊情况规则一览表

运算数类型	示　例	返　回　值
一个是对象，一个是布尔值	var student = new Object(); var result = student&&true;	返回对象类型，即 student

两个都是对象	var student1 = new Object();
	var student2= new Object();
	var result = student1&&student2;
		返回第二个对象，即 student2

一个是空值 null，一个是布尔值	var x = null; var result = x&&true;	null
存在 NaN	var x = 100 / 0; var result = x&&true;	NaN
存在未赋值 undefined	var x; var result = x&&true;	undefined

注：以上所有情况均不包括条件 1 为假（false），因为此时无论条件 2 是什么内容，最终返回值都是假（false）。

3．逻辑或运算符

在 JavaScript 中，逻辑或运算符（OR）使用双竖线符号（||）表示，用于连接符号前后

的两个条件判断，表示二选一的关系。当两个条件均为布尔值时，逻辑或的运算结果也是布尔值（true 或者 false）。判断结果如表 3-20 所示。

表 3-20　逻辑或（||）的布尔值对照表

条件 1	条件 2	返　回　值
真（true）	真（true）	真（true）
真（true）	假（false）	真（true）
假（false）	真（true）	真（true）
假（false）	假（false）	假（false）

由表 3-20 可见，在条件 1 和条件 2 本身均为布尔值的前提下，只有当两个条件均为假（false）时，逻辑或的返回值才为假（false），只要有一个条件为真（true），逻辑或的返回值就为真（true）。

另外还有一种特殊情况：当条件 1 为真（true）时，无论条件 2 是什么内容（例如 null 值、undefined、数字、对象等），最终返回值都是真（true）。原因是逻辑或也具有简便运算的特性，即如果第一个条件为真（true），直接判断逻辑或的运行结果为真（true），不再执行第二个条件。例如：

```
var x1=true;
var result=x1||x2;  //因为 x1 为 true，可以忽略 x2 直接判断最终结果
alert(result);      //该语句的执行结果为 true
```

由于条件 1 为真（true），逻辑或会直接判定最终结果为真（true），忽略条件 2。因此，即使本例中条件 2 的变量未声明也不影响代码的运行。

但是如果条件 1 为假（false），无法判断最终结果，此时仍然需要判断条件 2。例如上例中修改变量 x1 的值为假（false），代码如下。

```
var x1=false;
var result=x1||x2;  //因为未声明变量 x2，所以执行时发生错误
alert(result);      //该语句不会被执行
```

此时由于逻辑或需要判断条件 2 的值，所以会发现变量 x2 从未被声明过，从而在执行时发生错误，导致后续语句不会被执行。

和逻辑与运算符类似，如果存在某个条件是数字类型，则先将其转换为布尔值再继续判断。其中，数字 0 对应的是假（false），其他非 0 数字对应的都是真（true）。例如：

```
var x1=0;           //对应的是 false
var x2=99;          //对应的是 true
var result=x1||x2;  //结果是 true
```

逻辑或运算符的返回值也不一定是布尔值，如果其中某个条件的返回值不是布尔值，有可能出现其他返回值。逻辑或的运算规则如表 3-21 所示。

表 3-21　逻辑或（||）特殊情况规则一览表

运算数类型	示　例	返　回　值
条件 1 为 false，条件 2 为对象	var student = new Object(); var result = false||student;	返回对象类型，即 student

两个都是对象	var student1 = new Object();
	var student2= new Object();
	var result = student1||student2;
		返回第一个对象，即 student1

条件 1 为 false，条件 2 为 null	var x = null; var result = false||x;	null
条件 1 为 false，条件 2 为 NaN	var x = 100 / 0; var result = false||x;	NaN
条件 1 为 false，条件 2 为 undefined	var x; var result = false||x;	undefined

注：以上所有情况均不考虑条件 1 为真（true），因为此时无论条件 2 是什么内容，根据逻辑或的简便运算特性，最终返回值都是真（true）。

3.4.4　关系运算符

在 JavaScript 中关系运算符共有 4 种，即大于（>）、小于（<）、大于等于（>=）和小于等于（<=），用于比较两个值的大小，返回值一定是布尔值（true 或 false）。

1．数字之间的比较

数字之间的比较完全依据数学中比大小的规律，当条件成立时返回真（true），否则返回假（false）。例如：

```
var result1=99>0;        //符合数学规律，返回 true
var result2=1<100;       //不符合数学规律，返回 false
```

此时只要两个运算数都是数字即可，整数或小数都可以依据此规律进行比较，并且返回对应的布尔值。

2．字符串之间的比较

当两个字符串比大小时是按照从左往右的顺序依次比较相同位置上的字符，如果字符完全一样，则继续比较下一个。

如果两个字符串在相同位置上都是数字，则仍然按照数学上的大小进行比较。例如：

```
var x1="9";
var x2="1";
var result=x1>x2;        //返回 true
```

此时从数学概念上来说 9 大于 1，因此返回值是真（true）。

如果两个数字的位数不一样，仍然只对相同位置上的数字比大小，不按照数学概念看整体数值大小。例如：

```
var x1="9";
var x2="10";
var result=x1>x2;        //返回 true
```

虽然从数学概念上来说 10 应该大于 9，但是由于字符串同位置比较原则，此时比较的是变量 x1 中的 9 和变量 x2 中的 1，得出结论 9 大于 1，因此返回值仍然是真（true）。

由于 JavaScript 是一种大小写敏感的程序语言，所以如果相同位置上的字符大小写不同可以直接作出判断，因为大写字母的代码小于小写字母的代码。例如：

```
var x1="hello";
var x2="HELLO";
var result=x1>x2;        //返回 true
```

在该例中按照从左往右的顺序先比较两个字符串的第一个字符，即变量 x1 中的 h 和变量 x2 中的 H。由于大写字母的代码小于小写字母的代码，因此返回值是真（true）。此时已判断出结果，所以不再继续比较后续的字符。

如果大小写相同，则按照字母表的顺序进行比较，字母越往后越大。例如：

```
var x1="hello";
var x2="world";
var result=x1>x2;        //返回 false
```

在上例中同样按照从左往右的顺序先比较两个字符串的第一个字符，即变量 x1 中的 h 和变量 x2 中的 w。按照字母表的顺序 h 在先、w 在后，所示返回值是假（false）。此时已判断出结果，所以不再继续比较后续的字符。

如果不希望两个字符串之间的比较受到大小写字母的干扰，而是无论大小写都按照字母表顺序比大小，可以将所有字母都转换为小写或大写的形式，再进行大小的比较。

使用 toLowerCase()方法可以将所有字母转换为小写形式，例如：

```
var x1="ball";
var x2="CAT";
var result1=x1>x2;       //返回 true
var result2=x1.toLowerCase()>x2.toLowerCase();      //返回 false
```

本例给出了变量 result1 作为参照，当未进行大小写转换时，由于大写字母小于小写字母，即使字母 c 在字母表更后的位置，也只能返回真（true）；使用了 toLowerCase()方法将字母全部转换为小写形式后，结果符合字母表顺序排序的要求，返回假（false）。

使用 toUpperCase()方法可以将所有字母转换为大写形式，例如：

```
var x1="ball";
var x2="CAT";
var result1=x1>x2;       //返回 true
var result2=x1.toUpperCase()>x2.toUpperCase();      //返回 false
```

本例使用了 toUpperCase()将所有字母转换为大写再进行比较，与之前使用 toLowerCase()方法将所有字母转换为小写的原理相同，这里不再赘述。

3．字符串与数字的比较

当字符串与数字比大小时，总是先将字符串强制转换为数字再进行比较。例如：

```
var x1="100";
var x2=99;
var result1=x1>x2;  //返回 true
```

如果字符串中包含字母或其他字符导致无法转换为数字，则直接返回假（false）。例如：

```
var x1="hello";
var x2=99;
var result1=x1>x2;  //返回 false
```

因为变量 x1 的字符串在强制转换为数字时会变成 NaN 类型，当用 NaN 类型与数字类型比大小时默认返回假（false）。无论中间的关系运算符是哪一种，所产生的结果都是一样的，即使修改本例中的最后一行代码为相反的含义（var result1=x1 < x2;），返回值仍然为假（false）。

3.4.5 相等性运算符

在 JavaScript 中相等性运算符共有 4 种，即等于（==）、非等于（!=）、全等于（===）和非全等于（!==），用于判断两个值是否相等，返回值一定是布尔值（true 或 false）。

1．等于和非等于运算符

在 JavaScript 中，判断两个数值是否相等用双等于符号（==），只有两个数值完全相等时返回真（true）；判断两个数值是否不相等使用感叹号加等于号（!=），在两个数值不一样的情况下返回真（true）。

在使用等于或非等于运算符进行比较时，如果两个值均为数字类型，则直接进行数学逻辑上的比较判断是否相等。例如：

```
var x1=100;
var x2=99;
alert (x1 == x2);        //返回 false
```

若需要进行比较的数据存在其他数据类型（例如字符串、布尔值等），要先将运算符前后的内容尝试转换为数字再进行比较判断，转换规则如表 3-22 所示。

表 3-22　数据类型转换规则表

数 据 类 型	示　　例	转 换 结 果
布尔值（真）	true	1
布尔值（假）	false	0
字符串（纯数字内容）	"99"	99
字符串（非纯数字内容）	"99hello123"	NaN
空值	null	null
未定义的值	undefined	undefined

注：在进行数字转换时，null、undefined 不可以进行转换，需保持原值不变，并且在判断时 null 与 undefined 被认为是相等的。

在进行了数据类型转换后仍然不是数字类型的特殊情况判断规则如表 3-23 所示。

表 3-23　相等性特殊情况规则一览表

运算数类型	示　例	返　回　值

其中一个为 null，另一个为 undefined	var x1=null;
	var x2;
	var result = (x1==x2);
		true

两个值均为 null	var x1 = null;
	var x2 = null;
	var result = (x1==x2);
		true

两个值均为 undefined	var x1; var x2; var result = (x1==x2);	true

其中一个为数字，另一个为 NaN	var x1 = 5;
	var x2 = parseInt("a");
	var result = (x1==x2);
		false

两个值均为 NaN	var x1 = parseInt("a");
	var x2 = parseInt("b");
	var result = (x1==x2);
		false

2．全等于和非全等于运算符

全等号由 3 个连续的等号组成（===），也用于判断两个数值是否相同，作用和双等号（==）类似，但是全等号更加严格，在执行判断前不进行任何类型转换，两个数值必须数据类型相同并且内容也相同才返回真（true）。例如：

```
var x1=100;
var x2="100";
var result1=(x1 == x2);          //返回 true
var result2=(x1 === x2);         //返回 false
```

本例中变量 x1 为数字类型，变量 x2 是字符串类型。虽然它们的内容都是 100，参照变量 result1 使用了普通双等号（==），返回结果为真（true）；但是变量 result2 使用全等号（===）会判定为假（false），因为它们数据类型不相同，数字不能等于字符串。所以全等号更为严谨，只要数据类型不同则直接判断为假（false），不再进行数字转换。

非全等号由感叹号和两个连续的等号组成（!==），用于判断两个数值是否不同。它有两种情况返回真（true），一是两个数值的数据类型不相同，二是两个数值虽然数据类型一样，但是内容不相同；其他情况均返回假（false）。继续使用上例中的变量 x1 和 x2 判断非全等，代码如下。

```
var x1=100;
var x2="100";
var result1=(x1 != x2);          //返回 false
var result2=(x1 !== x2);         //返回 true
```

此时参照变量 result1 使用的是普通非等于号（!=），因此会先把变量 x2 转换为数字再进行判断，得出两者相等的结论，最后返回结果为假（false）。变量 result2 使用的是非全

等于号（!==），因此首先判断两个数值的数据类型。由于变量 x1 是数字，变量 x2 是字符串，数据类型不相同，因此直接返回真（true），不再继续判断数据的内容。

3.4.6 条件运算符

JavaScript 中的条件运算符语法和 Java 语言相同，语法格式如下。

```
变量=布尔表达式条件 ? 结果 1 : 结果 2
```

该格式使用问号（?）标记前面的内容为条件表达式，返回值以布尔值的形式出现。问号后面是两种不同的选择结果，使用冒号（:）将其隔开，如果条件为真则把结果 1 赋给变量，否则把结果 2 赋给变量。

例如使用条件运算符进行数字大小比较，代码如下。

```
var x1=5;
var x2=9;
var result=(x1>x2)? x1: x2;
```

本例中变量 result 将被赋予变量 x1 和 x2 中的最大值。表达式判断 x1 是否大于 x2，如果为真则把 x1 值赋给 result，否则把 x2 值赋给 result。显然 x1>x2 的返回值是 false，因此变量 result 最终会被赋成 x2 的值，最终答案为 9。

3.5 本章小结

本章首先介绍了 JavaScript 的基本数据类型，包括 Undefined、Null、String、Number 和 Boolean 类型；其次介绍了 JavaScript 的 3 种对象类型，分别是本地对象、内置对象和宿主对象。然后讲解了 JavaScript 不同类型之间的转换方法，包括转换成字符串、数字和强制类型转换；最后在 JavaScript 运算符部分根据运算符的不同功能分别介绍了赋值运算符、算术运算符、逻辑运算符、关系运算符、相等性运算符以及条件运算符。

习题 3

1．JavaScript 的基本数据类型有哪些？

2．试举出 5 种水果的名称，并使用 Array 数组对象进行存储。

3．请使用 Date 对象获取今天的年月日。

4．已知“var msg = "Merry Christmas";”，请分别解答以下内容。

（1）试获取字符串长度。

（2）试获取字符串中的第 5 个字符。

（3）试分别使用 indexOf()和 exec()方法判断字符串中是否包含字母 a。

5．请分别说出下列内容中变量 x 的运算结果。

（1）var x = 9+9;

（2）var x = 9+"9";

（3）var x = "9"+"9";

6．请分别说出下列数据类型转换的结果。

（1）parseInt("100plus101")

（2）parseInt("010")

（3）parseInt("3.99")

（4）parseFloat("3.14.15.926")

（5）parseFloat("A",16)

7．请分别说出下列布尔表达式的返回值。

（1）("100" > "99") && ("100" > 99)

（2）("100" == 100) && ("100" === 100)

（3）(!0) && (!100)

（4）("hello" > "javascript") || ("hello" > "HELLO")

8．转义字符\n 的作用是什么？怎样使用转义字符输出双引号？