项目三

输出九九乘法表

本项目主要内容

- 条件控制语句
- 循环控制语句
- 九九乘法表的输出

和其他编程语言一样，JavaScript 也具有各种语句来进行流程上的判断。从本质上看，语句定义了 JavaScript 的主要语法，语句通常使用一个或多个关键字来完成给定任务。语句可以很简单，如退出函数；也可以比较复杂，如指定重复执行某个命令的次数。

任务一　条件控制语句

条件控制语句依照某种条件判断某代码段是否执行。在实际生活中，往往存在需要选择分支的情况，一般来说，像抛硬币之类的事件存在着正面和反面两个分支，像选择出行路径之类的事件往往存在多个分支，分支的不同将决定程序的不同行为表现。JavaScript 常用的条件控制语句有两种：

（1）if-else 语句，这种语句的作用场景为，只有两个分支的程序选择。

（2）switch 语句，这种语句的作用场景为，具有多个分支的程序选择。

一、if-else 语句

对程序流程进行条件判断时，最常用的语句就是 if-else 语句。

在详细介绍 if-else 语句的用法之前，我们需要了解，一个选择结构通常包括多个分支，这些分支代表着程序运行时在不同条件下表现出的不同行为。下面从一个简单的算法开始讨论选择结构。

假设一个程序需要输出一个数的绝对值，可以得到如下的操作步骤（流程图如图 3-1 所示）：

（1）从程序外输入整数 *X*。

（2）判断输入的整数 *X* 是否小于 0，若小于 0，则执行步骤（3），否则执行步骤（4）。

（3）返回–*X*。

（4）返回 *X*。

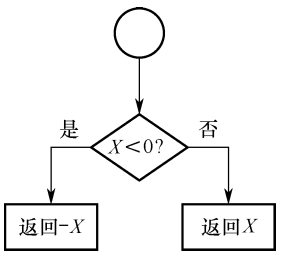

图 3-1　绝对值计算的流程图

图 3-1 表达的是上面例子中选择结构程序的流程图，在选择结构中，*X*<0（菱形部分）对应的是步骤（2），菱形的左右分别有是和否的两个分支，两个分支分别执行对应的两个操作步骤（是与否箭头指向的长方形部分），这两个步骤分别对应步骤（3）、步骤（4）。

由此可得，上面的判断框表示选择结构中的一种，即双分支选择结构，条件判断结果决定程序的走向。

由此可知，if-else 语句的语法如下：

```
if(condition) statement1 else statement2
```

其中的条件 condition 可以是任意的表达式，且该条件表达式的值不一定是布尔值。ECMAScript 会自动调用 Boolean()转换函数将这个表达式的结果转换为一个布尔值。if-else 语句的执行过程为：若对 condition 求值的结果是 true，则执行 statement1（语句 1）；若对 condition 求值的结果为 false，则执行 statement2（语句 2）。语句 1 和语句 2 既可以是一行代码，又可以是一个代码块（以一对花括号括起来的多行代码）。

if-else 语句的流程图如图 3-2 所示。

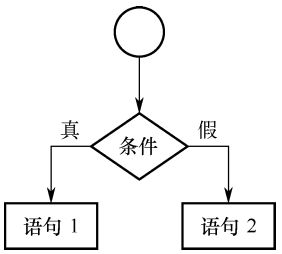

图 3-2　if-else 语句的流程图

【范例 3-1】 根据年龄显示不同的内容。

```
<body>
<script>
var age = 20;
if(age >= 18){//若 age >= 18 为 true，则执行 if 语句块
    alert('成年人');
}else{//否则执行 else 语句块
    alert('青少年');
}
</script>
</body>
```

在 Chrome 浏览器中的运行结果如图 3-3 所示。

图 3-3 根据年龄显示不同的内容

其中，else 子句是可选的，如果语句块中只包含一个语句，那么可以省略{}，修改范例 3-1 如下：

```
var age = 20;
if(age >= 18)
    alert('成年人');
else
    alert('青少年');
```

省略{}的危险之处在于，如果后来想添加一些语句，却忘了写{}，那么就改变了 if…else…的语义。例如，修改范例 3-1 如下：

```
var age = 20;
if(age >= 18)
    alert('成年人');
else
    console.log('年龄小于 18 岁'); //添加一行日志
    alert('青少年'); //这行语句已经不在 else 的控制范围内了
```

上述代码中的 else 子句实际上只负责执行“console.log（'年龄小于 18 岁'）;”，原有的“alert（'青少年'）;”已经不属于 else 子句的控制范围内了，无论条件判断的表达式结果是 true 还是 false，它都会执行。

相反地，有{}的语句就不会出错：

```
var age = 20;
if(age >= 18){
```

```
        alert('成年人');
    }else{
        console.log('年龄小于 18 岁');
        alert('青少年');
    }
```

所以，业界推崇的最佳写法是始终使用代码块，即使要执行的只有一行代码。因为这样可以消除人们的误解，否则可能让人分不清在不同条件下要执行哪些语句。

如果还需要进行更细致的判断，那么可以使用多个 if-else 语句的组合。可以把多个 if-else 语句写在一行代码中，例如：

```
    if(condition1) statement1 else if(condition2) statement2 else statement3
```

以上语句虽然不会有错误，但是我们推荐的写法则像下面这样：

```
    if(condition1){
        statement1
    }else if(condition2){
        statement2
    }else{
        statement3
    }
```

多个 if-else 语句的流程图如图 3-4 所示。

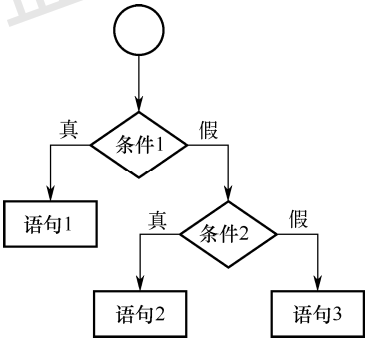

图 3-4　多个 if-else 语句的流程图

例如，我们再次修改范例 3-1，进行更细致的判断。

【范例 3-2】 根据年龄显示不同内容。

```
    <body>
    <script>
    var age = 3;
    if(age >= 18){
        alert('成年人');
    }else if(age >= 6){
```

```
        alert('青少年');
    }else{
        alert('儿童');
    }
    </script>
    </body>
```

在 Chrome 浏览器中的运行结果如图 3-5 所示。

图 3-5　根据年龄显示不同的内容

【范例 3-3】 在数学考试中，小明考了 86 分，给他做个评价，60 分以下为不及格，60（包含 60 分）～75 分为良好，75（包含 75 分）～85 分为很好，85（包含 85 分）～100 为优秀。

```
    <body>
    <script type = "text/javascript">
    var myscore = 86;
    if(myscore < 60){
        alert("成绩不合格，加油！");
    }else if(myscore >= 60 && myscore < 75){
        alert("成绩良好，不错呀");
    }else if(myscore> = 75 && myscore <85){
        alert("成绩很好，很棒");
    }else{
        alert("成绩优秀，超级棒");
    }
    </script>
    </body>
```

在 Chrome 浏览器中的运行结果如图 3-6 所示。

图 3-6　判断成绩

二、switch 语句

在日常生活中，针对回答条件是与否的问题，使用双分支结构就能够解决，但是在逻辑结构上，分支的形式不止双分支。在 JavaScript 中，可以通过 switch 关键字进行多分支的实现。

例如，监控用户的输入，若输入为大写 A、B、C，则替换为小写，否则直接返回。

用 if-else 语句也可以实现上述功能要求，但逻辑关系不容易表达清楚，所以，当有多种选择时，使用 switch 语句比 if-else 语句更方便。

switch 语句的优点为选择结构更加清晰，让人一目了然，并且执行速度相对较快。

switch 语句的语法如下：

```
switch(表达式){
    case 常量1:
        语句1;
        break;
    case 常量2:
        语句2;
        break;
    ……
    default:
        语句;
}
```


要注意以下几点：

（1）switch 括号内的“表达式”应该为整型或字符串型。

（2）switch 下面的花括号内是一个复合语句，意味着包含若干条语句，它是 switch 语句中的语句体。语句体内包括多个以关键字 case 开头的语句行和一个以 default 开头的语句行，case 后面跟着一个常量或常量表达式，在表达式后面需要跟一个冒号，如 case'A':，case 0:等。

（3）switch 语句执行时，先计算圆括号内表达的值，再将这个值与 case 后面的常量进行匹配，若匹配成功，则进入该 case 所表示的分支。

（4)若没有与任何 case 后面的常量相匹配，则执行 default 后面的语句。可以没有 default 及后面的语句，但此时若没有与任何 case 后面的常量相匹配，则不执行任何语句。

（5）每个 case 后面的常量必须互不相同，否则会出现矛盾的现象（同值不同的入口冲突）。

（6）case 只起标记作用，在执行 switch 语句时，根据 switch 表达式的值找到入口，在执行一个 case 语句后会顺序执行下去，直到遇到 break 跳出顺序执行。

（7）在 case 语句中，如果包含了一个以上的执行语句，可不必加花括号，程序执行时会顺序执行，加花括号也不会有影响。

（8）多个 case 语句可以公用一个执行语句，例如：

```
case  'A':
case  'B':
case  'C': b++;
```

【范例 3-4】 判断今天是星期几。

```
<body>
<script>
iwork = parseInt(prompt("请输入 0~6 范围内的值"));
switch(iwork){
      case 0:
         alert("星期天");
      break;
      case 1:
         alert("星期一");
         break;
      case 2:
         alert("星期二");
         break;
      case 3:
         alert("星期三");
         break;
      case 4:
         alert("星期四");
         break;
      case 5:
         alert("星期五");
         break;
      case 6:
         alert("星期六");
         break;
      default:
         alert("要输入合理值");
}
</script>
</body>
```

在 Chrome 浏览器中的运行结果如图 3-7、图 3-8 所示。

图 3-7　输入对应星期几的数字

图 3-8　判断是星期几

在 switch 语句中，几个 case 是可以公用一条指令的，相邻的 case 具有相同指令的，可以只写最后一条指令，然后写 break 结束选择结构。

【范例 3-5】 根据输入的月份判断季节。

```
<body>
<script>
var month = Number(prompt("请输入月份"));
var season = "";
switch (month){
    case 12:
    case 1:
    case 2:
        season = "冬";
        break;
    case 3:
    case 4:
    case 5:
        season = "春";
        break;
    case 6:
    case 7:
    case 8:
        season = "夏";
        break;
    case 9:
    case 10:
    case 11:
        season = "秋";
        break;
    default:
    season = "请输入正确的月份";
}
alert(season);
</script>
</body>
```

在 Chrome 浏览器中的运行结果如图 3-9、图 3-10 所示。

图 3-9　输入月份

图 3-10　根据月份判断季节

以上多个例子说明：

（1）由于每次执行 switch 语句时，并不是所有的 case 都能执行到，因此，应该避免使用带有副作用的 case，如函数调用表达式和赋值表达式，最安全的做法是在 case 中使用常量表达式。

（2）default 一般都出现在 switch 语句的末尾，位于所有 case 之后。当然这是最合理也是最常用的写法，实际上，default 可以放置在 switch 语句内的任何地方。

（3）在 switch 语句中，对每个 case 的匹配操作实际上是使用恒等运算符（===）比较，而不是使用相等运算符（==）比较，因此，表达式和 case 的匹配并不会做任何数据类型转换。

任务二　循环控制语句

对于程序设计初学者来说，循环结构在三大结构（顺序结构、选择结构、循环结构）中是最难掌握的。但是用来表示循环结构的语法并不难掌握，重点是要学会如何运用循环结构的设计思想来解决实际问题。

一、for 循环

for 循环要求在执行循环之前初始化变量和定义循环后要执行的代码。for 循环的语法如下：

```
for(初始状态; 判断条件; 状态改变){
    语句;
}
```

for 循环的执行顺序为：判断条件→执行语句→状态改变。因此，for 循环的执行过程可以用下面的过程来描述：

（1）计算“初始状态”。

（2）计算“判断条件”，“判断条件”若为真，则执行循环体中的“语句”，跳转到第（3）步；若为假，则循环结束，退出循环。

（3）计算“状态改变”。

（4）跳转到第（2）步执行。

for 循环的流程图如图 3-11 所示。

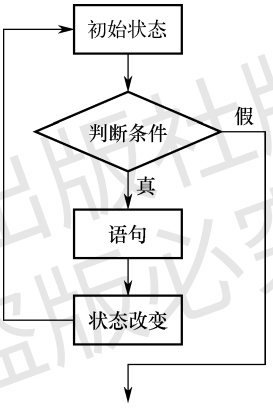

图 3-11　for 循环的流程图

例如：

```
var count = 10;
for(var i = 0; i < count; i++){
   alert(i);
}
```

以上代码定义了变量 i 的初始值为 0 。只有在判断条件表达式（i<count）返回 true 的情况下才会进行 for 循环，因此也有可能不会执行循环体中的代码。若执行了循环体中的代码，则一定会对循环后的表达式（i++）求值，即递增 i 的值。

i = 0 是初始状态，将变量 i 置为 0；i<count 是判断条件，满足就继续循环，不满足就退出循环；i++是每次循环后的状态改变，由于每次循环后变量 i 都会加 1，因此它终将在若干次循环后不满足判断条件 i<count，这时退出循环。

在 for 循环的变量初始化表达式中，也可以不使用 var 关键字。变量的初始化可以在外部执行。

例如：

```
var count = 10;
var i;
for(i = 0; i < count; i++){
    alert(i);
}
```

以上代码与在循环初始化表达式中声明变量的效果是一样的。由于 ECMAScript 中不存在块级作用域，因此在循环内部定义的变量也可以在外部访问到。

例如：

```
var count = 10;
for(var i = 0; i < count; i++){
    alert(i);
}
alert(i);//输出 10
```


在这个例子中，会有一个警告框显示循环完成后变量 i 的值，这个值是 10。这是因为，即使 i 是在循环内部定义的一个变量，但在循环外部仍然可以访问它。

由于 for 循环存在极大的灵活性，因此它是 ECMAScript 中常用的语句之一。

【范例 3-6】 累加求和，求 100 以内所有数相加的和。

```
<body>
<script>
var sum = 0;
for(var i = 1;i < 100;i++){
   sum += i;
}
alert(sum);
</script>
</body>
```

在 Chrome 浏览器中的运行结果如图 3-12 所示。

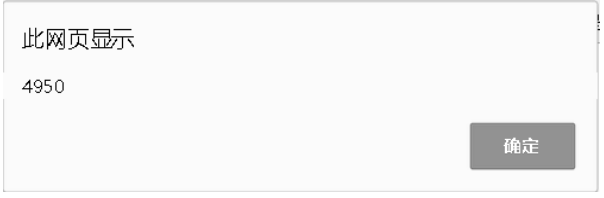

图 3-12　100 以内所有数相加的和

使用 for 循环时需要注意：

（1）for 循环中的“初始状态（循环变量赋初值）”“判断条件（循环条件）”和“状态改变（循环变量增量）”都是可选择项，全部可以省略，但“;”不能省略。

（2）省略“初始状态（循环变量赋初值）”，表示不对循环变量赋初值。

（3）省略“判断条件（循环条件）”，表示不对循环变量进行条件判断，注意这时可

能成为无限循环。

例如：

```
for(i = 1;;i++) sum = sum+i;
```

相当于：

```
i = 1;
while(1)
{ sum = sum + i;
  i++;}
```

（4）省略“状态改变（循环变量增量）”，表示不对循环变量进行修改，这时可在语句体中加入修改循环变量的语句。

例如：

```
for(i = 1;i <= 100;)
{ sum = sum + i;
  i++;}
```

（5）省略“初始状态（循环变量赋初值）”和“状态改变（循环变量增量）”。

例如：

```
for(;i <= 100;)
{ sum = sum + i;
  i++;}
```

相当于：

```
while(i <= 100)
{ sum = sum + i;
    i++;}
```

（6）三个表达式都省略。

例如：

```
for(;;)
```

相当于：

```
while(1)
```

（7）“初始状态”可以是设置循环变量初值的赋值表达式，也可以是其他表达式。

例如：

```
for(sum = 0;i <= 100;i++) sum = sum + i;
```

（8）“初始状态”和“状态改变”可以是一个简单表达式，也可以是逗号表达式。

（9）“判断条件”一般是关系表达式或逻辑表达式，但也可以是数值表达式或字符表达式，只要其值非零，就执行循环体。

下面介绍循环嵌套。for 循环嵌套就是一个 for 循环里面套着一个 for 循环，如我们都熟知的九九乘法表的输出，就可以用一个两重 for 循环来实现。外层的 for 循环控制行，内层的 for 循环控制列。格式如下：

```
for（;;）
{
        for（;;）
        {
        ……
        }
}
```

【范例 3-7】 输出九九乘法表。

```
<body style = "text-align:left">
<script>
for(var j = 1; j <= 9; j++){
      var str = '',s = '';
      for(var  i = 1;i <= j;i++){
             s = i + '*' + j + ' ';
          str += s;
      }
      document.write(str + '<br/>');
}
</script>
</body>
```

在 Chrome 浏览器中的运行结果如图 3-13 所示。

```
1*1
1*2 2*2
1*3 2*3 3*3
1*4 2*4 3*4 4*4
1*5 2*5 3*5 4*5 5*5
1*6 2*6 3*6 4*6 5*6 6*6
1*7 2*7 3*7 4*7 5*7 6*7 7*7
1*8 2*8 3*8 4*8 5*8 6*8 7*8 8*8
1*9 2*9 3*9 4*9 5*9 6*9 7*9 8*9 9*9
```

图 3-13　九九乘法表

二、while 循环

while 循环会在指定条件为真时循环执行代码块。while 循环属于前测试循环语句，也就是说，在循环体内的代码被执行之前，就会对出口条件求值。因此，循环体内的代码有可能永远不会被执行。while 循环的流程图如图 3-14 所示。while 循环的语法如下：

```
while(表达式)  语句
```

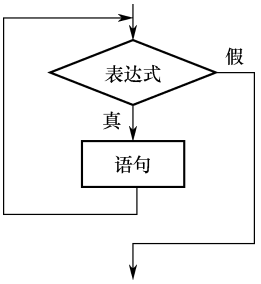

图 3-14　while 循环的流程图

while 循环的示例如下：

```
var i = 0;
while(i < 10){
  i += 2;
}
```

在这个例子中，变量 i 开始时的值为 0，每次循环都会递增 2。而只要 i 的值小于 10，循环就会继续下去。

使用 while 循环时，需要注意的问题如下：

（1）循环次数的控制要正确。使用循环结构设计程序，有时可以通过循环变量来控制循环次数。

（2）循环体包含一个以上的语句时，一定要用花括号括起来，否则，程序只将第一个语句作为循环体。

（3）在循环体内，要有使循环趋于结束的语句，否则，可能导致无限循环。

【范例 3-8】 累加求和，求 100 以内所有数相加的和。

首先分析此题的特点：

（1）此题有一个明显的特征，即重复执行加法动作，是将 100 个数进行累加的问题。那么我们就可以想到使用循环结构来解决本问题，循环执行加法运算，执行 100 次。

（2）既然知道了需要累加 100 次，那么接下来就要分析累加的数值有什么变化规律。通过观察，可以发现累加的数都有一个规律，就是后一个数是前一个数加 1。所以，我们可以在每一次循环的同时，对累加的数进行自加 1，就可以得到下一个数。

为了使思路可以更加清晰，画出该题具体实现的流程图，如图 3-15 所示。

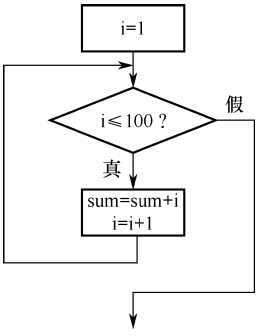

图 3-15　求 100 以内所有数相加的和的流程图

代码如下：

```
<body>
<script>
var sum = 0;
var i = 1;
while(i<100){
   sum += i;
   i++;
}
alert(sum);
</script>
</body>
```

在 Chrome 浏览器中的运行结果如图 3-16 所示。

图 3-16　100 以内所有数相加的结果

三、do-while 循环

do-while 循环是一种后测试循环语句，即只有在循环体中的代码执行之后，才会测试出口条件。换句话说，在对条件表达式求值之前，循环体内的代码至少会被执行一次。do-while 循环的流程图如图 3-17 所示。

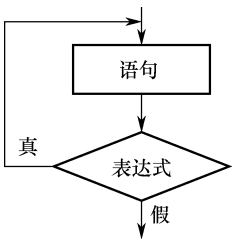

图 3-17　do-while 循环的流程图

do-while 循环的语法如下：

```
do{
    语句
}while(表达式);
```

do-while 循环的示例如下：

```
var i = 0;
do{
    i += 2;
}while(i < 10);
alert(i);
```

在这个例子中，只要变量 i 的值小于 10，循环就会一直继续下去。而且变量 i 的值最初为 0，每次循环都会递增 2。

为了避免误读，do-while 循环的循环体中即使只有一个语句，也要用花括号括起来；编写 do-while 循环时切勿忘记 while（表达式）后需要添加“;”。

注　意

do-while 循环与 while 循环的区别在于，do-while 循环先执行一次循环体，再进行表达式的判断，因此循环体中的语句至少要执行一次。在设计程序时，若不知道循环重复执行的次数，且第一次必须执行，则常采用 do-while 语句。

理解了 do-while 循环与 while 循环之间的区别之后，下面将范例 3-8 采用 do-while 循环的形式进行编写。

【范例 3-9】 累加求和，求 100 以内所有数相加的和。

程序实现的流程图如图 3-18 所示。

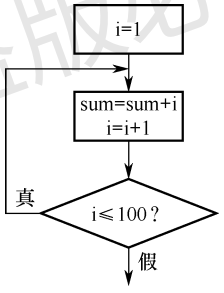

图 3-18　求 100 以内所有数相加的和的流程图（do-while 循环）

代码如下：

```
<body>
<script>
var sum = 0;
var i = 1;
do{
   sum += i;
   i++;
}while(i<100);
alert(sum);
</script>
</body>
```

在 Chrome 浏览器中的运行结果如图 3-19 所示。

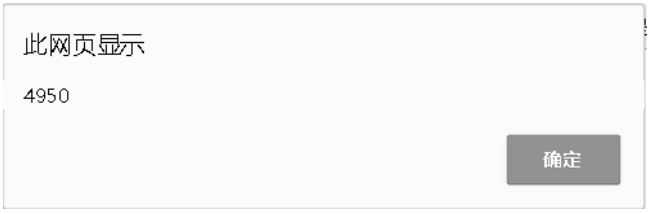

图 3-19　程序运行结果

四、break 和 continue 语句

break 和 continue 语句用于在循环中精确地控制代码的执行。其中，break 语句会使得循环立即退出，从而强制开始继续执行循环后面的语句。而 continue 语句虽然也会使循环立即退出，但退出循环后又会从循环的开始位置继续执行循环。

根据以上分析可知，continue 语句只结束本次循环，而不终止整个循环。break 语句则终止整个循环，不再执行循环体。

break 与 continue 语句的流程图如图 3-20 所示，通过流程图可以更加直观地看到两者之间的区别。

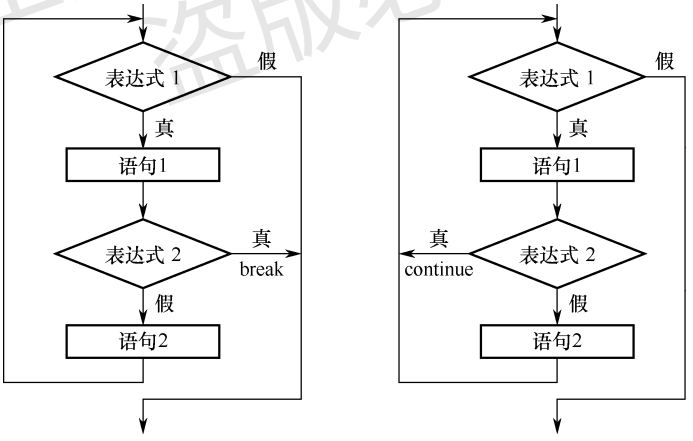

图 3-20　break 与 continue 语句的流程图

【范例 3-10】 从 1 循环到 10，当能把 5 除尽时跳出循环。

```
<body>
<script>
var num = 0;
for(var i = 1;i < 10;i++){
  if(i % 5 == 0){
    break;
  }
```

```
        num++;
    }
    alert(num);//结果为 4
    </script>
    </body>
```

在 Chrome 浏览器中的运行结果如图 3-21 所示。

图 3-21　跳出循环的结果

在 for 循环体内，有一个 if 语句来检查 i 的值是否可以被 5 整除（使用求模操作符）。若 i 不能被 5 整除，则执行 num 自加；若 i 能被 5 整除，则执行 break 语句退出循环。另外，变量 num 从 0 开始，用于记录循环执行的次数。在执行 break 语句之后直接跳出 for 循环，需要执行的下一行代码是 alert()，结果显示 4。也就是说，在变量 i 等于 5 时，循环共执行了 4 次；而 break 语句的执行，导致了循环在 num 再次递增之前就退出了。

如果在这里把 break 替换为 continue，那么会看到另外的结果。

【范例 3-11】 从 1 循环到 10，当能把 5 除尽时进入下一轮循环。

```
    <body>
    <script>
    var num = 0;
    for(var i = 1; i < 10; i++){
        if(i % 5 == 0){
            continue;
        }
        num++;
    }
    alert(num);//结果为 8
    </script>
    </body>
```

在 Chrome 浏览器中的运行结果如图 3-22 所示。

图 3-22　进入下一轮循环的结果

本例的结果显示为 8，也就是循环共执行了 8 次。当变量 i 等于 5 时，循环会在 num 再次递增之前退出，但接下来执行的是下一次循环，即 i 的值等于 6 的循环。于是，循环又继续执行，直到 i 等于 10 时自然结束。而 num 的最终值之所以是 8，是因为 continue 语句导致它少递增了一次。

break 和 continue 语句都可以与 label 语句联合使用，从而返回代码中特定的位置。这种联合使用的情况多发生在循环嵌套的情况下。

【范例 3-12】 continue 语句与 label 语句联合使用。

```
<body>

<script>
var num = 0;

labal:
for(var i = 0;i < 10;i++){
  for(var j = 0;j < 10;j++){
    if(i == 5 && j == 5) {
      continue labal;
    }
  num++;
  }
}
alert(num); //结果为 95

</script>
</body>
```


在这种情况下，continue 语句会强制继续执行循环——退出内部循环，执行外部循环。当 j 是 5 时，continue 语句执行，而这也就意味着内部循环少执行了 5 次，因此 num 的结果为 95。

虽然联合使用 break、continue 和 label 语句能够执行复杂的操作，但如果使用过度，也会给调试带来麻烦。在此，我们建议，如果使用 label 语句，一定要使用描述性的标签，同时不要嵌套过多的循环。

五、for-in 循环

for-in 循环是一种精准的迭代循环，主要用来枚举某一对象的属性。for-in 循环的语法如下：

```
for(property in expression) statement
```

比如我们需要显示 BOM 中 window 这一对象的所有属性，可以通过 fon-in 循环来实现：

```
for(var propName in window){
  document.write(propName);
}
```

在这个例子中，我们使用 for-in 循环来显示 BOM 中 window 对象的所有属性。每次执行循环时，都会将 window 对象中存在的某一个属性名赋值给变量 propName，这个循环过程会一直持续进行，直到 window 对象中的所有属性都被枚举输出了一遍为止。

与 for 循环类似，这里控制语句中的 var 操作符也不是必需的。但是，为了保证使用的是局部变量，比较推荐上面例子中的这种做法。由于 ECMAScript 对象的属性没有一定的顺序，因此，通过 for-in 循环输出的属性名的顺序是不可预测的。但是，window 对象的所有属性都会被返回一次，只是返回的先后次序可能会因浏览器的不同而不同。之前，如果表示要迭代的对象的变量值为 null 或 undefined，那么 for-in 循环会抛出错误。ECMAScript5 更正了这一行为，当变量值为 null 或 undefined 时不再抛出错误，只是不执行循环体。为了保证最大限度的兼容性，建议在使用 for-in 循环之前，先检测该对象的值不是 null 或 undefined。

【范例 3-13】 使用 for-in 循环访问数组元素。

```
<body>
<script>
var arr = {
    name: 'Jack',
    age: 20,
    city: 'Beijing'
};

var str = "";
for (var key in arr){
    str+ = key;
    str+ = " ";
}
alert(str);//'name', 'age', 'city'
</script>
</body>
```

在 Chrome 浏览器中的运行结果如图 3-23 所示。

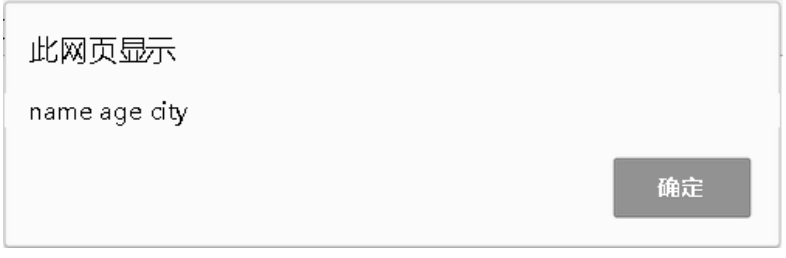

图 3-23　使用 for-in 循环访问数组元素

任务三　项目实施

一、任务目标

（1）掌握循环语句的使用。

（2）掌握输出 HTML 源代码的方法。

二、任务内容

在 JavaScript 的实际应用中，常需要将结果按一定的格式输出，如将数据以表格或列表的形式显示在页面上。若要以表格的形式输出九九乘法表，则要在范例 3-7 的基础上加入表格的 HTML 源代码，从而实现以表格的形式显示。

实现上述效果的关键在于，在两层循环的正确位置添加表格标签，以便正确输出表格。

三、操作步骤

（1）运行 Dreamweaver 软件，新建 HTML 标准网页文件。

（2）在网页<body></body>标签之间编写代码：

```
<script type = "text/javascript">
var str = "<table border = '1'>";
for(var i = 1;i <= 9;i++)
{
        str + = "<tr>";          //一行的开始
        for(var j = 1; j <= i; j++)
        {
            str += "<td>";       //单元格的开始
            str += j + "x" + i + " = " + i*j;
            str += "</td>";      //单元格的结束
        }
        str += "</tr>";          //一行的结束
}
str += "</table>";
document.write(str);
</script>
```

（3）运行结果如图 3-24 所示。

1x1=1								
1x2=2	2x2=4							
1x3=3	2x3=6	3x3=9						
1x4=4	2x4=8	3x4=12	4x4=16					
1x5=5	2x5=10	3x5=15	4x5=20	5x5=25				
1x6=6	2x6=12	3x6=18	4x6=24	5x6=30	6x6=36			
1x7=7	2x7=14	3x7=21	4x7=28	5x7=35	6x7=42	7x7=49		
1x8=8	2x8=16	3x8=24	4x8=32	5x8=40	6x8=48	7x8=56	8x8=64	
1x9=9	2x9=18	3x9=27	4x9=36	5x9=45	6x9=54	7x9=63	8x9=72	9x9=81

图 3-24　九九乘法表（表格形式）

四、拓展内容

修改代码，实现如图 3-25 所示的九九乘法表。

1x1=1								
1x2=2	2x2=4							
1x3=3	2x3=6	3x3=9						
1x4=4	2x4=8	3x4=12	4x4=16					
1x5=5	2x5=10	3x5=15	4x5=20	5x5=25				
1x6=6	2x6=12	3x6=18	4x6=24	5x6=30	6x6=36			
1x7=7	2x7=14	3x7=21	4x7=28	5x7=35	6x7=42	7x7=49		
1x8=8	2x8=16	3x8=24	4x8=32	5x8=40	6x8=48	7x8=56	8x8=64	
1x9=9	2x9=18	3x9=27	4x9=36	5x9=45	6x9=54	7x9=63	8x9=72	9x9=81

图 3-25　拓展内容结果