第4章 流程控制语句

所谓流程控制，主要就是用于控制整个程序的走向的。默认情况下，程序是从上往下，逐条执行，这种情况我们将其称之为顺序结构。但是并不是所有的程序都是使用顺序结构就能够完成的。在我们书写程序时往往还需要对整个程序进行分之，或者对某一段代码进行重复执行。所以，在这一章我们就一起来看一下JavaScript中的流程控制语句。

本章中我们将学习如下的内容：

- 条件语句
- 循环语句
- for-of循环
- 使用迭代方法进行遍历

4-1 条件语句

条件语句，主要就是给定一个判断条件，并在程序执行过程中判断该条件是否成立，然后根据判断结果来执行不同的操作，从而改变代码的执行顺序，实现更多的功能。一般来讲，条件语句可以分为3种：单分支语句，双分支语句以及多分支语句，我们一个一个来看。

4-1-1 单分支语句

单分支语句，由一个if组成，如果条件成立，则进入代码块开始执行语句，单分支语句的语法如下：

```
if(条件)
{
    //条件为真时执行的代码
}
```

示例：

```
let age = 20;
if(age >= 18)
{
    console.log("你已经是成年人了");
}
```

如果条件不是一个布尔值，那么会被自动转换为布尔值

```
if(1)
{
    console.log("Hello");
}
```

if语句里面的花括弧不是必须的，当只有一条执行语句的时候，可以不使用花括弧，但是不推荐这么做

```
if(true)console.log("Hello");
```

4-1-2 双分支语句

顾名思义，就是有两个分支线，如果if条件不成立，那么就会跳入到else语句里面，语法如下：

```
if(条件)
{
    //条件为真时要执行的代码
}
else{
    //条件为假时要执行的代码
}
```

示例：

```
let age = 16;
if(age >= 18)
{
    console.log("你已经成年了");
}
else{
    console.log("你还是未成年");
}
```

三目运算符

if...else语句的一个缩写方式，就是使用三目运算符 `?:`
语法：条件?(条件为真时执行的代码):(条件为假时执行的代码)
示例：

```
let age = 16;
let result = age>=18 ? "你已经成年了" : "你还是未成年";
console.log(result);//你还是未成年
```

练习：使用三目运算符来找出3个数中最大的那个数

```
let a = 3,b = 8,c = 5;
let max = a>b ? (a>c?a:c) : (b>c?b:c);
console.log(max);//8
```

4-1-3 多分支语句

多个if...else语句可以组合在一起，形成逻辑决策树

```
if(条件)
{
    //执行语句
}
else if(条件)
{
    //执行语句
}
else{
    //执行语句
}
```

需要注意的就是在多分支语句里面，如果进入到了某一个语句块，那么后面的条件就不会再进行判断，而是会直接跳出，多分支语句示例如下：

```
let readline = require("readline-sync");
console.log("请输入你的成绩:");
let score = readline.question("");
if(score > 100 || score < 0)
{
    console.log("成绩输入有误");
}
else if(score >= 90)
{
    console.log("优秀");
}
else if(score >= 70)
{
```

```
    console.log("良好");
}
else if(score >= 60)
{
    console.log("合格");
}
else{
    console.log("不合格");
}
```

switch语句

有一种比多分支语句结构更加清晰的语句结构，就是使用switch语句
语法如下：

```
switch(条件)
{
    case 1:
        //执行语句;
        break;
    case 2:
        //执行语句;
        break;
    case 3:
        //执行语句;
        break;
    default:
        //执行语句;
}
```

示例：

```
let num = 2;
switch(num)
{
    case 1:
        console.log("the number is one");
        break;
    case 2:
        console.log("the number is two");
        break;
    default:
        console.log("neither one nor two");
}
//the number is two
```

break关键字

`break`用于跳出某一个case，如果不书写`break`的话，进入case以后会继续进入后面的case语句。

`break`往往存在一个误区，就是觉得这个关键字是必须要写的，但是并不是必须要写。有些时候我们故意不写`break`关键字反而能给我们带来便利，例如下面的例子：

```
let day = 4;
switch(day)
{
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        console.log("今天是工作日");
        break;
    case 6:
    case 7:
        console.log("今天是周末");
    default:
        console.log("输入有误");
}
```

这里我们希望程序在变量day值为1-5时输出"今天是工作日"，在值为6和7的时候输出"今天是周末"。这个时候就可以通过省略`break`关键字来节省我们的代码量。

default关键字

`default`用于书写默认的条件，如果前面都不满足的话，就进入到`default`语句里面。

`default`往往也会存在误区，很多人认为`default`也是必须要写的。事实上`default`也是可以省略的，我们书写`default`，只是为了对`switch`没有进入`case`语句的情况做一种默认操作。

在JavaScript里面的`switch`语句中，和其他静态语言有所不同的是，case后面的值不一定必须是常量，可以是变量甚至是表达式，如下，只要和入口条件匹配即可

```
switch(false)
{
    case 2 < 3:
        console.log("进入了第一个入口");
        break;
    case 6 < 3:
        console.log("进入了第二个入口");
        break;
```

```
    default:
        console.log("进入了默认入口");
}
//进入了第二个入口
```

4-2 循环语句

循环语句，也是流程控制语句中不可或缺的一种结构。在JavaScript中实现循环的方式有好几种，我们一个一个来看。

4-2-1 为什么需要循环

在具体介绍JavaScript中的循环之前，首先我们来明确一个问题，那就是为什么需要循环。这里举一个简单的例子，例如我们要设计一个程序，计算从1加到10的值，或许你会写出：

```
let i = 1+2+3+4+5+6+7+8+9+10;
console.log(i);
```

但是如果是从1加到100呢？或者是从1加到1000呢？显然这种方法就力不从心了。这个时候，就需要循环语句登场了。这里我们介绍JavaScript中的3种循环结构，while循环，do..while循环以及for循环。

4-2-2 while循环

while循环是循环里面比较常见的一种循环，它的语法如下：

```
while(表达式)
{
    //循环体
}
```

含义：如果表达式成立，执行循环体，否则结束循环
while循环示例：从1加到100

```
let i = 1,sum = 0;
while(i <= 100)
{
    sum += i;
    i++;
}
console.log(sum);//5050
```

无穷循环

所谓无穷循环，被被称之为死循环。这是在我们无法退出循环时会遇到的一种情况。一般来讲，我们在设计循环的时候需要为循环设计一个出口，这样才能在当不满足循环条件的时候退出循环。如果没有正确的对循环设计一个出口，那么循环语句将无法退出，陷入到无穷循环，也就是循环里面。所以我们在设计循环的时候，一定要注意正确的书写循环条件，以便循环能在执行一定次数以后退出。

4-2-3 do..while循环

首先执行一次循环体，然后检测循环条件表达式的值是否为"真"，如果是真的话，则重复执行循环语句
do...while循环和前面两种循环语句略有区别，区别在于循环至少会执行一次。
它的一般形式为：

```
do{
    //执行语句;
}while(表达式);
```

举例：用do...while改写上面的程序

```
let i = 1,sum = 0;
do{
    sum += i;
    i++;
}while(i <= 100);
console.log(sum);//5050
```

4-2-4 for循环

for循环是所有循环里面最常见的一种循环。
for循环的一般形式为：

```
for(表达式1;表达式2;表达式3)
{
    //循环体
}
```

它执行的顺序为：首先执行表达式1，然后进行表达式2的判断，如果表达式2成立，那么执行循环体，循环体执行结束后，进行表达式3，然后回头再看表达式2是否成立，成立就执行循环体，不成立该循环就结束。

for循环举例：用for循环实现从1加到100

```
let sum = 0;
for(let i=1;i<=100;i++)
{
    sum += i;
}
console.log(sum);//5050
```

在for循环里面，每一个部分都是可选的，例如上面的代码，我们可以做如下的修改：

```
let i = 1,sum = 0;
for(;i<=100;)
{
    sum += i;
    i++;
}
console.log(sum);//5050
```

遍历数组

最早的时候，在JS里面就经常使用for循环来对数组进行遍历。示例如下：

```
let arr = [1,2,3,4,5];
for(let i=0;i<arr.length;i++)
{
    console.log(arr[i]);
}
```

这里有一个小技巧，就是在使用for循环来遍历数组的时候，使用一个变量来存储数组的长度，这样可以大大提升程序的效率，因为不用每次都去重新计算数组的长度

```
let arr = [1,2,3,4,5];
for(let i=0,j=arr.length;i<j;i++)
{
    console.log(arr[i]);
}
```

4-2-5 循环的嵌套

我们可以把一个循环放在另外一个循环里面，从而构成循环的嵌套。里面的循环称之为内层循

环，外面的循环称之为外层循环。外层循环每循环一次，内层循环就把自己的全部循环执行完。

循环嵌套示例：打印九九乘法表

```
let str = "";
for(let i=1;i<=9;i++)
{
    for(let j=1;j<=i;j++)
    {
        str += i + "*" + j + "=" + (i*j) + " ";
    }
    console.log(str);
    str = "";
}
// 1*1=1
// 2*1=2 2*2=4
// 3*1=3 3*2=6 3*3=9
// 4*1=4 4*2=8 4*3=12 4*4=16
// 5*1=5 5*2=10 5*3=15 5*4=20 5*5=25
// 6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
// 7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
// 8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
// 9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

上面有提到使用for循环来遍历数组。一般来讲，遍历一维数组就用一个for循环就够了，而如果要遍历一个二维数组，就需要使用到for循环的嵌套，也就是双层for循环。而如果是三维数组，则需要使用到三层for循环，以此类推。

使用双层for循环遍历一个二维数组

```
let arr = [
    ["Bill",20],
    ["Lucy",23],
    ["David",25],
    ["Lily",18]
];
for(let i=0;i<arr.length;i++)
{
    for(let j=0;j<arr[i].length;j++)
    {
        console.log(arr[i][j]);
    }
}
// Bill
// 20
```

```
// Lucy
// 23
// David
// 25
// Lily
// 18
```

4-2-6 break 和 continue

break

break语句我们在前面讲switch语句的时候已经见到过了，当时有讲到它的作用是跳出switch语句或者循环语句。有一点需要注意的是：break语句不能直接作用于if语句，除非这个if语句是被嵌套在循环语句里面的。

```
//错误
if(条件)
{
    //执行语句
    break;
}
```

```
//正确
for(表达式1;表达式2;表达式3)
{
    if(条件)
    {
        //执行语句
        break;
    }
}
```

continue语句

结束本次循环(不是终止整个循环)，即跳过循环体中continue语句后面的语句，开始下一次循环。
break语句和continue语句的区别：continue语句的意思是结束本次循环，break语句是结束整个循环。具体示例如下：

break语句：

```
for(1;2;3)
{
    a;
    b;
    break;//执行a,b后循环结束
    c;
    d;
}
```

continue语句：

```
for(1;2;3)
{
    a;
    b;
    continue;//执行a,b以后结束本次循环
    c;
    d;
}
```

break和continue在默认情况下，是终止掉离它最近的循环。特殊情况下，我们可以通过给每个循环取一个名字，然后通过名字来终止指定循环，示例如下：

```
// 给数组去重复
let arr = [1, 2, 2, 2, 3, 4, 5, 5, 6, 6, 7, 8];
let newArr = [];
outer: for (let i = 0; i < arr.length; i++)
{
    for (let j = 0; j < newArr.length; j++)
    {
        if (arr[i] === newArr[j])
        {
            continue outer; // 终止名字为outer的循环
        }
    }
    newArr.push(arr[i]);
}
console.log(newArr);
```

在上面的代码中，我们要给arr这个数组去除重复，除了前面所介绍过的使用set集合来快速去除以外，在ES6之前，数组去重都需要开发人员自己来书写方法。这里我们就利用的是一种最简单的去重方式，用旧数组和新数组进行比较，如果新数组里面不含有旧数组里面的元素，则将此元素放入新的数组。当我们比较的时候，发现新数组里面已经存在此元素时，就直接continue结束

外层的循环，进入外层循环的下一次循环。

4-3 for-of 语句

4-3-1 遍历数组

前面我们介绍了使用`for`循环来进行数组的遍历。从ES6开始，引入了一个改进的迭代器功能，称之为`for-of`循环。我们可以使用`for-of`可以很轻松的实现数组的遍历，示例如下：

```
let arr = [1,2,3,4,5];
for(let i of arr)
{
    console.log(i);
}
// 1
// 2
// 3
// 4
// 5
```

可以看到，这种语法更加的简洁方便。用一个变量来表示数组中每个元素。

除了上面介绍的for-of以外，ES6还新增加了for-in来对数组进行迭代。不过和for-of不同的是，for-in迭代的是数组的键

```
let arr = [1,2,3,4,5];
for(let i in arr)
{
    console.log(i);
}
// 0
// 1
// 2
// 3
// 4
```

除了上面介绍的`for-of`以及`for-in`以外，ES6中还专门提供了3个用于迭代可迭代元素的方法，分别是`keys()`，`values()`以及`entries()`方法。其中keys()是找到可迭代元素的键，values()是找到可迭代元素的值，entries()是同时找到可迭代元素的键和值。

在后面的章节中我们会对迭代器进行更加详细的介绍(我们会在第12章学习JS里面的迭代器)，这里先作为了解，掌握其用法即可。

> 注意：数组里面无法使用`values()`方法

示例：使用`keys()`遍历出数组的键

```
let arr = [3,5,8,1];
for(let i of arr.keys())
{
    console.log(i);
}
// 0
// 1
// 2
// 3
```

示例：使用`entries()`方法遍历出数组的键和值

```
let arr = [3,5,8,1];
for(let i of arr.entries())
{
    console.log(i);
}
// [ 0, 3 ]
// [ 1, 5 ]
// [ 2, 8 ]
// [ 3, 1 ]
```

4-3-2 遍历集合

集合也是可以枚举的，我们同样可以使用for-of来对集合进行遍历，如下

```
let s = new Set([1,2,3,4,5]);
for(let i of s)
{
    console.log(i);
}
// 1
// 2
// 3
// 4
// 5
```

除此之外，我们也可以使用集合里面自带的keys()，values()以及entries()方法来对集合进行遍历。顺便要说一下的是，在集合里面键和值是相同的。

`keys()` 方法遍历集合的键

```
let s = new Set(["Bill","Lucy","David"]);
for(let i of s.keys())
{
    console.log(i);
}
// Bill
// Lucy
// David
```

`values()` 方法遍历集合的值

```
let s = new Set(["Bill","Lucy","David"]);
for(let i of s.values())
{
    console.log(i);
}
// Bill
// Lucy
// David
```

`entries()` 方法同时遍历集合的键与值

```
let s = new Set(["Bill","Lucy","David"]);
for(let i of s.entries())
{
    console.log(i);
}
// [ 'Bill', 'Bill' ]
// [ 'Lucy', 'Lucy' ]
// [ 'David', 'David' ]
```

4-3-3 遍历映射

与集合一样，映射也是可以枚举的，所以可以用与集合类似的方式进行遍历。

使用for-of来遍历映射

```
let m = new Map([["name","xiejie"],["age",18]]);
for(let i of m)
```

```
{
    console.log(i);
}
// [ 'name', 'xiejie' ]
// [ 'age', 18 ]
```

`keys()` 方法遍历映射的键

```
let m = new Map([["name","xiejie"],["age",18]]);
for(let i of m.keys())
{
    console.log(i);
}
// name
// age
```

`values()` 方法遍历映射的值

```
let m = new Map([["name","xiejie"],["age",18]]);
for(let i of m.values())
{
    console.log(i);
}
// xiejie
// 18
```

`entries()` 方法同时遍历映射的键与值

```
let m = new Map([["name","xiejie"],["age",18]]);
for(let i of m.entries())
{
    console.log(i);
}
// [ 'name', 'xiejie' ]
// [ 'age', 18 ]
```

总结

1. 所谓流程控制，主要就是用于控制整个程序的走向的。
2. 条件语句可以分为单分支语句，双分支语句以及多分支语句。
3. 三目运算符可以看作是双分支语句的一个 缩写形式。

4. switch语句是一种结构更清晰的多分支语句。涉及到的关键字有case，break和default。
5. 常见的循环结构有while循环，do...while循环和for循环。
6. 循环中常见的关键字有break和continue，需要注意两者之间的区别。
7. ES6引入了新的遍历数组的方式for-of，可以遍历数组的每一个元素。还可以使用for-in来遍历数组的键。
8. ES6还提供了3个迭代器keys()，values()以及entries()来遍历可迭代的元素。