day03 循环控制语句

程序控制语句

- 条件控制语句 （根据对应的条件执行对应的代码片段）
- 循环控制语句 （根据循环条件循环执行多次对应的代码）

循环控制语句的流程

- 定义**初始值**
- 设置**迭代条件**（初始值的基础上迭代）
- 执行的代码 **循环体**
- 设置**终止条件** （布尔类型表达式 返回的是boolean的值（强制转换为boolean类型））

常用的循环控制语句

- while
- do while
- for
- ...

while循环

```
//外部定义初始值
var 初始值变量 = 值
while(终止条件（当他为false的时候就会结束对应的循环）){
    //执行的代码 循环体
    //迭代条件
}
```

示例

打印1-100的值

```
var i = 1
while(i<=100){
    console.log(i)
    i++
}
```

求和 1-100的求和

```
var i = 1
var sum = 0
while(i<=100){
    console.log(i)
    sum += i
    i++
}
console.log(sum)
```

- var关键词修饰的变量进行变量提升 伪全局变量
- var关键词修饰的变量会被预编译 但是不会读取赋值操作 而是强制赋值undefined
- 在循环中不建议声明变量 会覆盖之前的值 会加大空间复杂度

练习

求1-100的阶乘

求1-100之间5的倍数的阶乘

打印100-999之间的水仙花数

while循环嵌套

```
while(条件){
 while(条件){
    循环体
 }
}
```

- 循环嵌套时间复杂度会增加（不建议嵌套超过俩层）
- 循环嵌套执行的次数为外层循环次数*内层循环次数

示例

打印乘法口诀表 （外层控制行 内层控制列）

```
/*
乘法口诀表为9行9列 1行为1列 2行为2列 .... 9行为9列
*/
var row = 1,col //声明行和列
while (row <= 9) { //打印行
    col = 1 //列值返回初始值 为了下一行
    while (col <= row) { //列最大为行数
        document.write(row+'*'+col+'='+row*col+' ')//在文档上书写内容
        col++
    }
    document.write('<br/>') //换行
    row++
}
```

打印直角三角型

```
/*
*
***
*****
*******
*********
*/
var row = 0,col
while(row<5){
    col = 0
    //处理列 列和行的关系 行数*2+1
    while(col < row*2+1){
        document.write('*')
        col++
```

```
    }
    document.write('<br/>') //换行
    row++
}
```

do while循环

```
var 初始值变量 = 初始值
do{
    循环体
}while(条件)
```

do while和while的区别

- while 最少执行0次 do while 最少执行一次
- do while常用于 先执行对应的内容再判断是否循环（人机交互）

示例

打印1-100

```
var i = 1
do{
    console.log(i)
    i++
}while(i<=100)
```

练习

do while 打印1-100的3的倍数

计算1-100内容 3的倍数及7的倍数的和

叫号系统

```
/*
输入的编号来进行叫号
输入1 输出一号技师
输入2 输出金牌技师
输入3 输出88号技师
输入其他重新输入
*/
var code
do {
    code = prompt('请输入你需要叫的编号')
    switch (code * 1) {
        case 1:
            console.log('一号技师为您服务')
            break
        case 2:
            console.log('金牌技师为您服务')
            break
        case 3:
            console.log('88技师为您服务')
            break
    }
} while (code != 1  &&  code != 2 && code != 3)
```

do while也可以嵌套

```
/*
打印四行五列的矩形
*/
var row = 0,
    col, str = ''
do {
    col = 0
    do {
        str += '*'
        col++
    } while (col < 5)
        str += '\n'//换行
    row++
} while (row < 4)
    console.log(str)
```

for循环

任何循环之间可以互相转换 while循环适用于你不知道执行次数的时候 for循环适用于知道执行次数

```
for(初始值;迭代条件;迭代量){
    执行的代码 循环体
}
```

示例

打印1-100之间的数

```
for(var i=0;i<100;i++){
    console.log(i+1)
}
```

打印1-100之间能整除2和3的数

```
for(var i=1;i<=100;i++){
    if(i%2==0 && i%3==0){
        console.log(i)
    }
}
```

打印1-100之间的偶数和

```
var sum = 0
for(var i=1;i<=100;i++){
    if(i%2==0){
        sum+=i
    }
}
console.log(sum)
```

for循环嵌套

任意循环可以互相嵌套

```
for(初始值;迭代条件;迭代量){
    for(初始值;迭代条件;迭代量){
        //循环体
    }
}
```

示例

打印直角三角型

```
/*
*
***
*****
*******
*********
*/
for(var i=0;i<5;i++){
    for(var j=0;j<2*i+1;j++){
        document.write('*')
    }
    document.write('<br/>')
}
```

练习

打印等腰三角形

```
/*
    *
   ***
  *****
 *******
*********
*/
for(var i=0;i<5;i++){
    var line = ''
    //打印空格
    for(var j=0;j<4-i;j++){
        line += ' '
    }
    //打印*
    for(var j=0;j<2*i+1;j++){
        line += '*'
    }
    console.log(line)
}
```

找到1-100之间所有的素数 （只能被自身和1整除）

```
for(var i=1;i<=100;i++){
    //从2数到自身-1
    var count = 0 //计算因数的个数
    for(var j=2;j<i;j++){
        //用i除以j
        if(i%j==0){ //是有因数
            count++
        }
    }
    if(!count){ //判断因数是否还是0个 如果是0表示当前为素数
        console.log(i)
    }
}
```

打印乘法口诀表

```
for(var i=1;i<=9;i++){
    var line = ''
    for(var j=1;j<=i;j++){
        line+=i+'*'+j+'='+i*j+'\t'
    }
    console.log(line)
}
```

打印 1-100所有的偶数每行三个

```
var line = ''
var count = 0
for (var i = 1; i <= 100; i++) {
    if (i % 2 == 0) {
        count++
        line += (i+'\t')
        //满足了3个条件
        if (count==3) {
            //到达三个加上换行符
            line+='\n'
            //重新计数
            count = 0
        }
    }
}
console.log(line)
```

循环总结

- 循环是用于反复多次执行一段代码
- 循环的三种方案可以互相嵌套 以及三种方法可以随意转换 常用的为for和while
- while循环用于处理不知道对应的执行次数 for循环常用于知道对应的执行次数
- 循环要避免对应的死循环（死循环就是循环的迭代条件一直为true 没有办法停止）
- while死循环写法 for死循环写法

```
while(true){ //死循环

}
for(;;){ //死循环

}
```

- while循环的时间复杂度低于for循环 for循环的执行效率要低于while循环
- do while循环先做后判断 最少执行一次 while及for最少执行0次

用于循环中的关键词

break 跳出当前循环（跳出switch块）

示例

查找1-100内的素数

```
for(var i=1;i<=100;i++){
    var count = 0
    //再进行对应的整除 用i整除 2-i-1的值 只要有能整除的那么这个数就是不是素数 没有就是素数
    for(var j=2;j<i;j++){
        if(i%j==0){
            count++
            break
        }
    }
    if(count!=1){
        console.log(i)
    }
}
```

continue 跳过本次循环 进入下一次

示例

1-100 逢7过 里面带7和7的倍数跳过

```
for(var i=1;i<100;i++){
    if(i%7==0 || parseInt(i/10)==7 ||parseInt(i%10)==7){
        continue
    }
    console.log(i)
}
```

扩展内容

时间复杂度

时间复杂度讲的是恒定机器中的执行次数和对应的变量的比例 也就是说在恒定机器内执行次数越多 那么时间复杂度越高，那么对应的时间复杂度越高 他的执行效率就越低。将时间复杂度降低那么就可以提高对应的效率。

时间复杂度（用O来表示）跟对应的执行次数成正比

O1 常数阶 每行代码执行一次

```
console.log('123')//1次
console.log('123')//1次
console.log('123')//1次
console.log('123')//1次
```

On 线性阶 循环执行多次由一个n变量来控制

```
for(var i=0;i<n;i++){
    console.log('123')
}
```

Ologn 对数阶 由俩个变量来控制的 (递归)

```
var i = 2
while(i<n){
    i*=k
}
```

Onlogn 线性对数阶 线性阶包含对数阶

```
for(var i=0;i<n;i++){
    var j = 2
    while(j<n){
        j*=k
    }
}
```

On2 平方阶 俩个线性阶包含

```
for(var i=0;i<n;i++){
    for(var j=0;j<n;j++){
        console.log('123')
    }
}
```

On3立方阶 3个线性阶

Onk次方 k个线性阶

...

时间复杂度排序

O1<Ologn<On<Onlogn<On^2<On^3<On^k....

空间复杂度

空间复杂度讲的是在内存开辟上 有多个变量内存被同时开辟，开辟的内存越多对应的空间复杂度越高，占用的内存大小就越大。