前言

循环语句：通过循环语句可以反复的执行一段代码多次。

for循环

for循环的语法

语法：

```
for(①初始化表达式; ②条件表达式; ④更新表达式){
    ③语句...
}
```

执行流程：

```
①执行初始化表达式，初始化变量（初始化表达式只会执行一次）

②执行条件表达式，判断是否执行循环：
    如果为true，则执行循环③
    如果为false，终止循环

④执行更新表达式，更新表达式执行完毕继续重复②
```

for循环举例：

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

上方代码的解释：

for循环举例

```
for (var i = 1; i < 13; i = i + 4) {
    console.log(i);
}

```

上方代码的遍历步骤：

```
程序一运行，将执行var i = 1;这条语句， 所以i的值是1。
然后程序会验证一下i < 13是否满足，1<13是真，所以执行一次循环体（就是大括号里面的语句）。
执行完循环体之后，会执行i=i+4这条语句，所以i的值，是5。

程序会会验证一下i < 13是否满足，5<13是真，所以执行一次循环体（就是大括号里面的语句）。
执行完循环体之后，会执行i=i+4这条语句，所以i的值，是9。

程序会会验证一下i < 13是否满足，9<13是真，所以执行一次循环体（就是大括号里面的语句）。
执行完循环体之后，会执行i=i+4这条语句，所以i的值，是13。

程序会会验证一下i < 13是否满足，13<13是假，所以不执行循环体了，将退出循环。

最终输出输出结果为：1、5、9
```

接下来做几个题目。

题目1：

```
for (var i = 1; i < 10; i = i + 3) {
    i = i + 1;
    console.log(i);
}
```

输出结果：2、6、10

题目2：

```
for (var i = 1; i <= 10; i++) {

}
console.log(i);
```

输出结果：11

题目3：

```
for(var i = 1; i < 7; i = i + 3){

}
console.log(i);
```

输出结果：7

题目4：

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

死循环。

while循环语句

while循环

语法：

```
while(条件表达式){
    语句...
}
```

执行流程：

```
while语句在执行时，先对条件表达式进行求值判断：

    如果值为true，则执行循环体：
        循环体执行完毕以后，继续对表达式进行判断
        如果为true，则继续执行循环体，以此类推

    如果值为false，则终止循环
```

如果有必要的话，我们可以使用 break 来终止循环。

do...while循环

语法：

```
do{
    语句...
}while(条件表达式)

```

执行流程：

```
do...while语句在执行时，会先执行循环体：
    循环体执行完毕以后，在对while后的条件表达式进行判断：
        如果结果为true，则继续执行循环体，执行完毕继续判断以此类推
        如果结果为false，则终止循环

```

while循环和 do...while循环的区别

这两个语句的功能类似，不同的是：

- while是先判断后执行，而do...while是先执行后判断。

也就是说，do...while可以保证循环体至少执行一次，而while不能。

break 和 continue

> 这个知识点非常重要。

break

- break可以用来退出switch语句或退出**整个**循环语句（循环语句包括for循环、while循环。不包括if。if里不能用 break 和 continue，否则会报错）。
- break会立即终止离它**最近**的那个循环语句。
- 可以为循环语句创建一个label，来标识当前的循环（格式：label:循环语句）。使用break语句时，可以在break后跟着一个label，这样break将会结束指定的循环，而不是最近的。

举例1：通过 break 终止循环语句

```
for (var i = 0; i < 5; i++) {
    console.log('i的值:' + i);
    if (i == 2) {
        break;  // 注意，虽然在 if 里 使用了 break，但这里的 break 是服务于外面的 for 循环。
    }
}

```

打印结果：

```
i的值:0
i的值:1
i的值:2
```

举例2：label的使用

```
outer:
for (var i = 0; i < 5; i++) {
    console.log("外层循环 i 的值：" + i)
    for (var j = 0; j < 5; j++) {
        break outer; // 直接跳出outer所在的外层循环（这个outer是我自定义的label）
        console.log("内层循环 j 的值:" + j);
    }
}

```

打印结果：

```
外层循环 i 的值：0
```

continue

- continue可以用来跳过**当次**循环，继续下一次循环。
- 同样，continue默认只会离他**最近**的循环起作用。

举例：

```
for (var i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        continue;
    }
    console.log('i的值:' + i);
}
```

打印结果：

```
i的值:1

i的值:3

i的值:5

i的值:7

i的值:9
```