



Java的控制流 Control Flow

Control Flow 是程序运行时执行代码的顺序

▼ 顺序结构

- 默认情况下，Java 程序是**从上到下逐行执行的**

```
System.out.println("第一步");  
System.out.println("第二步");
```

▼ 选择结构（分支）

if 语句

```
if (condition) {  
    // 条件为 true 执行  
}  
  
int age = 20;  
if (age >= 18) {  
    System.out.println("成年人");  
}
```

if-else

```
if (condition) {  
    // 条件为 true  
} else {  
    // 条件为 false  
}  
  
int number = -5;  
if (number > 0) {  
    System.out.println("正数");  
} else {  
    System.out.println("不是正数");  
}
```

if-else if-else

```

if (condition1) {
    // result
} else if (condition2) {
    // result
} else {
    // result
}

int score = 75;
if (score >= 90) {
    System.out.println("优秀");
} else if (score >= 60) {
    System.out.println("及格");
} else {
    System.out.println("不及格");
}

```

switch 语句（适合枚举、整数、字符串）

▼ Java 中可以用 switch 语句的类型：

- int long char byte short 🖱️ 除了 float double boolean 以外
- enum 枚举
- String

```

int day = 3;
switch (day) {
    case 1:
        System.out.println("星期一");
        break;
    case 2:
        System.out.println("星期二");
        break;
    case 3:
        System.out.println("星期三");
        break;
    default:
        System.out.println("其他日子");
}

```

```

public static void main(String[] args) throws IOException {
    char i = (char) ('A' + new Random().nextInt( bound: 5));
    switch (i) {
        case 'A':
        case 'B':
        case 'C':
            System.out.println("ABC");
            break;
        default:
            System.out.println("DE");
            break;
    }
}

```

▼ 循环结构

▼ while 循环

```

while (condition) {
    // 条件为 true 时重复执行
}

int i = 1;
while (i <= 3) {
    System.out.println("第 " + i + " 次 while 循环");
    i++;
}

```

▼ do-while 循环 (先执行一次，再判断条件)

```

do {
    // 至少执行一次
} while (condition);

int j = 1;
do {
    System.out.println("第 " + j + " 次 do-while 循环");
    j++;
} while (j <= 3);

```

▼ for 循环 (适合已知次数的循环)

```

public class ForFlowDemo {
    public static void main(String[] args) { // 程序入口

        for (int i = 1; // 1.初始化：声明变量 i 并赋值为 1

            i <= 3; // 2.条件判断：每轮循环前都判断 i 是否 <= 3

```

```

        i++; // 4.更新：每轮循环结束后, 执行更新语句:i自增1

    { System.out.println("当前值为：" + i); // 3.执行循环体：条件为 true 时执行
    }

    System.out.println("循环结束"); // 5.结束语句:条件为 false 时退出循环，程序结束
}
}

```

🧠 执行流程图解

```

1 int i = 1;    ← 初始化（只执行一次）
2 i <= 3 ?      ← 判断：如果 true，执行循环体；false 就退出
3 执行循环体    ← 打印当前 i
4 i++          ← 自增：i + 1
回到 2         ← 再判断
2 i <= 3 ?      ← 判断：如果 true，执行循环体；false 就退出
3 执行循环体    ← 打印当前 i
4 i++          ← 自增：i + 1
回到 2         ← 再判断

```

🌀 实际执行过程逐步展开：

步骤	i 值	判断 (i <= 3)	打印内容	是否更新 i	说明
1	1	true	当前值为：1	i++ → 2	开始循环，i = 1
2	2	true	当前值为：2	i++ → 3	第二轮，i = 2
3	3	true	当前值为：3	i++ → 4	第三轮，i = 3
4	4	false	循环结束		i > 3，跳出循环

✅ 输出结果

```

当前值为：1
当前值为：2
当前值为：3
循环结束

```

▼ **for-each** 循环（用于遍历数组/集合）

```
int[] nums = {1, 2, 3};
for (int n : nums) {
    System.out.println(n);
}
```

▼ 跳转语句 `break` `continue`

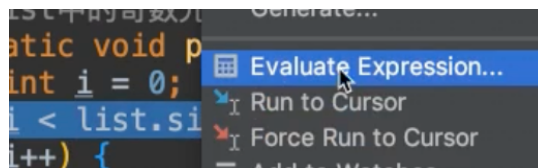
▼ `continue`

- 跳过包裹当前 `continue` 的第一层循环中的其余语句, 继续下一次循环

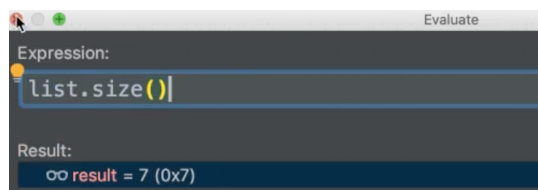
// 课上的example

```
public static void printOdd(List<Integer> list){
    for (int i = 0; i<list.size();i++){
        if (list.get(i) % 2 == 0){
            continue;
        }
        system.out.println(list.get(i));
    }
}

public static void main ( String[] args ){
    printOdd ( Array.asList(1,4,2,56,-1,3,12) );
}
```



如果想要知道代码运行过程中某个方法的结果可以右击 [Evaluate Expression](#)



```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue; // 跳过这一轮，不执行下面的打印
    }
    System.out.println("i = " + i);
}
```

```
}  
System.out.println("循环结束");
```

输出结果：


```
i = 1  
i = 2  
i = 4  
i = 5  
循环结束
```

注释说明：

```
i = 1, 打印  
i = 2, 打印  
i == 3, 执行 continue, 跳过打印  
i = 4, 打印  
i = 5, 打印
```

▼ break

- 立即结束包裹当前 break 的第一层循环

```
for (int i = 1; i <= 5; i++) {  这是包裹break的第一层循环  
    if (i == 3) {  
        break; // 直接跳出整个循环  
    }  
    System.out.println("i = " + i);  
}  
System.out.println("循环结束");
```

输出结果：

```
i = 1  
i = 2  
循环结束
```

注释说明：

- i = 1, 打印
- i = 2, 打印
- i == 3, 执行 break, 整个循环被终止！

- 后面的 4 和 5 都不会执行

▼ ⚠️ 判断包裹 break 的第一层循环

❓ 为什么说 `for` 是包裹 `break` 的第一层循环？`break` 明明是写在 `if` 里面？

✅ `break` 只能影响循环结构，而不是 `if`

`if` 是判断语句，不是循环

它只是判断某个条件是否成立，然后决定是否执行某段代码，不会重复执行

`for`、`while`、`do-while` 才是循环语句，它们是能让程序多次重复执行一段代码的结构

🔍 图解层级结构：

```
for (int i = 1; i <= 5; i++) {    // 👉 循环结构（真正能被 break 跳出的）
    if (i == 3) {                // 👉 判断结构，不是循环
        break;                  // 👉 break 只能跳出“最近的外层循环”——就是 for
    }
    System.out.println("i = " + i); // 👉 循环体
}
System.out.println("循环结束");
```

外层：for 循环（会重复执行）
↳ 内层：if 判断（只判断一次）

`break` 的效果：跳出“当前所在的最近的一层循环” → 就是 `for`

打印一个 **2行3列** 的表格数字，但当数字到达 `i==1 且 j==2` 时使用 `break`

✅ 嵌套循环 + break

```
public class BreakDemo {
    public static void main(String[] args) {
        for (int i = 1; i <= 2; i++) {    // 外层循环：控制行数（2行）
            for (int j = 1; j <= 3; j++) { // 内层循环：控制列数（3列）

                if (j == 2) {
                    break;                // 👉 只跳出“内层”for循环
                }

                System.out.println("i = " + i + ", j = " + j);
            }
        }
        System.out.println("循环结束");
    }
}
```

```
}  
}
```

🧠 执行过程模拟：

1. `i = 1`
 - `j = 1` → 打印 `i=1, j=1`
 - `j = 2` → 触发 `break` ! → 跳出内层 `j` 的循环，继续下一轮外层 `i`
2. `i = 2`
 - `j = 1` → 打印 `i=2, j=1`
 - `j = 2` → again `break` ! → 跳出内层

🖨️ 输出结果：

```
i = 1, j = 1  
i = 2, j = 1  
循环结束
```

📌 关键点：

行为	说明
<code>break</code> 执行位置	位于内层循环 (<code>j</code> 循环) 中
跳出的层级	只跳出了 内层 <code>for(j)</code> 循环，不会影响外层 <code>i</code> 循环
外层循环	仍然会继续下一轮

🎯 总结：

`break` 只跳出它所在的那一层循环
如果是在嵌套里，就跳出“离它最近的循环”

▼ `break label`

`break label` 可以跳出多层嵌套循环，这是 `break` 的加强版

`label` 是放在循环前面、加了冒号的一个标识符，用来给这个循环起名字

语法格式：

```
label名: for (...) {  
    // ...  
    break label名; // 一旦执行，就跳出这个标签对应的循环  
}
```



```
public static void main(String[] args) throws IOException {
    最外层的循环:
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            break 最外层的循环;
        }
    }
}
```

用 `break label` 跳出两层循环

模拟一个 3x3 的嵌套循环，但遇到某个条件时 **立即跳出全部循环（外层 + 内层）**

```
public class BreakLabelDemo {
    public static void main(String[] args) {

        outerLoop: // 🖱️ 这是外层循环的标签
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                System.out.println("i = " + i + ", j = " + j);

                if (i == 2 && j == 2) {
                    System.out.println("触发 break outerLoop !");
                    break outerLoop; 🖱️ 跳出 outerLoop 所代表的整个外层循环
                }
            }
        }

        System.out.println("循环结束");
    }
}
```

⚠️ 注意事项：

- 标签必须写在 **循环前面**
- `break 标签名;` 只能跳出 **循环结构**，不能用于跳出 `if` 或方法
- 标签名可以是任意合法的标识符（不建议用 Java 关键字）
- 标签版 `break` 是唯一能“一步退出多层循环”的方式

▼ `return`

- 结束当前方法，返回值

```
public class ReturnExample {
    public static void main(String[] args) {
        System.out.println("程序开始");
        if (true) {
```

```
        return; // 立即结束方法
    }
    System.out.println("这行不会被执行");
}
}
```