

# P85-86关键字与键盘输入

2021年4月27日 11:40

## 关键字

关键字的定义和特点 (不用背)

定义: 被Java语言赋予了特殊含义, 用做专门用途的字符串 (单词)

特点: 关键字中所有字母都为小写

用于定义数据类型的关键字

class	interface	enum	byte	short
int	long	float	double	char
boolean	void			

用于定义数据类型值的关键字

true	false	null		
------	-------	------	--	--

用于定义流程控制的关键字

if	else	switch	case	default
while	do	for	break	continue
return				

## 保留字

### 介绍

Java保留字: 现有Java版本尚未使用, 但以后版本可能会作为关键字使用。自己命名标识符时要避免使用这些保留字

byValue, Icast, future, generic, inner, operator, outer, rest, var, goto, const

## 键盘输入语句

### ● 介绍

在编程中, 需要接收用户输入的数据, 就可以使用键盘输入语句来获取。

**Input.java**, 需要一个 扫描器(对象), 就是 **Scanner**

### ● 步骤:

- 1) 导入该类的所在包, `java.util.*`
- 2) 创建该类对象 (声明变量)
- 3) 调用里面的功能

```
//步骤2: 创建Scanner类的对象
Scanner input = new Scanner(System.in);
//步骤3: 调用里面的功能
System.out.println("请输入姓名:");
String name = input.next();
System.out.println("请输入年龄:");
int age = input.nextInt();
System.out.println("请输入成绩:");
double score = input.nextDouble();
System.out.println("name:"+name);
System.out.println("age:"+age);
System.out.println("score:"+score);
```

### ● 案例演示:

要求: 可以从控制台接收用户信息, 【姓名, 年龄, 薪水】。

# P88-97进制转换

2021年4月27日 14:12

## 进制

- 进制介绍

对于整数，有四种表示方式：

1. 二进制：0,1，满2进1.以0b或0B开头。
2. 十进制：0-9，满10进1。
3. 八进制：0-7，满8进1.以数字0开头表示。
4. 十六进制：0-9及A(10)-F(15)，满16进1.以0x或0X开头表示。此处的A-F不区分大小写。

- 举例说明 BinaryTest.java

```
int n1 = 0b1010;
int n2 = 1010;
int n3 = 01010;
int n4 = 0x10101;
```

## 进制的图示

十进制	十六进制	八进制	二进制
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000

十进制	十六进制	八进制	二进制
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111
16	10	20	10000
17	11	21	10001

## 进制转换的介绍

- 第一组：
  1. 二进制转十进制
  2. 八进制转十进制
  3. 十六进制转十进制
- 第二组：
  1. 十进制转二进制
  2. 十进制转八进制
  3. 十进制转十六进制
- 第三组：
  1. 二进制转八进制
  2. 二进制转十六进制
- 第四组：
  1. 八进制转二进制
  2. 十六进制转二进制

## 十进制转换成二进制

规则：将该数不断除以2，直到商为0为止，然后将每步得到的余数倒过来，就是对应的二进制。

案例：请将 34 转成二进制 =

## 二进制转换成八进制

规则：从低位开始，将二进制数每三位一组，转成对应的八进制数即可。

案例：请将 0b11010101 转成八进制

0b11(3)010(2)101(5) => 0325

## 二进制转换成十六进制

规则：从低位开始，将二进制数每四位一组，转成对应的十六进制数即可。

案例：请将 0b11010101 转成十六进制

## 八进制转换成二进制

规则：将八进制数每1位，转成对应的一个3位的二进制数即可。

案例：请将 0237 转成二进制

02(010)3(011)7(111) = 0b01001111



## 十六进制转换成二进制

规则：将十六进制数每1位，转成对应的4位的一个二进制数即可。

案例：请将 0x23B 转成二进制

0x2(0010)3(0011)B(1011) =



# P98-101位运算、原码反码补码

2021年5月1日 16:28

## 原码、反码、补码(重点 难点)

网上对原码,反码,补码的解释过于复杂,我这里精简几句话:(背下来)  
对于有符号的而言:

1. 二进制的最高位是符号位: 0表示正数,1表示负数 (老韩口诀: 0->0 1->-)
2. 正数的原码, 反码, 补码都一样 (三码合一)
3. 负数的反码=它的原码符号位不变, 其它位取反(0->1,1->0)
4. 负数的补码=它的反码+1, 负数的反码 = 负数的补码 - 1
5. 0的反码, 补码都是0
6. java没有无符号数, 换言之, java中的数都是有符号的
7. 在计算机运算的时候, 都是以补码的方式来运算的.
8. 当我们看运算结果的时候, 要看他的原码①

## 位运算符

- java中有7个位运算(&、|、^、~、>>、<<和>>>)

✓ 分别是 按位与&、按位或|、按位异或^,按位取反~,它们的运算规则是:

- 按位与& : 两位全为1, 结果为1, 否则为0
- 按位或| : 两位有一个为1, 结果为1, 否则为0
- 按位异或^ : 两位一个为0,一个为1, 结果为1, 否则为0
- 按位取反~ : 0->1,1->0
- 比如:  $2 \& 3 = ?$      $\sim 2 = ?$      $\sim 2 = ?$      $2 | 3 = ?$      $2 \wedge 3 = ?$

BitOperator.java

✓ 完成前面的案例!

## 位运算符

- 还有3个位运算符 >>、<< 和 >>>, 运算规则:

1. 算术右移 >>: 低位溢出,符号位不变,并用符号位补溢出的高位
2. 算术左移 <<: 符号位不变,低位补0
3. >>> 逻辑右移也叫无符号右移,运算规则是: 低位溢出, 高位补 0
4. 特别说明: 没有 <<< 符号

- 应用案例 BitOperator02.java

int a=1>>2; //1 => 00000001 => 00000000 本质  $1 / 2 / 2 = 0$   
int c=1<<2; //1 => 00000001 => 00000100 本质  $1 * 2 * 2 = 4$

- 完成前面的案例

建议: 掌握老师讲解的即可, 不用再深入.

## 本章作业

### 1. 计算下列表达式的结果

$10/3 = 3$  ;  $10/5 = 2$  ;  $10\%2 = 0$  ;  $-10.5\%3 = ?$ ;  
//a % b 当 a 是小数时, 公式 =  $a - (\text{int})a / b * b$   
// $-10.5\%3 = -10.5 - (-10)/3 * 3 = -10.5 + 9 = -1.5$   
//注意: 有小数运算, 得到结果是近似值

### 2. 试说出下面代码的结果

```
int i=66;  
System.out.println(++i+i); // 执行 i = i + 1 => i = 67 => 134
```

## 本章作业

### 3. 在Java中, 以下赋值语句正确的是0。

- A) `int num1=(int)"18";` //错误 应该 `Integer.parseInt("18")` ;
- B) `int num2=18.0;` //错误 `double -> int`
- C) `double num3=3d;` //ok
- D) `double num4=8;` //ok `int -> double`
- E) `int i=48; char ch = i+1;` //错误 `int -> char`
- F) `byte b = 19; short s = b+2;` //错误 `int -> short`

## 本章作业

### 4. 试写出将String转换成double类型的语句, 以及将char类型转换成String的语句,举例说明即可, 写简单代码

```
String str = "18.8";//注意 字符串要可以被转成 double  
double d1 = Double.parseDouble(str);
```

```
char c1 = '韩';  
String str2 = c1 + "";
```



## 顺序控制

### ● 顺序控制介绍

程序从上到下逐行地执行，中间没有任何判断和跳转。

### ● 顺序控制举例和注意事项

Java中定义成员变量时采用合法的前向引用。如：

```
public class Test{
    int num1 = 12;
    int num2 = num1 + 2;
}
```

错误形式：

```
public class Test{
    int num2 = num1 + 2; //错误
    int num1 = 12;
}
```



## 分支控制if-else

### ● 单分支

#### ✓ 基本语法

```
if(条件表达式){
    执行代码块; (可以有多条语句)
}
```

- 说明：当条件表达式为true时，就会执行 {} 的代码。如果为false，就不执行。特别说明，如果 {} 中只有一条语句，则可以用不用 {}，建议写上 {}

#### ✓ 案例说明

请大家看个案例[If01.java]:

编写一个程序,可以输入人的年龄,如果该同志的年龄大于18岁,则输出 "你年龄大于18,要对自己的行为负责,送入监狱"

## 分支控制if-else

### ● 双分支

#### ✓ 基本语法

```
if(条件表达式) {
    执行代码块1;
}
else {
    执行代码块2;
}
```

**说明：**当条件表达式成立，即执行代码块1，否则执行代码块2。如果执行代码块只有一条语句，则 {} 可以省略，否则，不能省略

#### ✓ 案例演示

请大家看个案例[If02.java]:

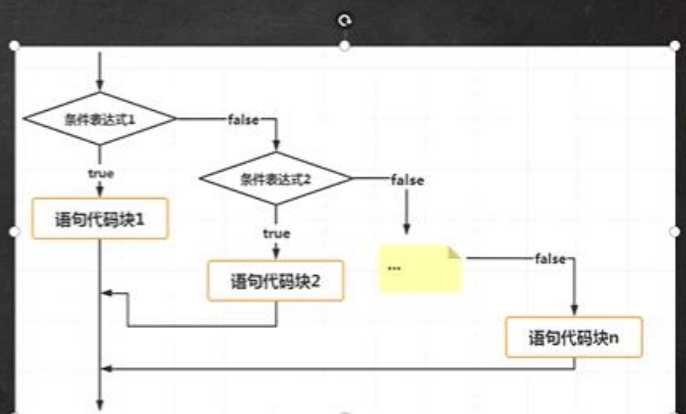
编写一个程序,可以输入人的年龄,如果该同志的年龄大于18岁,则输出 "你年龄大于18,要对自己的行为负责,送入监狱"。否则,输出 "你的年龄不大这次放过你了。"

## 分支控制if-else

### ✓ 多分支的流程图(重要!)

说明:

1. 当条件表达式1成立时, 即执行代码块1,
2. 如果表达式1不成立, 才去判断表达式2是否成立,
3. 如果表达式2成立, 就执行代码块2
4. 以此类推, 如果所有的表达式都不成立
5. 则执行 else 的代码块, 注意, 只能有一个执行入口。



## 嵌套分支。

### ● 基本介绍

在一个分支结构中又完整的嵌套了另一个完整的分支结构, 里面的分支的结构称为内层分支外面的分支结构称为外层分支。老师建议: 不要超过3层 (可读性不好)

### ● 基本语法

```

if(){
    if(){
        //if-else....
    }else{
        //if-else
    }
}
    
```

## 嵌套分支。

### ● 应用案例2

出票系统: 根据淡旺季的月份和年龄, 打印票价 [课后练习]

4 10 旺季:

成人 (18-60) : 60  
 儿童 (<18) : 半价  
 老人 (>60) : 1/3

淡季:

成人: 40  
 其他: 20

思路分析(1) 淡旺季 - if - else (2) 在旺季 中, 可以使用多分支处理三种情况 (3) 在淡季情况, 使用双分支处理即可



# P114-121 switch基本用法

2021年5月2日 22:53

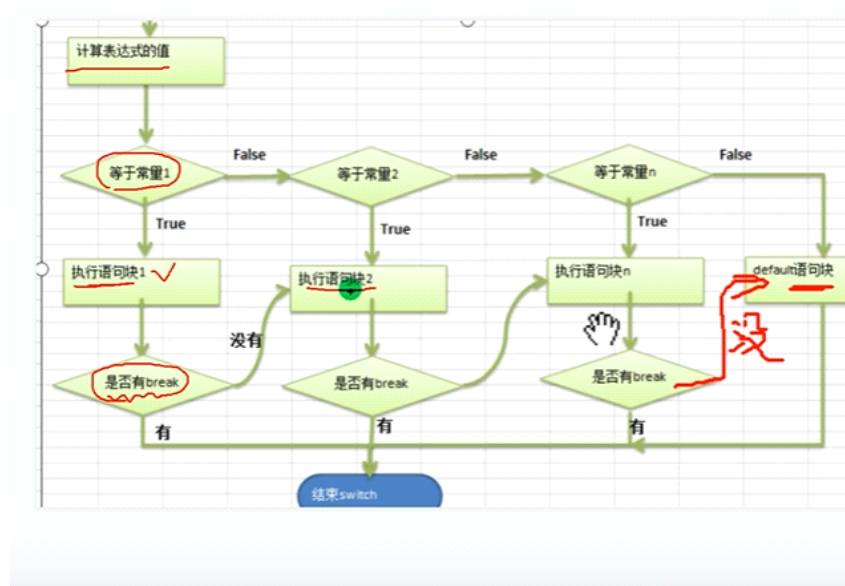
### switch分支结构

- 基本语法

```
switch(表达式){  
    case 常量1: //当...  
        语句块1;  
        break;  
    case 常量2;  
        语句块2;  
        break;  
    ...  
    case 常量n;  
        语句块n;  
        break;  
    default:  
        default语句块;  
        break;  
}
```

1. switch 关键字, 表示switch分支
2. 表达式 对应一个值
3. case 常量1 :当表达式的值等于常量1, 就执行 语句块1
4. break : 表示退出switch
5. 如果和 case 常量1 匹配, 就执行语句块1, 如果没有匹配, 就继续匹配 case 常量2
6. 如果一个都没有匹配上, 执行default

2 2.3 Hello



穿透, 没有break会直接执行语句块2、不会判断

- switch注意事项和细节讨论

```
//SwitchDetail.java
```

1. 表达式数据类型, 应和case 后的常量类型一致, 或者是可以自动转成可以相互比较的类型, 比如输入的是字符, 而常量是 int
2. switch(表达式)中表达式的返回值必须是: (byte,short,int,char,enum[枚举],String)

```
double c = 1.1;  
switch(c){ //错误  
    case 1.1 : //错误  
        System.out.println("ok3");  
        break;  
}
```

3. case子句中的值必须是常量,而不能是变量
4. default子句是可选的, 当没有匹配的case时, 执行 default
5. break语句用来在执行完一个case分支后使程序跳出switch语句块; 如果没有写 break, 程序会顺序执行到switch结尾

### switch分支结构

- switch和if的比较

1. 如果判断的具体数值不多, 而且符合byte、short、int、char、enum、String 这6种类型。虽然两个语句都可以使用, 建议使用switch语句。
2. 其他情况: 对区间判断, 对结果为boolean类型判断, 使用if, if的使用范围更广。



## for循环控制

基本介绍:听其名而知其意,就是让你的代码可以循环的执行.

- 看一个实际需求

请大家看个案例[For01.java]:编写一个程序,可以打印10句 "你好, 韩顺平教育!".  
请大家想想怎么做?

## for循环控制

- 基本语法

```
for (循环变量初始化; 循环条件; 循环变量迭代) {  
    循环操作(可以多条语句);  
}
```

- 老韩说明 ①

1. for 关键字, 表示循环控制
2. for有四要素: (1)循环变量初始化(2)循环条件(3)循环操作(4)循环变量迭代
3. 循环操作, 这里可以有多条语句, 也就是我们要循环执行的代码
4. 如果 循环操作(语句) 只有一条语句, 可以省略 {}, 建议不要省略

## for循环控制

- for循环执行流程分析

- 1) 使用for循环完成前面的题
- 2) 画出for流程图
- 3) 代码执行内存分析法

```
for (循环变量初始化; 循环条件; 循环变量迭代) {  
    循环操作(可以多条语句);  
}
```

① ② ③ ④



先循环一遍, 在进行迭代

## for循环控制

- 注意事项和细节说明

### ForDetail.java

- 1) 循环条件是返回一个布尔值的表达式
- 2) for(循环判断条件;) 中的初始化和变量迭代可以写到其它地方, 但是两边的分号不能省略。
- 3) 循环初始值可以有多条初始化语句, 但要求类型一样, 并且中间用逗号隔开, 循环变量迭代也可以有多条变量迭代语句, 中间用逗号隔开。
- 4) 使用内存分析法, 老师分析输出下面代码输出什么?

```
int count = 3;  
for (int i = 0, j = 0; i < count; i++, j += 2) {  
    System.out.println("i=" + i + " j=" + j);  
}
```

# P127-129while语法

2021年5月3日 21:29

## while循环控制

- 基本语法  
循环变量初始化;  
**while (循环条件) {**  
    循环体(语句);  
    循环变量迭代;  
**}**
- 老韩说明  
1) while 循环也有四要素  
2) 只是四要素放的位置, 不一样.

## while循环控制

- while循环执行流程分析  
**While01.java**  
1) 画出流程图  
2) 使用while循环完成前面的题  
3) 代码执行内存分析图

```
graph TD; A[循环变量初始化] --> B{循环条件}; B -- T --> C[循环体; 循环变量迭代]; C --> B; B -- F --> D([while结束]);
```

## while循环控制

- 课堂练习题[学员先做]  
**WhileExercise.java**  
1. 打印1—100之间所有能被3整除的数 [使用while, 老师评讲]  
2. 打印40—200之间所有的偶数 [使用while, 课后练习]



## do..while循环控制

### ● 基本语法

循环变量初始化;

do{

    循环体(语句);

    循环变量迭代;

}while(循环条件);

### ● 老韩说明:

#### 1. do while 是关键字

1. 也有循环四要素, 只是位置不一样

2. 先执行, 再判断, 也就是说, 一定会至少执行一次

3. 最后 有一个 分号 ;

4. while 和 do..while 区别举例: 要账

## do..while循环控制

### ● do...while循环执行流程分析

#### 1. 画出流程图

#### 2. 使用do...while循环完成前面的题

#### 3. 代码执行内存分析图

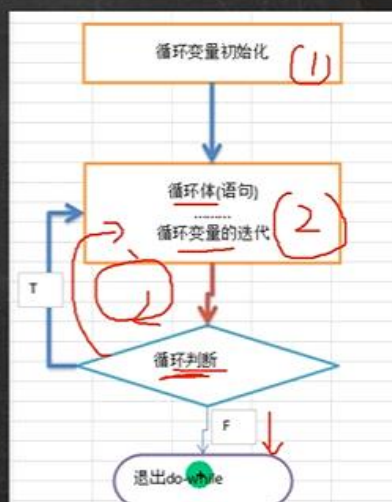
循环变量初始化;

do{

    循环体(语句);

    循环变量迭代;

}while(循环条件);



## do..while循环控制。

### ● 注意事项和细节说明

1) 循环条件是返回一个布尔值的表达式

2) do..while循环是先执行, 再判断, 因此它至少执行一次

### ● 课堂练习题【学员先做】

1) 打印1—100 [学生做]

2) 计算1—100的和 [学生做]

3) 统计1——200之间能被5整除但不能被3整除的个数 (DoWhileExercise01.java)

4) 如果李三不还钱, 则老韩将一直使出五连鞭, 直到李三说还钱为止  
[System.out.println("老韩问: 还钱吗? y/n")] do...while ..

DoWhileExercise02.java





# P134-137 多重循环控制

2021年5月4日 10:23

## 多重循环控制(难点! 重点!)

### ● 介绍

1. 将一个循环放在另一个循环体内, 就形成了嵌套循环。其中, for, while, do...while 均可以作为外层循环和内层循环。【建议一般使用两层, 最多不要超过3层, 否则, 代码的可读性很差】
2. 实质上, 嵌套循环就是把内层循环当成外层循环的循环体。当只有内层循环的循环条件为 false 时, 才会完全跳出内层循环, 才可结束外层的当次循环, 开始下一轮的循环[听不懂, 走案例]。

3. 设外层循环次数为m次, 内层为n次, 则内层循环体实际上需要执行m\*n次。

```
for(int i = 1; i <= 7; i++) {  
    for(int j = 1; j <= 2; j++) {  
        System.out.println("ok~~"); // 7 * 2 = 14  
    }  
}
```

## 多重循环控制

### ● 多重循环执行步骤分析:

请分析 下面的多重循环执行步骤, 并写出输出 => 韩老师的内存分析法

// 双层for MulFor.java

```
for(int i = 0; i < 2; i++) { // 先思考  
    for(int j = 0; j < 3; j++) {  
        System.out.println("i=" + i + j + " ");  
    }  
}
```



## 多重循环控制

### ● 应用实例:

// MulForExercise01.java

1. 统计3个班成绩情况, 每个班有5名同学, 求出各个班的平均分和所有班级的平均分[学生的成绩从键盘输入]。
2. 统计三个班及格人数, 每个班有5名同学。
3. 打印出九九乘法表[课后题]

```
1 1 2 3 4 5 6 7 8 9  
2 2 2 4 6 8 10 12 14 16  
3 3 3 6 9 12 15 18 21 24  
4 4 4 8 12 16 20 24 28 32  
5 5 5 10 15 20 25 30 35 40  
6 6 6 12 18 24 30 36 42 48  
7 7 7 14 21 28 35 42 49 56  
8 8 8 16 24 32 40 48 56 64  
9 9 9 18 27 36 45 54 63 81
```

## 多重循环控制

### ● 经典的打印金字塔

使用 for 循环完成下面的案例

请编写一个程序, 可以接收一个整数, 表示层数 (totalLevel), 打印出金字塔。 (Stars.java) [化繁为简, 先死活]



上机练习题:  
打印空心金字塔  
打印空心的菱形

- 1) 完成思路, 可以从矩形开始打
- 2) 代码实现
- 3) 请同学使用while来实现一把

## 跳转控制语句-break

- 看下面一个需求

随机生成1-100的一个数，直到生成了97这个数，看看你一共用了几次？

提示使用 `(int)(Math.random() * 100) + 1`

思路分析：

循环，但是循环的次数不知道。 -> `break`，当某个条件满足时，终止循环  
通过该需求可以说明其它流程控制数据的必要性，比如`break`

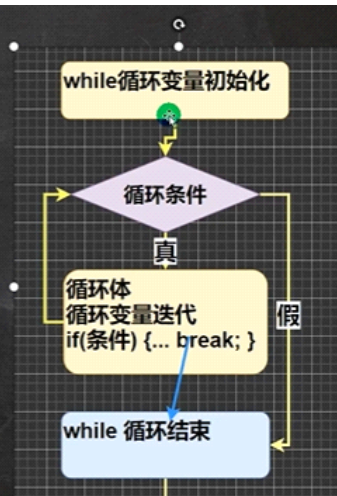
## 跳转控制语句-break

- 基本介绍：  
`break`语句用于终止某个语句块的执行，一般使用在`switch`或者循环`for`，`while`，`do-while`中

- 基本语法：

```
{
    .....
    break;
    .....
}
```

- 以`while`使用`break`为例，画出示意图



## 跳转控制语句-break

- 注意事项和细节说明：

1. `break`语句出现在多层嵌套的语句块中时，可以通过**标签**指明要终止的是哪一层语句块 **BreakDetail.java**

2. 标签的基本使用

```
label1: { .....
label2:  { .....
label3:  { .....
           break label2;
           .....
       }
    }
}
```

```
label1:
for(int j = 0; j < 4; j++){
label2:
    for(int i = 0; i < 10; i++){
        if(i == 2){
            break label1;
        }
        System.out.println("i = " + i);
    }
}
```

输出什么？并分析原因

- (1) `break` 语句可以指定退出哪层
- (2) `label1` 是标签，由程序员指定。
- (3) `break` 后指定到哪个`label` 就退出到哪里
- (4) 在实际的开发中，尽量不要使用标签。
- (5) 如果没有指定 `break`，默认退出最近的循环体

## 跳转控制语句-break

- 课堂练习题:

### BreakExercise.java

- 1) 1-100以内的数求和, 求出 当和 第一次大于20的当前数 【for + break】
- 2) 实现登录验证, 有3次机会, 如果用户名为"丁真", 密码"666"提示登录成功, 否则提示还有几次机会, 请使用for+break完成



## 跳转控制语句-continue

- 基本介绍:

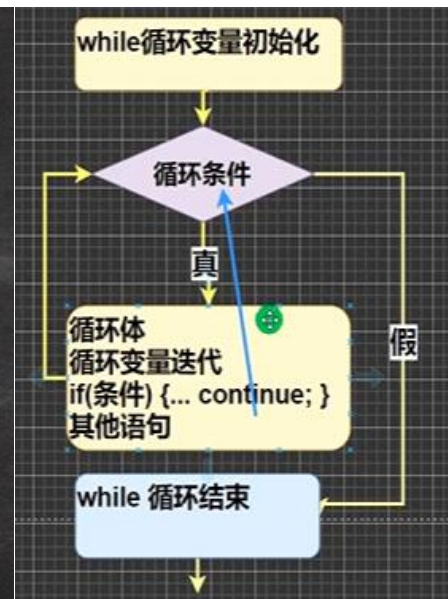
1) continue 语句用于结束本次循环, 继续执行下一次循环。

2) continue 语句出现在多层嵌套的循环语句体中时, 可以通过标签指明要跳过的是哪一层循环, 这个和前面的标签的使用的规则一样。

- 基本语法:

```
{
    .....
    continue;
    .....
}
```

- 以 while 使用 continue 为例, 画出示意图



## 跳转控制语句-continue

- 细节案例分析和说明:

```
label1:
for(int j = 0; j < 4; j++){
    label2:
    for(int i = 0; i < 10; i++){
        if(i == 2){
            //看看分别输出什么值, 并分析
            //continue ;
            //continue label2;
            continue label1;
        }
        System.out.println("i = " + i);
    }
}
```

说明: 结合流程分析图, 分析内存执行情况

## 跳转控制语句-return

- 介绍

return 使用在方法, 表示跳出所在的方法, 在讲解方法的时候, 会详细的介绍, 这里我们简单的提一下。注意: 如果 return 写在 main 方法, 退出程序..

Return01.java

```
for(int i=1;i<=5;i++){

    if(i==3) {
        System.out.println("韩顺平教育 "+i);
        return;//continue; break;
    }
    System.out.println("Hello World!");
}
System.out.println("go on..");
```

# P148-160数组介绍及使用

2021年5月6日 16:30

## • 数组介绍

数组可以存放**多个同一类型**的数据。数组也是一种数据类型，是引用类型。  
即：数(数据)组(一组)就是一组数据

## • 数组快速入门

### Array01.java

比如，我们可以用数组来解决上一个问题。 体验

## 数组的使用

### 使用方式1-动态初始化

#### 数组的定义

数据类型 数组名[] = new 数据类型[大小]

int a[] = new int[5]; //创建了一个数组，名字a,存放5个int

说明：这是定义数组的一种方法。为了让大家明白，我画数组内存图说明



#### 数组的引用(使用)

数组名[下标/索引] 比如：你要使用a数组的第3个数 a[2]

### 快速入门案例 Array02.java

循环输入5个成绩，保存到double数组,并输出

```
Scanner input = new Scanner(System.in);
//1.声明并开辟空间
double[] scores = new double[5];
//2.赋值
for(int i=0;i<5;i++){
    scores[i] = input.nextDouble();
}
//3.打印
for(int i=0;i<5;i++){
    System.out.println(scores[i]);
}
```

## 数组的使用

### • 使用方式2-动态初始化

#### ✓ 先声明数组

语法:数据类型 数组名[]; 也可以 数据类型[] 数组名;  
int a[]; 或者 int[] a;

#### ✓ 创建数组

语法: 数组名=new 数据类型[大小];  
a=new int[10];

#### ✓ 案例演示【前面修改即可】

## 数组的使用

### • 使用方式3-静态初始化

#### ✓ 初始化数组

语法: 数据类型 数组名[] = {元素值,元素值...}

int a[] = {2,5,6,7,8,89,90,34,56}; 如果知道数组有多少元素，具体值

上面的用法相当于: int a[] = new int[9];

a[0]=2;a[1]=5;a[2]=6; a[3]=7;a[4]=8;

a[5]=89;a[6]=90;a[7]=34;a[8]=56;

#### ✓ 快速入门案例【养鸡场】

//案例 Array01.java 讲过

double hens[] = {3, 5, 1, 3.4, 2, 50};

等价

double hens[] = new double[6];

hens[0] = 3; hens[1] = 5; hens[2] = 1; hens[3] = 3.4; hens[4] = 2; hens[5] = 50;



## 数组使用注意事项和细节

### ArrayDetail.java

1. 数组是多个相同类型数据的组合，实现对这些数据的统一管理
2. 数组中的元素可以是任何数据类型，包括基本类型和引用类型，但是不能混用。
3. 数组创建后，如果没有赋值，有默认值  
int 0, short 0, byte 0, long 0, float 0.0, double 0.0, char \u0000, boolean false, String null
4. 使用数组的步骤 1. 声明数组并开辟空间 2 给数组各个元素赋值 3 使用数组
5. 数组的下标是从0开始的。
6. 数组下标必须在指定范围内使用，否则报：下标越界异常，比如  
int [] arr=new int[5]; 则有效下标为 0-4
7. 数组属引用类型，数组型数据是对象(object)

## 数组应用案例

1. 创建一个char类型的26个元素的数组，分别放置'A'-'Z'。使用for循环访问所有元素并打印出来。提示：char类型数据运算 'A'+2 -> 'C'  
ArrayExercise01.java
2. 请求出一个数组int[]的最大值 {4,-1,9, 10,23}，并得到对应的下标。  
ArrayExercise02.java
3. 请求出一个数组的和和平均值。(养鸡场)



# P164-168数组赋值、拷贝、翻转

2021年5月6日 21:54

## 数组赋值机制

基本数据类型赋值，这个值就是具体的数据，而且相互不影响。

```
int n1 = 2; int n2 = n1;
```

数组在默认情况下是引用传递，赋的值是地址。

看一个案例，并分析数组赋值的内存图(重点)。

//代码 **ArrayAssign.java**

```
int[] arr1 = {1,2,3};
```

```
int[] arr2 = arr1;
```

## 数组拷贝

编写代码 实现数组拷贝(内容复制) **ArrayCopy.java**

• 将 `int[] arr1 = {10,20,30};` 拷贝到 `arr2` 数组, 要求数据空间是独立的。

```
// 创建一个新的数组 arr2, 开辟新的数据空间
```

```
// 大小 arr1.length;
```

```
int[] arr2 = new int[arr1.length];
```

遍历 `arr1` , 把每个元素拷贝到 `arr2` 对应的元素位置

```
for(int i = 0; i < arr1.length; i++) {
```

```
    arr2[i] = arr1[i];
```

```
}
```

## 数组反转

要求: 把数组的元素内容反转。 **ArrayReverse.java**

`arr {11,22,33,44,55,66} → {66, 55,44,33,22,11}`

//思考 2min

1) 方式1: 通过找规律反转 【思路分析】

2) 方式2: 使用逆序赋值方式 【思路分析, 学员自己完成】