今日学习目标

* 能够理解顺序结构
* 能够使用if语句第一种格式
* 能够使用if语句第二种格式
* 能够使用if语句第三种格式
* 能够使用switch选择结构
* 能够使用for循环语句
* 能够使用while循环语句
* 能够使用do-while循环语句
* 能够使用跳转控制语句
* 能够使用跳转控制语句

# 流程控制语句

## 概述

* 概述

在一个程序执行的过程中，各条语句的执行顺序对程序的结果是有直接影响的。也就是说程序的流程对运行结果有直接的影响。所以，我们必须清楚每条语句的执行流程。而且，很多时候我们要通过控制语句的执行顺序来实现我们要完成的功能。

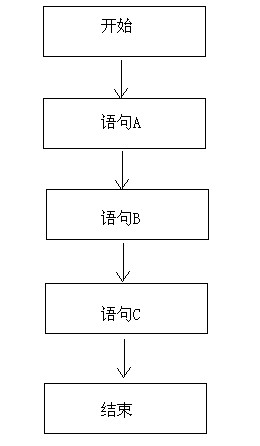
* 分类
  + 顺序结构
  + 选择结构
  + 循环结构

## 顺序结构

* 概述

是程序中最简单最基本的流程控制，没有特定的语法结构，按照代码的先后顺序，依次执行，程序中大多数的代码都是这样执行的。

* 总的来说：写在前面的先执行，写在后面的后执行
* 顺序结构流程图



**public** **class** Demo01Order {

**public** **static** **void** main(String[] args) {

System.***out***.println("开始");

System.***out***.println("语句A");

System.***out***.println("语句B");

System.***out***.println("语句C");

System.***out***.println("结束");

}

}

## 选择结构

### 概述

* 选择结构也被称为分支结构。
* 选择结构有特定的语法规则，代码要执行具体的逻辑运算进行判断，逻辑运算的结果有两个，所以产生选择，按照不同的选择执行不同的代码。
* Java语言提供了两种选择结构语句
  + if语句
  + switch语句

### 选择结构if

if语句有三种格式

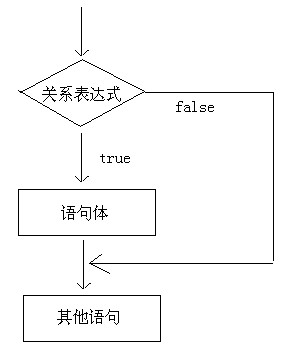
* If语句1
  + if语句第一种格式：

if(关系表达式) {

语句体

}

* + 执行流程
    - 首先判断关系表达式看其结果是true还是false
    - 如果是true就执行语句体
    - 如果是false就不执行语句体
  + 流程图



**public** **class** Demo01If {

**public** **static** **void** main(String[] args) {

System.***out***.println("开始");

// 定义两个变量

**int** a = 10;

**int** b = 20;

**if** (a == b) {

System.***out***.println("a等于b");

}

**int** c = 10;

**if** (a == c) {

System.***out***.println("a等于c");

}

System.***out***.println("结束");

}

}

* If语句2
  + if语句第二种格式：

if(关系表达式) {

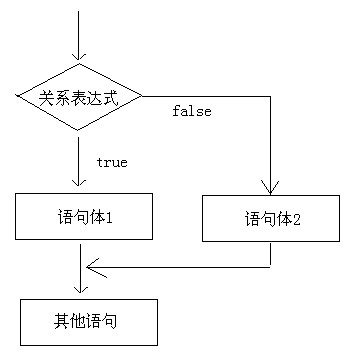
语句体1;

}else {

语句体2;

}

* + 执行流程
    - 首先判断关系表达式看其结果是true还是false
    - 如果是true就执行语句体1
    - 如果是false就执行语句体2
  + 流程图



**public** **class** Demo02If {

**public** **static** **void** main(String[] args) {

System.***out***.println("开始");

// 判断给定的数据是奇数还是偶数

// 定义变量

**int** a = 100;

// 给a重新赋值

a = 99;

**if** (a % 2 == 0) {

System.***out***.println("a是偶数");

} **else** {

System.***out***.println("a是奇数");

}

System.***out***.println("结束");

}

}

* If语句3
  + if语句第三种格式：

if (判断条件1) {

执行语句1

} else if (判断条件2) {

执行语句2

}

...

else if (判断条件n) {

执行语句n

} else {

执行语句n+1

}

* 执行流程
  + - 首先判断关系表达式1看其结果是true还是false
    - 如果是true就执行语句体1
    - 如果是false就继续判断关系表达式2看其结果是true还是false
    - 如果是true就执行语句体2
    - 如果是false就继续判断关系表达式…看其结果是true还是false
    - …
    - 如果没有任何关系表达式为true，就执行语句体n+1。
  + 流程图



* 例题:

x和y的关系满足如下：

x>=3 y = 2x + 1;

-1<x<3 y = 2x;

x<=-1 y = 2x – 1;

根据给定的x的值，计算出y的值并输出。

**public** **class** Demo03If {

**public** **static** **void** main(String[] args) {

// x和y的关系满足如下：

// x>=3 y = 2x + 1;

// -1<=x<3 y = 2x;

// x<=-1 y = 2x – 1;

// 根据给定的x的值，计算出y的值并输出。

// 定义变量

**int** x = 5;

/\*

int y;

if (x >= 3) {

y = 2 \* x + 1;

} else if (x >= -1 && x < 3) {

y = 2 \* x;

} else if (x <= -1) {

y = 2 \* x - 1;

}else {

y = 0;

}

\*/

/\*int y = 0;

if (x >= 3) {

y = 2 \* x + 1;

} else if (x >= -1 && x < 3) {

y = 2 \* x;

} else if (x <= -1) {

y = 2 \* x - 1;

}\*/

**int** y;

**if** (x >= 3) {

y = 2 \* x + 1;

} **else** **if** (x >= -1 && x < 3) {

y = 2 \* x;

} **else** {

y = 2 \* x - 1;

}

System.***out***.println("y的值是："+y);

}

}

* 总结
  + 第一种格式适合做一种情况的判断
  + 第二种格式适合做二种情况的判断
  + 第三种格式适合做多种情况的判断
* 练习

#### 键盘录入两个数据，获取这两个数据的较大值

**public** **class** Demo04If {

**public** **static** **void** main(String[] args) {

//创建对象

Scanner sc = **new** Scanner(System.***in***);

//接收数据

System.***out***.println("请输入第一个数据：");

**int** a = sc.nextInt();

System.***out***.println("请输入第二个数据：");

**int** b = sc.nextInt();

//采用if语句格式2实现

/\*

if(a>b){

System.out.println("较大的值是："+a);

}else {

System.out.println("较大的值是："+b);

}

\*/

//拿到较大的值之后，我未必想直接输出，所以我们定义变量接收这个较大的值

**int** max;

**if**(a>b){

max = a;

}**else** {

max = b;

}

//可能做其他的操作

//max += 100;

System.***out***.println("较大的值是："+max);

}

}

#### 键盘录入学生考试成绩，判断学生等级

90-100 优秀

80-90 好

70-80 良

60-70 及格

60以下 不及格

**public** **class** Demo05If {

**public** **static** **void** main(String[] args) {

//创建对象

Scanner sc = **new** Scanner(System.***in***);

//接收数据

System.***out***.println("请输入学生的考试成绩：");

**int** score = sc.nextInt();

//if语句格式3

/\*

if(score>=90 && score<=100){

System.out.println("你的成绩属于优秀");

}else if(score>=80 && score<90){

System.out.println("你的成绩属于好");

}else if(score>=70 && score<80){

System.out.println("你的成绩属于良");

}else if(score>=60 && score<70){

System.out.println("你的成绩属于及格");

}else {

System.out.println("你的成绩属于不及格");

}

\*/

//我们发现程序不够健壮,加入错误数据的判断

**if**(score<0 || score>100){

System.***out***.println("你的成绩是错误的");

}**else** **if**(score>=90 && score<=100){

System.***out***.println("你的成绩属于优秀");

}**else** **if**(score>=80 && score<90){

System.***out***.println("你的成绩属于好");

}**else** **if**(score>=70 && score<80){

System.***out***.println("你的成绩属于良");

}**else** **if**(score>=60 && score<70){

System.***out***.println("你的成绩属于及格");

}**else** {

System.***out***.println("你的成绩属于不及格");

}

}

}

### 选择结构switch

* 格式

switch(表达式) {

case 目标值1：

语句体1;

break;

case 目标值2：

语句体2;

break;

…

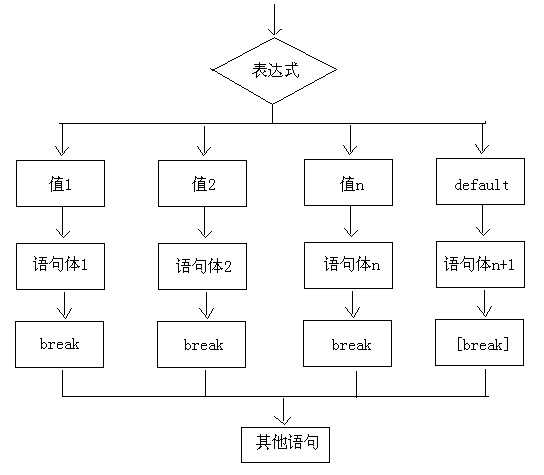
default：

语句体n+1;

break;

}

* 格式解释
  + switch表示这是switch语句
    - 表达式的取值：byte,short,int,char
    - JDK5以后可以是枚举
    - JDK7以后可以是String
  + case后面跟的是要和表达式进行比较的值
  + 语句体部分可以是一条或多条语句
  + break表示中断，结束的意思，可以结束switch语句
  + default语句表示所有情况都不匹配的时候，就执行该处的内容，和if语句的else相似。
* 执行流程
  + 首先计算出表达式的值
  + 其次，和case依次比较，一旦有对应的值，就会执行相应的语句，在执行的过程中，遇到break就会结束。
  + 最后，如果所有的case都和表达式的值不匹配，就会执行default语句体部分，然后程序结束掉。
* 流程图



* 练习
  + 根据键盘录入的数值1、2、3...7输出对应的星期一、星期二、星期三...星期日

**public** **class** Demo01Switch {

**public** **static** **void** main(String[] args) {

//创建键盘录入对象

Scanner sc = **new** Scanner(System.***in***);

//接收数据

System.***out***.println("请输入一个数字(1-7)：");

**int** weekday = sc.nextInt();

//switch语句实现选择

**switch**(weekday) {

**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**;

**case** 4:

System.***out***.println("星期四");

**break**;

**case** 5:

System.***out***.println("星期五");

**break**;

**case** 6:

System.***out***.println("星期六");

**break**;

**case** 7:

System.***out***.println("星期日");

**break**;

**default**:

System.***out***.println("你输入的数字有误");

**break**;

}

}

}

## 循环结构

### 概述

循环语句可以在满足循环条件的情况下，反复执行某一段代码，这段被重复执行的代码被称为循环体语句，当反复执行这个循环体时，需要在合适的时候把循环判断条件修改为false，从而结束循环，否则循环将一直执行下去，形成死循环。

### -循环语句的组成

* 循环变量初始化 (初始化表达式)：

一条或者多条语句，这些语句完成一些初始化操作。

* 循环出口(布尔表达式)：

这是一个boolean 表达式，这个表达式能决定是否执行循环体。

* 循环逻辑内容(循环体)：

这个部分是循环体语句，也就是我们要多次做的事情。

* 循环增量(步进表达式)：

这个部分在一次循环体结束后，下一次循环判断条件执行前执行。通过用于控制循环条件中的变量，使得循环在合适的时候结束。

### 循环结构for

* 格式

for(初始化表达式①；布尔表达式②；步进表达式④){

循环体③

}

* 执行流程

执行顺序： ①②③④>②③④>②③④… ②不满足为止。

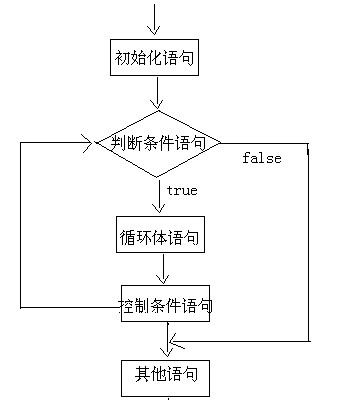
①负责完成循环变量初始化

②负责判断是否满足循环条件，不满足则跳出循环

③具体执行的语句

④循环后，循环条件所涉及变量的变化情况

* 流程图



* 练习
  + 在控制台输出10次HelloWorld

**public** **class** Demo01For {

**public** **static** **void** main(String[] args) {

//原始写法

System.***out***.println("HelloWorld");

System.***out***.println("HelloWorld");

System.***out***.println("HelloWorld");

System.***out***.println("HelloWorld");

System.***out***.println("HelloWorld");

System.***out***.println("HelloWorld");

System.***out***.println("HelloWorld");

System.***out***.println("HelloWorld");

System.***out***.println("HelloWorld");

System.***out***.println("HelloWorld");

System.***out***.println("-------------------------");

//用循环改进

**for**(**int** x=1; x<=10; x++) {

System.***out***.println("HelloWorld"+x);

}

}

}

* + 打印输出1-5和5-1
  + 求出1-100之间偶数和

**public** **class** Demo04For {

**public** **static** **void** main(String[] args) {

//1.定义一个初始化变量,记录累加求和,初始值为0

**int** sum = 0;

//2.利用for循环获取1-100之间的数字

**for** (**int** i = 1; i <= 100; i++) {

//3.判断获取的数组是奇数还是偶数

**if**(i%2==0){

//4.如果是偶数就累加求和

sum += i;

}

}

//5.循环结束之后,打印累加结果

System.***out***.println("sum:"+sum);

}

}

* + 统计“水仙花数”共有多少个

所谓的水仙花数是指一个三位数，其各位数字的立方和等于该数本身。

举例：153就是一个水仙花数。

153 = 1\*1\*1 + 5\*5\*5 + 3\*3\*3

**public** **class** Demo06For {

**public** **static** **void** main(String[] args) {

//1.定义一个统计变量,初始值为0

**int** count = 0;

//2.使用for循环获取所有的三位数

**for**(**int** x=100; x<1000; x++) {

//3.拆分三位数的百位,十位,个位

**int** ge = x%10;

**int** shi = x/10%10;

**int** bai = x/10/10%10;

//4.利用公式判断这个三位数是否为水仙花数,如果是,统计变量++

**if**((ge\*ge\*ge+shi\*shi\*shi+bai\*bai\*bai) == x) {

count++;

}

}

//5.循环结束之后,打印统计结果

System.***out***.println("水仙花数共有："+count+"个");

}

}

### 循环结构while

* 格式
  + 基本格式

while(布尔表达式) {

循环体语句;

}

* + 扩展格式

初始化表达式①

while(布尔表达式②){

循环体③

步进表达式④

}

* 执行流程

执行顺序： ①②③④>②③④>②③④… ②不满足为止。

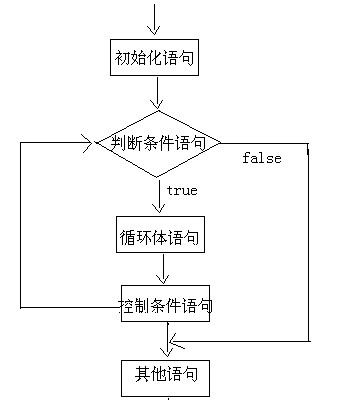
①负责完成循环变量初始化。

②负责判断是否满足循环条件，不满足则跳出循环。

③具体执行的语句。

④循环后，循环变量的变化情况。

* 流程图



* 练习
  + 输出10次HelloWorld

**public** **class** Demo01While {

**public** **static** **void** main(String[] args) {

//使用for循环打印10次HelloWorld

**for** (**int** i = 1; i <=10; i++) {

System.***out***.println("HelloWorld");

}

System.***out***.println("--------------------");

//while循环实现打印10次HelloWorld

**int** i = 1;

**while**(i<=10){

System.***out***.println("HelloWorld");

i++;

}

}

}

* + 求出1-100之和

**public** **class** Demo02While {

**public** **static** **void** main(String[] args) {

//回顾for循环 求1-100之和

//定义一个变量,记录累加求和

**int** sum = 0;

//利用for循环获取1-100之间的数字

**for** (**int** i = 1; i <= 100; i++) {

//累加求和

sum += i ;

}

//打印求和的变量

System.***out***.println("1-100的和是："+sum);

//使用while循环实现

//定义一个变量,记录累加求和

sum = 0;

//定义初始化表达式

**int** i = 1;

//使用while循环让初始化表达式的值变化

**while**(i<=100){

//累加求和

sum += i ;

//步进表达式改变变量的值

i++;

}

//打印求和的变量

System.***out***.println("1-100的和是："+sum);

}

}

* + 统计水仙花数有多少个(学生自己练习)

### 循环结构do-while

* 格式
  + 基本格式

do {

循环体语句;

}while(布尔表达式)**;**

* + 扩展格式

初始化表达式①

do{

循环体③

步进表达式④

}while(布尔表达式②)**;**

* 执行流程

执行顺序： ①③④>②③④>②③④… ②不满足为止。

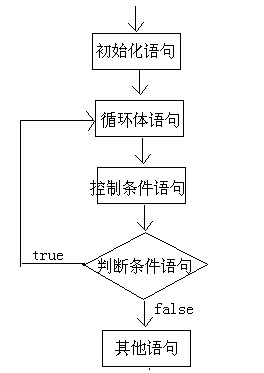
①负责完成循环变量初始化。

②负责判断是否满足循环条件，不满足则跳出循环。

③具体执行的语句

④循环后，循环变量的变化情况

* 流程图



* 练习
  + 输出10次HelloWorld

**public** **class** Demo01DoWhile {

**public** **static** **void** main(String[] args) {

//输出10次 HelloWorld

/\*

for(int x=1; x<=10; x++) {

System.out.println("HelloWorld");

}

\*/

//do...while改写

**int** x=1;

**do** {

System.***out***.println("HelloWorld");

x++;

}**while**(x<=10);

}

}

* + 求出1-100之和(学生自己练习)
  + 统计水仙花数有多少个(学生自己练习)

### 三种循环的区别

* + 虽然可以完成同样的功能，但是还是有小区别：
    - do…while循环至少会执行一次循环体。
    - for循环和while循环只有在条件成立的时候才会去执行循环体
  + for循环语句和while循环语句的小区别：
    - 使用区别：控制条件语句所控制的那个变量，在for循环结束后，就不能再被访问到了，而while循环结束还可以继续使用，如果你想继续使用，就用while，否则推荐使用for。原因是for循环结束，该变量就从内存中消失，能够提高内存的使用效率。
  + 循环使用推荐for -- while -- do..while

**public** **class** Demo02DoWhile {

**public** **static** **void** main(String[] args) {

/\*

int x = 3;

while(x<3) {

System.out.println("我爱柳岩");

x++;

}

System.out.println("--------------");

int y = 3;

do {

System.out.println("我爱柳岩");

y++;

}while(y<3);

\*/

**for**(**int** x=1; x<=10; x++){

System.***out***.println("爱生活，爱Java");

}

//这里的x无法继续访问

//System.out.println(x);

System.***out***.println("-----------------");

**int** y = 1;

**while**(y<=10) {

System.***out***.println("爱生活，爱Java");

y++;

}

System.***out***.println(y);

}

}

## 跳转控制语句

* 概述

我想让循环在某一步的时候结束或者跳过某些数据不要这些数据，现在就做不了这件事情。为了弥补这个缺陷，Java就提供了break，continue来实现控制语句的中断和跳转。

* + break 中断
  + continue 继续
* Break
  + break的使用场景：
    - 在选择结构switch语句中
    - 在循环语句中
    - 离开使用场景的存在是没有意义的
  + break的作用：
    - 跳出单层循环

**public** **class** Demo01Break {

**public** **static** **void** main(String[] args) {

//离开使用场景没有意义

//break;//break cannot be used outside of a loop or a switch

**for** (**int** i = 1; i <=10; i++) {

//需求:打印完两次HelloWorld之后结束循环

**if**(i == 3){

**break**;

}

System.***out***.println("HelloWorld"+i);

}

}

}

* Continue
  + continue的使用场景：
    - 在循环语句中
    - 离开使用场景的存在是没有意义的
  + continue的作用：
    - 单层循环对比break，然后总结两个的区别

break 结束当前循环

continue 结束本次循环,继续下一次循环

**public** **class** Demo02Continue {

**public** **static** **void** main(String[] args) {

//离开使用场景没有意义

//continue;//离开使用场景没有意义

**for** (**int** i = 1; i <=10; i++) {

//需求:不打印第三次HelloWorld

**if**(i == 3){

**continue**;

}

System.***out***.println("HelloWorld"+i);

}

}

}

* 练习: 按要求分析结果，并验证

**public** **class** Demo03BreakAndContinue {

**public** **static** **void** main(String[] args) {

**for** (**int** i = 1; i <= 10; i++) {

**if** (i % 3 == 0) {

// 分别写break，continue，说说输出几次

//break;//输出2次

**continue**;//输出7次

}

System.***out***.println("我爱Java!"+i);

}

}

}

## 死循环(永真循环)

* 概述

永远都停不下来的循环,就叫死循环. 死循环不一定就是错误,有可能有意为之.

* 死循环格式
  + while格式的死循环(常用)

while (true) {

循环体;

}

* + for格式的死循环(了解)

for(;;) {

循环体;

}

* 举例

