## 一、While循环

while( 布尔表达式 ) {

//循环内容

}

先判断后执行

循环体中必须有改变循环变量（循环体）的数值，若无则有可能形成死循环

**public** **class** Test {

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

**int** a = 10;

**while**(a < 20) {

System.***out***.println("a=" + a++);

}

}

}

**结果：10 11 12 13 14 15 16 17 18 19**

## 二、do…while循环

do {

//代码语句

} while(布尔表达式);

do…while 循环和 while 循环相似，不同的是，do…while 循环至少会执行一次。

先执行do循环体内的内容，然后判断条件，若条件成立，则继续循环；若条件不成立，则结束循环

示例一：

**public** **class** Test {

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

**int** a = 10;

**do** {

System.***out***.println(a++);

} **while**(a==10);

}

}

**结果：10**

示例二

**public** **class** Test {

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

**int** a = 10;

**do** {

System.***out***.println(a++);

} **while**(a < 20);

}

}

**结果：**

**10 11 12 13 14 15 16 17 18 19**

**While和do…while比较**

|  |  |  |
| --- | --- | --- |
| **比较** | **While** | **do…while** |
| **执行循序** | **先判断，再执行循环体** | **先执行循环体一次，再判断** |
| **最少执行次数** | **0** | **1** |

## 三、For循环

for(初始化; 布尔表达式; 更新) {

//代码语句

}

 for 循环说明：

* 最先执行初始化步骤。可以声明一种类型，但可初始化一个或多个循环控制变量，也可以是空语句。
* 然后，检测布尔表达式的值。如果为 true，循环体被执行。如果为false，循环终止，开始执行循环体后面的语句。
* 执行一次循环后，更新循环控制变量。
* 再次检测布尔表达式。循环执行上面的过程。

示例一（常用）：

**for** (**int** i = 0; i < 5; i++) {

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

}

示例二：

**int** i = 0;

**for** (; i < 5; i++) {

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

}

示例三：

**int** i = 0;

**for** (; i < 5; ) {

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

i++;

}

**四、增强for**

Java5 引入了一种主要用于数组的增强型 for 循环。

for(类型 声明语句 : 表达式) {

//代码句子

}

**声明语句：**声明新的局部变量，该变量的类型必须和数组元素的类型匹配。其作用域限定在循环语句块，其值与此时数组元素的值相等。

**表达式：**表达式是要访问的数组名，或者是返回值为数组的方法。

//定义一个数组

int [] numbers = {10, 20, 30, 40, 50};

for (int x : numbers ){

System.out.print(x);

System.out.print(",");

}

结果：10,20,30,40,50

## 五、break和continue

break和continue用于跳出语句块

区别：

break是跳出语句块不在进入；

continue是跳出本次，下次还会进入

break可用于while、do…while、for、增强for、switch…case

continue可用于while、do…while、for、增强for

**break示例：**

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

**for** (**int** i = 0; i < 10; i++) {

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

//跳出循环

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

**break**;

}

}

}

**continue示例:**

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

**for** (**int** i = 0; i < 10; i++) {

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

//跳出循环

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

**continue**;

}

}

}

## 六、嵌套for循环

使用for循环输出\*号，每行\*号递减

**for** (**int** i = 0; i < 4; i++) {

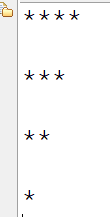
**for** (**int** j = i; j < 5-1; j++) {

System.***out***.print("\*");

}

System.***out***.println("\n");

}



## 七、switch…case语句

switch(expression){

case value : //语句

break; //可选

case value : //语句

break; //可选 //你可以有任意数量的case语句

default : //可选 //语句

}

switch 语句规则：

* switch 语句中的变量类型可以是： byte、short、int 或者 char。从 Java SE 7 开始，switch 支持字符串类型了，同时 case 标签必须为字符串常量或字面量。
* switch 语句可以拥有多个 case 语句。每个 case 后面跟一个要比较的值和冒号。
* case 语句中的值的数据类型必须与变量的数据类型相同，而且只能是常量或者字面常量。
* 当变量的值与 case 语句的值相等时，那么 case 语句之后的语句开始执行，直到 break 语句出现才会跳出 switch 语句。
* 当遇到 break 语句时，switch 语句终止。程序跳转到 switch 语句后面的语句执行。case 语句不必须要包含 break 语句。如果没有 break 语句出现，程序会继续执行下一条 case 语句，直到出现 break 语句。
* switch 语句可以包含一个 default 分支，该分支必须是 switch 语句的最后一个分支。default 在没有 case 语句的值和变量值相等的时候执行。default 分支不需要 break 语句。

示例一：

int iaction=4;

switch (iaction)

{

    case 1:System.out.print(iaction);break;

    case 4:System.out.print(iaction);break;

}

示例二：

char caction='a';

switch (caction)

{

    case 1:System.out.print(caction);break;

    case 'a':System.out.print(caction);break;

}

示例三：

String straction="abc";

switch (straction)

{

    case "1":System.out.print(straction);break;

    case "abc":System.out.print(straction);break;

}

示例四：

byte a = 2;

  switch(a){

  case 1 : System.out.println(" A ");

  case 2 : System.out.println(" B ");

  case 3 : System.out.println(" C ");

  case 4 : System.out.println(" D ");

  default : System.out.println(" default ");

  }

=========输出结果为:

B

C

D

default

--------------------------

## 八．if…else

if 语句后面可以跟 else 语句，当 if 语句的布尔表达式值为 false 时，else 语句块会被执行。

if(布尔表达式){

//如果布尔表达式的值为true

}else{

//如果布尔表达式的值为false

}

int x = 30;

if( x < 20 ){

System.out.print("这是 if 语句");

}else{

System.out.print("这是 else 语句");

}

if(布尔表达式 1){

//如果布尔表达式 1的值为true执行代码

}else if(布尔表达式 2){

//如果布尔表达式 2的值为true执行代码

}else if(布尔表达式 3){

//如果布尔表达式 3的值为true执行代码

}else {

//如果以上布尔表达式都不为true执行代码

}

int x = 30;

if( x == 10 ){

System.out.print("Value of X is 10");

}else if( x == 20 ){

System.out.print("Value of X is 20");

}else if( x == 30 ){

System.out.print("Value of X is 30");

}else{

System.out.print("这是 else 语句");

}

嵌套的 if…else 语句

if(布尔表达式 1){

////如果布尔表达式 1的值为true执行代码

if(布尔表达式 2){

////如果布尔表达式 2的值为true执行代码

}

}

int x = 30;

int y = 10;

if( x == 30 ){

if( y == 10 ){

System.out.print("X = 30 and Y = 10");

}

}

## 九、数组

数组是一组相关数据的集合，一个数组实际上就是一连串的变量，数组按照使用可以分为一维数组、二维数组、多维数组

Java 语言中提供的数组是用来存储固定大小的同类型元素。

**数组创建方式：**

1. datatype[] array = new datatype[arraysize];
2. datatype[] array = {value0, value1, value2, ….,valuek};

示例一：

public class TestArray {

public static void main(String[] args) {

// 数组大小

int size = 10;

// 定义数组

double[] myList = new double[size];

myList[0] = 5.6;

myList[1] = 4.5;

myList[2] = 3.3;

myList[3] = 13.2;

myList[4] = 4.0;

myList[5] = 34.33;

myList[6] = 34.0;

myList[7] = 45.45;

myList[8] = 99.993;

myList[9] = 11123; // 计算所有元素的总和

double total = 0;

for (int i = 0; i < size; i++) {

total += myList[i];

}

System.out.println("总和为： " + total);

}

}

## foreach 循环

JDK 1.5 引进了一种新的循环类型，被称为 foreach 循环或者加强型循环，它能在不使用下标的情况下遍历数组。

**public** **class** Test {

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

**int**[] num = {1, 2, 3, 4, 5, 6, 7}; //定义int型数组

**for** (**int** i : num) {

System.***out***.println("第" + (++i) + "个数字是：" + i);

}

}

}

## 数组作为函数的参数

数组可以作为参数传递给方法。

例如，下面的例子就是一个打印 int 数组中元素的方法:

public static void printArray(int[] array) {

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

System.out.print(array[i] + " ");

}

}

public class Test {

public static void main(String[] args) {

int[] num = {1, 2, 3, 4, 5, 6, 7}; //定义int型数组

for (int i : Nums(num)) {

System.out.println(i);

}

}

/\*\*

\* 数组作为参数

\* @param num

\* @return

\*/

public static int[] Nums(int[] num) {

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

num[i] = num[i] + 1;

}

return num;

}

}

## 多维数组

多维数组可以看成是数组的数组，比如二维数组就是一个特殊的一维数组，其每一个元素都是一个一维数组，例如：

String str[][] = new String[3][4];

### 多维数组的动态初始化（以二维数组为例）

1. 直接为每一维分配空间，格式如下：

type arrayName = new type[arraylenght1][arraylenght2];

type 可以为基本数据类型和复合数据类型，arraylenght1 和 arraylenght2 必须为正整数，arraylenght1 为行数，arraylenght2 为列数。

例如：

int a[][] = new int[2][3];

解析：

二维数组 a 可以看成一个两行三列的数组。

2. 从最高维开始，分别为每一维分配空间，例如：

String s[][] = new String[2][];

s[0] = new String[2];

s[1] = new String[3];

s[0][0] = new String("Good");

s[0][1] = new String("Luck");

s[1][0] = new String("to");

s[1][1] = new String("you");

s[1][2] = new String("!");

解析：

**s[0]=new String[2]** 和 **s[1]=new String[3]** 是为最高维分配引用空间，也就是为最高维限制其能保存数据的最长的长度，然后再为其每个数组元素单独分配空间 **s0=new String("Good")** 等操作。

### 多维数组的引用（以二维数组为例）

对二维数组中的每个元素，引用方式为 **arrayName[index1][index2]**，例如：

num[1][0];

## Arrays 类

java.util.Arrays 类能方便地操作数组，它提供的所有方法都是静态的。

具有以下功能：

* 给数组赋值：通过 fill 方法。

Arrays.fill(num, 1); //改变所有值

Arrays.*fill*(num, 1, 2, 1); //改变某个区间的值（只改变左边的值）

* 对数组排序：通过 sort 方法,按升序。

Arrays.sort(num);

Arrays.sort(num, 2, 5);

* 比较数组：通过 equals 方法比较数组中元素值是否相等。

Arrays.equals(num, num1);

* 查找数组元素：通过 binarySearch 方法能对排序好的数组进行二分查找法操作。

Arrays.binarySearch(num, 1);

具体说明请查看下表：

|  |  |
| --- | --- |
| **序号** | **方法和说明** |
| 1 | **public static int binarySearch(Object[] a, Object key)** 用二分查找算法在给定数组中搜索给定值的对象(Byte,Int,double等)。数组在调用前必须排序好的。如果查找值包含在数组中，则返回搜索键的索引；否则返回 (-(*插入点*) - 1)。 |
| 2 | **public static boolean equals(long[] a, long[] a2)** 如果两个指定的 long 型数组彼此*相等*，则返回 true。如果两个数组包含相同数量的元素，并且两个数组中的所有相应元素对都是相等的，则认为这两个数组是相等的。换句话说，如果两个数组以相同顺序包含相同的元素，则两个数组是相等的。同样的方法适用于所有的其他基本数据类型（Byte，short，Int等）。 |
| 3 | **public static void fill(int[] a, int val)** 将指定的 int 值分配给指定 int 型数组指定范围中的每个元素。同样的方法适用于所有的其他基本数据类型（Byte，short，Int等）。 |
| 4 | **public static void sort(Object[] a)** 对指定对象数组根据其元素的自然顺序进行升序排列。同样的方法适用于所有的其他基本数据类型（Byte，short，Int等）。 |