# Lesson07 Java 基础语法

## 今日内容介绍

* 流程控制语句（if,switch,for,while,break,continue）
* 循环嵌套
* 数组

# 第1章 流程控制语句

## 顺序结构

顺序结构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

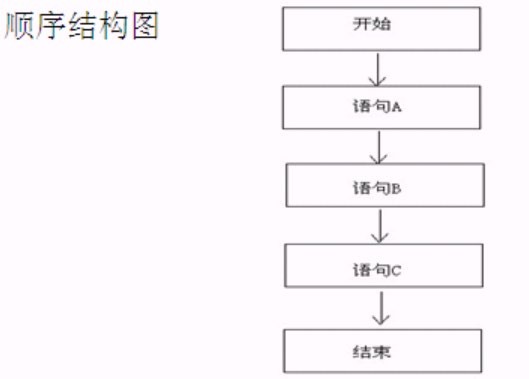
public class ShunXuDemo{

public static void main(String[] args){

System.out.println(“ 第 一 步 ”); System.out.prinktln(“ 第 二 步 ”); System.out.println(“最后一步”);

}

}



## 选择结构

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

### if 语句

接下来要学习的 if 条件语句分为三种语法格式，每一种格式都有它自身的特点，下面我们分别进行介绍。j

if 语句是指如果满足某种条件，就进行某种处理。例如，小明妈妈跟小明说“如果你考试得了

100 分，星期天就带你去游乐场玩”。这句话可以通过下面的一段伪代码来描述。

如果小明考试得了 100 分

妈妈星期天带小明去游乐场

在上面的伪代码中，“如果”相当于 Java 中的关键字 if，“小明考试得了 100 分”是判断条件，需要用()括起来，“妈妈星期天带小明去游乐场”是执行语句，需要放在{}中。修改后的伪代码如下：

if (小明考试得了 100 分) {

妈妈星期天带小明去游乐场

}

上面的例子就描述了 if 语句的用法，在 Java 中，if 语句的具体语法格式如下：

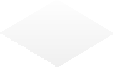
if (条件语句){

执行语句;

……

}

上述格式中，判断条件是一个布尔值，当判断条件为 true 时，{}中的执行语句才会执行。if 语句的执行流程如下图所示。



判断条件

true

false

执行语句 2

执行语句 1

图1-1 if 语句流程图

接下来通过一段代码，学习一下 if 语句的具体用法，IfDemo01.java

**public class** IfDemo01 {

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

**int** x = 5;

**if** (x < 10) { x++;

}

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

}

}

运行结果如下图所示。



图1-2 运行结果

在上述代码中，定义了一个变量 x，其初始值为 5。在 if 语句的判断条件中判断x 的值是否小于10，很明显条件成立，{}中的语句会被执行，变量 x 的值将进行自增。从运行结果可以看出，x 的值已由原来的 5 变成了 6。

### if…else 语句

if…else 语句是指如果满足某种条件，就进行某种处理，否则就进行另一种处理。例如，要判断一个正整数的奇偶，如果该数字能被 2 整除则是一个偶数，否则该数字就是一个奇数。if…else 语句具体语法格式如下：

if (判断条件){

执行语句 1

……

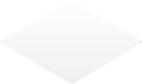
}else{

执行语句 2

……

}

上述格式中，判断条件是一个布尔值。当判断条件为true 时，if 后面{}中的执行语句 1 会执行。当判断条件为 false 时，else 后面{}中的执行语句 2 会执行。if…else 语句的执行流程如下图所示。



判断条件

true

false

执行语句 2

执行语句 1

图1-3 if…else 语句流程图接下来通过一段代码，来实现判断奇偶数的程序，IfDemo02.java

**public class** IfDemo02 {

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

**int** num = 19;

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

// 判断条件成立，num 被 2 整除

System.*out*.println("num 是一个偶数");

} **else** {

System.*out*.println("num 是一个奇数");

}

}

}

运行结果如下图所示。

图1-4 运行结果

上述代码中，变量 num 的值为 19，模以 2 的结果为 1，不等于 0，判断条件不成立。因此会执行else 后面{}中的语句，打印“num 是一个奇数”。

### if…else if…else 语句

if…else if…else 语句用于对多个条件进行判断，进行多种不同的处理。例如，对一个学生的考试成绩进行等级的划分，如果分数大于 80 分等级为优，否则，如果分数大于 70 分等级为良，否则， 如果分数大于 60 分等级为中，否则，等级为差。if…else if…else 语句具体语法格式如下：

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) {

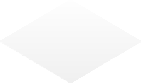
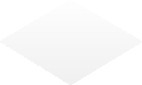
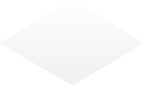
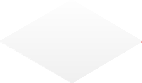
执行语句 n

} else {

执行语句 n+1

}

上述格式中，判断条件是一个布尔值。当判断条件 1 为 true 时，if 后面{}中的执行语句 1 会执行。当判断条件 1 为 false 时，会继续执行判断条件 2，如果为true 则执行语句 2，以此类推，如果所有的判断条件都为false，则意味着所有条件均未满足，else 后面{}中的执行语句 n+1 会执行。if…else if…else 语句的执行流程如下图所示。



判断条件1

true

false

true

判断条件2

false

true

......

false

true

判断条件n

false

执行语句n+1

执行语句n

......

执行语句2

执行语句1

图1-5 if…else if…else 语句的流程图接下来通过一段代码，来实现对学生考试成绩进行等级划分的程序， 从键盘录入一个成绩进行判断,根据要求打印结果:

成绩在[0,60)打印该成绩为差成绩在[60,70)打印该成为中 成绩在[70,80)打印该成绩为良成绩在[80,100]打印该成绩为优

**public class** IFDemo {

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

Scanner scan=**new** Scanner(System.*in*); **int** score=scan.nextInt(); if(score>=0 && score<60){

System.out.println("打印成绩为差");

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

System.out.println("打印成绩为中");

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

System.out.println("打印成绩为良");

}else if(score>=80 && score<=100){

System.out.println("打印成绩为优");

}

}

}

## 选择结构switch

switch 条件语句也是一种很常用的选择语句，它和 if 条件语句不同，它只能针对某个表达式的

值作出判断，从而决定程序执行哪一段代码。例如，在程序中使用数字 1~7 来表示星期一到星期天， 如果想根据某个输入的数字来输出对应中文格式的星期值，可以通过下面的一段伪代码来描述：

用于表示星期的数字

如果等于 1，则输出星期一如果等于 2，则输出星期二如果等于 3，则输出星期三如果等于 4，则输出星期四如果等于 5，则输出星期五如果等于 6，则输出星期六如果等于 7，则输出星期天

对于上面一段伪代码的描述，大家可能会立刻想到用刚学过得 if…else if…else 语句来实现，但是由于判断条件比较多，实现起来代码过长，不便于阅读。Java 中提供了一种switch 语句来实现这种需求，在 switch 语句中使用switch 关键字来描述一个表达式，使用 case 关键字来描述和表达式结果比较的目标值，当表达式的值和某个目标值匹配时，会执行对应case 下的语句。具体实现代码如下：

switch(用于表示星期的数字) { case 1 :

输出星期一; break;

case 2 :

输出星期二; break;

case 3 :

输出星期三

break; case 4 :

输出星期四;

break; case 5 :

输出星期五;

break; case 6:

输出星期六;

break; case 7:

输出星期天;

break;

}

* 上面改写后的伪代码便描述了 switch 语句的基本语法格式，具体如下：

switch (表达式){ case 目标值 1:

执行语句 1 break;

case 目标值 2:

执行语句 2 break;

．．．．．．

case 目标值 n:

执行语句 n break;

default:

执行语句 n+1 break;

}

在上面的格式中，switch 语句将表达式的值与每个 case 中的目标值进行匹配，如果找到了匹配的值，会执行对应case 后的语句，如果没找到任何匹配的值，就会执行 default 后的语句。switch 语句中的 break 关键字将在后面的做具体介绍，此处，我们只需要知道 break 的作用是跳出 switch 语句即可。

需要注意的是，在 switch 语句中的表达式只能是 byte、short、char、int 类型的值，如果传入其它类型的值，程序会报错。但上述说法并不严谨，实际上在 JDK5.0 中引入的新特性 enum 枚举也可以作为 switch 语句表达式的值，在 JDK7.0 中也引入了新特性，switch 语句可以接收一个 String 类型的值。

## 选择结构switch 练习

接下来通过一个案例演示根据数字来输出中文格式的星期，如下所示。SwitchDemo01.java

**public class** SwitchDemo01 {

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

**int** week = 5; **switch** (week) {

**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("星期天");

**break**; **default**:

System.*out*.println("输入的数字不正确...");

**break**;

}

}

}

运行结果如图所示。

图1-6 运行结果

上述代码中，由于变量 week 的值为 5，整个 switch 语句判断的结果满足第 17 行的条件，因此打印“星期五”，例程中的 default 语句用于处理和前面的case 都不匹配的值，将第 3 行代码替换为 int week = 8，再次运行程序，输出结果如下图所示。

图1-7 运行结果

## 循环结构

### 循环语句 while

while 循环语句和选择结构 if 语句有些相似，都是根据条件判断来决定是否执行大括号内的执行语句。区别在于，while 语句会反复地进行条件判断，只要条件成立，{}内的执行语句就会执行，直到条件不成立，while 循环结束。while 循环语句的语法结构如下：

while(循环条件){

执行语句

………

}

在上面的语法结构中，{}中的执行语句被称作循环体，循环体是否执行取决于循环条件。当循环 条件为true 时，循环体就会执行。循环体执行完毕时会继续判断循环条件，如条件仍为 true 则会继续执行，直到循环条件为false 时，整个循环过程才会结束。

while 循环的执行流程如下图所示。

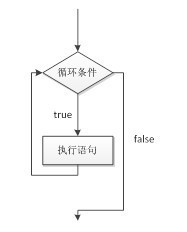


图1-8 while 循环的流程图

接下来通过一段代码，来实现打印 1~4 之间的自然数，WhileDemo.java

**public class** WhileDemo {

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

**int** x = 1; // 定义变量 x，初始值为 1 **while** (x <= 4) { // 循环条件

System.*out*.println("x = " + x); // 条件成立，打印 x 的值

x++; // x 进行自增

}

}

}

运行结果如下图所示。

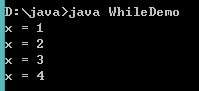


图1-9 运行结果

在上述代码中，x 初始值为 1，在满足循环条件 x <= 4 的情况下，循环体会重复执行，打印x 的值并让x 进行自增。因此打印结果中x 的值分别为 1、2、3、4。

大家要注意的是，代码 x++用于在每次循环时改变变量 x 的值，从而达到最终改变循环条件的目的。如果没有这行代码，整个循环会进入无限循环的状态，永远不会结束。

### 循环语句 for

for 循环语句是最常用的循环语句，一般用在循环次数已知的情况下。for 循环语句的语法格式如下：

for（初始化表达式; 循环条件; 操作表达式）{ 执行语句

………

}

在上面的语法结构中，for 关键字后面()中包括了三部分内容：初始化表达式、循环条件和操作表达式，它们之间用“;”分隔，{}中的执行语句为循环体。

接下来分别用①表示初始化表达式、②表示循环条件、③表示操作表达式、④表示循环体，通

过序号来具体分析 for 循环的执行流程。具体如下：

for（① ; ② ; ③）{

④

}

第一步，执行①

第二步，执行②，如果判断结果为 true，执行第三步，如果判断结果为 false，执行第五步第三步，执行④

第四步，执行③，然后重复执行第二步

第五步，退出循环

接下来通过一个案例对自然数 1~4 进行求和，如下所示。ForDemo01.java

**public class** ForDemo01 {

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

**int** sum = 0; // 定义变量 sum，用于记住累加的和

**for** (**int** i = 1; i <= 4; i++) { // i 的值会在 1~4 之间变化

sum += i; // 实现 sum 与 i 的累加

}

System.*out*.println("sum = " + sum); // 打印累加的和

}

}

运行结果如下图所示。

图1-10 运行结果

上述代码中，变量i 的初始值为 1，在判断条件i<=4 为true 的情况下，会执行循环体 sum+=i， 执行完毕后，会执行操作表达式i++，i 的值变为2，然后继续进行条件判断，开始下一次循环，直到i=5 时，条件i<=4 为 false，结束循环，执行 for 循环后面的代码，打印“sum=10”。

为了让初学者能熟悉整个 for 循环的执行过程，现将上述代码运行期间每次循环中变量 sum 和i

的值通过表 2-11 罗列出来。

表2-1 sum 和i 循环中的值

|  |  |  |
| --- | --- | --- |
| **循环次数** | **sum** | **i** |
| 第一次 | 1 | 1 |
| 第二次 | 3 | 2 |
| 第三次 | 6 | 3 |
| 第四次 | 10 | 4 |

### for 循环案例 for 循环实现 1-5 之间数据求和

/\*

* 需求：求出 1-5 之间数据之和

\*

* 分析：

\*

\*

\*

\*

\*/

A:定义求和变量，初始化值是 0

B:获取 1-5 之间的数据，用 for 循环实现C: 把每一次获取到的数据，累加起来就可以了D: 输出求和变量即可

**public class** ForTest2 {

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

//定义求和变量，初始化值是 0

**int** sum = 0;

//获取 1-5 之间的数据，用 for 循环实现

**for**(**int** x=1; x<=5; x++) {

//把每一次获取到的数据，累加起来就可以了

//sum = sum + x;

/\*

\* 第一次：sum = 0 + 1 = 1

\* 第二次：sum = 1 + 2 = 3

\* 第三次：sum = 3 + 3 = 6

\* 第四次：sum = 6 + 4 = 10

\* 第五次：sum = 10 + 5 = 15

\*/

sum += x;

}

//输出求和结果

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

}

}

* + - 1. **for 循环实现 1-10 之间偶数和**

**package** com.itheima\_04;

/\*

* 需求：求出 1-10 之间偶数和

\*

* 分析：
* A:定义求和变量，初始化值是 0
* B:获取 1-10 之间的数据，用 for 循环实现
* C:把获取到的数据进行判断，看是否是偶数
* 如果是，就累加
* D:输出求和结果

\*/

**public class** ForTest3 {

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

//定义求和变量，初始化值是 0

**int** sum = 0;

//获取 1-100 之间的数据，用 for 循环实现

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

//把获取到的数据进行判断，看是否是偶数

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

sum += x;

}

}

//输出求和结果System.***out***.println("sum:"+sum);

}

}

* + 1. **无限循环**

最简单无限循环格式：

while(true){}

或

for(;;){}

无限循环存在的原因是并不知道循环多少次，而是根据某些条件，来控制循环。

## 跳转语句（break、continue）

跳转语句用于实现循环执行过程中程序流程的跳转，在 Java 中的跳转语句有 break 语句和

continue 语句。接下来分别进行详细地讲解。

#### 1、 break 语句

当 break 出现在循环语句中，作用是跳出循环语句，执行后面的代码，接下来通过下面一段代码，实现将当变量x 的值为 3 时，使用break 语句跳出循环，代码如下所示。BreakDemo.java

**public class** BreakDemo {

**public static void** main(String[] args) { **int** x = 1; // 定义变量 x，初始值为 1 **while** (x <= 4) { // 循环条件

System.*out*.println("x = " + x); // 条件成立，打印 x 的值

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

**break**;

}

x++; // x 进行自增

}

}

}

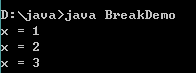
运行结果如下图所示。

图1-11 运行结果

在上述带代码中，通过while 循环打印x 的值，当x 的值为 3 时使用break 语句跳出循环。因此打印结果中并没有出现“x=4”。

#### 2、 continue 语句

continue 语句用在循环语句中，它的作用是终止本次循环，执行下一次循环。接下来通过一个练习对 1~100 之内的奇数求和，ContinueDemo.java

**public class** ContinueDemo {

**public static void** main(String[] args) { **int** sum = 0; // 定义变量 sum，用于记住和**for** (**int** i = 1; i <= 100; i++) {

**if** (i % 2 == 0) { // i 是一个偶数，不累加

**continue**; // 结束本次循环

}

sum += i; // 实现 sum 和 i 的累加

}

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

}

}

运行结果如下图所示。



图1-12 运行结果

上述代码中，使用 for 循环让变量 i 的值在 1~100 之间循环，在循环过程中，当 i 的值为偶数时， 将执行continue 语句结束本次循环，进入下一次循环。当 i 的值为奇数时，sum 和i 进行累加，最终得到 1~100 之间所有奇数的和，打印“sum = 2500”。

在嵌套循环语句中，continue 语句后面也可以通过使用标记的方式结束本次外层循环，用法与

break 语句相似，在此不再举例说明。

## 循环嵌套

嵌套循环是指在一个循环语句的循环体中再定义一个循环语句的语法结构。while、do…while、

for 循环语句都可以进行嵌套，并且它们之间也可以互相嵌套，如最常见的在 for 循环中嵌套 for 循

环，格式如下：

for(初始化表达式; 循环条件; 操作表达式) {

………

for(初始化表达式; 循环条件; 操作表达式) {

执行语句

………

}

………

}

接下来通过一个练习，来实现使用“\*”打印矩形，如下所示。ForForDemo.java

1. **public class** ForForDemo {
2. **public static void** main(String[] args) {
3. **int** i, j; // 定义两个循环变量

4 **for** (i = 1; i <= 3; i++) { // 外层循环

5 **for** (j = 1; j <= 4; j++) { // 内层循环

6 System.*out*.print("\*"); // 打印\* 7 }

8 System.*out*.print("\n"); // 换行

9 }

10 }

11 }

\*\*\*\*

\*\*\*\*

\*\*\*\*

在上述代码中定义了两层 for 循环，分别为外层循环和内层循环，外层循环用于控制打印的行数，内层循环用于打印“\*”,过程如下:

从外到内执行

* + 1. int j=0 2.j<3 true

3.进入内层循环 int i=0 4.i<4 true

5.打印\*

6.i++ i=1

7.i<4 true 8.打印\*

9.i++ i=2

10.i<4 true 11.打印\*

12.i++ i=3

10.i<4 true 11.打印\*

12.i++ i=4

13.i<4 false 14.System.out.println();

15.j++ j=1

16.j<3 true

17.重复 4~14 步

18.j++ j=2

19.j<3 true

20.重复 4~14 步

21.j++ j=3

1. j<3 false
2. 外层 for 结束,整个 for 结束

# 第2章 数组

在生活中，我们可能会碰到如下的场景。

现在需要统计某公司员工的工资情况，例如计算平均工资、最高工资等。假设该公司有50 名员工，用前面所学的知识完成，那么程序首先需要声明 50 个变量来分别记住每位员工的工资，这样做会显得很麻烦。

其实在Java 中，我们可以使用一个数组来记住这 50 名员工的工资。数组是指一组数据的集合， 数组中的每个数据被称作元素。在数组中可以存放任意类型的元素，但同一个数组里存放的元素类 型必须一致。

## 数组的定义

在Java 中，可以使用以下格式来定义一个数组。如下

数据类型[] 数组名 = new 数据类型[元素个数或数组长度];

int[] x = new int[100];

上述语句就相当于在内存中定义了 100 个int 类型的变量，第一个变量的名称为 x[0]，第二个变量的名称为 x[1]，以此类推，第 100 个变量的名称为 x[99]，这些变量的初始值都是 0。为了更好地理解数组的这种定义方式，可以将上面的一句代码分成两句来写，具体如下：

int[] x; // 声明一个 int[]类型的变量

x = new int[100]; // 创建一个长度为 100 的数组

接下来，通过两张内存图来详细地说明数组在创建过程中内存的分配情况。

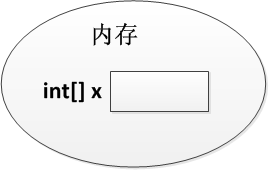
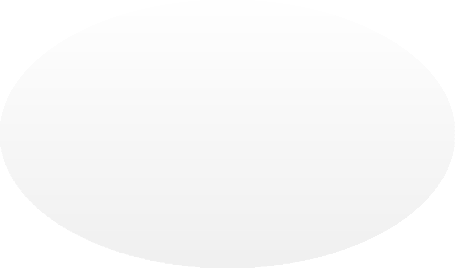
第一行代码 int[] x; 声明了一个变量x，该变量的类型为 int[]，即一个 int 类型的数组。变量x 会占用一块内存单元，它没有被分配初始值。内存中的状态如下图所示。

图1-13 内存状态图

第二行代码 x = new int[100]; 创建了一个数组，将数组的地址赋值给变量 x。在程序运行期间可以使用变量x 来引用数组，这时内存中的状态会发生变化，如下图所示。



内存

**new int[100]**

（数组的首地址） **0**

**X[0]**

**0x3000**

**int[] x**

**0 X[1]**

……

……

**0 X[98]**

**0 X[99]**

**0x3000**

图1-14 内存状态图

在上图中描述了变量x 引用数组的情况。该数组中有 100 个元素，初始值都为 0。数组中的每个元素都有一个索引(也可称为角标)，要想访问数组中的元素可以通过“x[0]、x[1]、……、x[98]、x[99]” 的形式。需要注意的是，数组中最小的索引是 0，最大的索引是“数组的长度-1”。在 Java 中，为了方便我们获得数组的长度，提供了一个 length 属性，在程序中可以通过“数组名.length”的方式来获得数组的长度，即元素的个数。

接下来，通过一个案例来演示如何定义数组以及访问数组中的元素，如下所示。

ArrayDemo01.java

1. **public class** ArrayDemo01 {
2. **public static void** main(String[] args) {
3. **int**[] arr; // 声明变量
4. arr = **new int**[3]; // 创建数组对象
5. System.*out*.println("arr[0]=" + arr[0]); // 访问数组中的第一个元素
6. System.*out*.println("arr[1]=" + arr[1]); // 访问数组中的第二个元素
7. System.*out*.println("arr[2]=" + arr[2]); // 访问数组中的第三个元素
8. System.*out*.println("数组的长度是：" + arr.length); // 打印数组长度

9 }

10 }

运行结果如下图所示。

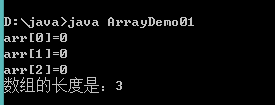


图1-15 运行结果

在上述代码中声明了一个int[]类型变量 arr，并将数组在内存中的地址赋值给它。在 5~7 行代码

中通过角标来访问数组中的元素，在第 8 行代码中通过 length 属性访问数组中元素的个数。从打印结果可以看出，数组中的三个元素初始值都为 0，这是因为当数组被成功创建后，数组中元素会被自动赋予一个默认值，根据元素类型的不同，默认初始化的值也是不一样的。具体如下表所示。

表2-2 元素默认值

|  |  |
| --- | --- |
| **数据类型** | **默认初始化值** |
| byte、short、int、long | 0 |
| float、double | 0.0 |
| char | 一个空字符，即’\u0000’ |
| boolean | false |
| 引用数据类型 | null，表示变量不引用任何对象 |

如果在使用数组时，不想使用这些默认初始值，也可以显式地为这些元素赋值。接下来通过一 个程序来学习如何为数组的元素赋值，如下所示。ArrayDemo02.java

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. **public class** ArrayDemo02 { 2. **public static void** main(String[] args) { 3. **int**[] arr = **new int**[4]; // 定义可以存储 4 个整数的数组 4. arr[0] = 1; // 为第 1 个元素赋值 1 5. arr[1] = 2; // 为第 2 个元素赋值 2 6. // 下面的代码是打印数组中每个元素的值 | | | | |
| 7 |  | System.*out*.println("arr[0]=" | + | arr[0]); |
| 8 |  | System.*out*.println("arr[1]=" | + | arr[1]); |
| 9 |  | System.*out*.println("arr[2]=" | + | arr[2]); |
| 10 |  | System.*out*.println("arr[3]=" | + | arr[3]); |
| 11 | } |  |  |  |
| 12 } |  |  |  |  |

运行结果如下图所示。

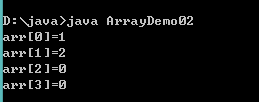


图1-16 运行结果

在上述代码中，第 3 行代码定义了一个数组，此时数组中每个元素都为默认初始值 0。第 2、3 行代码通过赋值语句将数组中的元素 arr[0]和 arr[1]分别赋值为 1 和 2，而元素arr[2]和 arr[3]没有赋值，其值仍为 0，因此打印结果中四个元素的值依次为 1、2、0、0。

在定义数组时只指定数组的长度，由系统自动为元素赋初值的方式称作动态初始化。

在初始化数组时还有一种方式叫做静态初始化，就是在定义数组的同时就为数组的每个元素赋 值。数组的静态初始化有两种方式，具体格式如下：

1、类型[] 数组名 = new 类型[]{元素，元素，……};

2、类型[] 数组名 = {元素，元素，元素，……};

上面的两种方式都可以实现数组的静态初始化，但是为了简便，建议采用第二种方式。接下来 通过一段代码来演示数组静态初始化的效果，如下所示。ArrayDemo03.java

1. **public class** ArrayDemo03 {
2. **public static void** main(String[] args) {

3 **int**[] arr = { 1, 2, 3, 4 }; // 静态初始化

|  |  |  |  |
| --- | --- | --- | --- |
| 4 |  | // 下面的代码是依次访问数组中的元素 |  |
| 5 |  | System.*out*.println("arr[0] = " + | arr[0]); |
| 6 |  | System.*out*.println("arr[1] = " + | arr[1]); |
| 7 |  | System.*out*.println("arr[2] = " + | arr[2]); |
| 8 |  | System.*out*.println("arr[3] = " + | arr[3]); |
| 9 | } |  |  |
| 10 } |  |  |  |

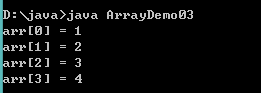
运行结果如下图所示。

图1-17 运行结果

上述代码中采用静态初始化的方式为数组每个元素赋予初值，分别是 1、2、3、4。需要注意的是，第 3 行代码千万不可写成 int[] arr = new int[4]{1,2,3,4};，这样写编译器会报错。原因在于编译器会认为数组限定的元素个数[4]与实际存储的元素{1,2,3,4}个数有可能不一致，存在一定的安全隐患。

## 数组遍历

在操作数组时，经常需要依次访问数组中的每个元素，这种操作称作数组的遍历。接下来通过 一个案例来学习如何使用 for 循环来遍历数组，如下所示。ArrayDemo04.java

**public class** ArrayDemo04 {

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

**int**[] arr = { 1, 2, 3, 4, 5 }; // 定义数组

// 使用 for 循环遍历数组的元素

**for** (**int** i = 0; i < arr.length; i++) { System.*out*.println(arr[i]); // 通过索引访问元素

}

}

}

运行结果如下图所示。



图1-18 运行结果

上述代码中，定义一个长度为5 的数组 arr，数组的角标为 0~4。由于 for 循环中定义的变量i 的值在循环过程中为 0~4，因此可以作为索引，依次去访问数组中的元素，并将元素的值打印出来。

## 数组求和

## 数组最值

在操作数组时，经常需要获取数组中元素的最值。接下来通过一个案例来演示如何获取数组中 元素的最大值，如下所示。ArrayDemo05.java

**public class** ArrayDemo05 {

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

**int**[] arr = { 4, 1, 6, 3, 9, 8 }; // 定义一个数组

**int** max = arr[0]; // 定义变量 max 用于记住最大数，首先假设第一个元素为最大值

// 下面通过一个 for 循环遍历数组中的元素

**for** (**int** x = 1; x < arr.length; x++) {

**if** (arr[x] > max) { // 比较 arr[x]的值是否大于 max max = arr[x]; // 条件成立，将 arr[x]的值赋给 max

}

}

System.*out*.println("max=" + max); // 打印最大值

}

}

运行结果如下图所示。



图1-19 运行结果

上述代码中，定义了一个临时变量 max，用于记住数组的最大值。通过 for 循环获取数组中的最大值，赋值给max 变量。

首先假设数组中第一个元素arr[0]为最大值，然后使用 for 循环对数组进行遍历，在遍历的过程中只要遇到比max 值还大的元素，就将该元素赋值给max。这样一来，变量max 就能够在循环结束时记住数组中的最大值。需要注意的是，在 for 循环中的变量i 是从 1 开始的，这样写的原因是程序已经假设第一个元素为最大值，for 循环中只需要从第二个元素开始比较，从而提高程序的运行效率。

## 数组的常见问题

数组在编写程序时应用非常广泛，灵活地使用数组对实际开发很重要。接下来，本节将针对数 组的常见操作进行详细地讲解，如数组的遍历、最值的获取等。

### 2.5.1 空指针异常

在使用变量引用一个数组时，变量必须指向一个有效的数组对象，如果该变量的值为 null，则意

味着没有指向任何数组，此时通过该变量访问数组的元素会出现空指针异常，接下来通过一个案例 来演示这种异常，如下所示。ArrayDemo07.java

1. **public class** ArrayDemo07 {
2. **public static void** main(String[] args) {
3. **int**[] arr = **new int**[3]; // 定义一个长度为 3 的数组
4. arr[0] = 5; // 为数组的第一个元素赋值
5. System.*out*.println("arr[0]=" + arr[0]); // 访问数组的元素
6. arr = **null**; // 将变量 arr 置为 null
7. System.*out*.println("arr[0]=" + arr[0]); // 访问数组的元素

8 }

9 }

运行结果如下图所示。

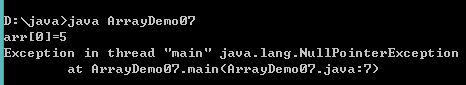


图1-20 运行结果

通过上图所示的运行结果可以看出，上述代码中第 4、5 行代码都能通过变量 arr 正常地操作数组。第 6 行代码将变量置为 null ，当第 7 行代码再次访问数组时就出现了空指针异常NullPointerException。

### 2.5.2 数组越界异常

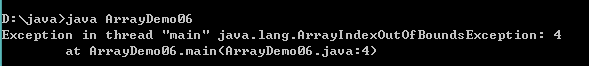
每个数组的索引都有一个范围，即 0~length-1。在访问数组的元素时，索引不能超出这个范围， 否则程序会报错，如下所示。ArrayDemo06.java

1. **public class** ArrayDemo06 {
2. **public static void** main(String[] args) {
3. **int**[] arr = **new int**[4]; // 定义一个长度为 4 的数组
4. System.*out*.println("arr[0]=" + arr[4]); // 通过角标 4 访问数组元素

5 }

6 }

运行结果如下图所示。



上图运行结果中所提示的错误信息是数组越界异常 ArrayIndexOutOfBoundsException，出现这个异常的原因是数组的长度为 4，其索引范围为 0~3，而上述代码中的第 4 行代码使用索引 4 来访问元素时超出了数组的索引范围。

所谓异常指程序中出现的错误，它会报告出错的异常类型、出错的行号以及出错的原因，关于 异常在后面的章节会有详细地讲解。

## 二维数组

在程序中可以通过一个数组来保存某个班级学生的考试成绩，试想一下，如果要统计一个学校 各个班级学生的考试成绩，又该如何实现呢？这时就需要用到多维数组，多维数组可以简单地理解 为在数组中嵌套数组。在程序中比较常见的就是二维数组，接下来针对二维数组进行详细地讲解。

### 2.6.1 二维数组的定义格式

二维数组的定义有很多方式，接下来针对几种常见的方式进行详细地讲解，具体如下： 第一种方式：

int[][] arr = new int[3][4];

上面的代码相当于定义了一个 3\*4 的二维数组，即二维数组的长度为 3，二维数组中的每个元素又是一个长度为 4 的数组，接下来通过一个图来表示这种情况，如下图所示。



图1-21 二维数组

第二种方式：

int[][] arr = {{1,2},{3,4,5,6},{7,8,9}};

上面的二维数组中定义了三个元素，这三个元素都是数组，分别为{1,2}、{3,4,5,6}、{7,8,9}，接下 来通过一个图来表示这种情况，如图 2-54 所示。

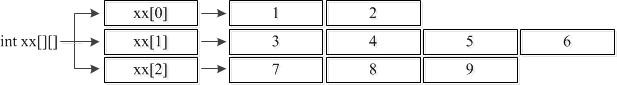


图1-22 二维数组

对二维数组中元素的访问也是通过角标的方式，如需访问二维数组中第一个元素数组的第二个 元素，具体代码如下：

arr[0][1];

### 2.6.2 二维数组元素的访问

操作二维数组时，经常需要获取数组中元素的值。接下来通过一个案例来演示如何获取数组中 元素值，如下所示。ArrayDemo08.java

class ArrayDemo08 {

public static void main(String[] args){

//定义二维数组的方式

int[][] arr = new int[3][4];

System.out.println( arr );

System.out.println("二维数组的长度: " + arr.length);

//获取二维数组的 3 个元素System.out.println( arr[0] ); System.out.println( arr[1] ); System.out.println( arr[2] );

System.out.println("打印第一个一维数组的元素值"); System.out.println( arr[0][0] );

System.out.println( arr[0][1] );//访问的为二维数组中第 1 个一维数组的第 2 个元素

System.out.println( arr[0][2] ); System.out.println( arr[0][3] );

System.out.println("打印第二个一维数组的元素值"); System.out.println( arr[1][0] ); System.out.println( arr[1][1] ); System.out.println( arr[1][2] ); System.out.println( arr[1][3] );

System.out.println("打印第三个一维数组的元素值"); System.out.println( arr[2][0] ); System.out.println( arr[2][1] ); System.out.println( arr[2][2] ); System.out.println( arr[2][3] );

}

}

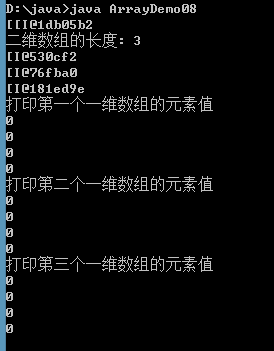
运行结果如下图所示：

图1-23 运行结果

### 2.6.3 二维数组元素遍历

学习完了数组元素的访问，我们来学习下数组的遍历及数组的元素累加和操作。

class ArrayDemo09 {

public static void main(String[] args){

//二维数组的求累加和并遍历

int[][] arr2 = { {1,2},{3,4,5},{6,7,8,9,10} };

int sum2 = 0;

for (int i=0; i<arr2.length; i++) {

for (int j=0; j<arr2[i].length; j++) { System.out.println(arr2[i][j])

}

}

}

}