# 《Android 系统下 Java 编程详解》

作者：华清远见



## 第 4 章 运算符、表达式与流程控制

本章简介

本章介绍了 Java 运算符的概念和用法，并论述在表达式中各种运算符结合使用的情况，分析了运算符的优先级；介绍了分支语句和循环语句等流程控制语句的用法。

## 4.1 运算符

Java 语言提供了丰富的运算符环境，其中主要包括四大类运算符，即算术运算符、位运算符、关系运算符和逻辑运算符。本章主要介绍 Java 运算符的概念和各种运算符的用法，并对各种运算符在表达式中的结合性和优先级做了论述；分支语句和循环语句及 continue/break 的用法。

### 4.1.1 知识准备：算术运算符

Java 常见的算术运算符有 5 种，如表 4-1 所示。

**表 4-1 常见算数运算符**

| 运算符 | 含义 |
|---|---|
| + | 加法运算 |
| — | 减法运算 |
| * | 乘法运算 |
| / | 除法运算 |
| % | 取模运算 |

这些算术运算符可以用于 Java 基本数据类型中的数值型（byte、short、char、int、long、float、double）数据。对于+、-和*运算符，都是很容易理解的，它们分别接受两个操作数，通过运算后返回得到新值。需要注意的是除法运算和取模运算。

#### 1．除法运算

在数学运算中，0 作为除数是没有意义的，在 Java 程序中，对于以 0 作为除数的情况，根据操作数的数据类型，做了不同的处理：对于整形的运算，它将会出现异常；而对于浮点型数据的运算，它将得到一个无穷大值或者 NaN（not a number）。示例如下：

源文件：Division.Java

```
public class Division {
 public static void main(String[] args) {
      System.out.println("123.0/0 = " + 123.0 / 0);
      System.out.println("123/0 = " + 123/ 0);
 }
}
```

编译运行这个程序，将得到如下的结果：

```
123.0/0 = Infinity
Exception in thread "main" Java.lang.ArithmeticException: / by zero
 at Division.main(Division.Java:5)
```

关于 Exception，是 Java 的异常处理机制，将在后面的章节详细讲解。

#### 2．取模运算

取模运算即求余运算，对于 Java 语言来说，其操作数可以是浮点数，计算结果也将是浮点数。

源文件：Mode.Java

```
public class Mode {
 public static void main(String args[]) {
      System.out.println("123.5 mod 4 = " + 123.5 % 4);
      System.out.println("123 mod 4 = " + 123 % 4);
 }
}
```

编译运行这个程序，将得到如下的输出：

```
123.5 mod 4 = 3.5
123 mod 4 = 3
```

取模运算可以用于判别奇偶数，一个整数 *n* 对 2 取模，如果余数为 0，则表示 *n* 为偶数，否则 *n* 为奇数。此外，判别素数，求最大公约数的运算中也会用到取模运算。

**注意：**

**取模运算也会执行除法操作，所以，对于整形数据来说，也不能使用 0 作为取模运算中的“除数”，否则也会出现和除法运算一样的异常。**

### 3. 二元运算符简捷赋值方式

+、−、*、/、%运算如果用在赋值语句中，还可以使用二元运算符的简捷方式来实现，比如：

```
a = a+5;
```

可以使用如下的运算式表示：

```
a +=5;
```

它们在运算结果上是相等的。其他 4 个运算符也可以像上面这个例子中的运算符一样使用，也就是说，将运算符放在“=”的左边，如 a*=5、a/=5 等。

## 4.1.2 知识准备：递增、递减运算符

在编写 Java 程序时，经常需要对一个变量加一或者减一，这时通常使用递增或递减运算符来完成。其中递增运算符对操作数加 1，递减运算符从操作数减 1。

递增和递减操作符有两种形式：“前缀版”和“后缀版”。“前递增”表示++运算符位于变量或表达式的前面；而“后递增”表示++运算符位于变量或表达式的后面。类似地，“前递减”意味着−−运算符位于变量或表达式的前面；而“后递减”意味着−−运算符位于变量或表达式的后面。对于“前递增”和“前递减”（如++A 或−−A），会先执行运算，再生成值。而对于“后递增”和“后递减”（如 a++或 a−−），会先生成值，再执行运算，示例如下：

```
int counter =20;
counter++;
```

此时，counter 的值为 21。

前缀方式和后缀方式的作用都是对操作数加上或减去 1，区别在于用在表达式中的时候。例如：

```
int a = 10;
int b = 10;
int m = 2*++a;
int n = 2*b++;
```

此时，m 的值是 22，n 的值是 20，a 和 b 的值都是 11。这是因为，在进行 m = 2*++a 运算时，程序会先将 a 加上 1，然后再进行乘法运算。而对于 n=2*b++的后缀递增运算，则会先取出 b 的数值进行乘法运算，然后再将 b 递增 1。所以，此时 m 的值是 22（m=2*(10+1)），n 的值是 20（n=2*10），a 和 b 的值都为 11。

**注意：**

**递增/递减操作符只能用于变量而不能用在数字本身，如这样的用法是错误的：10--，5++。**

## 4.1.3 知识准备：关系和布尔运算符

Java 中提供了完整的关系运算符，共有 6 种，用来对两个简单类型操作数进行比较运算，所组成的表达式结果为 boolean 类型的值 true 或 false，如表 4-2 所示。

**表 4-2 Java 关系运算符**

| 运算符 | 名称 | 示例 | 功能 |
|---|---|---|---|

| | | | |
|---|---|---|---|
| < | 小于 | a<b | a 小于 b 时返回真；否则返回假 |
| <= | 小于等于 | a<=b | a 小于等于 b 时返回真；否则返回假 |
| > | 大于 | a>b | a 大于 b 时返回真；否则返回假 |
| >= | 大于等于 | a>=b | a 大于等于 b 时返回真；否则返回假 |
| == | 等于 | a==b | a 等于 b 时返回真；否则返回假 |
| != | 不等于 | a!=b | a 不等于 b 时返回真；否则返回假 |

关系运算符的优先顺序为：

（1）前 4 种关系运算符的优先级别相同，后两种也相同。前 4 种高于后两种。

（2）关系运算符的优先级低于算术运算符。

（3）关系运算符的优先级高于赋值运算符。

来看一些简单的例子：

```
2>3;       //返回 false
2==3;      //返回 false
2!=3;      //返回 true
```

这里要提醒一些有编程经验的读者需要注意的是，在 Java 中，“不等于”是用“!=”表示的，而不是一些编程语言的“<>”，而等于使用“==”而非“=”，在 Java 中，“=”用于赋值操作，而非关系运算符。“==”和“!=”除了用于简单类型的操作数外，还可以用于比较引用类型数据。

**注意：**

**除了“==”和“!=”外，其他的关系运算符都不能用在 boolean 类型的操作数中。**

### 1. 逻辑运算符

逻辑运算符也称为布尔运算符，是指进行逻辑运算的符号。逻辑运算符主要包括：!、&、|、^、&&、||，这些运算符分别实现“非”、“与”、“或”、“异或”、“短路与”、“短路或”等逻辑运算。和关系运算一样，逻辑运算结果也是布尔类型值 true 或 false，参与逻辑运算的数据也必须是 boolean 类型。关于逻辑运算符的种类和功能说明如表 4-3 所示。

**表 4-3　逻辑运算符的种类和功能说明**

| 符号 | 名称 | 功能说明 |
|---|---|---|
| ! | 非 | 只操作一个数据，对数据取反 |
| & | 与 | 两个条件同时为 true 才为 true，否则为 false |
| \| | 或 | 两个条件有一个为 true 则为 true，否则为 false |
| ^ | 异或 | 两个条件真值不相同，则异或结果为真。反之，为假 |
| && | 短路与 | 同&，两个条件同时为 true 才为 true，否则为 false |
| \|\| | 短路或 | 两个条件有一个为 true 则为 true，否则为 false |

在布尔运算符中，需要特别说明的是，短路与“&&”和短路或“||”，这两个运算符是按照“短路”的方式进行求值的，也就是说，如果第一个表达式已经可以判断出整个表达式的值时，就不进行后面的运算了。例如，当对表达式 a&&b 进行运算时，如果表达式 a 的值为 false，将不对 b 的值进行计算。而当对表达式 a||b 进行运算时，如果 a 的值为 true，将不对 b 的值进行计算。

## 4.1.4　任务一：短路布尔运算

### 1. 任务描述

编写一个程序，包含几个返回值为布尔类型的方法，方法中向控制台输出调用信息。在主函数中通过含“短路与”和“短路或”的逻辑表达式调用这些方法，验证短路布尔运算的特点。随后将短路布尔运算符换乘普通布尔运算符，比较输出结果。

### 2．技能要点

- 使用布尔运算符。
- 比较短路布尔运算符与普通布尔运算符的异同。

### 3．任务实现过程

（1)编写一个名为 LogicalOperators 的类，定义 3 个返回值为 boolean 类型的方法 Msg1、Msg2 和 Msg3，返回值分别为 true、false、false，并在每个方法里都输出一条语句，显示该方法被调用。在 Mail()方法中，调用 3 个方法，将 3 个方法的返回值进行短路与运算和短路或运算。查看输出结果和方法调用情况。

源文件：LogicalOperators.Java

```
public class LogicalOperators {

 public static void main(String[] args) {
      LogicalOperators lg= new LogicalOperators();
      System.out.println("短路或运算");
      System.out.println(lg.Msg1() || lg.Msg2() || lg.Msg3());
      System.out.println("短路与运算");
      System.out.println(lg.Msg1() && lg.Msg2() && lg.Msg3());
 }
 boolean Msg1() {
      System.out.println("显示信息 1");
      return 1 < 2;// true
 }

 boolean Msg2() {
      System.out.println("显示信息 2");
      return 1 == 2;// false
 }

 boolean Msg3() {
      System.out.println("显示信息 3");
      return 1 > 2;// false
 }
}
```

（2）编译运行上面的程序，将得到如下的输出：

```
短路或运算
显示信息 1
true
短路与运算
显示信息 1
显示信息 2
false
```

分析：上面的程序中，因为方法 Msg1()的值为 true，而“或”运算中如果有一个表达式为真（true），则整个表达式均为真（true），因此，无须计算后面方法 Msg2()和方法 Msg3()两个表达式就可以得到整个表达式的值了。而第二条“短路与”语句，因为在逻辑“与”运算中，只需要一个表达式的值为假（false），则整个表达式的值都为假（false）。Msg1()为真（true），所以将进行第二个表达式的运算，它将调用方法 Msg2()，而此时，Msg2()方法的返回值为假（false），所以将不用进行后面的运算了。

（3）将 LogicalOperators.Java 中的短路布尔运算符修改成普通布尔运算符。即将如下语句修改，程序其他语句不变：

```
 System.out.println("短路或运算");
 System.out.println(lg.Msg1() | lg.Msg2() | lg.Msg3());
 System.out.println("短路与运算");
 System.out.println(lg.Msg1() & lg.Msg2() & lg.Msg3());
```

输出结果如下：

```
短路或运算
显示信息 1
显示信息 2
显示信息 3
true
短路与运算
显示信息 1
显示信息 2
显示信息 3
false
```

分析：运算符“&&”和“&”、“||”和“|”所求得的结果是一样的，它们的区别在于，“&”和“|”不会进行“短路”运算，而是会计算运算符两边的各个参数的值。

### 4.1.5 知识准备：三元运算符

Java 还支持三元运算符“?:”（也称为条件运算符），这个运算符的用法如下：

```
condition? a:b
```

其中条件 condition 是一个布尔表达式， 如果 condition 为 true，则表达式的值为 a；否则，表达式的值为 b。来看一个简单的例子：

```
public class demo {
public static void main(String[]args){
int a=10,b=20,y;
a>b?(y=a):(y=b;)  //1，这样写是错误的
y=a>b?a:b; //2，这样写是正确的 }
}
```

可以看到，a>b 的时候，执行 y=a ，本例中，a<b，所以条件表达式 a>b 的值为 false，执行 y=b，此时 y 的值为 20。.

### 4.1.6 知识准备：位运算符

位运算是以二进制位为单位进行的运算，其操作数和运算结果都是整型值。可以使用运算符直接处理组成这些整数的各个二进制位。适用的数据类型有 byte、short、char、int、long。

位运算符共有 7 个，分别是位与（&）、位或（|）、位非（~）、位异或（^）、右移（>>）、左移（<<）、0 填充的右移（>>>）。其功能说明如表 4-4 所示。

**表 4-4 位运算符功能表**

| 运算符 | 名称 | 示例 | 说明 |
|---|---|---|---|
| & | 位与 | x&y | 把 x 和 y 按位求与 |
| \| | 位或 | x\|y | 把 x 和 y 按位求或 |
| ~ | 位非 | ~x | 把 x 按位求非 |
| ^ | 位异或 | x^y | 把 x 和 y 按位求异或 |
| >> | 右移 | x>>y | 把 x 的各位右移 y 位 |
| << | 左移 | x<<y | 把 x 的各位左移 y 位 |
| >>> | 右移 | x>>>y | 把 x 的各位右移 y 位，左边填 0 |

其中位运算的位与（&）、位或（|）、位非（~）、位异或（^）与逻辑运算的相应操作的真值表完全相同，其差别只是位运算操作的操作数和运算结果都是二进制整数，而逻辑运算相应操作的操作数和运算结果都是逻辑值。

而>>、<<、>>>也被称为移位运算符，作用是左移运算是将一个二进制位的操作数按指定移动的位数向左移位，移出位被丢弃，右边的空位一律补 0。右移运算是将一个二进制位的操作数按指定移动的位数向右移动，移出位被丢弃，左边移出的空位或者一律补 0，或者补符号位。在使用补码作为机器数的机器中，正数的符号位为 0，负数的符号位为 1。程序示例如下。

（1）有如下程序段：

```
int x = 64;       //x 等于二进制数的 01000000
int y = 70;       //y 等于二进制数的 01000110
int z = x&y       //z 等于二进制数的 01000000
```

即运算结果为 z 等于二进制数 01000000。位或、位非、位异或的运算方法类同。

（2）右移是将一个二进制数按指定移动的位数向右移位，移掉的被丢弃，左边移进的部分或者补 0（当该数为正时），或者补 1（当该数为负时）。这是因为整数在机器内部采用补码表示法，正数的符号位为 0，负数的符号位为 1。例如，对于如下程序段：

```
int x = 70;           //x 等于二进制数的 01000110
int y = 2;
int z = x>>y          //z 等于二进制数的 00010001
```

即运算结果为 z 等于二进制数 00010001，即 z 等于十进制数 17。

对于如下程序段：

```
int x = -70;          //x 等于二进制数的 11000110
int y = 2;
int z = x>>y          //z 等于二进制数的 11101110
```

即运算结果为 z 等于二进制数 11101110，即 z 等于十进制数-18。要透彻理解右移和左移操作，读者需要掌握整数机器数的补码表示法。

（3）0 填充的右移（>>>）是不论被移动数是正数还是负数，左边移进的部分一律补 0。

**注意：**

**没有与>>>对应的<<<操作。**

### 4.1.7 知识准备：赋值运算符

在前面的章节中，已经在很多地方都用到了赋值运算符。赋值运算符“=”将“=”右边的值赋给（更准确地说是“复制到”）左边的变量。“=”右边的值可以是任何的变量、常量或者一个可以产生值的表达式，而“=”的左边必须是一个明确的、命名的变量，不可以为常量，如 a= 10 是合法的，而 10 = a 却是不允许的。

对于基本数据类型的赋值，其非常简单，它直接将“=”右边的值复制到左边的变量中；对于引用数据类型的赋值，操作的并非是对象本身，而是它的“对象引用”，它实际上是将“=”右边的引用（而非对象本身）复制到左边的变量中。

#### 1．扩展赋值运算

将赋值运算符和其他的运算符结合起来，就可以作为一种特别的“扩展”赋值运算符。扩展赋值运算符有：+=、-=、*=、/=、%=、&=、|=、^=、>>=、<<=、>>>=等。注意，并非所有的运算符都可以和赋值运算符结合成扩展赋值运算符。

扩展赋值运算符的引入只是为了简化赋值运算的表达形式，将“a=a operator b;”简化为“ a operator=b;”，其作用是相同的。

#### 2．运算中的数据类型转换

在第 3 章中，已经知道了，数值简单数据类型数据之间是可以相互转换的。那么，在表达式中，它是如何转换的呢？比如，一个表达式中既有 float 类型的数据，又有 double 类型的数据，那么，得出来的结果到底是什么数据类型呢？

Java 在编译期间就会进行扩展类型检查，并且数据从一种类型转换到另一种类型时有严格的限制。在 Java 中，存在两种不同类型的类型转换：

隐式转换：在对包含非 boolean 简单数据类型（primitive type）的表达式求值的时候，Java 会进行大量的隐式类型转换。这些转换有很大的限制，但最基本的原则是这种转换必须是提升（widening，或称为扩大）而不是下降（narrowing，或称为缩小）转换。也就是说，隐式转换只能将一种简单数据类型转换到比它范围更大的类型。

强制类型转换：当隐式转换不能被所要求的表达式支持，或者是有特殊的需求，则需要进行强制的类型转换，这时候需要使用类型转换运算符进行强制转换。

对于一元运算符，例如++或--，隐式转换比较简单： byte、short、char 类型的数被转换成 int 的，而其他类型的数据保持不变。

对于二元运算符，情况比较复杂，但是这种转换基本上遵循如下的基本方式：表达式中最长的类型为表达式的类型。下面是具体的运算规则：

（1）如果两个操作数中有一个是 double 类型的，则另一个也将会转换成 double 类型，最后的运算结果也是 double 类型的，也就是说，表达式的类型为 double 类型。

（2）如果两个操作数中有一个是 float 类型的，则另一个操作数也会转换成 float 类型，此时表达式类型是 float 类型。

（3）如果两个操作数中有一个是 long 类型的，另一个将会转换成 long 类型，此时，表达式的类型也为 long 类型，否则，两个操作数都会转换成 int 类型。

（4）对于 byte/char/short 类型的数据，在进行计算时都会转换成 int 类型来计算，得出的结果也是 int 类型。

下面来看一个例子：

源文件：TypeConversion.Java

```
public class TypeConversion {
 public static void main(String[] args) {
      short s = 11;
      long l = 111;
      int  i = 1;
      byte b1 = 2, b2 = 3;
      char c = 'c';
      System.out.println(l * s);  // 将会得到一个 long 类型的数值
      int j = b1 + c;             //byte 类型+char 类型结果为 int 类型
      // byte f = b+e;            //将会报错，因为计算得出的结果应该是 int 类型
      int m = 1123456789;
      float n = m;                // 将会损失精度，得到的结果是 1.12345677E9
      System.out.println(j);
      System.out.println(n);
 }
}
```

在这个例子中，如果将两个 byte 类型的数据相加，将结果赋给一个 byte 类型的变量，编译的时候将会出错，这是因为两个 byte 类型的值相加返回的是 int 类型的值。而如果将一个整型的值赋给一个 float 类型的变量，则会保留正确的数值级数，但是，从例子中的结果可以看出，转换后已经损失了一些精度。

虽然不能将一个会产生 int 类型结果的表达式的值赋给 byte 类型变量，但是实际上，可以将整型值直接赋值给 byte/short/char 等“更窄”类型的变量，而不需要进行强制类型转换，只要不超出其表数范围即可。

例如：

```
byte b1 = 33;           //合法
short s = 456;          //合法
char ch = 345;          //合法
byte b2 = 142           //非法，超出 byte 型数据表数范围
```

隐式转换不但发生在表达式中，还发生在方法调用的时候。比如，当调用方法时的参数不是方法定义中所规定的参数类型的时候。

在上面已经讲过，Java 可以自动“提升”数据类型。但是，经常需要将数据从较长的类型转换到较短的类型，如将 double 类型的数据转换成 int 类型的数据，这时，Java 不会自动完成这个动作（默认情况下只会将 int 类型的数据转换成 double 类型的），所以，需要在程序中对其进行强制转换。当然，这种操作可能会引起信息的丢失，所以，应该尽量小心使用。

除了简单数据类型外，类型转换还可以引用于引用类型的数据。任何对象都可以被转换成它的基类或任何它所实现的接口。一个接口也可以被转换成它所扩展的任何其他接口。

### 4.1.8 任务二：简单数据类型和引用数据类型的赋值操作

#### 1．任务描述

写一段程序，分别给简单类型变量和引用类型变量进行赋值操作，要求输出赋值后变量值，并体现出引用类型赋值后变量指向同一对象。

#### 2．技能要点

- 简单的赋值操作。
- 了解引用类型变量赋值和简单类型变量赋值的区别。

#### 3．任务实现过程

（1）编写源程序，定义了一个类“Clock”，它有一个“time”的属性。在类“Assignment”的 main() 方法中，定义了两个 int 简单数据类型的变量 a、b，并给 b 赋值 100，然后将 b 的值赋给变量 a，此时实际上是将 b 的值的一个“副本”复制给了 a，因此，a 和 b 中任何一方的变化，都不会影响到另一方。

（2）定义了两个 Clock 引用类型的变量 c1、c2，并给 c1 初始化了一个对象引用，然后，将 c1 的值赋给 c2，此时，这个操作实际上是将 c1 的对象引用复制给了 c2，此时，c1 和 c2 所指向的是同一个对象！因此，无论通过变量 c1 还是 c2 去改变对象，改变的都是同一个对象。

源文件：Assignment.Java

```
public class Assignment {

 public static void main(String[] args) {
      // 简单数据类型
      int a, b = 100;
      a = b;
      b = 10;
      System.out.println("a = " + a);
      System.out.println("b= " + b);

      Clock c1 = new Clock(10);
      Clock c2;
      c2 = c1;
      c1.setTime(12);
      System.out.println("Clock1 的 time=" + c1.getTime());
      System.out.println("Clock2 的 time=" + c2.getTime());
 }
}
 class Clock {
      private int time;

      // 构造器
      public Clock(int clockTime) {
           time = clockTime;
      }

      public int getTime() {
           return time;
      }

      public void setTime(int time) {
           this.time = time;
      }
}
```

（3）编译并运行上面的类“Assignment”，将得到如下的输出：

```
a = 100
b= 10
Clock1 的 time=12
Clock2 的 time=12
```

### 4.1.9 知识准备：运算符的优先顺序

除了上面的这些运算符外，Java 还提供其他非常丰富的运算符来进行其他运算。

Java 运算符在风格和功能上都与 C 和 C++极为相似。下面按优先顺序列出了各种运算符：

```
分隔符：  []  ()  ;  ,
从右到左结合：  ++  -- + - ~ ! (data type)
从左到右结合：  *  /  %
从左到右结合：  +  -
从左到右结合：  <<  >>  >>>
从左到右结合：  <  >  <=  >= instanceof
从左到右结合：  ==  !=
从左到右结合：  &
从左到右结合：  ^
从左到右结合：  |
从左到右结合：  &&
从左到右结合：  ||
从左到左结合：  ?:
从右到左结合：  =  *=  /=  %=  +=  -=  <<=  >>=  >>>=  &=  ^=  |=
```

**注意：**

instanceof 是 Java 编程语言特有的运算符。

### 4.1.10 技能拓展任务：字符串连接运算符

#### 1．任务描述

运算符“+”除了用于数值类型的加法运算外，在字符串类型（String）数据中，它还是一个用于连接字符串的特殊的运算符。在表达式中用“+”连接两个操作数，其中有一个操作数是字符串类型（String），Java 自动将另一个操作数也转换成字符串，然后将这两个字符串相连起来生成一个新的字符串。

要求通过实例来验证这一转换过程，编写程序实现输出一月和二月的手机数据流量值。

#### 2．技能要点

- 使用“+”连接字符串。

#### 3．任务实现过程

（1）写一个类 StringConnect.Java，在该类中可以通过将数字和一个空字符串相连的方式，将数字转换成字符串类型。

源文件：StringConnect.Java

```
public class StringConnect {

 public static void main(String[] args) {
      double jan = 98.987;
      double feb = 76;                      // 自动将 int 型的数值 1 提升到 double 类型 1.0
      double total = jan + feb;
      String flow = "January Dataflow is: " + jan;
// 上面得到一个字符串："Price is:9.987"
      String sflow = "The Total DataFlow is: " + total;
// 上面得到一个字符串："Total Price is:10.987"
      System.out.println(flow);
      System.out.println(sflow);
      System.out.println("" + jan + feb);   // 打印出一个字符串："9.9871.0"
```

```
        System.out.println(jan + feb +"");    // 打印出一个字符串："10.987"
    }
}
```

（2）运行程序，输出结果是：

```
January Dataflow is: 98.987
The Total DataFlow is: 174.987
98.98776.0
174.987
```

从上面的例子中可以看到，String 和一个数字类型的数据进行“+”运算，将会得到一个新的字符串，这个字符串由旧的字符串和这个数字组成。

再来看这行程序：

```
System.out.println("" + jan + feb);
```

根据运算符从左到右的结合原则，空字符串“""”首先和 jan 进行运算，得到一个字符串，这个字符串的内容就是“98.987”，然后，这个字符串再和数字 y 进行运算，此时得到一个由 x 和 y 组合成的新的字符串：98.98776.0。

比较一下下面这条语句：

```
System.out.println(jan + feb +"");
```

这条语句首先进行数值的相加运算，得到一个新的数值：174.987，然后再和空字符串进行连接运算，此时得到一个新的字符串，内容为“174.987”。

## 4.2 表达式

在前面的内容中，一直在使用一个概念：表达式。表达式就是运算符和操作数的结合。Java 中的表达式分为算数表达式和逻辑表达式两种。当代码执行的时候，由 Java 解释器进行求值，如果结果可以预先计算的话，可以由编译器来进行求值。

### 4.2.1 知识准备：表达式中运算符的结合性

所有的数学运算都认为是从左到右结合的，在 Java 中，大部分运算也是从左到右结合的，只有单目运算符、赋值运算符和条件运算符例外。

乘法和加法是两个可结合的运算，也就是说，这两个运算符左右两边的操作符可以互换位置而不会影响到结果。

### 4.2.2 知识准备：表达式中运算符的优先顺序

在进行表达式的转换过程中，必须了解各种运算的优先顺序，使转换后的表达式能满足数学公式的运算要求。表 4-5 列出了运算符的优先级。

**表 4-5 运算符的优先级**

| 运算符说明 | Java 运算符 | 运算符说明 | Java 运算符 |
|---|---|---|---|
| 分隔符 | . [] () , ; | 按位与 | & |
| 单目运算符 | + - ~ ! ++expr --expr | 按位异或 | ^ |
| 创建或类型转换 | new (type)expr | 按位或 | \| |
| 乘法 / 除法 | * / % | 条件与 | && |

| | | | |
|---|---|---|---|
| 加法 / 减法 | + - | 条件或 | \|\| |
| 移位 | << >> >>> | 条件 | ?: |
| 关系 | < <= >= > instanceof | 赋值 | = |
| 等价 | == != | | |

问：如果是同级的运算符要怎么办呢？

答：如果同级的运算，运算符是按从左到右依次进行；多层括号时由里向外进行。

## 4.3 分支语句

从结构化程序设计角度出发，程序有 3 种结构：顺序结构、选择结构、循环结构。顺序结构的程序设计是最简单的，只要按照解决问题的顺序写出相应的语句就行，它的执行顺序是自上而下，依次执行，在这里不再介绍。本节主要介绍程序设计结构中的选择结构。

选择结构主要通过分支语句实现程序流程控制 ，即根据一定的条件有选择地执行或跳过特定的语句。Java 分支语句分为两种。

if-else 语句：一种控制程序流程的最基本的方法，else 子句可有可无。

switch 语句：另一种效率程序流程控制语句，当必须在程序中检测一个整型表达式的多个值时将会用到它。

下面首先来看 if 语句，然后来讨论 switch 语句。

### 4.3.1 知识准备：if 语句

if 条件语句是最常用的一种分支语句，它的基本格式有 3 种。

（1）形式一：

```
if (boolean 条件表达式 ){
     语句 A;
}
```

在 if 后面的条件语句中，必须是一个可以转换成 boolean 的表达式，这个表达式需要用括号括起来。当表达式值为 false 时，执行语句 A，否则跳过语句 A。

（2）形式二：

```
if (boolean 条件表达式){
     语句 A;
} else {
     语句 B;
}
```

表达式为 true，执行 A；表达式为 false，执行 B。

（3）形式三：

```
if(boolean 条件表达式)
     语句 1
else if(boolean 表达式 2)
     语句 2
else if(boolean 表达式 3)
     语句 3
…
else if(boolean 表达式 n)
     语句 n
```

```
else
    语句 n
```

表达式 1 为 true，则执行语句 1，表达式 1 为 false，判断表达式 2，表达式 2 为 true，执行语句 2，表达式 2 为 false，判断表达式 3，如此直至表达式 n 为 false，则执行最后一个 else 后的语句 n。

### 4.3.2 任务三： if 语句的用法

#### 1．任务描述

使用 if-else 语句，对手机电池电量使用情况进行提示。当手机电池电量大于 30%时，显示手机电量充足；当电量处于 10%~30%时，提示手机电量低；当电量处于 10%以下时，提示更换电池；当电量小于 5%时，提示电量耗尽将自动关机。

#### 2．技能要点

- 使用 if…else 多分支语句。

#### 3．任务实现过程

（1）定义一个名为 Battery 的类，该类中有一个名为 warning 的方法，该方法入口处传入一个 int 类型的参数，该参数表示当前手机电量。该方法中使用 if…else 语句进行多分支条件的判断。当电池电量小于等于 5 时，直接输出信息；当电量大于 5 时，进行下一层判断，电量小于 10 时，输出信息；当电量大于 10 时，再继续进行判断；如果电量小于 30，输出信息，电量大于 30 时，不再进行判断，执行最后的 else 语句，输出信息。

（2）在 main()方法里声明了 4 个 int 类型变量，覆盖了 4 个分支条件。调用 warning 方法，将这 4 个变量依次作为参数传入。

源文件：Battery.Java

```
public class Battery {

 public static void main(String[] args) {
      int a = 40,b=20,c=6,d=5;
      Battery bat = new Battery();
      bat.warning(a);
      bat.warning(b);
      bat.warning(c);
      bat.warning(d);
 }
 public void warning(int power){

      if (power <= 5) {
           System.out.println("*The power runs out,about to shutdown!*");
      } else if (power < 10) {
           System.out.println("*Replace the battery!*");
      } else if(power < 30){
           System.out.println("*The power is low*");
      } else
           System.out.println("*The power is enough*");
 }
}
```

（3）运行程序，它将向控制台输出如下的信息：

```
*The power is enough*
*The power is low*
*Replace the battery!*
*The power runs out,about to shutdown!*
```

### 4.3.3 知识准备：switch 语句

对于多选择分支的情况，可以用 if 语句的 if-else 形式或 if 语句嵌套处理，但大多数情况下略显麻烦。为此，Java 提供了另一种方法——switch 语句，也称开关语句。一个 switch 语句由一个控制表达式和一个由 case 标记描述的语句块组成。和 if 不同，switch 后面的控制表达式求出的值应该是整型而不是 boolean 类型。对控制表达式求得的值，决定了哪一个 case 分支将被执行。每一个 case 都用唯一的常量表达式或常量来标示，用于控制这个 case 是否必须被执行。程序将执行到那些表达式值与控制表达式的值相匹配的 case 分支中。如果不存在这样的匹配，则将执行 default 后面的代码块。如果没有 default 标记（并不推荐这样做），则控制被传递到 switch 块后的第一条语句，也就是退出了 switch 块。控制表达式必须是可以转化为 byte、short、char 或 int 类型的值的表达式。

在 switch 中，需要了解 4 个关键字——switch、case、break 和 default，它们的意思分别是开关、情况、中断、默认（值）。用一句话套起来的说法就是根据开关值的不同、执行不同的情况、直到遇上中断；如果所有的情况都不符合开关值，那么就执行默认的分支。注意在每一个 case 标记的代码块中，最后都有一个 break 语句。这条语句具有重大的作用，如果没有这条语句，当与 case 分支相连续的代码块执行完毕后，将会继续运行与下一个 case 分支相联系的代码块。

### 4.3.4 任务四：switch 分支语句实例

#### 1. 任务描述

使用分支语句 switch，输出一周车辆限行尾号（默认周一、周六限行）。

#### 2. 技能要点

- 使用 switch 分支语句及 case、default、break 关键字。

#### 3. 任务实现过程

（1）定义一个名为 MorningGreetings 的类，在该类中定义一个 greetings 方法，该方法传入一个 int 类型变量 n。使用 switch 语句对 n 进行判断。如果 n 是 1，则代表周一，限行车号为 n=1 和 m=5+n;n=2,3,4 时计算法方法相同。n=5 时，m=5−n。输出 m、n。

（2）当 n=6,7 时，此时为周末，输出信息都显示车辆自由行驶。将“case 6 ：”的执行语句设为空，则程序会执行与紧随其后的 case 语句相同的动作，n=7 时的动作。

（3）当传入 n 的值不是 1~7 时，程序执行 default 之后的语句，提示错误信息。

源文件：MorningGreetings.Java

```
public class MorningGreetings {

 public static void main(String[] args) {
      MorningGreetings mg = new MorningGreetings();
      int day1=1,day2=5,day3=6,day4=7,day5=9;
      mg.greetings(day1);
      mg.greetings(day2);
      mg.greetings(day3);
      mg.greetings(day4);
      mg.greetings(day5);
 }
 public void greetings(int n){
      int m;
      switch (n) {
      case 1: {
           m = n+5;
           System.out.println("Monday,Driving restrictions is "+n+","+m);
           break;
      }
      case 2: {
           m = n+5;
           System.out.println("Tuesday,Driving restrictions is "+n+","+m);
           break;
      }
```

```
        case 3: {
             m = n+5;
             System.out.println("Wednesday,Driving restrictions is "+n+","+m);
             break;
        }
        case 4: {
             m = n+5;
             System.out.println("Thursday,Driving restrictions is "+n+","+m);
             break;
        }
        case 5: {
             m = n-5;
             System.out.println("Friday,Driving restrictions is "+n+","+m);
             break;
        }
        case 6:
        case 7: {
             System.out.println("Great!Today is freedom!");
             break;
        }
        default:
             System.out.println("Not of the week");
        }
   }
}
```

（4）执行该程序，显示如下输出：

```
Monday,Driving restrictions is 1,6
Friday,Driving restrictions is 5,0
Great!Today is freedom!
Great!Today is freedom!
Not of the week
```

## 4.4 循环语句

4.3 节中使用分支语句实现了选择结构的程序控制，本节中，将关注三大节本结构之中的循环结构。实现循环结构的语句是循环语句，循环语句是由循环体和循环终止条件两部分组成的。其功能是在循环条件满足的情况下，反复执行一段代码，直到不再满足循环条件为止。循环可分为三类：

- 条件变为 false 之前重复执行语句。
- 条件变为 true 之前重复执行语句。
- 按照指定的次数重复执行语句。

Java 编程语言支持 3 种循环构造类型：for、while 和 do while。3 种循环结构均通过一个条件表达式控制。在 while()和 for()结构中，条件判断均先于语句块执行，所以，有可能语句块一次也不执行。在 do…while()中，语句块先于条件判断，所以，语句块至少执行一次。

### 4.4.1 知识准备：for 语句

for 语句的基本格式如下：

```
        for ( 初始化语句 ;循环条件; 迭代语句){
             循环体;
        }
```

其中：初始化语句是循环的初始状态，循环条件是条件判断的布尔表达式，如果表达式的值为 true，则执行后面的语句，接下来进行后面迭代语句。如果条件判断表达式第一次求值就为 false，那么 for 循环不会进行任何的迭代，后面的循环体和迭代语句也不会执行任何操作。

一次循环结束后，下一次循环开始前，执行迭代部分的语句，然后判断循环条件表达式的值，决定是否进行下一次循环。

请看下面 for 循环的例子：

```
int counter = 1;
for (int i = 5;i>1;i--){
     counter = counter * i;
}
```

上面程序片断的作用是实现一个简单的阶乘运算：n*(n-1)*(n-2)*…*1，在这里，n 的值为 5，因此，它运算的结果是 5*(5-1)*(5-2)*(5-3)*(5-4)=5*4*3*2*1=120。

上面的例子中，初始化语句只有一个初始化的值，条件判断表达式也只有一个条件，步进代码也是每次递减一个数字。但是，其实 Java 允许在 for 语句的循环控制的各个部分放置任何表达式，如下例：

```
for(int b = 0,s = 0,p = 0;(b<10)&&(s<4)&&(p<10);p++){
     //代码块
     //更新 b 和 s 的值
}
```

在这个例子中，初始化的变量有 3 个，但是，只能有一个声明语句，所以，如果需要在初始化表达式中声明多个变量，那么这些变量必须是同一种数据类型的。

for 循环并没有限制它的 for 语句的每一部分都必须提供一个表达式，这 3 个部分其实都可以为空，此时，是一个无限的循环。这在语法上是没有错误的，只是这种循环在实际应用中会引起很多的问题，应该避免这种会引起无限循环的 for 语句。

下面来看一个例子，了解一下 for 语句中各部分为空时的控制情况：

```
int sum=0;
//注意看 for 语句，它的步进部分是空的
for (int i=1;i<=n;){
     sum=sum+i;
     //将步进放到了 for 程序块中
     i++;
}
```

注意 for 语句，它的步进部分是空的，将这个步进运算放到了程序块中。一个无限 for 循环的例子如下：

```
for(;;){
     //程序块
}
```

这种情况一般是要避免的，不应该让程序出现这种无限循环的情况。

在 for 语句内定义的变量，它的作用范围仅限于 for 语句块、表达式及 for 子句的语句部分。在 for 循环终止后，它们将不可被访问。如果需要在 for 循环外部使用循环计数器的值，可以将这个变量定义在 for 循环体外，如下：

```
int k;
for(k = 0;k<10;k++){
     //statements
}
//此时可以再使用变量 k
```

另外，如果变量定义在 for 循环体内，则在另外一个循环体中还可以使用相同的变量名称，如下：

```
for(int k=0;k<10;k++)  {
     …
}

for(int k = 100;k>0;k--)    {
     …
}
```

上面的代码段是合法的。

### 4.4.2 任务五： for 循环语句实例

1．任务描述

手机用户设定在一小时内，每隔 10 分钟，手机闹钟响铃一次。

2．技能要点

- 使用 for 循环语句。
- for 循环语句与 if 分支语句结合使用。

3．任务实现过程

（1）定义一个名为 Alarm 的类，该类中使用 for 进行响铃循环输出。for 语句的初始条件为 1，即从第一分钟开始计时，当时间小于 60 分钟时，判断此时时间是否可以被 10 整除，如果可以整除，则响铃，不能整除，则时间加 1，重新开始循环。

（2）定义一个 int 类型变量 j 来记录响铃次数。每次响铃之后使 j++。

```
public class Alarm {
 public static void main(String[] args) {
      int j =1;
      for(int i=1;i<=60;i++){
            if(i%10==0){
              System.out.println("第"+j+"次响铃");
              j++;
            }
          }
 }
}
```

（3）程序输出结果：

```
第 1 次响铃
第 2 次响铃
第 3 次响铃
第 4 次响铃
第 5 次响铃
第 6 次响铃
```

## 4.4.3 知识准备： while 语句

while 循环语句的格式：

```
while (条件判断表达式 ){
      循环体;
      迭代运算;
 }
```

while 语句首先测试条件表达式，这个表达式的值必须是布尔类型的，如果为 true，则运行代码块中的程序，并且一般需要进行迭代运算，以改变条件表达式中的变量的值，直到表达式中的值变为 false。

如果刚开始条件表达式就为 false，则 while 循环一次也不会被执行。

来看一个 while 循环的例子：

```
…
//while 循环
int counter=0;
//初始化一个变量
int i=1;
//利用这个变量构成一个条件表达式
while(i<=10)  {
      counter=counte1+i;
      //将 i 加 1
      i=i++;
}
System.out.println("After the While Loop,the counter is:"+counter);
…
```

for 循环和 while 循环是等价的，可以将如下的 for 循环：

```
for(init_expr;test_expr;alter_expr){
  statements;
}
```

改写成如下的 while 循环方式：

```
init_expr;
while(test_expr){
  statements;
  alter_expr;
}
```

这两种方式是完全可以相互替换的。for 循环和其他两个循环控制语句不同的地方在于，它可以在控制表达式中定义变量，而 while/do…while 不能这样做。

### 4.4.4 知识准备：do…while 语句

do…while 循环语句的格式如下：

```
[初始化表达式]
do{
      循环体;
      迭代运算;
}while(条件表达式);
```

Do…while 循环类似于 while 循环，在 while 后面也得跟一个 boolean 类型的条件表达式。do…while 循环首先执行里面的代码段，然后再判断条件表达式是否为 true，如果为 true，则返回到 do 语句来执行，否则，退出整个循环。因为 do…while 循环是先运行里面的代码块，然后再判断条件，所以，do…while 循环至少会执行一次，这是 do…while 循环和 while、for 循环最大的区别所在。

来看下面这个例子：

```
int counter=0;
int j=1;
do {
     counter = counter +j;
     j=j+1;
}while(j<=10);
System.out.println("After the Do Loop,the counter is:"+ counter);
```

比较一下这个例子和上面 while 的例子，这两个例子中的条件表达式都是一样的，但是，它们运行后得到的结果也是一样的。在 while 循环中，得出的运算结果是 55，而 do…while 得出的结果也是 55。但是，如果将各自的条件改成(i<=0)和(j<=0)，则 do…while 循环将会返回一个 1 的结果，而 while 循环却只能返回一个 0 的结果，这是因为 do…while 是“先执行、后判断”，而 while 却是“先判断、后执行”。

### 4.4.5 知识准备： break/continue 语句

使用 break 语句可以终止 switch 语句和终止循环的子语句块，甚至是普通的程序块。关于如何终止 switch 语句，请参考前面的 switch 的内容。

#### 1．break 语句

在循环中，经常需要在某种条件出现时，强行终止循环的运行，而不是等到循环的判断条件为 false 时，这个时候，可以通过 break 来完成这个功能。

break 语句通常用在循环语句和开关语句中。例如，前面的章节已经提到的开关语句 switch 中，break 可以使程序跳出 switch 而执行 switch 以后的语句，这样可以防止程序进入死循环而无法退出。当 break 语句用于 do-while、for、while 循环语句中时，可使程序终止循环而执行循环

下面来看一个 break 用于循环语句中的例子：

```
int sum1 = 0,n=10;
for (int i=1;i<=n;i++){
```

```
        sum1=sum1+i;
        if(i%2==0)break;
    }
```

以上例子中，如果 i 能够被 2 整除，就跳出 for 循环。因此，实际上，for 循环只能循环两次，得到的 sum1 的值是 3。

从上面的例子中可知 break 可以跳出循环体，但是如果是多层循环嵌套的时候，break 的作用就有局限性了，因为它只能跳出单层循环。如果希望可以跳出指定层数的循环到这里对于有 C++或其他编程经验的读者，可能会想到了 goto 语句。goto 语句是无条件转移语句，break/continue 语句可以理解为弱化了的 goto 语句。但是，在结构化程序设计中不主张使用 goto 语句，以免造成程序流程的混乱。在 Java 中，goto 语句虽然是保留字，但并没有使用它，而是提供了一种类似 goto 功能的实现方法，就是使用 break/ continue 与标签结合。在本质上而言，它和 goto 语句的跳跃是不同的，它是一种循环中断的方式而已。它和 goto 语句的相同点在于，它们都使用了标签（label）。

所谓标签（label），就是后面跟了一个冒号“:”的标识符，例如：

```
oneLabel:
```

从语法上看，在 Java 程序中，标签可以放在任意的地方，但是，一般而言，标签只有放在循环语句之前，才能真正起到应有的作用，如下：

```
LableOne:
循环
{
…
}
```

来看一个用在嵌套循环中的和标签结合的 break 例子：

```
        outer:
        for(int i = 0;i<10;i++) {
            System.out.println("Outer loop:");
            inner:
                while(true) {
                    int k = System.in.read();
                    System.out.println("Inner Loop:"+k);
                    if(k=='b') break inner;
                    if(k=='q') break outer;
                }
        }
```

在这个例子中，从控制台接收一个输入，如果输入 b，则退出内层的 while 循环，如果输入 q，则退出外层的循环（也就是终止整个循环）。

另外，如果需要终止普通的语句块（既不是 switch 也不是循环语句），则必须使用标签：

```
Label1:    {
 Label2:{
        Lable 3:{
            …
            }
        }
 }
```

### 2．continue 语句

continue 语句用来略过循环中剩下的语句，停止当前迭代，重新开始新的循环，这和 break 语句的完全跳出循环是不一样的。

continue 仅仅出现在 while/do…while/for 语句的子语句块中。也可以使用和标签结合的方式来选择需要终止的嵌套循环的层级。

下面来看一个例子：

```
    int sum1=0;
    int sum2=0;
    //Continue
```

```
    for (int j=1;j<=10;j++)     {
         if(j%2==0)continue;
         sum2=sum2+j;
    }
    System.out.println(sum2);
```

在这个例子中，如果在 j 可以被 2 整除，则不进行后面的相加操作，而重新返回到循环的开头，判断 j=3 时是否满足循环条件。因此，它运算后的值为 25。

问：continue 和 break 的区别是什么呢？

答：如果遇到 continue，程序将立即返回循环的入口，继续执行；如果遇到 break，程序将立即跳出循环，从循环后的第一条语句开始执行。自己多编几个小程序去体会吧！

### 4.4.6 技能拓展任务：continue 结合标签的使用

#### 1. 任务描述

使用 break/continue 语句和标签结合，实现在短信中查找关键字的功能。

#### 2. 技能要点

- break/continue 语句跳出循环。
- 标签和 continue 语句的结合使用。

#### 3. 任务实现过程

（1）定义一个名为 SearchKeyWord 的类，这个程序的作用是从字符串 msg 中搜索指定的子串 keyword，从要搜索的字符串 keyword（程序中设定为“key”）第一个子符开始去匹配 msg 的第一个字符，如果第一个字符都不匹配，就不用再比较第二个字符了（利用 continue）。如果第一个字符匹配，则比较第二个字符，如果第二个字符不匹配，则不用再往下比较，否则，往下比较第三个字符，依此类推。如果找到完全匹配的子字符串，则退出整个循环（break）并且返回 true。然后根据是否返回 true 打印出“Found it”或“Didn't find it”。

```
public class SearchKeyword {
  public static void main(String[] args) {
       String msg = "Look for a keyword in msg";
       String keyword = "key";
       boolean foundIt = false;

       int max = msg.length() - keyword.length();

       test: for (int i = 0; i <= max; i++) {
            int n = keyword.length();
            int j = i;
            int k = 0;
            while (n-- != 0) {
                 if (msg.charAt(j++) != keyword.charAt(k++)) {
                      // 跳出的本次循环是 for 循环，而不是 while 循环
                      continue test;
                 }
            }
            foundIt = true;
            // 跳出整个循环
            break test;
       }
       System.out.println(foundIt ? "Found it" : "Didn't find it");
  }
}
```

（2）运行程序。如果 substring 的值为“key”（如程序中所示），则会在控制台上打印出“Found it”，如果修改 keyword 的值为“123”，则会打印出“Didn't find it”。

# 4.5 本章小结

本章介绍了运算符的基本概念；着重介绍了算术运算符，递增递减运算符，关系和布尔运算符，位运算符和赋值运算符等主要运算符，通过实例使读者了解这些运算符的使用方法；并介绍了各种运算符的优先级。描述了表达式的概念和表达式中运算符的结合性与优先级。分析应用于 Java 流程控制中的分支语句和循环语句，通过实际代码使读者了解了 if-else、switch、for、while 等语句的用法。通过布置任务使读者掌握技巧，学会在编程中使用这些语句。

## 课后练习题

### 一、选择题

1．阅读下面代码：

```
boolean var1=false;
boolean var2=true;
boolean result1=var1&&var2;
boolean result2=!var2;
boolean result2=(var1&var2)&(!var2);
System.out.println("result1="+result1+";result2="+result2);
```

以上代码运行的结果是（　　）。

A．result1=false;result2=false　　B．result1=true;result2=true

C．result1=true;result2=false　　D．result1=false;result2=true

2．下列代码的执行结果是（　　）。

```
public class Test1{
public static void main(String args[]){
float t=6.5f;
int q=5;
System.out.println((t++)*(--q));
}
}
```

A．30　　B．30.0　　C．26.0　　D．26

3．int j=2;

```
switch(j){
case 2:
System.out.println("北京");
case 2+1:
System.out.println("上海");
break;
default:
System.out.println("南京");
break;
}
```

运行以上代码输出结果为（　　）。

A．北京　　B．北京<br>上海　　C．北京<br>上海<br>南京　　D．编译错误

4．以下选项中循环结构合法的是（　　）。

A．

```
while (int i<5){
i++;
System.out.println("i is "+i);
}
```

B．

```
int j=3;
while(j){
System.out.println(j);
}
```

C.

```
int j=0;
for(int k=0;j+k!=10;j++,k++){
System.out.println("j is"+j+"k is"+k);
}
```

D.

```
int j=0;
do{
System.out.println("j is"+j++);
if(j==3){ loop;}
}while(j<10);
```

5. 下列程序段的输出结果是（ ）。

```
int x=5,y=3;
  int a;
  System.out.println(a=x>y?x:y) ;
```

A. 3　　B. 5　　C. 8　　D. 2

二、编程题

1. 编写程序实现比较两个数的大小，输出比较结果，比如输入 5,3，输出 5 比 3 大。
2. 编写程序打印 100 以内的 3 的倍数，3，6，9，…，99。
3. 编写一个程序为一周中的每天打印一句问候语（要求用 switch 语句）。

# 联系方式