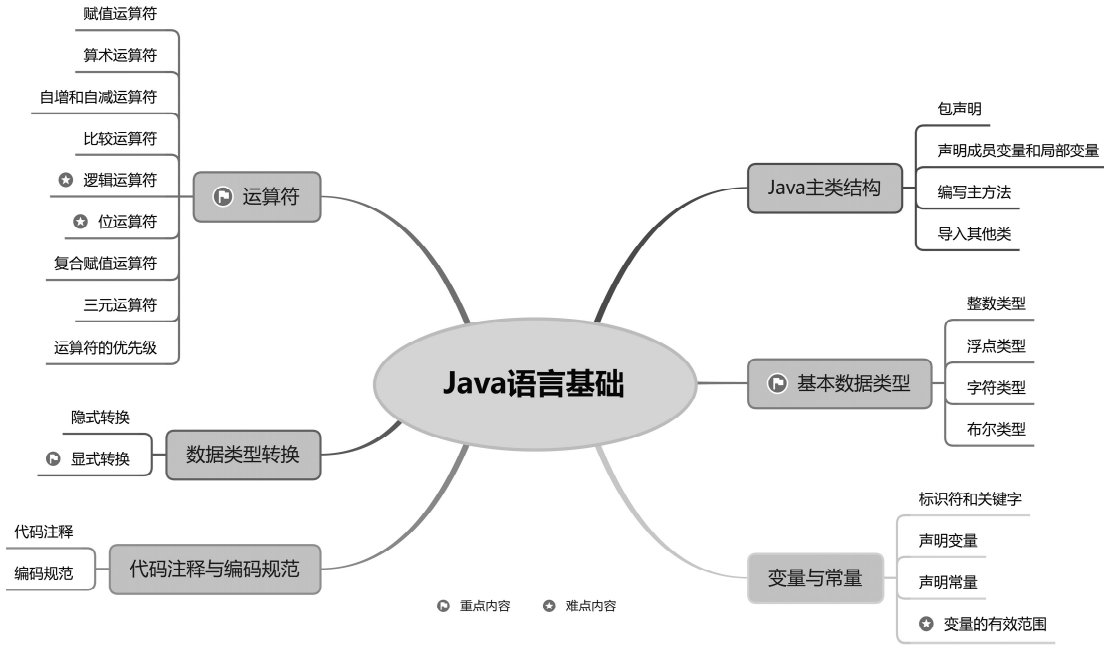
# 第 3 章

# Java 语言基础

很多人认为在学习 Java 语言之前必须先学习 C 或 C++语言，其实并非如此。之所以存在这种错误的认识，是因为很多人在学习 Java 语言之前都学过 C 或 C++语言。事实上，Java 语言比 C 或 C++语言更容易掌握。要掌握并熟练应用 Java 语言，就需要对 Java 语言的基础进行充分的了解。本章将对 Java 语言基础进行比较详细的讲解，初学者应该对本章的各个小节进行仔细的阅读、思考，这样才能达到事半功倍的效果。

本章的知识架构及重难点如下。




## 3.1 Java 主类结构

Java 语言是面向对象的程序设计语言，Java 程序的基本组成单元是类，类体中又包括属性与方法两部分（本书将在第 6 章中逐一介绍）。每一个应用程序都必须包含一个 main()方法，含有 main()方法的类称为主类。下面通过程序来介绍 Java 主类结构。

**【例 3.1】创建主类并调用其主方法（实例位置：资源包\TM\sl\3\1）**

在 Eclipse 下依次创建项目 item、包 Number 和类 First。在类体中输入以下代码，实现在控制台上

输出“你好 Java”。

```
package Number;
public class First {
        static String s1 = "你好";
        public static void main(String[] args) {
                String s2 = "Java";
                System.out.println(s1);
                System.out.println(s2);
        }
}
```

运行结果如下：

```
你好
Java
```

**注意**

代码中的所有标点符号都是英文字符。不要在中文输入法状态下输入标点符号，如双引号和分号，否则会导致编译错误。

文件名必须和类名 First 相同，即 First.java。还要注意大小写，Java 是区分大小写的。

### 1．包声明

一个 Java 应用程序是由若干个类组成的。在例 3.1 中就是一个类名为 First 的类，语句 package Number 为声明该类所在的包，package 为包的关键字（关于包的详细讲解可参见第 11 章）。

### 2．声明成员变量和局部变量

通常将类的属性称为类的全局变量（成员变量），将方法中的属性称为局部变量。全局变量被声明在类体中，局部变量被声明在方法体中。全局变量和局部变量都有各自的应用范围。在例 3.1 中，s1 是成员变量，s2 是局部变量。

### 3．编写主方法

main()方法是类体中的主方法。该方法从“{”开始，至“}”结束。public、static 和 void 分别是 main()方法的权限修饰符、静态修饰符和返回值修饰符，Java 程序中的 main()方法必须被声明为 public static void。String[] args 是一个字符串类型的数组，它是 main()方法的参数（在后续章节中将对其进行详细的讲解）。Java 程序首先从 main()方法开始执行。

### 4．导入 API 类库

在 Java 语言中可以通过 import 关键字导入相关的类。在 JDK 的 API 中（应用程序接口）提供了 130 多个包，如 java.swing、java.io 等。可以通过 JDK 的 API 文档来查看这些包中的类，把握类的继承结构、类的应用、成员变量表、构造方法表等，并对每个变量的使用目的进行了解，API 文档是程序开发人员不可或缺的工具。

**误区警示**

Java 语言是严格区分大小写的。例如，不能将关键字 class 等同于 Class。

# 3.2 基本数据类型

在 Java 中有 8 种基本数据类型来存储数值、字符和布尔值，如图 3.1 所示。

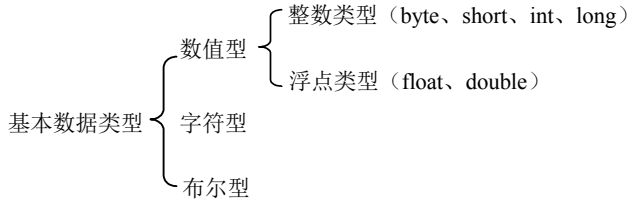



图 3.1 Java 基本数据类型

## 3.2.1 整数类型

整数类型简称整型，用来存储整数数值，即没有小数部分的数值。它们可以是正数，也可以是负数。整型数据根据它所占内存大小的不同，可分为 byte、short、int 和 long 4 种类型。它们具有不同的取值范围，如表 3.1 所示。

表 3.1 整型数据类型

| 数 据 类 型 | 内存空间（8 位等于 1 字节） | 取 值 范 围 |
|---|---|---|
| byte | 8 位 | −128～127 |
| short | 16 位 | −32768～32767 |
| int | 32 位 | −2147483648～2147483647 |
| long | 64 位 | −9223372036854775808～9223372036854775807 |

下面分别对这 4 种整型数据类型进行介绍。

### 1．int 型

定义 int 型变量有以下 4 种语法：

```
int x;                          //定义 int 型变量 x
int x,y;                        //同时定义 int 型变量 x,y
int x = 10,y = -5;              //同时定义 int 型变量 x,y 并赋予初值
int x = 5 + 23;                 //定义 int 型变量 x，并赋予公式（5+23）计算结果的初值
```

int 型变量在内存中占 4 字节，也就是 32 位，在计算机中 bit 是由 0 和 1 来表示的，所以 int 型值 5 在计算机中是这样显示的：

```
00000000 00000000 00000000 00000101
```

int 型是 Java 整型值的默认数据类型。当对多个尚未定义数据类型的整数做运算时，运算的结果将默认为 int 类型。例如，下面这行代码：

```
System.out.println(15 + 20);            //输出 35
```

等同于如下代码：

```
int a = 15;
int b = 20;
int c = a + b;
System.out.println(c);                  //输出 35
```

### 2．byte 型

byte 型的定义方式与 int 型的定义方式相同。定义 byte 类型变量，代码如下：

```
byte a;
byte a, b, c;
byte a = 19, b = -45;
```

### 3．short 型

short 型的定义方式与 int 型的定义方式相同。定义 short 类型变量，代码如下：

```
short s;
short s, t, r;
short s = 1000, t = -19;
short s = 20000 / 10;
```

### 4．long 型

由于 long 类型变量的取值范围比 int 类型变量的取值范围大，且属于高精度数据类型，因此在赋值时要和 int 型做出区分，需要在整数后加 L 或者 l（小写的 L）。定义 long 类型变量，代码如下：

```
long number;
long number, rum;
long number = 12345678l, rum = -987654321L;
long number = 123456789L * 987654321L;
```

**注意**

整数在 Java 程序中有 3 种表示形式，分别为十进制、八进制和十六进制。

（1）十进制：十进制的表现形式大家都很熟悉，如 120、0、−127。除了数字 0，不能以 0 作为其他十进制数的开头。

（2）八进制：如 0123（转换成十进制数为 83）、−0123（转换成十进制数为−83）。八进制数必须以 0 开头。

（3）十六进制：如 0x25（转换成十进制数为 37）、0Xb01e（转换成十进制数为 45086）。十六进制数必须以 0X 或 0x 开头。

## 3.2.2　浮点类型

浮点类型简称浮点型，用来存储含有小数部分的数值。Java 语言中浮点类型分为单精度浮点类型（float）和双精度浮点类型（double），它们具有不同的取值范围，如表 3.2 所示。

表 3.2　浮点型数据类型

| 数 据 类 型 | 内存空间（8 位等于 1 字节） | 取 值 范 围 |
|---|---|---|
| float | 32 位 | 1.4E-45～3.4028235E38 |
| double | 64 位 | 4.9E-324～1.7976931348623157E308 |

在默认情况下，小数都被看作 double 型，若想使用 float 型小数，则需要在小数后面添加 F 或 f。另外，可以使用后缀 d 或 D 来明确表明这是一个 double 类型数据，但加不加 d 或 D 并没有硬性规定。

而定义 float 型变量时，如果不加 F 或 f，系统会认为它是一个 double 类型数据，并出错。定义浮点类型变量，代码如下：

```
float f1 = 13.23f;
double d1 = 4562.12d;
double d2 = 45678.1564;
```

**误区警示**

浮点值属于近似值，在系统中运算后的结果可能与实际有偏差。

**【例 3.2】**根据身高体重计算 BMI 指数（**实例位置：资源包\TM\sl\3\2**）

创建 BMIexponent 类；声明 double 型变量 height 以记录身高，单位为米；声明 int 型变量 weight 以记录体重，单位为千克；根据 BMI=体重/(身高×身高)计算 BMI 指数。实例代码如下：

```
public class BMIexponent {
    public static void main(String[] args) {
        double height = 1.72;                                   //身高变量，单位：米
        int weight = 70;                                        //体重变量，单位：千克
        double exponent = weight / (height * height);           //BMI 计算公式
        System.out.println("您的身高为：" + height);
        System.out.println("您的体重为：" + weight);
        System.out.println("您的 BMI 指数为：" + exponent);
        System.out.print("您的体重属于：");
        if (exponent < 18.5) {                                  //判断 BMI 指数是否小于 18.5
            System.out.println("体重过轻");
        }
        if (exponent >= 18.5 && exponent < 24.9) {              //判断 BMI 指数是否为 18.5～24.9
            System.out.println("正常范围");
        }
        if (exponent >= 24.9 && exponent < 29.9) {              //判断 BMI 指数是否为 24.9～29.9
            System.out.println("体重过重");
        }
        if (exponent >= 29.9) {                                 //判断 BMI 指数是否大于 29.9
            System.out.println("肥胖");
        }
    }
}
```

运行结果如下：

```
您的身高为：1.72
您的体重为：70
您的 BMI 指数为：23.661438615467823
您的体重属于：正常范围
```

## 3.2.3 字符类型

### 1．char 型

字符类型（char）用于存储单个字符，占用 16 位（两个字节）的内存空间。在定义字符型变量时，要用单引号表示，如's'表示一个字符。但是"s"则表示一个字符串，虽然只有一个字符，但由于使用双引号，因此它仍然表示字符串，而不是字符。

使用 char 关键字可定义字符变量，其语法如下：

```
char x = 'a';
```

由于字符 a 在 Unicode 表中的排序位置是 97，因此允许将上面的语句写成：

```
char x = 97;
```

同 C 和 C++语言一样，Java 语言也可以把字符作为整数对待。由于 Unicode 编码采用无符号编码，可以存储 65536 个字符（0x0000～0xffff），因此 Java 中的字符几乎可以处理所有国家的语言文字。若想得到一个 0～65536 的数所代表的 Unicode 表中相应位置上的字符，必须使用 char 型显式转换。

**【例 3.3】查看字符与 Unicode 码互转的结果（实例位置：资源包\TM\sl\3\3）**

在项目中创建类 Gess，编写如下代码，将 Unicode 表中某些位置上的字符以及一些字符在 Unicode 表中的位置输出到控制台上。

```
public class Gess {                                                    //定义类
    public static void main(String[] args) {                          //主方法
        char word = 'd', word2 = '@';                                  //定义 char 型变量
        int p = 23045, p2 = 45213;                                     //定义 int 型变量
        System.out.println("d 在 Unicode 表中的顺序位置是：" + (int) word);
        System.out.println("@在 Unicode 表中的顺序位置是：" + (int) word2);
        System.out.println("Unicode 表中的第 23045 位是：" + (char) p);
        System.out.println("Unicode 表中的第 45213 位是：" + (char) p2);
    }
}
```

运行结果如下：

```
d 在 Unicode 表中的顺序位置是：100
@在 Unicode 表中的顺序位置是：64
Unicode 表中的第 23045 位是：娅
Unicode 表中的第 45213 位是：?
```

String 类型为字符串类型，可以用来保存由多个字符组成的文本内容，其用法与字符类型类似，但文本内容需要用双引号标注。关于字符串的详细用法请参考本书第 10 章内容。

### 2．转义字符

转义字符是一种特殊的字符变量，它以反斜杠“\”开头，后跟一个或多个字符。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”。例如，printf 函数的格式串中用到的“\n”就是一个转义字符，意思是“回车换行”。Java 中的转义字符如表 3.3 所示。

表 3.3　转义字符

| 转 义 字 符 | 含　　义 | 转 义 字 符 | 含　　义 |
|---|---|---|---|
| \ddd | 1～3 位八进制数据所表示的字符，如\123 | \r | 回车 |
| \uxxxx | 4 位十六进制数据所表示的字符，如\u0052 | \n | 换行 |
| \' | 单引号字符 | \b | 退格 |
| \\ | 反斜杠字符 | \f | 换页 |
| \t | 垂直制表符，将光标移到下一个制表符的位置 | | |

将转义字符赋值给字符变量时，与字符常量值一样需要使用单引号。

**【例 3.4】输出'\'字符和'★'字符（实例位置：资源包\TM\sl\3\4）**

'\'字符的转移字符为'\\'，'★'字符的 Unicode 码为 2605，实例代码如下：

```
public class Demo {
```

```
    public static void main(String[] args) {
        char c1 = '\\';                                    //将转义字符 '\\' 赋值给变量 c1
        char char1 = '\u2605';                             //将转义字符 '\u2605' 赋值给变量 char1
        System.out.println(c1);                            //输出结果\
        System.out.println(char1);                         //输出结果★
    }
}
```

运行结果如下：

```
\
★
```

### 3.2.4 布尔类型

布尔类型又称逻辑类型，简称布尔型，通过关键字 boolean 来定义布尔类型变量。布尔类型只有 true 和 false 两个值，分别代表布尔逻辑中的“真”和“假”。布尔值不能与整数类型进行转换。布尔类型通常被用在流程控制中，作为判断条件。定义布尔类型变量，代码如下：

```
boolean b;                                                 //定义布尔型变量 b
boolean b1, b2;                                            //定义布尔型变量 b1、b2
boolean b = true;                                          //定义布尔型变量 b，并赋给初值 true
```

**编程训练（答案位置：资源包\TM\sl\3\编程训练）**

【训练 1】统计粮仓的粮食　一个圆柱形粮仓，底面直径为 10 米，高为 3 米，该粮仓体积为多少立方米？如果每立方米屯粮 750 千克，该粮仓一共可储存多少千克粮食？

【训练 2】谁该缴税　员工 a 与员工 b 的月薪分别为 4500 元和 5500 元，判断哪位员工需要缴纳个人所得税，哪位员工不需要缴纳个人所得税。（假设工资、薪金所得的个税起征点为 5000 元）

## 3.3 变量与常量

在程序执行过程中，其值不能被改变的量称为常量，其值能被改变的量称为变量。变量与常量的命名都必须使用合法的标识符。本节将向读者讲解标识符与关键字、变量与常量的声明、变量的有效范围。

### 3.3.1 标识符和关键字

#### 1．标识符

标识符可以简单地被理解为一个名字，它是用来标识类名、变量名、方法名、数组名、文件名的有效字符序列。

Java 语言规定标识符由任意顺序的字母、下画线（_）、美元符号（$）和数字组成，并且第一个字符不能是数字。标识符不能是 Java 中的关键字（保留字）。

下面是合法标识符：

```
name
user_age
$page
```

下面是非法标识符：

```
4word
String
User name
```

在 Java 语言中，标识符中的字母是严格区分大小写的，如 good 和 Good 是不同的两个标识符。Java 语言使用 Unicode 标准字符集，最多可以标识 65535 个字符。因此，Java 语言中的字母不仅包括通常的拉丁文字 a、b、c 等，还包括汉语、日语以及其他许多语言中的文字。

### 2．关键字

关键字又称保留字，是 Java 语言中已经被赋予特定意义的一些单词，不可以把这些单词作为标识符来使用。3.2 节介绍数据类型时提到的 int、boolean 等都是关键字。Java 语言中的关键字如表 3.4 所示。

表 3.4　Java 关键字

| 关　键　字 | 说　　明 | 关　键　字 | 说　　明 |
|---|---|---|---|
| abstract | 表明类或者成员方法具有抽象属性 | class | 用于声明类 |
| assert | 断言，用来调试程序 | const | 保留关键字，没有具体含义 |
| boolean | 布尔类型 | continue | 回到一个块的开始处 |
| break | 跳出语句，提前跳出一块代码 | default | 默认，如在 switch 语句中表示默认分支 |
| byte | 字节类型 | do | do…while 循环结构使用的关键字 |
| case | 用在 switch 语句之中，表示其中的一个分支 | double | 双精度浮点类型 |
| catch | 用在异常处理中，用来捕捉异常 | else | 用在条件语句中，表明当条件不成立时的分支 |
| char | 字符类型 | enum | 用于声明枚举 |
| extends | 用于创建继承关系 | public | 公有权限修饰符 |
| final | 用于声明不可改变的最终属性，如常量 | return | 返回方法结果 |
| finally | 声明异常处理语句中始终会被执行的代码块 | short | 短整数类型 |
| float | 单精度浮点类型 | static | 静态修饰符 |
| for | for 循环语句关键字 | strictfp | 用于声明 FP_strict（单精度或双精度浮点数）表达式遵循 IEEE 754 算术标准 |
| goto | 保留关键字，没有具体含义 | super | 父类对象 |
| if | 条件判断语句关键字 | switch | 分支结构语句关键字 |
| implements | 用于创建类与接口的实现关系 | synchronized | 线程同步关键字 |
| import | 导入语句 | this | 本类对象 |
| instanceof | 判断两个类的继承关系 | throw | 抛出异常 |
| int | 整数类型 | throws | 方法将异常处理抛向外部方法 |
| interface | 用于声明接口 | transient | 声明不用序列化的成员域 |
| long | 长整数类型 | try | 尝试监控可能抛出异常的代码块 |
| native | 用于声明一个方法是由与计算机相关的语言（如 C/C++/FORTRAN 语言）实现的 | var | 声明局部变量 |
| new | 用于创建新实例对象 | void | 表明方法无返回值 |
| package | 包语句 | volatile | 表明两个或多个变量必须同步发生变化 |
| private | 私有权限修饰符 | while | while 循环语句关键字 |
| protected | 受保护权限修饰符 | | |

## 3.3.2 声明变量

变量的使用是程序设计中一个十分重要的环节。声明变量就是要告诉编译器（compiler）这个变量的数据类型，这样编译器才知道需要配置多少空间给它，以及它能存放什么样的数据。在程序运行过程中，空间内的值是变化的，这个内存空间就称为变量。为了便于操作，给这个空间取个名字，称为变量名。变量名必须是合法的标识符。内存空间内的值就是变量值。在声明变量时可以不用赋值，也可以直接赋予初值。

例如，声明一个整数类型变量和声明一个字符类型变量，代码如下：

```
int age;                                    //声明 int 型变量
char char1 = 'r';                           //声明 char 型变量并赋值
```

编写以上程序代码，究竟会产生什么样的效果呢？要了解这个问题，就需要对变量的内存配置有一定的认识。用图解的方式将上述程序代码在内存中的状况表现出来，如图 3.2 所示。

由图 3.2 可知，系统的内存大略可被分为 3 个区域，即系统（OS）区、程序（program）区和数据（data）区。当执行程序时，程序代码会被加载到内存的程序区中，数据暂时被存储在数据区中。假设上述两个变量被定义在方法体中，则程序被加载到程序区中。当执行此行程序代码时，会在数据区配置空间给出这两个变量。

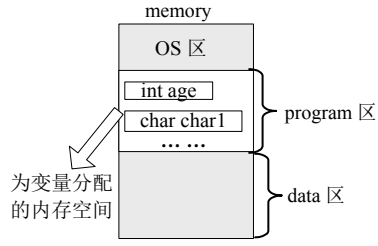



图 3.2　变量占用的内存空间

对于变量的命名并不是随意的，应遵循以下几条规则：

- ☑ 变量名必须是一个有效的标识符。
- ☑ 变量名不可以使用 Java 中的关键字。
- ☑ 变量名不能重复。
- ☑ 应选择有意义的单词作为变量名。

**说明**

在 Java 语言中允许使用汉字或其他语言文字作为变量名，如“int 年龄 ＝21”，在程序运行时不会出现错误，但建议读者尽量不要使用这些语言文字作为变量名。

Java 10 提供了一个方便好用的新特性：使用 var 声明局部变量。使用 var 声明局部变量的语法如下：

```
var 变量名称 = 值
```

需要注意的是，var 是关键字，它相当于一种动态类型。编译器会根据赋给变量的值推断出变量的类型，因此使用 var 声明局部变量时必须赋予值。

例如，在 main()方法中，先使用 var 声明一个变量，变量的值为“好好学习，天天向上”，再使用输出语句输出这个变量的值。代码如下：

```
public class Demo {
    public static void main(String[] args) {
        var str = "好好学习，天天向上";
        System.out.println(str);
    }
}
```

运行结果如下：

```
好好学习，天天向上
```

此外，还需要注意的是：var 不能用于声明成员变量，如图 3.3 所示；使用 var 声明的局部变量不能作为方法的返回值，如图 3.4 所示。

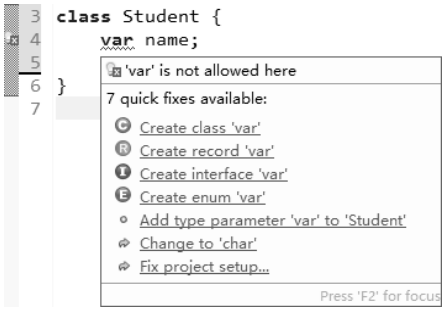



图 3.3　var 不能用于声明成员变量

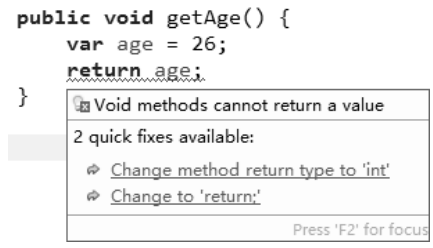



图 3.4　使用 var 声明的局部变量不能作为方法的返回值

## 3.3.3　声明常量

在程序运行过程中一直不会改变的量被称为常量（constant），通常也被称为“final 变量”。常量在整个程序中只能被赋值一次。在为所有的对象共享值时，常量是非常有用的。

在 Java 语言中声明一个常量，除了要指定数据类型，还需要通过 final 关键字进行限定。声明常量的标准语法如下：

```
final 数据类型 常量名称 [= 值]
```

常量名通常使用大写字母，但这并不是必需的。很多 Java 程序员使用大写字母表示常量，是为了清楚地表明正在使用常量。

例如，声明常量 π（程序中用 PI 表示），代码如下：

```
final double PI = 3.1415926D;                    //声明 double 型常量 PI 并赋值
```

当变量被 final 关键字修饰时，该变量就变成了常量，必须在定义时就设定它的初值，否则将会产生编译错误。从下面的实例中可看出变量与常量的区别。

**【例 3.5】尝试给常量赋值，观察是否会发生错误（实例位置：资源包\TM\sl\3\5）**

在项目中创建类 Part，在类体中创建变量 age 与常量 PI。在主方法中分别对变量和常量进行赋值，通过输出信息可测试变量与常量的有效范围。

```
public class Part {                                  //新建类 Part
    //声明常量 PI，此时如不对 PI 进行赋值，则会出现错误提示
    static final double PI = 3.14;
    static int age = 23;                             //声明 int 型变量 age 并进行赋值

    public static void main(String[] args) {         //主方法
        final int number;                            //声明 int 型常量 number
        number = 1235;                               //对常量进行赋值
        age = 22;                                    //再次对变量进行赋值
        number = 1236;                               //错误代码，number 为常量，只能赋值一次
        System.out.println("常量 PI 的值为：" + PI);    //输出 PI 的值
        System.out.println("赋值后 number 的值为:" + number);  //输出 number 的值
        System.out.println("int 型变量 age 的值为：" + age);   //输出 age 的值
    }
}
```

运行结果如下：

```
Exception in thread "main" java.lang.Error: 无法解析的编译问题：
        final 局部变量 number 可能已经被赋过值
        at Part.main(Part.java:10)
```

从这个结果中可以看到，Part 类被运行后发生了错误，异常日志中记载 Part 类出现编译问题，此编译问题正是常量 number 被二次赋值。

**说明**

英文版原版的异常日志为：

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The final local variable number may already have been assigned
        at Part.main(Part.java:10)
```

## 3.3.4 变量的有效范围

由于变量被定义出来后只是暂存在内存中，等到程序执行到某一个点，该变量会被释放掉，也就是说变量有它的生命周期。因此，变量的有效范围是指程序代码能够访问该变量的区域，若超出该区域，则在编译时会出现错误。在程序中，一般会根据变量的“有效范围”将变量分为“成员变量”和“局部变量”。

### 1．成员变量

在类体中所声明的变量被称为成员变量，成员变量在整个类中都有效。类的成员变量又可分为两种，即静态变量和实例变量。例如下面这段代码：

```
class Demo{
        int x = 45;
        static int y = 90
}
```

其中，x 为实例变量，y 为静态变量（也被称为类变量）。如果在成员变量的类型前面加上关键字 static，这样的成员变量被称为静态变量。静态变量的有效范围可以跨类，甚至可到达整个应用程序之内。静态变量除了能在声明它的类内存取，还能直接以“类名.静态变量”的方式在其他类内使用。

### 2．局部变量

在类的方法体中声明的变量（方法内部定义，在“{”与“}”之间的代码中声明的变量）称为局部变量。局部变量只在当前代码块中有效，也就是只能在“{”与“}”之间的代码中使用它。

在类的方法中声明的变量，包括方法的参数，都属于局部变量。局部变量只在当前定义的方法内有效，不能用于类的其他方法中。局部变量的生命周期取决于方法，当方法被调用时，Java 虚拟机会为方法中的局部变量分配内存空间，当该方法的调用结束后，则会释放方法中局部变量占用的内存空间，局部变量也将会被销毁。

局部变量可与成员变量的名字相同，此时成员变量将被隐藏，即这个成员变量在此方法中暂时失效。

变量的有效范围如图 3.5 所示。

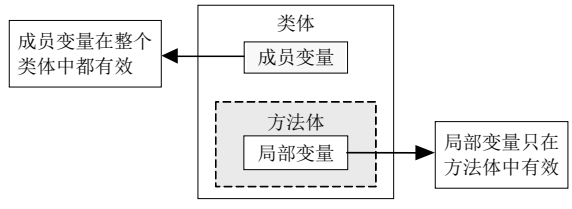



图 3.5　变量的有效范围

【例 3.6】把成员变量“排挤掉”的局部变量（**实例位置：资源包\TM\sl\3\6**）

在项目中创建类 Val，并分别定义名称相同的 成员变量与局部变量，当名称相同时成员变量将被隐藏。

```
public class Val {                                          //新建类
    static int times = 3;                                   //定义成员变量 times
    public static void main(String[] args) {                //主方法
        int times = 4;                                      //定义局部变量 times
        System.out.println("times 的值为：" + times);        //输出 times 的值
    }
}
```

运行结果如下：

```
times 的值为：4
```

**编程训练（答案位置：资源包\TM\sl\3\编程训练）**

【训练 3】比较字符和整数　比较'g'和 103 是否相等。

【训练 4】输出连续的字符　在控制台中输出“ABCDEFG”。

# 3.4 运　算　符

运算符是一些特殊的符号，主要用于数学函数、一些类型的赋值语句和逻辑比较方面。Java 中提供了丰富的运算符，如赋值运算符、算术运算符、比较运算符等。本节将向读者介绍这些运算符。

## 3.4.1 赋值运算符

赋值运算符以符号“=”表示，它是一个二元运算符（对两个操作数做处理），其功能是将右方操作数所含的值赋给左方的操作数。例如：

```
int a = 100;
```

该表达式是将 100 赋值给变量 a。左方的操作数必须是一个变量，而右边的操作数则可以是任何表达式，包括变量（如 a、number）、常量（如 123、'book'）、有效的表达式（如 45 * 12）。

由于赋值运算符“=”处理时会先取得右方表达式处理后的结果，因此一个表达式中若含有两个以上的“=”运算符，会从最右方的“=”开始处理。

【例 3.7】使用赋值运算符同时为两个变量赋值（**实例位置：资源包\TM\sl\3\7**）

在项目中创建类 Eval，在主方法中定义变量，使用赋值运算符为变量赋值。

```
public class Eval {                                         //创建类
    public static void main(String[] args) {                //主方法
        int a, b, c;                                        //声明 int 型变量 a、b、c
        a = 15;                                             //将 15 赋值给变量 a
        c = b = a + 4;                                      //将 a 与 4 的和赋值给变量 b，然后赋值给变量 c
        System.out.println("c 值为：" + c);                  //输出变量 c 的值
        System.out.println("b 值为：" + b);                  //输出变量 b 的值
    }
}
```

运行结果如下：

```
c 值为：19
b 值为：19
```

**说明**

在 Java 中可以把赋值运算符连在一起使用。如：

```
x = y = z = 5;
```

在这个语句中，变量 x、y、z 都得到同样的值 5，但在实际开发中建议开发者分开对其进行赋值，这样可以让代码的层次更清晰。

## 3.4.2 算术运算符

Java 中的算术运算符主要有+（加）、-（减）、*（乘）、/（除）、%（求余），它们都是二元运算符。Java 中算术运算符的功能及使用方式如表 3.5 所示。

表 3.5 Java 算术运算符

| 运 算 符 | 说 明 | 实 例 | 结 果 |
|---|---|---|---|
| + | 加 | 12.45f + 15 | 27.45 |
| - | 减 | 4.56 - 0.16 | 4.4 |
| * | 乘 | 5L * 12.45f | 62.25 |
| / | 除 | 7 / 2 | 3 |
| % | 取余数 | 12 % 10 | 2 |

其中，“+”和“-”运算符还可以作为数值的正负符号，如+5、−7。

**注意**

在进行除法运算时，0 不能做除数。例如，对于语句“int a = 5 / 0;”，系统会抛出 ArithmeticException 异常。

下面通过一个小程序来介绍算术运算符的使用方法。

**【例 3.8】**使用算术运算符模拟计算器（**实例位置：资源包\TM\sl\3\8**）

创建 ArithmeticOperator 类，让用户输入两个数字，分别用 5 种运算符对这两个数字进行计算。

```
import java.util.Scanner;
public class ArithmeticOperator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);                          //创建扫描器，获取控制台输入的值
        System.out.println("请输入两个数字，用空格隔开(num1 num2)：");    //输出提示
        double num1 = sc.nextDouble();                                //记录输入的第一个数字
        double num2 = sc.nextDouble();                                //记录输入的第二个数字
        System.out.println("num1+num2 的和为：" + (num1 + num2));       //计算和
        System.out.println("num1-num2 的差为：" + (num1 - num2));       //计算差
        System.out.println("num1*num2 的积为：" + (num1 * num2));       //计算积
        System.out.println("num1/num2 的商为：" + (num1 / num2));       //计算商
        System.out.println("num1%num2 的余数为：" + (num1 % num2));     //计算余数
        sc.close();                                                   //关闭扫描器
```

```
    }
}
```

运行结果如图 3.6 所示，图中数字 23 和 15 为用户在控制台中输入的值。

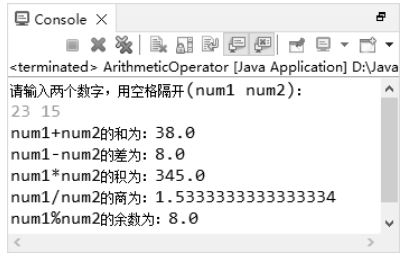



图 3.6　运行结果

**说明**

代码中出现的 Scanner 扫描器类可以让程序获得用户在控制台输入的值，关于 Scanner 类的详细用法请参考本书“第 11 章 常用类库”的相关内容。

## 3.4.3　自增和自减运算符

自增、自减运算符是单目运算符，可以放在操作元之前，也可以放在操作元之后。操作元必须是一个整型或浮点型变量。自增、自减运算符的作用是使变量的值增 1 或减 1。放在操作元前面的自增、自减运算符，会先将变量的值加 1（减 1），然后使该变量参与表达式的运算。放在操作元后面的自增、自减运算符，会先使变量参与表达式的运算，然后将该变量加 1（减 1）。例如：

```
++a(--a)                    //表示在使用变量 a 之前，先将 a 的值加（减）1
a++(a--)                    //表示在使用变量 a 之后，将 a 的值加（减）1
```

粗略地分析，“++a”与“a++”的作用都相当于 a = a + 1。假设 a = 4，则：

```
b = ++a;                    //先将 a 的值加 1，然后赋给 b，此时 a 值为 5，b 值为 5
```

再看另一个语法，同样假设 a = 4，则：

```
b = a++;                    //先将 a 的值赋给 b，再将 a 的值变为 5，此时 a 值为 5，b 值为 4
```

## 3.4.4　比较运算符

比较运算符属于二元运算符，用于程序中的变量之间、变量和自变量之间以及其他类型的信息之间的比较。比较运算符的运算结果是 boolean 型。当运算符对应的关系成立时，运算结果为 true，否则为 false。所有比较运算符通常作为判断的依据用在条件语句中。比较运算符共有 6 个，如表 3.6 所示。

表 3.6　比较运算符

| 运　算　符 | 作　　用 | 举　　例 | 操 作 数 据 | 结　　果 |
|---|---|---|---|---|
| > | 比较左方是否大于右方 | 'a' > 'b' | 整型、浮点型、字符型 | false |
| < | 比较左方是否小于右方 | 156 < 456 | 整型、浮点型、字符型 | true |
| == | 比较左方是否等于右方 | 'c' == 'c' | 基本数据类型、引用型 | true |
| >= | 比较左方是否大于或等于右方 | 479 >= 426 | 整型、浮点型、字符型 | true |
| <= | 比较左方是否小于或等于右方 | 12.45 <= 45.5 | 整型、浮点型、字符型 | true |
| != | 比较左方是否不等于右方 | 'y'! = 't' | 基本数据类型、引用型 | true |

**【例 3.9】**使用不同的比较运算符判断两个整数的关系（**实例位置：资源包\TM\sl\3\9**）

在项目中创建类 Compare，在主方法中创建整型变量，使用比较运算符对变量进行比较运算，并

输出运算后的结果。

```
public class Compare {                                          //创建类
    public static void main(String[] args) {
        int number1 = 4;                                        //声明 int 型变量 number1
        int number2 = 5;                                        //声明 int 型变量 number2
        //依次输出变量 number1 与变量 number2 的比较结果
        System.out.println("number1>number2 的返回值为：" + (number1 > number2));
        System.out.println("number1< number2 返回值为："+ (number1 < number2));
        System.out.println("number1==number2 返回值为："+ (number1 == number2));
        System.out.println("number1!=number2 返回值为："+ (number1 != number2));
        System.out.println("number1>= number2 返回值为："+ (number1 >= number2));
        System.out.println("number1<=number2 返回值为："+ (number1 <= number2));
    }
}
```

运行结果如下：

```
number1>number2 的返回值为：false
number1< number2 返回值为：true
number1==number2 返回值为：false
number1!=number2 返回值为：true
number1>= number2 返回值为：false
number1<=number2 返回值为：true
```

## 3.4.5 逻辑运算符

返回类型为布尔型的表达式（如比较运算符）可以被组合在一起构成一个更复杂的表达式。这是通过逻辑运算符来实现的。逻辑运算符包括&（&&）（逻辑与）、||（逻辑或）、!（逻辑非）。逻辑运算符的操作元必须是 boolean 型数据。在逻辑运算符中，除了“!”是一元运算符，其他都是二元运算符。表 3.7 给出了逻辑运算符的用法和含义。

表 3.7 逻辑运算符

| 运 算 符 | 含 义 | 用 法 | 结 合 方 向 |
|---|---|---|---|
| &&、& | 逻辑与 | op1 && op2 | 从左到右 |
| \|\| | 逻辑或 | op1 \|\| op2 | 从左到右 |
| ! | 逻辑非 | !op | 从右到左 |

结果为 boolean 型的变量或表达式可以通过逻辑运算符组合为逻辑表达式。

用逻辑运算符进行逻辑运算时，结果如表 3.8 所示。

表 3.8 使用逻辑运算符进行逻辑运算

| 表达式 1 | 表达式 2 | 表达式 1 && 表达式 2 | 表达式 1 \|\| 表达式 2 | !表达式 1 |
|---|---|---|---|---|
| true | true | true | true | false |
| true | false | false | true | false |
| false | false | false | false | true |
| false | true | false | true | true |

逻辑运算符“&&”与“&”都表示“逻辑与”，那么它们之间的区别在哪里呢？从表 3.8 中可以看出，当两个表达式都为 true 时，“逻辑与”的结果才会是 true。使用逻辑运算符“&”会判断两个表达式；而逻辑运算符“&&”则是针对 boolean 类型的类进行判断的，当第一个表达式为 false 时则不去判

断第二个表达式，直接输出结果，从而节省计算机判断的次数。通常将这种在逻辑表达式中从左端的表达式可推断出整个表达式的值的情况称为“短路”，而将那些始终需要执行逻辑运算符两边的表达式才能推断出整个表达式的值的情况称为“非短路”。“&&”属于“短路”运算符，而“&”属于“非短路”运算符。

**【例 3.10】**使用不同的比较运算符判断两个整数的关系（**实例位置：资源包\TM\sl\3\10**）

在项目中创建类 Calculation，在主方法中创建 3 个整数，分别记录男生人数、女生人数和总人数，使用逻辑运算符来判断“男生人数大于女生人数并且总人数大于 30 人”和“男生人数大于女生人数或者总人数大于 30 人”这两种情况是否存在。

```
public class Calculation {
    public static void main(String[] args) {
        int boys = 15;                                          //男生人数
        int girls = 17;                                         //女生人数
        int totle = boys + girls;                               //总人数
        boolean result1 = ((boys > girls) && (totle > 30));     //男生人数多于女生，且总人数大于 30
        boolean result2 = ((boys > girls) || (totle > 30));     //男生人数多于女生，或总人数大于 30
        System.out.println("男生人数大于女生人数并且总人数大于 30 人：" + result1);      //输出结果
        System.out.println("男生人数大于女生人数或者总人数大于 30 人：" + result2);
    }
}
```

运行结果如下：

```
男生人数大于女生人数并且总人数大于 30 人：false
男生人数大于女生人数或者总人数大于 30 人：true
```

## 3.4.6　位运算符

位运算符除“按位与”和“按位或”运算符外，其他只能用于处理整数的操作数，包括 byte、short、char、int 和 long 等数据类型。位运算是完全针对位方面的操作。整型数据在内存中以二进制的形式进行表示，如 int 型变量 7 的二进制表示是 00000000 00000000 00000000 00000111。

左边最高位是符号位，最高位是 0 表示正数，若为 1 则表示负数。负数采用补码表示，如-8 的二进制表示为 11111111 11111111 1111111 11111000。这样就可以对整型数据进行按位运算。

### 1.“按位与”运算

“按位与”运算的运算符为“&”，为双目运算符。“按位与”运算的运算法则是：如果两个整型数据 a、b 对应位都是 1，则结果位才是 1，否则为 0。如果两个操作数的精度不同，则结果的精度与精度高的操作数相同，如图 3.7 所示。

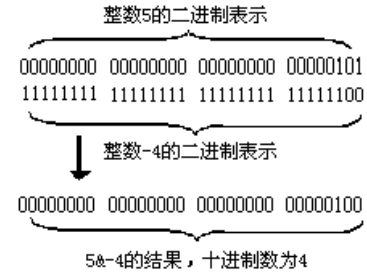



图 3.7　5 & -4 的运算过程

### 2.“按位或”运算

“按位或”运算的运算符为“|”，为双目运算符。“按位或”运算的运算法则是：如果两个操作数对应位都是 0，则结果位才是 0，否则为 1。如果两个操作数的精度不同，则结果的精度与精度高的操作数相同，如图 3.8 所示。

### 3．“按位取反”运算

“按位取反”运算也称“按位非”运算，运算符为“~”，为单目运算符。“按位取反”就是将操作数二进制中的 1 修改为 0，0 修改为 1，如图 3.9 所示。

### 4．“按位异或”运算

“按位异或”运算的运算符是“^”，为双目运算符。“按位异或”运算的运算法则是：当两个操作数的二进制表示相同（同时为 0 或同时为 1）时，结果为 0，否则为 1。若两个操作数的精度不同，则结果的精度与精度高的操作数相同，如图 3.10 所示。

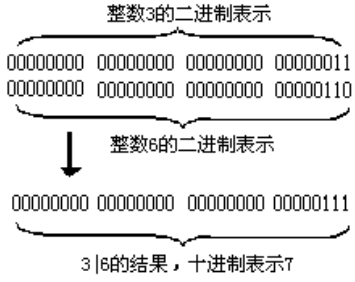



图 3.8　3 | 6 的运算过程

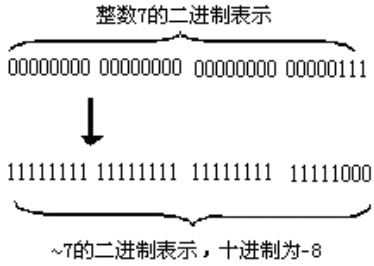



图 3.9　~7 的运算过程


图 3.10　10 ^ 3 的运算过程

### 5．移位操作

除了上述运算符，还可以对数据按二进制位进行移位操作。Java 中的移位运算符有以下 3 种：

☑ <<：左移。

☑ >>：右移。

☑ >>>：无符号右移。

左移就是将运算符左边的操作数的二进制数据，按照运算符右边操作数指定的位数向左移动，右边移空的部分补 0。右移则复杂一些。当使用“>>”符号时：如果最高位是 0，右移空的位就填入 0；如果最高位是 1，右移空的位就填入 1，如图 3.11 所示。

Java 还提供了无符号右移“>>>”，无论最高位是 0 还是 1，左侧被移空的高位都填入 0。

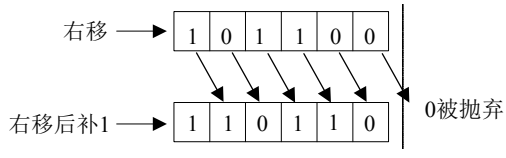



图 3.11　右移

**技巧**

移位可以实现整数除以或乘以 $2^n$ 的效果。例如，y << 2 与 y*4 的结果相同，y >> 1 的结果与 y / 2 的结果相同。总之：一个数左移 *n* 位，就是将这个数乘以 $2^n$；一个数右移 *n* 位，就是将这个数除以 $2^n$。

## 3.4.7　复合赋值运算符

和其他主流编程语言一样，Java 中提供了复合赋值运算符。所谓复合赋值运算符，就是将赋值运算符与其他运算符合并成一个运算符来使用，从而同时实现两种运算符的效果。Java 中的复合运算符如表 3.9 所示。

表 3.9　复合赋值运算符

| 运　算　符 | 含　　义 | 举　　例 | 等 价 效 果 |
|---|---|---|---|
| += | 相加结果赋予左侧 | a += b; | a = a + b; |
| -= | 相减结果赋予左侧 | a -= b; | a = a – b; |
| *= | 相乘结果赋予左侧 | a *= b; | a = a * b; |
| /= | 相除结果赋予左侧 | a /= b; | a = a / b; |
| %= | 取余结果赋予左侧 | a %= b; | a = a % b; |
| &= | 与结果赋予左侧 | a &= b; | a = a & b; |
| \|= | 或结果赋予左侧 | a \|= b; | a = a \| b; |
| ^= | 异或结果赋予左侧 | a ^= b; | a = a ^ b; |
| <<= | 左移结果赋予左侧 | a <<= b; | a = a << b; |
| >>= | 右移结果赋予左侧 | a >>= b; | a = a >> b; |
| >>>= | 无符号右移结果赋予左侧 | a >>>= b; | a = a >>> b; |

以“+=”为例，虽然“a += 1”与“a = a + 1”二者最后的计算结果是相同的，但是在不同的场景下，两种运算符都有各自的优势和劣势：

（1）低精度类型自增。

在 Java 编译环境中，整数的默认类型时 int 型，因此下面的赋值语句会报错：

```
byte a = 1;                    //创建 byte 型变量 a
a = a + 1;                     //让 a 的值+1，错误提示：无法将 int 型转换成 byte 型
```

在没有进行强制转换的条件下，a+1 的结果是一个 int 值，无法直接赋给一个 byte 变量。但是如果使用“+=”实现递增计算，就不会出现这个问题。

```
byte a = 1;                    //创建 byte 型变量 a
a += 1;                        //让 a 的值+1
```

（2）不规则的多值相加。

“+=”虽然简洁、强大，但是有些时候是不好用的，比如下面这个语句：

```
a = (2 + 3 - 4) * 92 / 6;
```

这条语句如果改成使用复合赋值运算符，代码就会显得比较烦琐，代码如下：

```
a += 2;
a += 3;
a -= 4;
a *= 92;
a /= 6;
```

## 3.4.8　三元运算符

三元运算符的使用格式如下：

```
条件式 ? 值 1 : 值 2
```

三元运算符的运算法则是：若条件式的值为 true，则整个表达式取值 1，否则取值 2。例如：

```
boolean b = 20 < 45 ? true : false;
```

上述程序表达式“20 < 45”的运算结果返回真，那么 boolean 型变量 b 取值为 true。相反，如果表达式的运算结果返回为假，则 boolean 型变量 b 取值为 false。

三元运算符等价于 if…else 语句，例如上述代码等价于：

```
boolean a;              //声明 boolean 型变量
if(20<45)               //将 20 < 45 作为判断条件
   a = true;            //条件成立，将 true 赋值给 a
else
   a = false;           //条件不成立，将 false 赋值给 a
```

## 3.4.9 运算符的优先级

Java 中的表达式就是使用运算符连接起来的符合 Java 规则的式子。运算符的优先级决定了表达式中运算执行的先后顺序。通常，优先级由高到低的顺序依次是：

- ☑ 增量和减量运算。
- ☑ 算术运算。
- ☑ 比较运算。
- ☑ 逻辑运算。
- ☑ 赋值运算。

如果两个运算有相同的优先级，那么左边的表达式要比右边的表达式先被处理。表 3.10 显示了在 Java 中众多运算符特定的优先级。

表 3.10 运算符的优先级

| 优 先 级 | 描 述 | 运 算 符 | 优 先 级 | 描 述 | 运 算 符 |
|---|---|---|---|---|---|
| 1 | 圆括号 | () | 9 | 按位与运算 | & |
| 2 | 正负号 | +、- | 10 | 按位异或运算 | ^ |
| 3 | 一元运算符 | ++、--、! | 11 | 按位或运算 | \| |
| 4 | 乘除 | *、/、% | 12 | 逻辑与运算 | && |
| 5 | 加减 | +、- | 13 | 逻辑或运算 | \|\| |
| 6 | 移位运算 | >>、>>>、<< | 14 | 三元运算符 | ? : |
| 7 | 比较大小 | <、>、>=、<= | 15 | 赋值运算符 | = |
| 8 | 比较是否相等 | ==、!= | | | |

**技巧**

在编写程序时尽量使用圆括号来指定运算次序，以免产生错误的运算顺序。

**编程训练（答案位置：资源包\TM\sl\3\编程训练）**

【训练 5】计算机车加速度　平均加速度，即速度的变化量除以这个变化所用的时间。现有一辆轿车用了 8.7 秒从每小时 0 千米加速到每小时 100 千米，计算并输出这辆轿车的平均加速度。

【训练 6】求解二元一次方程组　使用克莱姆法则求解二元一次方程组。

$$\begin{cases} 21.8x + 2y = 28 \\ 7x + 8y = 62 \end{cases}$$

提示：克莱姆法则求解二元一次方程组的公式如下：

$$\begin{cases} ax+by=e \\ cx+dy=f \end{cases} \Rightarrow x=\frac{ed-bf}{ad-bc}, y=\frac{af-ec}{ad-bc}$$

# 3.5　数据类型转换

类型转换是将一个值从一种类型更改为另一种类型的过程。例如，可以将 String 类型的数据“457”转换为数值型，也可以将任意类型的数据转换为 String 类型。

如果从低精度数据类型向高精度数据类型转换，则永远不会溢出，并且总是成功的；而把高精度数据类型向低精度数据类型转换时，则会有信息丢失，有可能失败。

数据类型转换有两种方式，即隐式转换与显式转换。

## 3.5.1　隐式类型转换

从低级类型向高级类型的转换，系统将自动执行，程序员无须进行任何操作。这种类型的转换被称为隐式转换。下列基本数据类型会涉及数据转换，不包括逻辑类型和字符类型。这些类型按精度从低到高排列的顺序为 byte < short < int < long < float < double。

例如，可以将 int 型变量直接赋值给 float 型变量，此时 int 型变量将隐式转换成 float 型变量。代码如下：

```
int x = 50;                          //声明 int 型变量 x
float y = x;                         //将 x 赋值给 y，y 的值为 50.0
```

隐式转换也要遵循一定的规则来解决在什么情况下将哪种类型的数据转换成另一种类型的数据。表 3.11 列出了各种数据类型隐式转换的一般规则。

表 3.11　隐式类型转换规则

| 操作数 1 的数据类型 | 操作数 2 的数据类型 | 转换后的数据类型 |
|---|---|---|
| byte、short、char | int | int |
| byte、short、char、int、 | long | long |
| byte、short、char、int、long | float | float |
| byte、short、char、int、long、float | double | double |

下面通过一个简单实例介绍数据类型隐式转换。

**【例 3.11】**使用隐式转换提升数值的精度（**实例位置：资源包\TM\sl\3\11**）

在项目中创建类 Conver，在主方法中创建不同数值型的变量，实现将各变量隐式转换。

```
public class Conver {
    public static void main(String[] args) {
        byte mybyte = 127;                    //定义 byte 型变量 mybyte，并把允许的最大值赋给 mybyte
        int myint = 150;                      //定义 int 型变量 myint，并赋值
        float myfloat = 452.12f;              //定义 float 型变量 myfloat，并赋值
        char mychar = 10;                     //定义 char 型变量 mychar，并赋值
        double mydouble = 45.46546;           //定义 double 型变量，并赋值
        //输出运算结果
        System.out.println("byte 型与 float 型数据进行运算结果为：" + (mybyte + myfloat));
        System.out.println("byte 型与 int 型数据进行运算结果为：" + mybyte * myint);
```

```
            System.out.println("byte 型与 char 型数据进行运算结果为：" + mybyte / mychar);
            System.out.println("double 型与 char 型数据进行运算结果为：" + (mydouble + mychar));
        }
}
```

运行结果如下：

```
byte 型与 float 型数据进行运算结果为：579.12
byte 型与 int 型数据进行运算结果为：19050
byte 型与 char 型数据进行运算结果为：12
double 型与 char 型数据进行运算结果为：55.46546
```

**技巧**

要理解类型转换，读者可以这么想象，大脑前面是一片内存，源和目标分别是两个大小不同的内存块（由变量及数据的类型来决定），将源数据赋值给目标内存的过程，就是用目标内存块尽可能多地套取源内存中的数据。

### 3.5.2 显式类型转换

当把高精度的变量的值赋给低精度的变量时，必须使用显式类型转换运算（又称强制类型转换）。语法如下：

```
(类型名)要转换的值
```

例如，将高精度数字转换为低精度数字。代码如下：

```
int a = (int)45.23;                //此时输出 a 的值为 45
long y = (long)456.6F;             //此时输出 y 的值为 456
int b = (int)'d';                  //此时输出 b 的值为 100
```

执行显式类型转换时，可能会导致精度损失。除 boolean 类型外，其他基本类型都能以显式类型转换的方法实现转换。

**误区警示**

当把整数赋值给一个 byte、short、int、long 型变量时，不可以超出这些变量的取值范围，否则必须进行强制类型转换。例如:

```
byte b = (byte)129;
```

**编程训练（答案位置：资源包\TM\sl\3\编程训练）**

【训练 7】输出连续的英文字母　使用 char 型声明'a'～'g'，然后输出它们相加后的结果。

【训练 8】货车装箱子　一辆货车运输箱子，载货区宽 2 米、长 4 米，一个箱子宽 1.5 米、长 1.5 米，请问载货区一层可以放多少个箱子？

## 3.6 代码注释与编码规范

在程序代码中适当地添加注释，可以提高程序的可读性和可维护性。好的编码规范可以使程序更

易阅读和理解。本节将介绍 Java 中的几种代码注释方法以及应该注意的编码规范。

## 3.6.1　代码注释

通过在程序代码中添加注释可提高程序的可读性。注释中包含了程序的信息，可以帮助程序员更好地阅读和理解程序。在 Java 源程序文件的任意位置处都可添加注释语句。因为 Java 编译器不会对注释语句进行编译，所以代码中的所有注释语句都不会对程序产生任何影响。Java 语言提供了 3 种添加注释的方法，分别为单行注释、多行注释和文档注释。

### 1．单行注释

“//”为单行注释标记，从符号“//”开始直到换行的所有内容均作为注释而被编译器忽略。语法如下：

```
//注释内容
```

例如，以下代码为声明的 int 型变量添加注释：

```
int age ;                    //定义 int 型变量，用于保存年龄信息
```

### 2．多行注释

“/* */”为多行注释标记，符号“/*”与“*/”之间的所有内容均为注释内容。注释中的内容可以换行。语法如下：

```
/*
注释内容 1
注释内容 2
…
*/
```

**注意**

（1）在多行注释中可嵌套单行注释。例如：

```
/*
   程序名称：Hello world   //开发时间：2021-03-05
*/
```

（2）多行注释中不可以嵌套多行注释，以下代码是错误的：

```
/*
    程序名称：Hello world
    /* 开发时间：2021-03-05；作者：张先生 */
*/
```

### 3．文档注释

“/** */”为文档注释标记。符号“/**”与“*/”之间的内容均为文档注释内容。当文档注释出现在声明（如类的声明、类的成员变量的声明、类的成员方法的声明等）之前时，会被 Javadoc 文档工具读取作为 Javadoc 文档内容。除注释标记不同外，文档注释的格式与多行注释的格式相同。对于初学者而言，文档注释并不是很重要，了解即可。

**说明**

一定要养成良好的编程习惯。软件编码规范中提到“可读性第一，效率第二”，所以程序员必须在程序中添加适量的注释来提高程序的可读性和可维护性。程序中，注释要占程序代码总量的20%～50%。

## 3.6.2 编码规范

在学习开发的过程中要养成良好的编码习惯，规整的代码格式会为程序日后的维护工作提供极大的便利。在此对编码规则做了以下总结，供读者学习。

☑ 每条语句尽量单独占一行，并且每条语句都要以分号结束。

**注意**

程序代码中的分号必须是在英文状态下输入的，初学者经常会将“;”写成中文状态下的“；”，此时编译器会报出 Invalid Character（非法字符）这样的错误信息。

☑ 在声明变量时，尽量使每个变量单独占一行，即使有多个数据类型相同的变量，也应将其各自放置在单独的一行上，这样有助于添加注释。对于局部变量，应在声明它们的同时为它们赋予初始值。

☑ 在 Java 代码中，空格仅提供分隔使用，无其他含义，开发者应控制好空格的数量，不要写过多的无用空格。例如：

```
public static void main ( String args[ ] )
```

等价于

```
public static void main(String args[])
```

☑ 为了方便日后的维护，不要使用技术性很高、难懂、易混淆的语句。因为程序的开发者与维护者可能不是同一个人，所以应尽量使用简洁、清晰的代码编写程序需要的功能。

☑ 对于关键的方法要多加注释，这样有助于阅读者了解代码的结构与设计思路。

代码应该写在哪？这可能是第一次学习编程的读者最大的疑惑了。笔者对 Java 代码的主要结构做出了几点总结：

☑ package 语句要写在类文件的第一行。如果类不在任何包中，可以不写 package 语句。

☑ import 语句要写在类上方、package 语句下方。

☑ 方法必须写在类的{ }之内。在方法的{ }内不可以创建其他方法。

☑ 类的成员变量必须定义在类的{ }之内、方法的{ }之外的位置。在方法的{ }之内定义的变量均为局部变量。

☑ 除了上面几种类型的代码，其他类型代码都应该写在某个{ }中（如代码块或方法体之内）。其他类型的代码包括局部变量、内部类等。

如果你现在无法读懂这几点总结，请不要焦虑，这里出现的很多概念、语句在后面的内容中都会重点讲解。只要勤加练习，这些注意事项自然就会掌握。

# 3.7　实践与练习

**（答案位置：资源包\TM\sl\3\实践与练习）**

综合练习 1：象棋口诀　先使用 char 型变量定义“马”“象”“卒”3 个棋子，再输出“马走日，象走田，小卒一去不复还”的象棋口诀。

综合练习 2：输出汇款单　向张三卡号为 1234567890987654321 的银行卡里汇款 10000 元，控制台输出如下所示的汇款单：

```
      中国工商银行


----------------------------
 日期：     2021-03-10
 户名：     张三
 账号：     1234567890987654321
 币种：     RMB
 存款金额：       10000.0
 存款序号：       010
 柜员号：  12345
----------------------------
```

综合练习 3：输出个人信息　控制台输出如下所示的个人基本信息。

```
      个人基本信息
-------------------------
姓名：      李四
性别：      男
年龄：      25
身高：      1.76 米
体重：      65.5 千克
是否已婚：false
```

综合练习 4：计算月收入　小李每月的工资是 4500 元，每月的奖金是 1000 元，每月要缴纳的五险一金合计是 500 元，计算小李每月的最终收入是多少元？

综合练习 5：计算商和余数　应用除法运算符可以计算两个数的商，应用取余运算符可以计算两个数相除所得的余数。根据这两个运算符做一个数字转置的练习，将 123 的各数字顺序前后颠倒后进行输出。

综合练习 6：判断成绩能否及格　当分数大于或等于 60 时，成绩及格，否则不及格。现一名学生的分数是 80 分，使用三元运算符判断这名学生的成绩能否及格。

综合练习 7：话费充值　向手机中充值 10 元。通话 0.2 元/分钟，通话时长已有 30 分钟；流量已使用 10MB，流量费用为 0.3 元/MB。计算话费余额还可以通话的时长。

综合练习 8：货车装西瓜　一货车的车厢长 400 厘米、宽 160 厘米、高 30 厘米，现有 100 个直径约为 23 厘米的西瓜。问：这辆货车满载时能装多少个西瓜？实际能装多少个西瓜？