第3章 Java程序设计基础

本章学习目的与要求

要想应用Java语言进行程序设计，必须学习其基本语言知识。通过本章的学习，能够掌握Java语言的基本语法和基本编程结构和方法，学会简单的Java语言程序设计。这部分内容是程序设计的基本知识和技巧，应该牢固掌握。

本章主要内容

本章主要介绍以下内容：

- 标识符、关键字。
- Java语言的数据类型。
- Java语言的运算符和表达式。
- 程序控制流程(顺序、选择和循环)。
- 数组。

3.1 标识符、关键字及数据类型

任何程序设计语言，都是由语言规范和一系列开发库组成的。例如标准C，除了语言规范外，还有很多函数库；Java语言也不例外，也是由Java语言规范和Java开发类库组成的。

学习任何程序设计语言，都要从这两方面着手。关于常用的Java开发类，会在后面加以介绍，这里主要介绍Java基础语法。本章重点讲解Java标识符、关键字与数据类型，运算符与表达式以及程序设计使用的控制语句。

3.1.1 标识符

程序中所用到的每一个变量、方法、类和对象的名称都是标识符(Identifier)，都应该有相应的名称作为标识。把为程序中的实体——变量、常量、方法、类和对象等所起的名

称为标识符。简单地说,标识符就是一个名称。

Java 语言的标识符应遵循如下规则:

- 标识符由英文大小写字母 A～Z 或 a～z、下画线(_)、美元符号($)和数字 0～9 组成,还可以包括中文字符等其他 Unicode 8.0 字符集。并且这些字符序列开头必须是英文字母、下画线或美元符号。变量名不能使用空格、加号(+)、减号(-)、逗号(,)等符号。
- 标识符区分大小写。例如,sum、Sum 和 SUM 是 3 个不同的标识符。为避免引起混淆,程序中最好不要出现仅靠大小写区分的相似标识符。
- 不允许使用 Java 关键字(后面介绍)来命名。因为关键字是系统已经定义的具有特殊含义的标识符(Java 语言的关键字参看 3.1.2 节)。另外,还有一些名称虽然不是关键字,但是系统已经把它们留作特殊用途,例如系统使用过的函数名等,用户也不要使用它们作为标识符(如 main),以免引起混乱。
- 标识符没有长度限制,但不建议使用太长的标识符。
- 标识符命名原则应该以直观且易于拼读为宜,做到"见名知意",最好使用英文单词及其组合,这样便于记忆和阅读。
- Java 中标识符命名通常约定:常量用大写字母,变量用小写字母开始,类以大写字母开始。
- 标识符不能包含空格,只能包含 $,不能包含 #、&、@等其他特殊字符。

例如,下面是合法的标识符:

b, a3, _switchstudentName,Student_Name,_my_value, $address。

下面是非法的标识符:

ok?,"abc", a.b, 2teacher,a+b, room#,abstract(这是一个关键字)、_。

Java 程序中的命名规范

为了提高 Java 程序的可读性,Java 源程序有以下一些约定成俗的命名规定。

(1) 包名:包名是全小写的名词,中间可以由点分隔开。例如,java.awt.event。

(2) 类名:首字母大写,通常由多个单词合成一个类名,要求每个单词的首字母也要大写。例如,class HelloWorldApp。

(3) 接口名:命名规则与类名相同。例如,interface Collection。

(4) 方法名:往往由多个单词合成,第一个单词通常为动词,首字母小写,中间的每个单词的首字母都要大写。例如,balanceAccount,isButtonPressed。

(5) 变量名:全小写,一般为名词。例如,length。

(6) 常量名:基本数据类型的常量名为全大写,如果是由多个单词构成,可以用下画线隔开。例如,int YEAR, int WEEK_OF_MONTH;如果是对象类型的常量,则是大小写混合,由大写字母把单词隔开。

3.1.2 关键字

关键字又称保留字(Reserved Word),它们具有专门的意义和用途,不能当作一般的标识符使用。

Java 常用的保留字如表 3-1 所示，完整的关键字信息请参阅 Java 语言相关文档。

表 3-1　Java 常用的保留字

abstract	break	byte	boolean
catch	case	class	char
continue	default	double	do
else	extends	transient	final
finally	float	for	if
implements	import	instanceof	int
interface	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	void	volatile	while

说明：

- 保留字包括关键字和未使用的保留字。
- true、false 和 null 通常为小写，而不是像在 C++ 语言中那样大写。严格地讲，它们不是关键字，而是文字，但这种区别只是理论上的。
- Java 没有 sizeof 运算符；所有类型的长度和表示是固定的。
- 在 Java 编程语言中不使用 goto、const 和 enum 作为关键字，尽管它们在其他语言中常用，但不能用 goto、const 和 enum 作为变量名。它们是 Java 现在还未使用的保留字，此外还有 byValue、future、generic、inner、outer、operator、rest、var。

3.1.3　数据类型

Java 语言提供了丰富的数据类型，主要分为基本类型（又称原始类型）和引用类型，此外还有空类型，如图 3-1 所示。

1. 常量

数值不能再变化的变量称为常量，需要使用关键字 final 修饰。其定义格式为：

```
final Type varName=value [, varName [ =value] …];
```

例如：

```
final int MAXLEN=1000;
final double PI=3.1415926;
```

一般情况下，常量名用大写字母标识，如果由几个单词组成，那么不同单词之间使用_

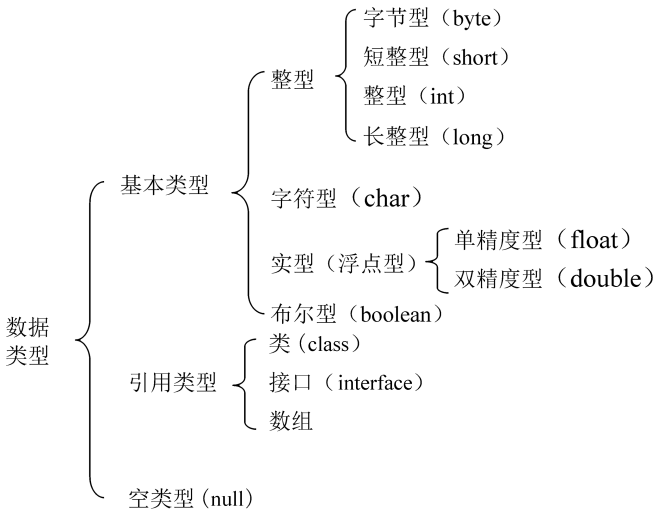

图 3-1　数据类型

连接。

例如，NAME，STUDENT_NAME 等。

2. 变量

在程序中使用变量之前必须对变量预先定义或赋值，一方面符合程序运行的要求，另一方面也使得程序的可读性更强。

变量是程序中的基本存储单元，其定义包括变量名、变量类型和作用域 3 个部分，其格式如下：

```
Type varName =value [, varName [ =value] …];
```

其中：

- Type 代表变量数据类型。Java 的数据类型有基本类型（又称原始类型）和引用类型，此外还有空类型。
- varName 代表变量名，也就是标识符。在一定的作用域内，变量名必须唯一。
- value 代表变量值。对变量的定义实际上分为两步：第一步是变量声明（declaration），如“int x;”。第二步是变量赋值（assignment），如“x＝10;”。第一次赋值也称为初始化。

有时也会合并这两步，例如：

```
int x =10;  int y =x;
```

- 变量的作用域：指变量在程序中的作用范围，变量分为全局变量和局部变量。全局变量是作用于全程序范围的变量（即在整个程序中均有效），它在函数体外声明。也就是说，在同一个页面的所有脚本都可以随时使用它。局部变量是在函数内定义的变量，它仅在该函数内的语句起作用。局部变量和全局变量可以同名，当在函数中定义的局部变量与全局变量同名时，函数内的变量名引用指的是该函数内的

局部变量，而不是全局变量。而局部变量定义以外的变量引用则指定为全局变量。

3. 基本类型

Java 定义了 8 个基本数据类型：字节型(byte)、短整型(short)、整型(int)、长整型(long)、字符型(char)、单精度型(float)、双精度型(double)和布尔型(boolean)。

这些基本类型可分为以下 4 组。

(1) 整型：包括字节型、短整型、整型和长整型，是有符号整数。

(2) 浮点型：包括单精度型和双精度型，代表有小数精度要求的数字。

(3) 字符型：包括字符型，代表字符集的符号，例如字母和数字。

(4) 布尔型：包括布尔型，是一种特殊的类型，表示真/假值。

可以按照定义使用它们，也可以构造数组或类的类型来使用它们。这些基本类型也是用来创建所有其他类型数据的基础。

基本数据类型代表单值，而不是复杂的对象。Java 是完全面向对象的，但基本数据类型不是，它们类似于其他大多数非面向对象语言的基本数据类型。这样做的原因是出于效率方面的考虑。在面向对象中引入基本数据类型不会对执行效率产生太多的影响。当然，Java 也提供了对基本数据类型的封装类型，后面会详细介绍。

所有基本类型所占的位数都是确定的，并不因操作系统的不同而不同(见表 3-2)。

表 3-2　基本数据类型的位数和范围

数 据 类 型	所 占 位 数	数的取值范围
char	16	0～65 535
byte	8	$-2^7 \sim 2^7-1$(−128～127)
short	16	$-2^{15} \sim 2^{15}-1$(−32 768～32 767)
int	32	$-2^{31} \sim 2^{31}-1$
long	64	$-2^{63} \sim 2^{63}-1$
float	32	$-3.4e^{038} \sim 3.4e^{038}$
double	64	$-1.7e^{308} \sim 1.7e^{308}$
boolean	8	true，false

下面依次讨论每种数据类型。

1) 整型

计算机中的数据都以二进制形式存储。在 Java 程序中，为了便于表示和使用，整型数据可以用以下几种形式表示，编译系统会自动将其转换为二进制形式存储。

(1) 整型常量。

整数可能是在典型的程序中最常用的类型。用来表达整数的方式有 3 种：十进制(人们平时使用的方式，基数是 10)、八进制(Octal，基数是 8)和十六进制(Hexadecimal，基数是 16)。

十进制整型数的数字由 0～9 表示，十进制无前缀。八进制是一种常用的表示形式，表示八进制数时，加前缀 0，八进制数的数字由 0～7 表示。表示十六进制数时，加前缀 0x 或 0X，十六进制数的数字由 0～9 和 a～f 或 A～F 组成。整型常量按进制分成 3 类，具体如

表 3-3 所示。

表 3-3 整型常量按进制分类

分　类	表示方法	说　明	举　例
十进制整数	一般表示形式	逢十进一	100 表示十进制数 100
八进制整数	以 0 开头	逢八进一	0100 表示八进制数 100
十六进制整数	以 0x 开头	逢十六进一	0x100 表示十六进制数 100

① 十进制整数形式：与数学上的整数表示相同。例如，123，－567，0。

② 八进制整数形式：在数码前加数字 0。例如，0123 是八进制数，表示十进制数 83；八进制数－011 表示十进制数－9，012 表示十进制数 10。

③ 十六进制整数：以 0x 或 0X 开头。例如，0x123 表示是十六进制数，表示十进制数 291；十六进制数－0X12 表示十进制数－18。

(2) 整型变量。

整型变量类型为 byte、short、int 或 long，其中 byte 在机器中占 8 位，short 占 16 位，int 占 32 位，long 占 64 位。如果没有明确指出一个整数值的类型，那么它默认为 int 类型。

整型变量的定义如下：

```
byte by1;                    //定义变量 by1 为 byte 型
short a;                     //定义变量 a 为 short 型
int x =123;                  //定义变量 x 为 int 型,且赋初值为 123
long y =123L;                //定义变量 y 为 long 型,且赋初值为 123
long z =123l;                //定义变量 z 为 long 型,且赋初值为 123
```

可以看到，一个整数值可以被赋给一个 long 变量。但是，定义一个 long 变量，需要告诉编译器变量的值是 long 型，可以通过在变量的后面加一个大写 L 或小写 l 来做到这一点。长整型必须以 L 作为结尾，如 9L、156L。

2) 实型

(1) 实型常量。

由于计算机中的实型数据是以浮点形式表示的，即小数点的位置是可以浮动的，因此，实型常量既可以称为实数，也可以称为浮点数。浮点数常量有 float(32 位)和 double(64 位)两种类型，分别称为单精度浮点数和双精度浮点数。表示浮点数时，要在后面加上 f(F)或者 d(D)。

Java 语言的实型常量有两种表示形式：标准记数法形式和科学记数法形式。

① 标准记数法形式：由数字和小数点组成。小数点前表示整数部分，小数点后表示小数部分，具体格式如下：

```
<整数部分>.<小数部分>
```

其中，小数点不可省略，＜整数部分＞和＜小数部分＞不可同时省略。

② 科学记数法形式：即指数形式，表示形式包含数值部分和指数部分。数值部分表示方法同十进制小数形式，指数部分是一个可正可负的整型数，这两部分用字母 e 或 E 连接

起来。具体格式如下：

<整数部分>.<小数部分>e<指数部分>

其中，e 左边部分可以是＜整数部分＞.＜小数部分＞，也可以只是＜整数部分＞，还可以是.＜小数部分＞；e 右边部分可以是正整数或负整数，但不能是浮点数，如表 3-4 所示。

表 3-4　实型常量表示方法

表 示 方 法	说　　明	举　　例
标准记数法形式	由数字和小数点组成	0.123，.123，123.，123.0
科学记数法形式	由尾数、字母 e 或 E 和指数组成	1.23e3 或 1.23E3

说明：10e2 表示 10 乘以 10 的 2 次幂，1.56e3 表示 1.56 乘以 10 的 3 次幂。

注意：在表示指数形式时，e 前必须有数字，e 后必须为整数。例如，10e2 也可写为 10e＋02。

提示：使用指数形式来表示很大或很小的数比较方便。

(2) 实型变量。

实型数据类型为 float(单精度型)或 double(双精度型)，其中 float 在机器中占 32 位，double 占 64 位。

实型变量的定义如下：

```
double x =0.123;          //定义变量 x 为 double 型,且赋初值为 0.123
float y =0.123F;          //定义变量 y 为 float 型,且赋初值为 0.123
```

实型变量的所占字节数及取值范围如表 3-5 所示。

表 3-5　实型变量类型的所占字节数及取值范围

类　　型	所占字节数	数的取值范围	举　　例
float	4	$-3.4e^{038}\sim3.4e^{038}$	float x1，x2；
double	8	$-1.7e^{308}\sim1.7e^{308}$	double y1，y2；

注意：Java 中的实型变量默认是双精度型(double)。为了指明一个单精度型变量，必须在数据后面加 F 或 f；若加 d 或 D，则为 double 类型。

例如，3.6d，2e3f，.6f，1.68d，7.012e＋23f，这些表示都是合法的。

3) 字符型数据

(1) 字符常量

字符常量是由英文字母、数字、转义序列、特殊字符等的字符所表示，它的值就是字符本身。字符常量是用单引号括起来的单个字符，Java 中的字符占用两字节。例如，'J'、'@'、'1'。另外，Java 中还有以反斜杠(\)开头的具有特殊含义的字符，称为转义字符。常用的转义字符如表 3-6 所示。

表 3-6 Java 中的转义字符

字符形式	说明
\n	换行
\t	横向跳格(即跳到下一个 tab 位置)
\b	退格
\r	回车
\f	走纸换页
\\	反斜杠字符(\)
\'	单引号(撇号)字符
\"	双引号字符
\0	空字符
\ddd	1～3 位八进制数所代表的字符(d 为 0～7)
\uxxxx	1～4 位十六进制数所代表的字符(x 为 0～f)

表中列出的转义字符,意思是将反斜杠(\)后面的字符转换成另外的意义。例如,\n 中的 n 不代表字母 n,而作为“换行”符。

表 3-6 中的最后两行是用 ASCII 码(八进制和十六进制)表示的一个字符。例如,\101 和\u41 都代表 ASCII 码(十进制)为 65 的字符 A。注意,\0 或\000 是代表 ASCII 码为 0 的控制字符,即“空操作”字符,它将用在字符串中。

注意字符常量与字符串常量的区别。字符串常量是由双引号括起来的字符序列,如"abc"、"a"、"Let's learn java!"。

注意:不要将字符常量与字符串常量混淆。'a'是字符常量,"a"是字符串常量,二者不同。后面可以使用字符串常量初始化一个 String 类的对象。关于字符串处理将在后面的章节中详细介绍。

(2) 字符变量

Java 语言的字符变量只有一种定义形式:

char 变量名;

字符变量用来存放字符常量,注意只能存放一个字符,在机器中占 16 位,如表 3-7 所示。

表 3-7 字符型变量类型所占字节数及取值范围

类型	所占字节数	说明	数据的取值范围	举例
Char	2	存放单个字符	0～65 535	char c1,c2='a';

例如,字符型变量的定义如下:

```
char  c='a';          //定义变量 c 为 char 型,且赋初值为'a'
char  c1='\n';        //定义变量 c1 为 char 型,且赋初值为'\n',转义字符,表示换行
```

说明：Java 的 char 与 C 或 C++ 中的 char 不同。在 C/C++ 中，char 的位宽是 8 位整数。但 Java 不同，Java 使用 Unicode 码代表字符。Unicode 定义的国际化字符集能表示迄今为止人类语言的所有字符集。它是几十个字符集的统一，如拉丁文、希腊语、阿拉伯语、古代斯拉夫语、希伯来语、日文片假名、匈牙利语等，因此它要求 16 位。这样，Java 中的 char 类型是 16 位，其范围是 0～65 535 且没有负数。人们熟知的标准字符集 ASCII 码的范围仍然是 0～127，扩展的 8 位字符集 ISO-Latin-1 的范围是 0～255。

【例 3-1】 字符型变量的使用。

```
package sample;
public class CharTest1 {
  public static void main(String args[]) {
    char ch1,ch2;
    ch1 =65;        //A 字符的 ASCII 代码值
    ch2 ='B';
    System.out.println("ch1 and ch2:" +ch1 +" " +ch2);
  }
}
```

程序运行结果：

```
ch1 and ch2: A B
```

说明：变量 ch1 被赋值 65，它是 ASCII 码(Unicode 码也一样)用来代表字母 A 的值。前面已提到，ASCII 字符集占用了 Unicode 字符集的前 127 个值，因此以前使用过的一些字符概念在 Java 中同样适用。

尽管 char 不是整数，但在许多情况下可以对它们进行运算操作。例如，可以将两个字符相加，或者对一个字符变量值进行增量操作。

【例 3-2】 字符型变量的运算。

```
package sample;
public class CharTest2 {
  public static void main(String args[]) {
    char ch1;
    ch1 ='A';
    ch1++;
    System.out.println("ch1 is: " +ch1);
  }
}
```

程序运行结果：

```
ch1 is: B
```

在该程序中，变量 ch1 首先被赋值为 A，然后递增 1。结果是 ch1 将代表字符 B，即在 ASCII(以及 Unicode)字符集中 A 的下一个字符。

前面介绍了几种基本类型，在一个程序中可能会出现多种类型，来看下面的一个实例。

【例 3-3】 几种基本类型变量的定义与使用。

```
package sample;
public class AssignTest {
  public static void main(String args []) {
    int x, y;                          //定义整型变量
    float z =3.414f;                   //定义单精度型变量并赋初值
    double w =3.1415;                  //定义双精度型变量并赋初值
    char c;                            //定义字符型变量
    String str;                        //定义字符串变量
    String str1 ="bye";                //定义字符串变量并赋初值
    c ='A';                            //赋初值
    str ="Hello! Welcome!";
    x =6;
    y =1000;
    System.out.println("int x=" +x);
    System.out.println("int y=" +y);
    System.out.println("float z=" +z);
    System.out.println("double w=" +w);
    System.out.println("char c=" +c);
    System.out.println("string str=" +str);
  }
```

程序运行结果：

```
int x=6
int y=1000
float z=3.414
double w=3.1415
char c=A
string str=Hello! Welcome!
```

4）布尔型

布尔型数据只有两个值 true 和 false，且它们不对应于任何整数值。

布尔型变量的定义如下：

```
boolean b1=true;     //声明变量 b1 为 boolean 类型,它被赋予初值 true
```

同样：

```
boolean b2=false;    //声明变量 b2 为 boolean 类型,它被赋予值 false
```

【例 3-4】 布尔型数据的使用。

```
package sample;
public class BooleanTest {
  public static void main(String args[]) {
    boolean a;
    a =true;
    System.out.println("It is true.");
    a =false;
    System.out.println("It is false.");
  }
}
```

程序运行结果：

```
It is true.
It is false.
```

4. 基本数据类型之间的转换

整型、实型、字符型数据可以混合运算。在运算中，不同类型的数据先转换为同一类型，然后进行运算，这时会遇到类型转换(casting)这个概念。

如果参与运算的两种类型是兼容的，那么 Java 将自动地进行转换。例如，把 int 类型的值赋给 long 类型的变量总是可行的。然而，不是所有的类型都是兼容的。例如，没有将 double 类型转换为 byte 类型的定义。幸运的是，在不兼容的类型之间进行转换仍然是可能的，这就必须使用一个强制类型转换，它能完成两个不兼容的类型之间的显式变换。下面介绍自动类型转换和强制类型转换。

1) 自动类型转换

自动类型转换也称为默认或隐式类型转换(implicit casting)，如果下列两个条件都能满足，那么将一种类型的数据赋给另外一种类型变量时，将执行自动类型转换。

(1) 这两种类型是兼容的。

(2) 目的类型数的范围比来源类型的大。

对于基本类型的范围(低→高)：

byte，short，char→int→long→float→double

当以上两个条件都满足时，拓宽转换(widening conversion)发生。例如，int 类型的范围比所有 byte 类型的合法范围大，因此不要求显式强制类型转换语句。

对于拓宽转换，数字类型(包括整数和浮点数类型)都是彼此兼容的，但是数字类型和字符类型或布尔类型是不兼容的。字符类型和布尔类型也是互相不兼容的。

当不同类型的数据在运算符的作用下构成表达式时要进行类型转换，即把不同的类型先转换成统一的类型，然后再进行运算。

通常数据之间的转换遵循的原则是“类型提升”，即如果一个运算符有两个不同类型的操作数，那么在进行运算之前，先将较低类型(所占内存空间字节数少)的数据提升为较高的类

型(所占内存空间字节数少),从而使两者的类型一致(但数值不变),然后再进行运算,其结果是较高类型的数据。类型的高低是根据其数据所占用的空间大小来判定的,占用空间越多,类型越高;反之,占用空间越少,则类型越低,如图 3-2 所示。

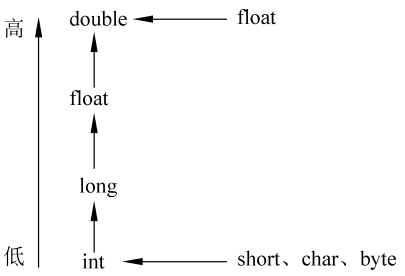

图 3-2 标准类型数据转换规则

当较高类型的数据转换成较低类型的数据时,称为降格。Java 语言中类型提升时,一般其值保持不变;但类型降格时就可能失去一部分信息。

2) 强制类型转换

强制类型转换也称为显式类型转换(explicit casting)。尽管自动类型转换对人们编程很有帮助,但并不能满足所有的编程需要。例如,如果需要将 int 类型的值赋给一个 byte 类型的变量,这样的转换不会自动进行,因为 byte 类型的变化范围比 int 类型的要小。这种转换有时称为"缩小转换",需要将源数据类型的值变小才能适合目标数据类型。

为了完成两种不兼容类型之间的转换,就必须进行强制类型转换。强制类型转换也称为显式类型转换(explicit casting),它的通用格式如下:

```
(target-type) value
```

其中,目标类型(target-type)指定了要将指定值转换成的类型。例如:

```
float x=5.65;          //x 为 float 类型
int y;                 //y 为 int 类型
y=(int)x+10;           //先将 x 的值转换为 int 型,在与 10 相加结果赋值给 y
```

上面是把实型 x 强制转换成整型,要把 x 前面的 int 用括号括起来。强制类型转换的一般形式为(类型名)(表达式),表达式应该用括号括起来。x 的类型仍为 float 型,所以值仍等于 5.65。

注意:强制类型是暂时的、一次性的,不会改变其后边表达式的类型。

下面的实例将 int 类型强制转换成 byte 类型。如果整数的值超出了 byte 类型的取值范围,它的值将会因为对 byte 类型值域取模(整数除以 byte 得到的余数)而减小。

```
int a;
byte b;
b =(byte) a;
```

当把浮点值赋给整数类型时,一种不同的类型转换发生了:截断(truncation)。由于整数没有小数部分,这样,当把浮点值赋给整数类型时,它的小数部分会被舍去。例如,如果将值 3.45 赋给一个整数,其结果值只是 3,0.45 被丢弃了。当然,如果浮点值太大而不能适合目标整数类型,那么它的值将会因为对目标类型值域取模而减小。

【例 3-5】 强制类型数据的转换。

```
package sample;
public class ExplicitDataCastingTest {
```

```
  public static void main(String args[]) {
    byte b;
    int i =257;
    double d =123.456;
    System.out.println("\nCasting of int to byte.");
    b =(byte) i;
    System.out.println("b is:" +b);
    System.out.println("\nCasting of double to int.");
    i =(int) d;
    System.out.println("i is:" +i);
  }
}
```

程序运行结果：

```
Casting of int to byte.
b is: 1
Casting of double to int.
i is: 123
```

我们看看每一个类型转换。当值 257 被强制转换为 byte 类型变量时，其结果是 257 除以 256(256 是 byte 类型的变化范围)的余数 1。当把变量 d 转换为 int 类型时，它的小数部分被舍弃了。

【例 3-6】 不同类型变量的转换。

```
package sample;
public class CastingTest {
      public void implictCasting() {
            byte a =0x60;
            int ia =a;
            char b ='a';
            int c =b;
            long d =c;
            long e =1000000000L;
            float f =e;
            double g =f;
            String s ="hello";
            Object o =s;
      }
      public void explicitCasting() {
            long l =1000000L;
            int i =l;                  //错误!应该为(int)l;
            double d =12345.678;
            float f =d;                //错误!应该为(float)d;
```

```
        Object o =new String("Hello");
        String str =o;          //错误!应该为(String)o;
    }
}
```

该程序有编译错误,已经在代码中标出。

3.2 运算符与表达式

Java 语言提供了丰富的运算符和表达式,这为编程带来了方便和灵活,主要包括算术运算符、关系运算符、逻辑运算符、位运算符、赋值运算符、条件运算符和其他运算符等。

运算符可按其操作数个数的多少分为 3 类:单目运算符(一个操作数)、双目运算符(两个操作数)和三目运算符(3 个操作数)。

由这些运算符和操作数按一定的语法形式连接起来的式子称为**表达式**。一个常量或一个变量名字是最简单的表达式,其值即该常量或变量的值。表达式的值还可以用作其他运算的操作数,嵌套在一起形成更复杂的表达式。

3.2.1 算术运算符及其表达式

算术运算符包括+(加)、-(减)、*(乘)、/(除)、%(模)、++(递增)、--(递减)等。算术运算符的运算数必须是数字类型。算术运算符不能用在布尔类型上,但是可以用在 char 类型上,因为在 Java 语言中,char 类型实质上是 int 类型的一个子集。

常见的算术运算符有双目算术运算符以及自增和自减运算符,表 3-8 为双目算术运算符。

表 3-8 双目算术运算符

运算符	名称	运算规则	运算对象	运算结果	举例	结果
*	乘	乘法	整型或实型	整型或实型	2.5 * 3.0	7.5
/	除	除法			2.5/5	0.5
%	模(求余)	整数取余	整型	整型	10%3	1
+	加	加法	整型或实型	整型或实型	2.5+1.2	3.7
-	减	减法			5-4.6	0.4

基本算术运算符加、减、乘、除可以对所有的数字类型数据进行操作。加、减运算符也用作表示单个操作数的正、负号。特别要注意的是,对整数进行除法(/)运算时,所有的余数都要被舍去,而对于浮点数除法则可以保留余数。

【例 3-7】 算术运算符的使用。

```
package sample;
public class MathTest {
  public static void main(String args[]) {
    int a =3+5;
```

```
    int b =a * 2;
    int c =b / 10;
    double  d=b/10;
    System.out.println("a =" +a);
    System.out.println("b =" +b);
    System.out.println("c =" +c);
    System.out.println("d =" +d);
  }
}
```

程序运行结果：

```
a =8
b =16
c =1
d =1.6
```

模运算符%可以获取整数除法的余数，它同样适用于浮点类型数据（这与 C/C++ 不同，C/C++ 语言要求%两侧均为整型数据）。

【例 3-8】 模运算符的用法。

```
package sample;
public class ModTest {
  public static void main(String args[]) {
    int x =23;
    double y =23.56;
    System.out.println("x mod 5 =" +x %5);
    System.out.println("y mod 5 =" +y %5);
  }
}
```

程序运行结果：

```
x mod 5 =3
y mod 5 =3.56
```

说明：Java 对加运算进行了扩展，能够完成字符串的连接，例如"Java"＋"Applet"结果为字符串"Java Applet"。

3.2.2 自增和自减运算符

＋＋和－－是 Java 的自增和自减运算符。

作用：自增运算符对其运算数加 1，自减运算符对其运算数减 1。

两种运算类型说明如下。

① 前置运算：＋＋i，－－i。

表示先使变量的值增 1 或减 1，再使用该变量。

② 后置运算：i＋＋，i－－。

表示先使用该变量参加运算，再将该变量的值增 1 或减 1。

自增和自减运算符见表 3-9。

表 3-9　自增和自减运算符

运　算　符	名　　称	运 算 规 则	对象个数	运算结果	举　　例	结　　果
＋＋	增 1(前缀)	先增值后引用	单目	同运算对象的数据类型	a＝2；x＝＋＋a；	x＝3
＋＋	增 1(后缀)	先引用后增值			a＝2；x＝a＋＋；	x＝2
－－	减 1(前缀)	先减值后引用			a＝2；x＝－－a；	x＝1
－－	减 1(后缀)	先引用后减值			a＝2；x＝a－－；	x＝2

注意：自增和自减运算符中的 4 个符号同级，且高于双目算术运算符。自增和自减运算符只作用于变量，而不能作用于常量或表达式上。

下面将对它们进行详细讨论。先来看递增和递减运算符的操作。语句

```
x++;
```

与下面语句相同：

```
x =x +1;
```

同样，语句

```
x--;
```

与下面语句相同：

```
x =x -1;
```

在上面例子中，自增或自减运算符采用前缀(prefix)或后缀(postfix)格式都是相同的。但是，当自增或自减运算符作为一个较大表达式的一部分时，就会有重要区别。如果自增或自减运算符放在其运算数前面，Java 就会在获得该运算数的值之前执行相应的操作，并将其用于表达式的其他部分。如果运算符放在其运算数后面，Java 就会先获得该操作数的值再执行递增或递减运算。例如：

```
x =10;
y =++x;
```

在上面例子中，y 将被赋值为 11，因为在将 x 的值赋给 y 以前，要先执行递增运算。这样，语句“y ＝＋＋x;”和下面两句是等价的：

```
x =x +1;
y =x;
```

但是,当写成如下这样时:

```
x =10;
y =x++;
```

在执行递增运算以前,先将 x 的值赋给了 y,因此 y 的值还是 10。当然,在这两个例子中,x 都被赋值为 11。在本例中,语句"y =x++;"与下面两个语句等价:

```
y =x;
x =x +1;
```

【例 3-9】 自增运算符的使用。

```
package sample;
public class IncTest {
  public static void main(String args[]) {
    int a =1;
    int b =2;
    int c;
    int d;
    c =++b;
    d =a++;
    c++;
    System.out.println("a =" +a);
    System.out.println("b =" +b);
    System.out.println("c =" +c);
    System.out.println("d =" +d);
  }
}
```

程序运行结果:

```
a =2
b =3
c =4
d =1
```

说明:单独的自增和自减运算,前置和后置等价。例如,"a++;"和"++a;"等价,都相当于"a=a+1;"。

相关知识 自增运算符(++)和自减运算符(--)只能用于变量,不能用于常量或表达式,例如 5++或(a+b)++都是不合法的。它们的结合方向是"自右至左"。它们常用于后面章节的循环语句中,使循环变量自动增加 1;也用于指针变量,使指针指向下一个地址。

3.2.3 关系运算符及其表达式

所谓"关系运算"(relational operator)实际上就是"比较运算"。将两个值进行比较,判

断其比较的结果是否符合给定的条件。关系运算符包括>、<、>=、<=、==、!=等。关系运算符决定值和值之间的关系。例如,决定相等、不相等及排列次序等。关系运算符如表 3-10 所示。

表 3-10 关系运算符

运 算 符	名 称	运算规则	运算结果	举 例	表达式值
<	小于	满足则为真,结果为 1;不满足则为假,结果为 0	逻辑值(整型)	a=1;b=2;a<b;	true
<=	小于或等于			a=1;b=2;a<=b;	true
>	大于			a=1;b=2;a>b;	false
>=	大于或等于			a=1;b=2;a>=b;	false
==	等于			a=1;b=2;a==b;	false
!=	不等于			a=1;b=2;a!=b;	true

这些关系运算符产生的结果是布尔类型值。关系运算符常常用在 if 控制语句和各种循环语句的表达式中。

Java 中的任何类型,包括整型、浮点型、字符型及布尔型,都可用==来比较是否相等,用!=来比较是否不等。

注意:Java 比较是否相等的运算符是用两个等号,而不是一个符号(注意,一个等号是赋值运算符)。只有数字类型可以使用排序运算符进行比较。也就是说,只有整数、浮点数和字符运算数可以用来比较哪个大或哪个小。等于运算符是==,即为代数式中的两个等号。通常容易在使用等于运算符时写成一个等号,使程序出现意想不到的错误。

使用关系运算符构成的关系表达式的值是逻辑值。要么为“真”,要么为“假”。

例如,下面的程序段对变量 c 的赋值是有效的。

```
int a =5;
int b =3;
boolean c =a <b;
```

在本例中,a<b(其结果是 false)的结果存储在变量 c 中。

【例 3-10】 关系运算符的计算。

```
/**
 * Java 中关系运算符的使用
 * /
package sample;
public class RelationOpTest{
  public static void main(String args[]){
    int a=1;
    int b=2;
    int c=3;
    boolean d=a<b;          //true
    boolean e=a>b;          //false
```

```
        boolean f=b==c;       //false
        boolean g=b!=c;       //true
        boolean h=b>=c;       //false
        boolean i=b<=c;       //true
        boolean j=a==b;       //false
        System.out.println("d="+d);
        System.out.println("e="+e);
        System.out.println("f="+f);
        System.out.println("g="+g);
        System.out.println("h="+h);
        System.out.println("i="+i);
        System.out.println("j="+j);
        }
    }
```

程序运行结果：

```
d=true
e=false
f=false
g=true
h=false
i=true
j=false
```

3.2.4 逻辑运算符

逻辑运算符用来进行逻辑运算，逻辑运算也称为布尔运算。用逻辑运算符连接操作数组成的表达式称为逻辑表达式。逻辑表达式的值或称逻辑运算的结果也只有真和假两个值。当逻辑运算的结果为真时，用 1 作为表达式的值；当逻辑运算的结果为假时，用 0 作为表达式的值。当判断一个逻辑表达式的结果时，则是根据逻辑表达式的值为非 0 时表示真，为 0 时表示假。逻辑运算符如表 3-11 所示。

表 3-11　逻辑运算符

运算符	名称	运算规则	运算结果	结合方向	举　例	表达式值
!	非	逻辑非	逻辑值（整型）	从右向左	a=1;!a;	false
&&	与	逻辑与		从左向右	a=1;b=0;a&&b;	false
||	或	逻辑或			a=1;b=0;a||b;	true

从表 3-11 可以看出，逻辑运算符包括!、&&、||。Java 提供了逻辑非(!)、逻辑与(&&)和逻辑或(||)3 个运算符。

逻辑非代表取反，如果当前运算数为真，取反后的值为假；反之，如果当前运算数为假，取反后的值为真。

在逻辑或运算中，如果第一个运算数为真，则不管第二个运算数是真还是假，其运算结果都为真。

同样，在逻辑与运算中，如果第一个运算数为假，则不管第二个运算数是真还是假，其运算结果都为假。

因此，如果采用||和 && 形式，那么一个运算数就能决定表达式的值，只有在需要时才对第二个运算数求值。当右边的运算数取决于左边的运算数是真或者假时，这点是很有用的。例如，下面的程序语句说明了逻辑运算符的优点，用它可以防止被 0 除的错误。

```
if (x != 0 && num / x > 12)
```

既然用了逻辑与运算符，就不会有当 x 为 0 时产生的运行时异常。

【例 3-11】 Java 逻辑运算符的使用。

```
package sample;
public class LogicTest{
  public static void main(String[] args){
    int i = 2;
    int j = 3;
    System.out.println("i = " + i);
    System.out.println("j = " + j);
    System.out.println("i != j is " + (i != j));
    System.out.println("(i<10&&j<10) is "+((i < 10) && (j < 10)));
    System.out.println("((i+j)>10) is "+((i+j)>10));
    System.out.println("(!(i==j)) is "+(!(i==j)));
  }
}
```

程序运行结果：

```
i = 2
j = 3
(i < 10 && j < 10) is true
((i+j)>10) is false
(!(i==j)) is  true
```

注意：除了逻辑非外，逻辑运算符的优先级低于关系运算符。逻辑非这个符号比较特殊，它的优先级高于算术运算符。逻辑运算符的优先级为!→&&→||。

3.2.5 位运算符

位运算符包括>>、<<、>>>、&、|、^、～等。Java 定义的位运算符（bitwise operator）直接对整数类型的位进行操作，这些整数类型包括 long、int、short、char 和 byte。表 3-12 列出了位运算符及其含义。