第3章 Java使用初步

3.1 Java 应用程序的组成元素

Java 应用程序一般由三部分组成：注释、import 语句和类声明。本节结合实例对这三部分进行简要介绍。

【例 3-1】 设想我们需要编写一个 Java 程序计算任意两个整数之和。在程序运行时首先提示用户输入两个数的值，程序计算出结果，并在屏幕上显示出来。

下面是计算两数之和的程序代码[1]：

```
/*
File: Addition.java
功能：计算两个整数之和
*/                                                    ← 多行注释
import java.util.Scanner;                             ← import 语句
//以下为类声明部分
public class Addition {
    //开始运行程序的主方法                             ← 单行注释
    public static void main(String args[]){
        //创建 Scanner 对象 input
        Scanner input = new Scanner(System.in);
        int number1;                          //第一个加数      ← 单行注释
        int number2;                          //第二个加数
        int sum;                              //sum 存储累加和
        System.out.print("输入第一个加数：");
        number1 = input.nextInt();

        System.out.print("输入第二个加数：");
        number2 = input.nextInt();

        sum = number1 + number2;              //进行相加运算

        //输出运算结果
        System.out.print("累加结果：");
        System.out.println(sum);
    }
} //类声明结束
```

其中，第一部分为注释，第二部分为 import 语句，第三部分为类声明，类声明中也包含注释。

1. 注释

程序中除了计算机要执行的指令，还包含注释，我们在注释中陈述程序的目的，解释代码的含义，并提供任何其他描述来帮助程序员理解程序。

Java 程序的注释分为 3 种，分别是多行注释、单行注释和文档注释。

顾名思义，多行注释可以占多行，以标记“/*”开始，以另一个标记“*/”结束，中间的部分就是注释。开始和结束注释标记是成对匹配的，即每一个开始标记必须有一个对应的结束标记。

单行注释的标记是双斜线“//”。位于双斜线标记和行末的任何文本都是注释。单行注释放在被注释语句的上面或右侧。

文档注释是一种专门的注释，可以出现在类声明之前，也可以出现在其他程序元素之前。文档注释由标记“/**”开始，由标记“*/”结束，可以占一行，也可以占多行，具体使用见 4.7 节。

注释只供程序员使用，而被计算机所忽略。在程序中不写注释不影响程序的运行，但会影响程序的阅读和理解。只编写正确运行的程序是不够的，还需要编写程序文档，给程序加注释是程序文档的重要部分。程序文档的其他部分包括程序图、程序员的工作日志、设计文档和用户手册。如果能够一次编写完程序，并永远使用，而不需要修改，那么编写不带注释的程序还是可以忍受的。然而，在现实世界中，从未做过任何变更的程序几乎是不存在的。例如，你可能决定增加新的特性和功能，或者要修改用户与程序交互的方式。即使不改进程序，当检测到程序中的某些错误时，还是不得不修改程序。另外，对于商用程序，大多数情况下，修改程序的人通常并不是开发程序的人。当程序员需要修改他自己或其他人的程序时，首先必须理解程序，而程序文档对于理解程序起到了绝对必要的辅助作用。

2. import 语句

只要可能，就通过使用预先定义的类开发面向对象的程序，包括系统定义的类和程序员定义的类。当没有合适的预先定义的类可以使用时，就需要定义自己的类。在 Java 中，将类组织到包(Packages)中，Java 系统具有很多包。还可以将我们自己的类逻辑组合到包中，这样就可以被其他程序方便地复用。

为了使用包中的类，通过使用下面的格式将类引入到我们的程序中：

```
import <package name>.<class name>
```

包可以包括子包，形成包的层次。在引用嵌入包中的类时，使用多个点。例如，编写语句

```
import java.util.Scanner;
```

来引用 java.util 包中的 Scanner 类。即 util 包在 java 包中。带有类所属的所有包名的点符

号表示法称为类的全限定名。使用类的全限定名常常很麻烦,特别是当我们不得不在程序中多次引用相同的类时。可以使用 import 语句来避免这个问题。

在 Addition.java 程序中,由于使用了 import 语句将 java.util.Scanner 类引入到程序中,在使用 Scanner 类时只写类名即可,其所属的包可以省略。如果没有使用 import 语句,则创建 Scanner 对象 input 的语句需要改为:

```
Scanner input = new java.util.Scanner(System.in);
```

如果想从同一个包中引入多个类,则可以使用星号将包中的所有类都引入,而不需要分别为每一个类使用一个 import 语句。使用星号引入类的语句如下:

```
import <package name>.*;
```

例如,语句

```
import java.util.*;
```

则从 java.util 包中引入了所有类。

3. 类声明

Java 程序由一个类或多个类组成。有些是预先定义的类,而有些是我们自己定义的类。在实例程序 Addition.java 中有两个类:Scanner 和 Addition。Scanner 是一个标准类,而 Addition 是我们自己定义的类。为了定义一个新类,必须在程序中进行类声明。类声明的语法如下:

```
class <class name> {
  <class member declarations>
}
```

其中,<class name>是类名;<class member declarations>是类成员声明的一个序列;单词 class 是保留字,用于标记类声明的开始。类成员或者是数据成员,或者是方法成员。可以使用不是保留字的任何合法标识符来命名类。

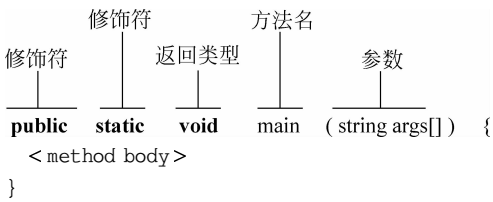
程序中的一个类必须被指定为主类。实例程序的主类是 Addition。如果将一个类指定为主类,则必须定义一个称为 main 的方法。因为当 Java 程序运行时,首先运行主类的 main 方法。要定义一个方法,就必须在类中声明它。

方法声明的语法如下:

```
<modifiers><return type><method name>(<parameters>){
   <method body>
}
```

其中,<modifiers>是一系列修饰符,指定不同种类的方法;<return type>是方法返回值的类型;<method name>是方法名;<parameters>是参数表;<method body>是指令序列。

主方法的声明如下：



这里不解释修饰符、返回类型和参数的含义，在本书的进展中将逐渐给出详细解释。

3.2 基本数据类型与表达式

3.2.1 基本数据类型

在 Java 中，数字、字符和布尔值都不属于对象，而是一种基本类型值。数字类型包括 6 种，分别为字节型(byte)、短整型(short)、整型(int)、长整型(long)、单精度浮点型(float)和双精度浮点型(double)。后两种类型的数据统称为实数。表 3-1 列出了 Java 语言中的 8 种基本类型。

表 3-1 Java 语言的基本类型

类 型	长 度	范 围
byte	1 字节	−128～127
short	2 字节	−32 768～32 767
int	4 字节	−2 147 483 648～2 147 483 647
long	8 字节	−9 223 372 036 854 775 808L～9 223 372 036 854 775 807L
float	4 字节	−3.402 823 47E + 38F～3.402 823 47E+ 38F
double	8 字节	−1.797 693 134 862 315 70E + 308～1.797 693 134 862 315 70E + 308
char	2 字节	'\u0000'～'\uFFFF'
boolean	1 位	true，false

在表示 long 常量时使用一个后缀 L；在表示 float 常量时使用一个后缀 F，例如 3.14159F。

字符类型数据(char)用于存储单个字符，字符以代码形式存储。字符常量包含在一对单引号中，例如'a'。在 ASCII 字符集中，字符 65 对应于字母 A，49 对应于数字 1。Java 字符数据类型是 16 位的，最小值为 0，最大值为 65 535，放置 Unicode 符号。Unicode 是 ASCII 字符集的扩展集，用于处理多种语言。

Java 也提供转义字符，以反斜杠(\)开头，将其后的字符转变成另外的含义。表 3-2 列出了 Java 中常用的转义字符。

表 3-2 Java 中常用的转义字符

转义字符	含 义	转义字符	含 义
\'	单引号	\f	送页符
\"	双引号	\t	水平制表符
\\	反斜杠	\b	退格
\r	回车	\un1 n2 n3 n4	Unicode 码
\n	换行符		

3.2.2 变量与常量

变量名和常量名必须是 Java 语言中的合法标识符,因此这里先介绍 Java 语言中的标识符。

1. 标识符(Identifier)

标识符是一个名称,其第一个字符必须是下列字符中的一个:大写字母(A～Z)、小写字母(a～z)、下划线(_)或者美元符号($),后面的字符可以是上述字母或者数字(0～9)中的一个。

在标识符中有一部分被系统定义,用户不能使用,被称为保留字或关键字。关键字列表如下[1]:

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

$var1、_var2、aInt、student_Number 都是合法标识符。而下列标识符是不合法的:

2student (数字不能作为标识符第一个字符)

try (关键字不能作为标识符)

var# (含有非法字符#)

2. 变量(Variables)

假设我们想计算两个整数的和与差,这两个整数分别是 x 和 y,则可以使用下面的语句声明整型变量 x 和 y:

```
int  x , y ;
```

一旦进行了声明,就为 x 和 y 分配内存单元来存储数据值,这些内存单元就称为变量

(variable),并且 x 和 y 就是我们给这些内存单元命名的名称。可以使用任何合法有效的标识符作为变量名。当完成上述的变量声明以后,只能给 x 和 y 赋整型数据,不能赋实型数据。

声明变量的一般语法格式为:

```
<data type>  <variables>;
```

<variables>是用逗号隔开的标识符序列。程序中用到的每个变量都必须声明。

可以根据需要以多个声明语句来定义变量。下面是声明不同数据类型的例子:

```
int       i, j, k;
int       x, y;
float     f1, f2;
long      bigInteger;
double    bigNumber;
```

也可以在声明变量时进行初始化。例如,可以在下面的声明语句中将整型变量 x 和 y 分别初始化为 10 和 34:

```
int x = 10, y = 34;
```

3. 常量(Final Variables)

变量的值是可以变化的,在程序运行的过程中,在不同的时刻可以将不同的值赋给同一个变量。但是,在有些情况下,我们希望某些值保持不变,这时就可以使用常量(Constant)。声明常量的方式与变量相似,但要增加保留字 final。在声明常量时,必须给它赋值。下面是常量声明的例子:

```
final int       CAPACITY = 35;                          //容量限制
final double    RATE_OF_CALL = 0.4;                     //活期存款利率
```

常量分为 3 种:命名常量(Named Constant)、符号常量(Symbolic Constant)和字面常量(Literal Constant)。CAPACITY 及 RATE_OF_CALL 称为命名常量和符号常量。字面常量是使用实际的值来描述的常量。上面声明语句中的 35 和 0.4 都是字面常量。

在程序中使用符号常量能够提高程序的可读性,并使程序易于修改。例如,如果活期存款利率改变了,只需修改 RATE_OF_CALL 的声明语句即可,程序中用到活期存款利率的地方都不需要修改。如果没有声明及使用符号常量 RATE_OF_CALL,在程序中使用活期存款利率的地方都使用了字面常量 0.4,修改起来就比较麻烦;一方面,可能有很多地方都用到了;另一方面,程序中有 0.4 的地方代表的含义不一定是活期存款利率。这样,在修改程序中可能发生漏改或错改的隐患。

3.2.3 表达式

表达式由运算符和运算数组成。运算数可以是常量、变量、方法调用或者是另一个用括号括起来的表达式。

运算符也称为操作符,指明对操作数所进行的运算。按照功能,可以把运算符分为算术运算符、赋值运算符、关系运算符、逻辑运算符、位运算符和条件运算符。

1. 算术运算符

表 3-3 列出了 Java 中所用到的算术运算符[3]。从表 3-3 中可以看到,当两个整数相除时,结果是整数商,即小数部分都被截掉了。两个整数之间的除法称为整除。当两个运算数中任意一个或两个都是实数时,相除的结果是实数。

模运算的结果是除法的余数。尽管可以对实数进行模运算,但是大多数情况下,模运算只涉及整数。

表 3-3 算术运算符

运　　算	Java 运算符	例　　子	表达式的结果 (x=15, y=6, z=2.5)
加	+	x + y	21
减	-	x - y	9
乘	*	x * y	90
		y * z	15.0
除	/	x / y	2
		x / z	6.0
模(求余)	%	x % y	3
		y % x	6
递增	++	++x	16
		x++	15
递减	--	--y	5
		y--	6
求正数	+	+z	2.5
求负数	-	-z	-2.5

加、减、乘、除及模运算都是二元运算,递增、递减、求正数、求负数都是一元运算,所不同的是,递增和递减运算修改操作数本身的值,++x 和 x++都使 x 的值递增 1,--y 和 y--都使 y 的值递减 1,而其他运算不改变操作数本身的值。

当两个以上的运算符出现在表达式中时,按照先乘除后加减的规则确定它们的计算顺序。例如,在表达式 x +3 * y 中,首先进行乘法运算,然后进行加法运算。表达式中可以用括号()改变运算次序。另外,适当地使用括号可以使表达式结构清晰,增强程序的可读性。图 3-1 给出了算术运算符及括号的优先规则。

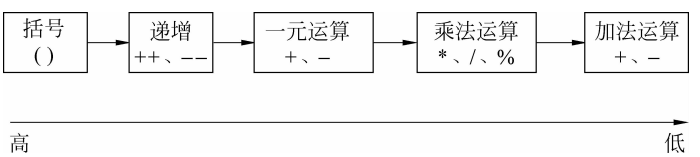


图 3-1 算术运算符及括号的优先规则

2. 赋值运算符

赋值运算符(=)的左侧为变量,右侧为表达式。赋值运算符的作用是将右侧表达式的结果赋值给左侧的变量,例如:

```
int    a,b;
a = 8;
b = 3 + 5 * a;
```

赋值运算之后,a 的值为 8,b 的值为 43。

在编写程序时,我们经常以一个确定的值对变量值递增或递减。例如,要将 sum 的值增加 10,可以写成:

```
sum = sum + 10;
```

可以用复合赋值运算符来改写这个语句,从而不用在赋值号的左右两边重复同一个变量:

```
sum + = 10;
```

Java 中常用的复合赋值运算符有+=、-=、*=、/=、%=。

如果想将一个值赋给多个变量,可以串联使用赋值号,例如:

```
a = b = c = 3;
```

与下面 3 个语句等价:

```
c = 3;
b = 3;
a = 3;
```

赋值运算符是从右向左计算的。赋值运算符的优先级低于任何其他运算符。

赋值运算符也可以出现在表达式中,例如:

```
a = 3 + (b = 10);          //表达式的值是 13,a 的值是 13,b 的值是 10
a = (b = 14)/(c = 7);      //表达式的值是 2,a 的值是 2,b 的值是 14,c 的值是 7
```

3. 关系运算符

关系运算符分为算术比较运算符和类型比较运算符。关系表达式的结果只能是布尔型。算术比较运算符如表 3-4 所示。

表 3-4 算术比较运算符

运　　算	Java 运算符	例　　子	结果(x=15, y=6)
大于	>	x > y	true
大于等于	>=	x >= y	true
小于	<	x < y	false
小于等于	<=	x <= y	false
等于	==	x == y	false
不等于	!=	x ! = y	true

算术比较运算符的优先级低于算术运算符。在算术比较运算符中,运算符<、>、<=、>=的优先级高于运算符==、!=。

类型比较运算符只有一个:instanceof,用法举例如下:

```
e instanceof Point        //Point 是一个类
```

如果 e 是 Point 类的一个实例,结果为 true,否则结果为 false。

4. 逻辑运算符

逻辑运算符有 3 个:"与"运算符(&&)、"或"运算符(||)、"非"运算符(!),具体如表 3-5 所示。

表 3-5 逻辑运算符

运算	Java 运算符	说明	例子	结果(x=15, y=6)
与运算	&&	两个操作数的值都为 true,运算结果为 true,否则结果为 false	(x > 10) && (y < 10)	true
			(x > 10) && (y < 5)	false
或运算	||	两个操作数的值都为 false,运算结果为 false,否则结果为 true	(x > 10) || (y < 5)	true
			(x < 10) || (y < 5)	false
非运算	!	操作数的值为 false,运算结果为 true;操作数的值为 true,运算结果为 false	!(x > 10)	false
			!(y < 5)	true

非运算(!)为一元运算符,其优先级与++、--相同,与运算(&&)的优先级高于或运算(||),两者的优先级都低于算术比较运算符。在实际应用中,适当使用括弧(如表 3-5 中的例子)会使运算的优先级更清晰。

5. 条件运算符(?:)

条件运算符为三目运算符,语法形式如下:

```
expression ? statement1: statement2
```

首先计算表达式 expression,它的结果应该为一个布尔值,如果该值为 true,则执行语句 statement1,否则执行语句 statement2。语句 statement1 和 statement2 需要返回相同的数据类型。下面是条件运算符的应用举例:

```
boolean    isStudent;
int    salary;
isStudent = true;
salary = (isStudent?500:1000);
```

3.2.4 类型转换

在很多时候都需要进行类型转换。当对不同类型的数据进行混合运算时,在运算之前,

系统会进行类型转换；在进行赋值运算时，当表达式运算结果的类型和被赋值的变量类型不一致时，则需要将表达式结果的类型转换成变量所对应的类型。

类型转换或者塑型(Typecasting)是将一种数据类型的值转换成另一种数据类型的值的过程。Java 中的塑型有隐式(Implicit)和显式(Explicit)两种。有些类型转换可以自动进行，通常将这种类型转换称为隐式类型转换。图 3-2 箭头所示方向表示可以自动进行隐式转换。按照这个方向，整型之间或实型之间的转换没有任何损失，例如，将 short 转换为 int 或将 float 转换为 double。但从 int 转换到 float 或者 double 将损失精度。

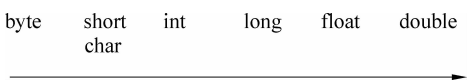


图 3-2 隐式类型转换

如果需要按照图 3-2 箭头相反的方向进行类型转换，如将 double 类型的数据转换成 int 类型或 float 类型的数据，则不能进行隐式转换，需要进行显式转换。显式转换使用塑型运算符(Typecast Operator)来实现运算数的类型转换。塑型运算符的语法格式为：

```
(<data type>) <expression>
```

例如：

```
double x = 10.0/3.0;   //x 的值为 3.3333333333333335
int n = (int) x;       //n 的值为 3
float f = (float) x    //f 的值为 3.3333333
```

布尔类型和数字类型之间不能进行转换。当字符串(不属于基本数据类型)与数值进行混合运算时(通常为“字符串＋操作数”)，操作数会被自动转换为字符串类型，这种情况在输出语句中经常遇到。例如，要输出前面 3 个变量的值，可以使用下面的输出语句：

```
System.out.println("x = " + x);
System.out.println("n = " + n);
System.out.println("f = " + f);
```

也可以使用换行符，将上面要输出的内容写在一个输出语句中：

```
System.out.println("x = " + x + "\n" + "n = " + n + "\n" +"f = " + f);
```

但如果需要将字符串转换成数值时，如将字符串"123"转换成数字 123，则不能使用塑型运算符，需要使用包裹类进行转换。包裹类的介绍见 3.4.4 小节。

3.3 控制流程语句

Java 语言中，if 语句和 switch 语句用于选择条件的执行；while 语句、do…while 语句和 for 语句用于循环。