第 2 章　Java 语言编程基础

本章目标：

- 熟练掌握 Java 的标识符、关键字和数据类型的使用
- 掌握通过命令行进行数据输入输出的方法
- 熟练掌握运算符、表达式和各种语句的使用
- 熟练掌握 Java 数组的使用方法

数据类型、表达式、控制语句以及数组是任何一门程序设计语言均包含的基本内容，也是程序设计的基础。本章主要介绍 Java 语言的基础知识，包括变量、数据类型、常量、运算符、流程控制语句以及数组等。

2.1　标识符、关键字和注释

2.1.1　标识符

Java 语言采用标识符对变量、方法、对象、类和接口进行命名。Java 中对标识符的规定如下：

（1）标识符由字母、数字、_和$组成，其中第一个字符不能是数字。

（2）标识符大小写敏感，例如 Student 和 student 是两个不同的标识符。

（3）标识符没有最大长度限制，但通常不超过 15 个字符。

（4）Java 关键字、保留字等不能作为用户自定义标识符使用。

注意：如果确实需要分隔，可以用下划线代替空格，如“MAX_SIZE”。

为便于理解，标识符通常具有一定的含义，例如 student_name。如果将标识符定义成 aaa、bbb，就容易混淆。

合法的标识符示例：

```
intTest    Student    var3    $name    _sss
```

不合法的标识符示例：

```
3name                //数字不能作为第一个字符
my#                  //含有非法字符
switch               //Java 关键字
```

Java 采用的字符集为 Unicode 码。Unicode 码是一个国际标准字符集，在这种字符集中，每个字符占 2 个字节。Unicode 字符集中共包含 65536 个字符，其中前 128 位与 ASCII 码完全兼容。此外，Unicode 字符集还涵盖了中文、日文、朝鲜文、德文、希腊文等多国语言中的符号。

2.1.2　关键字与保留字

1. 关键字

在 Java 语言中，有一部分标识符是系统定义的，它们有着专门的意义和用途，不能当作

一般的标识符使用，这些标识符称为关键字（key word）。表 2-1 中列出了 Java 语言中的大部分关键字，完整的关键字信息请参阅 Java 语言相关文档。

表 2-1　Java 关键字

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

2. 保留字

保留字（reserved word）是指 Java 语言预留的但暂时没有使用的关键字。对于保留字，用户也不能将其作为自定认标识符使用。表 2-2 列出了 Java 语言中的保留字。

表 2-2　Java 保留字

byValue	const	future	generic	goto
inner	outer	operator	rest	var

2.1.3　注释

在 Java 语言中，注释语句是一种特殊的语句，注释的内容不会被 Java 编译器编译，只是用来帮助其他阅读或使用该程序的人理解源程序的含义和作用。

Java 中采用下面三种注释方式。

1. 单行注释

标记“//”后的整行语句为注释，编译程序将不予编译。

2. 块注释

凡是在标记“/*”和标记“*/”之间的内容都是注释，编译程序将不予编译。注意，这种注释不能互相嵌套。

3. doc 注释

doc 注释以标记“/**”开始，到标记“*/”结束，内容位于其中。doc 注释一般位于整个文档的最前面，它是 Java 特有的，主要是为支持 JDK 工具 javadoc 而采用的一种注释方式。javadoc 能识别注释中用标记“@”标识的一些特殊变量，并把 doc 注释加入它所生成的 HTML 文件中。使用 JDK 提供的 javadoc.exe 可以制作源文件的 HTML 格式文档。

假如在 C:\java 目录下有源文件 Student.java，用 javadoc 生成 Student.java 的 HTML 格式文档的命令为：

```
javadoc Student.java
```

这时，在文件夹 java 中将生成若干个 HTML 文档，打开 index.html 文件即可查看到该文档的相关信息。

使用 javadoc 时，也可以使用参数-d 指定生成文档所在的目录，例如：

```
javadoc -d    E:\javadoc Student.java
```

这时，生成的 HTML 文件位于 E 盘的 javadoc 目录下。

注意：只有 doc 注释中书写的内容可以出现在 HTML 文档中。

2.2　基本数据类型

2.2.1　数据类型概述

数据类型指明了变量或表达式的状态和行为。与其他高级语言类似，Java 语言中定义了基本数据类型，通常用户是不能加以修改的。同时，Java 中还定义了引用类型，引用类型是用户根据自己的需要定义并实现的类型，这将在后面的章节中进行介绍。Java 语言的数据类型如图 2-1 所示。

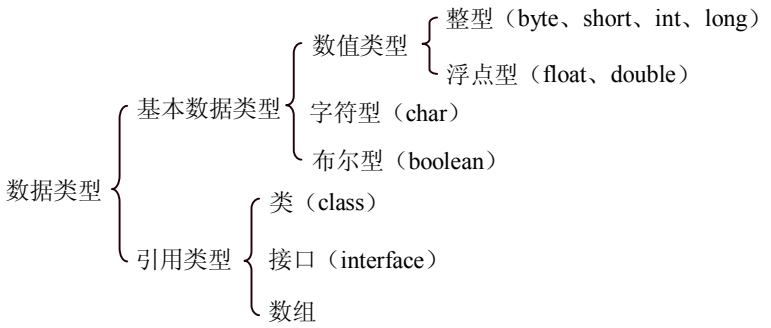

图 2-1　Java 语言的数据类型

Java 语言的基本数据类型如表 2-3 所示。

表 2-3　Java 基本数据类型

数据类型	说明	占内存空间（位数）
byte	字节型	1 字节（8 位）
short	短整型	2 字节（16 位）
int	整型	4 字节（32 位）
long	长整型	8 字节（64 位）
float	单精度浮点型	4 字节（32 位）
double	双精度浮点型	8 字节（64 位）
boolean	布尔型	1 字节（8 位）
char	字符型	2 字节（16 位）

2.2.2 整数类型

整数类型又称为整型，整型数据是一种数值型数据，是指那些没有小数部分的数据。Java 为整型数据提供了 byte、short、int、long 四种类型。其中，int 类型是最常使用的一种整数类型，它所表示的数据范围足够大，而且适合于 32 位、64 位处理器，但对于大型计算，常会遇到很大的整数，超出 int 类型所表示的范围，这时需要使用 long 类型。由于不同的机器对于多字节数据的存储方式不同，在分析网络协议或文件格式时，为了解决不同机器上的字节存储顺序问题，用 byte 类型来表示数据是最合适的，但是通常情况下，由于 byte 类型表示的数据范围很小，容易造成溢出，应尽量避免使用。short 类型通常很少被使用，它限制数据的存储为先高字节、后低字节，这样在某些机器中极易出错。整数类型的变量定义举例如下：

```
int   i;                    //指定 i 为 int 型
long a=0;                   //指定 a 的类型为长整型，且初值为 0
```

注意：与 C 语言不同，Java 中所有的数值都是有符号的。

2.2.3 浮点类型

浮点类型又称为实型，指的是包含小数部分的数据，分为 float 和 double 两种类型。双精度类型 double 比单精度类型 float 具有更高的精度和更大的表示范围，但在精度要求不太高的情况下，使用 float 类型有着速度快、占用存储空间小的优点。为 float 类型的变量赋值需加上后缀 f，例如：

```
float f1=125.39;            //错误，没有加后缀
float f2=66.88f;            //正确，加后缀 f
```

2.2.4 布尔类型

布尔类型数据只有两个值——true（真）和 false（假）。布尔类型一般用于逻辑测试，在流程控制中也常用到它。布尔类型的定义举例如下：

```
boolean f= false;           //定义 f 为布尔类型
```

注意：与其他高级语言不同，Java 中的布尔值和数值之间不能来回转换，即 false 和 true 不对应于任何零或非零的整数值。

2.2.5 字符类型

字符类型使用关键字 char 进行定义，它在机器中占 16 位，其范围为 0～65535。字符类型的定义如下：

```
char c = 'a';               //指定变量 c 为 char 型
```

char 类型的字符与 int 类型可相互转换，一般情况下，char 类型的值可自然转换成 int 类型，而从 int 类型转换成 char 类型需要强制执行。

【例 2-1】字符型数据与整型数据的相互转换

```
1    public class Example2_1 {
2        public static void main (String[] args){
3            char chinaWord='我';
4            int p1=25105;
5            System.out.println("汉字\"我\"在 unicode 表中的顺序位置:"+(int)chinaWord);
6            System.out.println("unicode 表中第 25105 位置上的字符是:"+(char)p1);
```

```
7         }
8     }
```

运行结果：

```
汉字"我"在 unicode 表中的顺序位置:25105
unicode 表中第 25105 位置上的字符是:我
```

说明：将 int 型转换为 char 型，表示获取 Unicode 字符集中指定位置处的字符，将 char 型转换为 int 型，表示获取该字符在 Unicode 字符集中的位置。

2.2.6　基本数据类型间的相互转换

Java 是一种强类型语言，当把一个表达式赋值给一个变量或是在一个方法中传递参数时，都要求两者的数据类型相互匹配。如果要求数据类型的绝对匹配，程序的设计将会非常困难。Java 允许某些不同类型的数据之间按一定规则进行转换以简化程序设计。对于 Java 中的数值类型，可以进行两种转换：自动转换和强制转换。

1. 自动转换

自动转换是指数据在一定的条件下自动升级为具有更高精度类型的数据。类型的精度由低到高的排列顺序是：

byte→short→char→int→long→float→double

按照优先关系，低精度数据要转换成高精度数据时，进行自动类型转换。如果赋值变量的数据类型精度高于表达式结果数据类型的精度，则表达式结果的数据类型将被自动转换为赋值变量的数据类型。

例如：

```
long len=64L;
float f=32.0f;
f=len+f;              //表达式 len+f 中有两种类型，len 自动升级为 float 类型
```

2. 强制转换

如果赋值变量的数据类型精度低于表达式结果数据类型的精度，则表达式结果的数据类型必须强制转换为赋值变量的数据类型。

强制转换是在代码中把原来某一精度类型的数据用指定的操作符转换为另一精度类型的数据。把高精度类型的数据转换为低精度类型的数据时必须使用强制转换，此时，数据可能会丢失部分信息。强制转换操作符的使用格式如下：

(目标数据类型)表达式

它的作用是把表达式的值转换为目标数据类型，例如：

```
int i=(int)32.6;              //把表达式的值 32.6 转换为 int 类型，该值为 32
char c=(char)(12.5+54);       //把表达式的值 66.5 转换为 char 类型，该值为'B'
double d=2.3;
int j=(int)d;                 //将 double 型变量强制转换为 int 类型
```

注意：在进行强制类型转换时，并不改变变量的类型，如上例中变量 d 仍为 double 类型。

2.2.7　从命令行输入输出数据

1. 输入数据

Java 中使用 Scanner 类从命令行输入数据。Scanner 类是一种解析基本数据类型和字符串的简单文本扫描器，它使用分隔符模式将输入分解为标记，默认情况下该分隔符模式与空格匹

配，然后使用不同的 next 方法将得到的标记转换为不同类型的值。使用以下语句可以为该类创建一个对象：

```
Scanner reader=new Scanner(System.in);
```

Scanner 类的 nextBoolean()、nextByte()、nextShort()、nextInt()、nextLong()、nextFloat()和 nextDouble()等方法可以读取用户在命令行输入的各种基本类型的数据。

注意：上述所有方法在执行时都会出现线程堵塞的情况，程序等待用户在命令行输入数据并按 Enter 键确认。

2. 输出数据

System.out.println()或 System.out.print()方法用来输出串值和表达式的值，二者的区别是前者输出数据后换行，后者则不换行。Java 中允许使用“+”将变量、表达式或一个常量数值与一个字符串连接起来一起输出。

注意：Java 中同时支持 printf 方法进行数据输出操作，其具体使用方法可查看 JDK 帮助文档。

【例 2-2】读入用户在命令行输入的数据并输出

```
1    import java.util.Scanner;
2    public class Example2_2 {
3        public static void main(String args[]) {
4            Scanner reader = new Scanner(System.in);
5            int number = reader.nextInt();                    //读入用户输入的整型数据
6            System.out.println("用户输入的是:" + number);
7        }
8    }
```

运行结果：

```
6↙
用户输入的是:6
```

说明：使用 nextInt 方法接收用户在命令行中输入的整型数据。

注意：Java 中没有提供接收字符类型数据的方法，Scanner 对象可以使用 next().charAt(0)接收字符数据。

2.3 运算符与表达式

按照运算符功能划分，Java 中常用的运算符有下面几类：

（1）算术运算符。

（2）关系运算符。

（3）布尔逻辑运算符。

（4）位运算符。

（5）赋值运算符。

（6）条件运算符。

（7）其他运算符。

Java 中的表达式是变量、常量、运算符以及方法调用的序列，它执行指定的计算并返回某个确定的值。表达式的类型由运算以及参与运算的操作数的类型决定，可以是基本数据类型，也可以是引用类型。

2.3.1　算术运算符与算术表达式

算术表达式由操作数和算术运算符组成。在算术表达式中，操作数只能是整型或浮点型数据。Java 的算术运算符分为二元运算符和一元运算符两种。

1. 二元算术运算符

二元算术运算符涉及两个操作数。Java 中的二元算术运算符共有五种，如表 2-4 所示。

表 2-4　二元算术运算符

运算符	表达式	功能
+	op1 + op2	加
-	op1 - op2	减
*	op1 * op2	乘
/	op1 / op2	除
%	op 1% op2	求余

注意：Java 与 C、C++语言不同，对取模运算符“%”来说，其操作数可以为浮点数，如 10% 3.5=3.0。

这些算术运算符适用于所有数值型数据。需要注意的是，如果操作数全为整型，那么，只要其中有一个为 long 型，则表达式结果也为 long 型；其他情况下，即使两个操作数全是 byte 型或 short 型，表达式结果也为 int 型。如果操作数为浮点型，那么，只要其中有一个为 double 型，表达式结果就是 double 型；只有两个操作数全是 float 型或其中一个是 float 型而另外一个是整型时，表达式结果才是 float 型。

例如：

```
int a=7/2;                  //结果为 3
double b=7/2.0;             //结果为 3.5
double f=b%a;               //结果为 0.5
```

注意：当“/”运算和“%”运算中除数为 0 时，程序可编译通过，但在运行时会产生算术异常（异常将在第 5 章介绍）。

2. 一元算术运算符

一元算术运算符涉及的操作数只有一个。Java 中的一元算术运算符共有四种，如表 2-5 所示。

表 2-5　一元算术运算符

运算符	名称	表达式	功能
+	一元加	+op1	取正值
-	一元减	-op1	取负值
++	增量	++op1，op1++	加 1
--	减量	--op1，op1--	减 1

一元加（+）和一元减（-）运算符仅仅表示某个操作数的符号，其操作结果为该操作数的

正值或负值。增量运算符（++）将操作数加 1，减量运算符（--）将操作数减 1。例如，假设 a = 5、b = 10，计算表达式 c = (a++)*(--b)，则计算过程分为以下三步：

① b = b-1 = 9

② c = a*b = 45

③ a = a+l = 6

2.3.2 关系运算符与关系表达式

关系表达式由操作数和关系运算符组成，关系运算符用来对两个操作数进行比较。关系运算符的操作结果是布尔类型，即如果运算符对应的关系成立，则关系表达式结果为 true，否则为 false。关系运算符都是二元运算符，共有六种，具体如表 2-6 所示。

表 2-6 关系运算符

运算符	表达式	功能
>	op1>op2	比较 op1 是否大于 op2
<	op1<op2	比较 op1 是否小于 op2
>=	op1>=op2	比较 op1 是否大于等于 op2
<=	op1<=op2	比较 op1 是否小于等于 op2
= =	op1==op2	比较 op1 是否等于 op2
!=	op1!=op2	比较 op1 是否不等于 op2

例如：

表达式 3 > 5 的值为 false；

表达式 3 <= 5 的值为 true；

表达式 3 == 5 的值为 false；

表达式 3 != 5 的值为 true。

注意：关系运算的结果返回 true 或 false，而不是 C 或 C++语言中的非零值或零值。

2.3.3 逻辑运算符与逻辑表达式

逻辑表达式由关系表达式和逻辑运算符组成。逻辑运算符用来连接关系表达式，对关系表达式的值进行布尔逻辑运算。逻辑运算符共有三种，即逻辑与（&&）、逻辑或（||）和逻辑非（!），当两个布尔操作数进行操作时，其操作结果也都是布尔型的，具体的运算结果如表 2-7 所示。

表 2-7 布尔值的逻辑运算表

关系表达式 1 的值（op1）	关系表达式 2 的值（op2）	op1&&op2	op1||op2	!op1
true	false	false	true	false
true	true	true	true	false
false	false	false	false	true
false	true	false	true	true

其中，“&&”和“||”为二元运算符，实现逻辑与、逻辑或。“!”为一元运算符，实现逻辑非。逻辑运算符常与关系运算符一起使用，作为流程控制语句的判断条件。例如：

```
if(a==b&&a!=c){ }
```

需要说明的是，Java 中的“&&”和“||”运算采用电工学中的“短路”方式进行计算，运算时先求出运算符左边的表达式的值。对于或运算，如果左边表达式的值为 true，则整个逻辑表达式的结果确定为 true，不会再对运算符右边的表达式进行运算；同样，对于与运算，如果左边表达式的值为 false，则不会再对运算符右边的表达式求值，整个逻辑表达式的结果已确定为 false。例如：

```
((y=2)==0) &&((x=5)==5);        //x 的初值是 2，那么经过该运算后，x 的值仍然是 2
((y=2)==2) &&((x=5)==5);        //x 的初值是 2，那么经过该运算后，x 的值将变为 5
```

2.3.4　位运算符与位表达式

位运算表达式由位运算符和整型操作数组成，位运算符可以对整型数据的二进制位进行操作。Java 中把位运算符分为位逻辑运算符和移位运算符两类，如表 2-8 所示。

表 2-8　位运算符

运算符	功能	表达式
～	按位取反	～op
&	按位与	op1&op2
|	按位或	op1|op2
^	按位异或	op1^op2
>>	op1 按位右移 op2 位	op1>>op2
<<	op1 按位左移 op2 位	op1<<op2
>>>	op1 添零右移 op2 位	op1>>>op2

1．位逻辑运算符

（1）按位取反运算符～。

“～”是一元运算符，对数据的每个二进制位取反，即把 1 变为 0，把 0 变为 1。例如：

～10011110 的值为 01100001

（2）按位与运算符&。

两个操作数中，如果两个对应位都为 1，该位的结果为 1，否则为 0。例如：

00101011 & 00010111 的值为 00000011

（3）按位或运算符 |。

两个操作数中，只有两个位都为 0 时，该位的结果为 0，否则为 1。例如：

10101010 | 00010111 的值为 10111111

（4）按位异或运算符 ^ 。

两个操作数中，如果两个对应位相同，结果为 0，否则为 1。例如：

10101010 ^ 00010111 的值为 10111101

由异或运算法则可知：a^a=0，a^0=a。因此，a^b^b=a，即用同一个数对 a 进行两次“异或”运算，结果仍然是 a。

注意：Java 中的按位与和按位或运算符也可进行逻辑与和逻辑或运算，但不具备短路特性。

2. 移位运算符

（1）右移运算符>>。右移运算符用来将一个数的二进制位序列右移若干位。例如 a=a>>2，使 a 的各二进制位右移 2 位。在对数据进行右移操作时，移到右端的低位被舍弃，最高位将移入原来高位的值。例如：

a = 00110011，则 a >> 2 的值为 00001100

b = 11110011，则 b >> 2 的值为 11111100

（2）左移运算符<<。左移运算符用来将一个数的二进制位序列左移若干位。例如 a = a << 2，使 a 的各二进制位左移 2 位，右端补 0。例如：

a = 10001111，则 a << 2 的值为 00111100

（3）添零右移运算符>>>。添零右移运算符用来将一个数的各二进制位添零右移若干位。通常，添零右移也被称为无符号右移运算符。例如：

a = 00110111，则 a >>> 2 的值为 00001101

b = 11010011，则 b >>> 2 的值为 00110100

注意：Java 中没有提供<<<运算符。

2.3.5 赋值运算符与赋值表达式

赋值表达式由变量、赋值运算符和表达式组成，其定义的格式为：

变量=表达式；

其中，表达式的值的类型应与左侧的变量类型一致或者可以转换为左侧的变量类型。赋值运算符分为赋值运算符“=”和扩展赋值运算符两种。

1. 赋值运算符

赋值运算符“=”用来把一个表达式的值赋给一个变量。当赋值运算符两侧的类型不一致时，如果左侧变量类型的级别高于右侧表达式的值，右侧的数据将被转化为与左侧相同的高级数据类型后再赋给左侧变量，否则需要对右侧表达式的值进行强制转换。例如：

```
byte b=12;
int i=b;                          //自动类型转换
int a=13;
byte c=(byte)a;                   //强制类型转换
```

2. 扩展赋值运算符

在赋值运算符“=”前加上其他运算符，即构成了扩展赋值运算符。例如：

a+=3 等价于 a=a+3。

表 2-9 列出了 Java 中的扩展赋值运算符及等效的表达式。

表 2-9 扩展赋值运算符

运算符	表达式	等效表达式
+=	op1+=op2	op1=op1+op2
-=	op1-=op2	op1=op1-op2
*=	op1*=op2	op1=op1*op2
/=	op1/=op2	op1=op1/op2

续表

运算符	表达式	等效表达式
%=	op1%=op2	op1=op1%op2
&=	op1&=op2	op1=op1&op2
|=	op1|=op2	op1=op1|op2
^=	op1^=op2	op1=op1^op2
>>=	op1>>=op2	op1=op1>>op2
<<=	op1<<=op2	op1=op1<<op2
>>>=	op1>>>=op2	op1=op1>>>op2

2.3.6　条件运算符与条件表达式

条件运算符是三元运算符，用“?”和“:”表示。三元条件表达式的一般形式为:

表达式 l? 表达式 2: 表达式 3

其中，表达式 l 为关系或逻辑表达式，其计算结果为布尔值。如果该值为 true，则计算表达式 2，并将计算结果作为整个条件表达式的结果；如果为 false，则计算表达式 3，并将计算结果作为条件表达式的结果。例如：

```
int a=20;
boolean b=a>16?160:180;
```

等号右边为条件表达式，a>16 的计算结果为 true，所以计算后变量 b 的值为 160。

2.3.7　其他运算符

1. 括号运算符()

在 Java 语言中，括号运算符()的运算优先级最高，在表达式中使用时，用于改变运算符的运算顺序。

2. 下标运算符[]

[]是数组下标运算符，定义数组时用于声明数组元素的个数，引用数组元素时用于提供数组的下标值。

3. 对象运算符 instanceof

对象运算符 instanceof 用于测试一个指定的对象是否是指定类（或它的子类）的一个实例化的对象，若是则返回 true，否则返回 false。例如：

```
String s = "abcde";
boolean b = s instanceof String;
```

由于 s 是字符串类 String 的一个实例化对象，所以 b 的值为 true。

4. 内存分配运算符 new

Java 语言使用 new 运算符为数组和对象分配内存空间。

2.3.8　运算符的优先级与结合性

Java 的表达式就是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。例如：

x<y&&!z 相当于(x<y)&&(!z)

通常，在编写程序时，没有必要去记忆运算符的优先级别，可以尽量使用括号运算符来实现规定的运算次序，以免产生难以阅读或含糊不清的计算顺序。运算符的结合性决定了并列相同级别运算符的先后顺序。例如，加减的结合性是从左到右，8-5+3 相当于(8-5)+3。逻辑否运算符的结合性是从右到左，!!x 相当于!(!x)。

在对一个复杂表达式进行运算时，要按运算符的优先顺序从高到低进行，同级的运算符则按结合性进行。表 2-10 列出了 Java 中运算符的优先级与结合性。

表 2-10　运算符的优先级与结合性

运算符	优先级	结合性
. [] ()	1	
++ -- ! ~ instanceof	2	右到左
new	3	左到右
* / %	4	左到右
+ -	5	左到右
>> << >>>	6	左到右
> < >= <=	7	左到右
== !=	8	左到右
&	9	左到右
^	10	左到右
|	11	左到右
&&	12	左到右
||	13	左到右
?:	14	左到右
= += -= *= /= %= ^=	15	右到左
&= |= <<= >>= >>>=	16	右到左

2.4　控制语句

Java 中的程序控制语句分为 3 类：选择语句、循环语句和跳转语句。有了这些控制语句，Java 程序能更加灵活地实现用户所需要的各种功能。

2.4.1　选择语句

选择语句提供了这样一种控制机制，它根据条件或表达式值的不同选择执行不同的语句序列，其他与条件值或表达式值不匹配的语句序列将被跳过不执行。选择语句分为以下四种情况：

1. 单分支 if 语句

Java 语言中，最简单的选择语句是 if 语句。if 语句的格式如下：

```
if(条件)
    statement;
```

或者

```
if(条件) {
    block
}
```

第一种情况下，在条件为真时，执行一条语句 statement，否则跳过 statement 执行下面的语句。第二种情况下，在条件为真时，执行多条语句组成的代码块 block，否则跳过 block 执行下面的语句。

【例 2-3】使用 if 语句输出一个数的绝对值

```
1   public class Example2_3 {
2       public static void main(String args[]) {
3           int a = 8;
4               if (a< 0) {
5                   a = -a;
6               }
7           System.out.println("绝对值为：" + a );
8       }
9   }
```

运行结果：

```
绝对值为：8
```

2. 双分支 if-else 语句

Java 语言中，较常使用的选择语句是 if-else 语句。if-else 语句的格式如下：

```
if(条件)
    statement1;  或   {block1}
else
    statement2;  或   {block2}
```

当条件为真时，执行语句 statement1（或代码块 block1），然后跳过 else 和 statement2（或代码块 block2）执行下面的语句；当条件为假时，跳过语句 statement 1（或代码块 block1）执行 else 后面的 statement2（或代码块 block2），然后继续执行下面的语句。

注意：else 子句不能单独作为语句使用，它必须和 if 子句配对使用。if 和 else 后面的代码块里如果只有一个语句，{ }可以省略不写，但为了增强程序的可读性最好不要省略。

【例 2-4】使用 if-else 表达式判断成绩

```
1   public class Example 2_4{
2       public static void main(String args[]) {
3           int math=85;
4           if(math>60) {
5               System.out.println("数学及格了");
6           }
7           else {
8               System.out.println("数学不及格");
9           }
10      }
11  }
```

运行结果：

数学及格了

3. 多分支 if-else if 语句

当需要处理多个分支时，可以使用 if-else if 语句。if-else if 语句采用的格式如下：

```
if(条件 1)
    statement1;      或    {block1}
else if(条件 2)
    statement2;      或    {block2}
    ⋮
else if(条件 N)
    statementN;      或    {blockN}
else
    statementN+1; 或    {blockN+1}
```

4. switch 语句

处理多个分支时，使用 if-else if 语句显得非常繁琐。和 C、C++语言相同，Java 也提供了 switch 语句。switch 语句根据表达式的值从多个分支中选择一个分支来执行。它的一般格式为：

```
switch (expression){
    case  value1:
            statement1;
            break;
    case  value2:
            statement2;
            break;
     ⋮
    case  value N:
            statementN;
            break;
    default:
            default statement;
}
```

在使用 switch 语句时需注意以下几点：

（1）表达式 expression 的返回值类型只能是 char、byte、short 或 int 类型。

（2）switch 语句先取得表达式 expression 的返回值，然后根据返回值依次与每个 case 语句所对应的 value1，value2，…，valueN 值匹配，如果匹配成功则执行对应的代码块。

（3）case 语句所对应的 value 值必须为常量，而且各 value 值应当不同。

（4）break 语句用来在执行完相应的 case 分支语句后跳出 switch 语句，否则将顺序执行后面的语句。

（5）default 是可选的，当表达式的值与任何的 value 值都不匹配时，则执行 default 代码块。如果没有 default 语句时，则程序不做任何操作，直接跳出 switch 代码块。

switch 语句的功能可以用 if-else if 语句来实现，但在某些情况下，使用 switch 语句更简练，可读性更强，而且程序的执行效率也更高。

注意：在 JDK7.0 中，switch 语句中表达式的返回值类型也可以是 String 类型。

【例 2-5】使用 switch 语句判断成绩等级

```
1    import java.util.*;
2    class Example2_5 {
3        public static void main(String args[]) {
4            Scanner reader = new Scanner(System.in);
5            int score = reader.nextInt();                    //从键盘中读入成绩
6            switch (score/10) {
7                case 9:
8                    System.out.print("A");break;
9                case 8:
10                   System.out.print("B");break;
11               case 7:
12                   System.out.print("C");break;
13               case 6:
14                   System.out.print("D");break;
15           }
16       }
17   }
```

运行结果：

```
99↙
A
```

2.4.2　循环语句

在程序设计中，有时需要反复执行一段相同的代码，直到满足一定的条件为止，这种操作可以通过循环语句实现。Java 中提供的循环分为不确定循环和确定循环两种。

1．不确定循环

当不知道一个循环会被重复执行多少次时，可以选择 Java 提供的不确定循环。不确定循环有两种形式供选择：一种是 while 循环，另一种是 do-while 循环。

（1）while 循环。

while 循环又称“当型”循环，它的一般格式为：

```
while (termination){
    body;
    [iteration];
}
```

① 当布尔表达式（termination）的值为 true 时，循环执行大括号中的语句，其中迭代部分（iteration）是任选的。若某次判断布尔表达式的值为 false，则结束循环的执行。

② while 循环首先计算终止条件，当条件满足时，才去执行循环体中的语句或代码块；若首次计算条件就不满足，则大括号中的语句或代码块一次都不会被执行。

（2）do-while 循环。

do-while 循环又称“直到型”循环，它的一般格式为：

```
do{
    body;
    [iteration];
}
while(termination);
```

① do-while 循环首先执行循环体，然后计算终止条件，若结果为 ture，则循环执行大括号中的语句或代码块，直到布尔表达式的结果为 false。

② 与 while 循环不同的是，do-while 循环的循环体至少被执行一次。

2. 确定循环

如果事先知道循环会被重复执行多少次，可以选择 Java 提供的确定循环结构——for 循环。for 循环的一般格式为：

```
for(initialization; termination; iteration){
    body;
}
```

对 for 循环的说明如下：

（1）for 循环执行时，首先执行初始化操作（initialization），然后判断终止条件（termination）是否满足，如果满足，则执行循环体中的语句，最后执行迭代部分（iteration）。完成一次循环后，重新判断终止条件。

（2）可以在 for 循环的初始化部分声明一个变量，它的作用域为整个 for 循环。

（3）for 循环通常用于循环次数确定的情况，但也可以根据循环结束条件用于循环次数不确定的情况。

（4）在初始化部分和迭代部分可以使用逗号分隔多个操作。例如：

```
for(i=0, j=10; i<j; i++){ }
```

（5）初始化、终止以及迭代部分都可以为空语句（但分号不能省略）。例如：

```
for(; i<=5; ){ }
```

注意：初始化、终止以及迭代部分三者均为空的时候，相当于一个无限循环。

【例 2-6】用 for 循环求 1000 之内的所有完全数

```
1    class Example2_6{
2        public static void main (String args[]){
3            System.out.println("1000 之内的完全数为：");
4            for(int i= 1;i<1000;i++){                    //遍历 1～1000 的所有整数
5                int y= 0;
6                for(int j=1;j<i;j++)
7                    if(i%j==0)
8                        y+=j;                            //将该数的所有真约数相加
9                if(y == i){                              //若该数为完全数，将其输出
10                   System.out.println(i);
11               }
12           }
13       }
14   }
```

运行结果：

```
1000 之内的完全数为：
6
28
496
```

说明：本例输出 1 到 1000 之间的所有完全数。完全数（perfect number），又称完美数或完备数，是一些特殊的自然数，它所有的真约数（即除了自身以外的约数）的和恰好等于它本身。

2.4.3　跳转语句

跳转语句用来控制流程的走向，Java 提供了三种跳转语句。

1．break 语句

break 语句虽然可以独立使用，但通常用于 switch 语句和循环语句中，控制程序的执行流程转移。break 语句的应用有下列两种情况：

（1）break 语句用在 switch 语句中，其作用是强制退出 switch 语句，执行 switch 语句后的语句。

（2）break 语句用在单层循环结构的循环体中，其作用是强制退出循环结构。若程序中有内外两重循环，而 break 语句写在内循环中，则执行 break 语句只能退出内循环，而不能退出外循环。

【例 2-7】利用 break 语句求一个数的最大真约数

```
1    import java.util.*;
2    class Example2_7 {
3        public static void main(String args[]) {
4            int i = 0;
5            int x = 0;
6            Scanner in = new Scanner(System.in);
7            i = in.nextInt();
8            x = i - 1;
9            while (x > 0) {
10               if (i % x == 0)
11                   break;              //当找到能够整除输入数据的整数时，退出循环
12               x--;
13           }
14           System.out.println(i + "的最大真约数是" + x);
15       }
16   }
```

运行结果：

```
6↙
6 的最大真约数是 3
```

说明：本例从键盘输入一个整数，并求出这个整数的最大真约数。将从键盘输入的数值赋值给变量，后面的代码用来求出 i 的真约数，最大真约数就是最大的能够整除 i 的数，其算法是用从 i-1 开始的数逐个去除 i，直到找到能够整除 i 的数为止，这个数就是 i 的最大真约数。

2．continue 语句

continue 语句用于结束本次循环，即跳过循环体后面尚未执行的语句，执行下一次循环。continue 语句也称为循环的短路语句，只能用在循环结构中。在循环结构中，当程序执行到 continue 语句时就返回到循环体的入口处，执行下一次循环，因而循环体内写在 continue 语句后的语句不被执行。

【例 2-8】利用 continue 输出 100 到 200 之间的奇数

```
1    class Example2_8{
2        public static void main(String args[]){
3            for(int i=101;i<=200;i++){
4                if(i%2==0)
5                    continue;                    //若是偶数，不进行输出操作，直接转入下一次循环
6                System.out.println(i);
7            }
8        }
9    }
```

3．return 语句

return 语句用于从方法（子程序）中返回到它的调用者。return 语句的执行情况可分为以下两种：

（1）带值返回：return 语句跳出当前的方法返回到调用者时，返回任一类型的数值。

（2）不带值返回：return 语句跳出当前的方法返回到调用者时，不带任何值。

【例 2-9】return 语句的应用

```
1    class Example 2_9{
2        public static void main(String args[]){
3            boolean test=true;
4            System.out.println("我被执行！ ");
5            if(test){
6                return;
7            }
8            System.out.println("return 执行之后!");    //这条语句不被执行
9        }
10   }
```

运行结果：

```
我被执行!
```

说明：在执行到 return 之后，就从该方法中跳出来，程序后面的语句不再被执行。

2.5 数组

Java 中的数组是一种引用类型，它是由类型相同的元素组成的有顺序的数据集合。数据元素既可以是 Java 中的基本数据类型，也可以是引用类型，甚至可以是其他的数组类型。Java 中的数组具有如下特性：

（1）每个元素的数据类型都是相同的。

（2）数组中的各个元素都是有顺序的。

（3）所有元素共用一个数组名，利用数组名和数组下标来唯一地引用数组中的各个元素。

（4）数组要经过声明以及分配内存后，才能被使用。

2.5.1 一维数组

一维数组是数组中最简单的数组，一维数组的定义分为数组声明和为数组分配内存空间两步。

1．一维数组的声明

一维数组的一般声明格式为：

type arrayName[ ];

或者

type[ ] arrayName;

其中 type 表示数组元素的类型，可以是 Java 中任意的数据类型，包括基本数据类型和引用类型；数组名 arrayName 为一个合法的标识符；[ ]指明该变量是一个数组类型变量，[ ]可以写在类型后或标识符后。例如：

```
int myArray[ ];              //声明了一个 int 类型数组 myArray，[ ]位于标识符后
double[ ] price;             //声明了一个 double 类型数组 price，[ ]位于类型后
```

与 C、C++语言不同，Java 在数组声明时仅仅声明了数组的名字和元素的类型，并没有对数组元素分配内存、生成实例。数组元素内存分配由 new 语句完成，因此数组声明中[]不用说明数组元素个数，而且如上声明的数组是不能访问它的任何元素的，必须在对数组元素分配内存后，才能引用数组的元素。

2．数组元素内存分配

为数组元素分配内存的方法很简单，只要用 new 操作符为数组元素分配内存即可。格式如下：

arrayName= new type[arraysize];

其中 arrayName 为已声明的数组名，type 为数组元素的类型，arraysize 为数组的长度。这条语句的作用是为数组 arrayName 分配 arraysize 个 type 类型大小的空间。例如：

```
myArray = new int[50];              //给 myArray 分配 50 个 int 类型的数据空间
```

也可以将数组声明与为数组分配内存空间合成一条语句，例如：

```
int myArray[] = new int[50];
```

注意：和 C 语言不同的是，Java 允许使用 int 型变量指定数组的大小。

例如：

```
int size=20;
double number[ ] = new double[size];
```

3．一维数组的初始化

当一个数组用 new 为数组分配内存后，自动用数据类型的缺省值初始化所有的数组元素。各种数据类型的初始值如表 2-11 所示。

表 2-11　数组元素默认初值

类型	初始化值	类型	初始化值
int	0	double	0.0d
long	0L	boolean	false
float	0.0f	引用类型	null

例如：

```
int myArray[ ] = new int[50];          //myArray 的 50 个 int 型元素的初始化为 0
```

另一种创建数组的方法是直接赋值，初值的个数就是数组的大小，初始值必须用大括号括住，用逗号作为分隔符。

```
int myArray[ ]={1,2,3,4,5,6};
```

4. 一维数组元素的引用

当定义了一个数组，为数组及其元素分配空间并初始化后，就可以引用数组中的每一个元素了。数组中元素的引用方式为：

arrayName[index]

其中，index 为数组下标，可以是整型常数或表达式。如 myArray[0]，myArray[i]（i 为整型），myArray[5*2+1]等。但是，数组下标只能从 0 开始，一直到数组的长度减 1。若 myArray 数组包含 4 个元素，则各元素分别是：

myArray[0]，myArray[1]，myArray[2]，myArray[3]。

注意：如果引用 myArray[4]将引发 ArrayIndexOutOfBoundsException 异常。

【例 2-10】采用冒泡排序算法将 10 个整数按照从小到大的顺序排列

```
1    public classExample2_10 {
2        public static void main(String[] args) {
3            int[] array = { 3, 1, 6, 2, 9, 0, 7, 4, 5, 8 };
4            System.out.print("数组排列前的顺序为：");
5            for (int i = 0; i < array.length; i++) {                    //输出排序前数组中的每个元素
6                System.out.print(array[i] + " ");
7            }
8            int temp;                                                   //存储交换的变量值
9            for (int i = 0; i < array.length-1; i++) {                  //比较 n-1 轮
10               for (int j = 0; j < array.length - i - 1; j++) {        //利用 length 属性控制循环次数
11                   if (array[j] > array[j + 1]) {
12                       temp = array[j];
13                       array[j] = array[j + 1];
14                       array[j + 1] = temp;
15                        }
16               }
17           }
18               System.out.println();
19           System.out.print("数组排列后的顺序为：");
20           for (int i = 0; i < array.length; i++) {                        //输出排序后数组中的每个元素
21               System.out.print(array[i] + " ");
22           }
23       }
24   }
```

运行结果：

```
数组排列前的顺序为：3 1 6 2 9 0 7 4 5 8
数组排列后的顺序为：0 1 2 3 4 5 6 7 8 9
```

说明：Java 对数组元素要进行越界检查以保证安全性。同时，每一个数组都有一个属性 length 指明数组的长度，例如：array.length 的值为 10，指明了数组 array 的长度。

2.5.2 数组的数组

在 Java 语言中，并没有一般程序设计语言中的多维数组，但 Java 可以实现相应多维数组

的功能，把多维数组看成数组的数组。例如二维数组是一个特殊的一维数组，其每一个元素又是一个一维数组。由于在 Java 中需要为数组元素分配相应的空间，分配空间可以在声明数组的同时进行，也可以用 new 操作符为数组元素分配内存，这就使得多维数组中每维数组的长度可以不同，数组空间也不是连续分配的（当然一维数组的空间仍然是连续分配的）。下面主要以二维数组为例来说明多维数组的使用。

1．二维数组声明

二维数组的声明方式为：

```
type arrayName[ ][ ];
```

或

```
type[ ][ ] arrayName;
```

其中类型 type 可以是 Java 中任意的数据类型；数组名 arrayName 为一个合法的标识符。例如：

```
int[ ][ ] myarray2;              //声明了一个整型二维数组
```

2．数组元素内存分配与引用

二维数组的内存空间分配与一维数组相似，也是使用 new 操作符，分配空间的方法有以下两种：

（1）直接为每一维分配空间，格式如下：

```
arrayName = new type[length1][length2];
```

其中 length1 是数组第一维的长度，length2 是数组第二维的长度。

（2）从最高维开始，分别为每一维分配空间，格式如下：

```
arrayName = new type[lengthl][];
arrayName[0] = new type[length20];
arrayName[1] = new type[length2l];
……
arrayVame[lengthl-1]=new type[length2n];
```

二维数组的初始化有以下两种方式：

（1）使用 new 操作符对数组元素初始化。

与一维数组相同，用 new 为数组分配内存后，自动用数据类型的缺省值初始化所有的数组元素。例如：

```
myArray2 = new int[5][];              //生成含 5 个一维数组的数组 myArray2
myArray2[0] = new int[2];             //生成 myArray2 的第 0 维的一维数组 myArray2[0]
myArray2[1] = new int[6];             //生成 myArray2 的第 1 维的一维数组 myArray2[1]
myArray2[1][0] = 10;                  //myArray2[1][0]初始化为 10
```

（2）定义数组的同时对数组初始化。

同一维数组一样，二维数组也可以在定义的同时进行初始化操作。例如：

```
int myArr[ ][ ]={{1,2},{3,4,5},{6,7}};
```

需要说明的是，在这种情况下，不必具体指出数组每一维的大小，系统会根据初始化时的初值个数计算出数组每一维的大小，并为数组及其元素分配相应的空间。

注意：在 Java 语言中，从最高维至最低维分配空间。分配空间时当前最高维必须指定长度，其余低维可不指定长度；但不允许最高维不指定长度而其余低维指定长度。

2.5.3 数组排序

可以使用循环实现对数组的排序，也可以使用循环查找一个数据是否在一个排序的数组中。除了前面介绍的冒泡排序算法外，Arrays 类中定义的 sort 方法实现了对数组的快速排序。Arrays 类位于 java.util 包中，提供了包括排序、查找等一系列对数组进行操作的方法。Arrays 类的常用方法包括：

- public static void sort(double a[ ]) 把参数 a 指定的 double 类型数组按升序排序。
- public static void sort(double a[ ], int start, int end) 把参数 a 指定的 double 类型数组中索引 start 至 end-1 的元素的值按升序排序。
- public static int binarySearch(double a[ ], double number) 判断参数 number 指定的数是否在参数 a 指定的数组中。如果 number 在数组 a 中，返回该元素的索引，否则返回一个负数。

【例 2-11】二分查找数组中的元素

```
1   import java.util.*;
2   public class Example2_11 {
3       public static void main(String args[]) {
4           int a[] = { 3, 5, 7, 6, 8, 9, 1, 2, 4 };
5           Arrays.sort(a);                          //对数组进行排序
6           for(int x:a){                            //遍历数组元素
7               System.out.print(x+" ");
8           }
9       }
10  }
```

运行结果：

```
1 2 3 4 5 6 7 8 9
```

说明：Java 语言允许使用 for 语句遍历数组，如本例第 6 行所示。在使用 for 语句遍历数组元素时，声明的循环变量的类型必须和数组的类型相同，表示循环变量依次取数组的每一个元素的值。

2.6 综合实例

【案例描述】

在歌唱比赛中，共有 6 位评委打分，在计算选手得分时，去掉一个最高分，去掉一个最低分，然后剩余的 4 个评委的分数进行平均，就是该选手的最后得分。要求在输入每位选手的打分后，输出该选手的最后得分。

【案例分析】

将评委打的分数存在数组中，然后对数组按升序进行排序，排序后的数组中，第一个元素为最低分，最后的一个元素为最高分，通过循环，求得数组中剩余 4 个元素的平均值即可。

【参考代码】

```
import java.util.*;
public class Score {
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        int score[] = new int[6];
        int sum = 0;
        for (int i = 0; i < score.length; i++) {
            System.out.println("请输入第" + (i + 1) + "位评委的分数");
            score[i] = reader.nextInt();
        }
        Arrays.sort(score);
        System.out.println("去掉一个最高分" + score[5]);
        System.out.println("去掉一个最低分" + score[0]);
        for (int i = 1; i < score.length - 1; i++) {
            sum += score[i];
        }
        System.out.println("该选手最后的得分是：" + sum / 4);
    }
}
```

运行结果：

```
请输入第 1 位评委的分数
95↙
请输入第 2 位评委的分数
96↙
请输入第 3 位评委的分数
97↙
请输入第 4 位评委的分数
93↙
请输入第 5 位评委的分数
94↙
请输入第 6 位评委的分数
92↙
去掉一个最高分 97
去掉一个最低分 92
该选手最后的得分是：94
```

本章小结

Java 的数据类型包括基本数据类型和引用类型两种，其中，基本数据类型包括整数类型、浮点类型、字符型和布尔型四种。表达式由运算符和操作数组成，常量或变量本身就是一种简单的表达式，复杂的表达式可以包含简单的表达式。

Java 中 if 语句和 switch 语句构成了选择结构，while 语句、do-while 语句和 for 语句构成

了各种循环结构，无论是选择结构还是循环结构都可以嵌套使用。同时，Java 中还定义了 break、continue 和 return 语句完成跳转操作。

数组是一种常见的数据组织形式，组成数组的任意数据称为数组的元素。在计算机系统中，一个数组在内存中占有一段连续的存储空间。

拓展

Java 编程规范

编程规范可以提高代码的可读性，使得开发人员快速理解新代码。好的代码风格不仅会使代码较“健壮”，更为重要的是在修改时不容易出错。在现代软件开发中，维护工作会占用80%的时间，而且开发者和维护者通常不是同一个程序员。这意味着需要经常阅读和修改他人开发的程序，因此，开发规范的代码是非常必要的。常用的编程规范包括：①命名规范。②变量定义规范。③代码编写格式。④注释规范。⑤方法规范。⑥程序编写规范。具体可参考本书附录 1。

习题二

【基础】

1．下面（　　）是 Java 语言的关键字。

A．Double　　B.this　　C.string　　D.bool

2．下面（　　）是 Java 语言中正确的标识符。

A．byte　　B．new　　C．next　　D．rest-1

3．下面（　　）能定义一个字符变量 chr。

A．char chr='abcd';　　B．char chr='\uabcd';

C．char chr="abcd";　　D．char chr="\uabcd";

4．下面（　　）不能定义一个 float 型变量 f1。

A．float f1=3.1415E10　　B．float f1=3.14f

C．float f1=3.1415F　　D．f1=3.14F

【应用】

1．设计 x、y、z 的值分别为 true、true 和 false，试计算下列逻辑表达式的值。

（1）x&&y||!z&&true

（2）!x||!y&&!z

（3）(!x&&!y)||(!y&&!z)

（4）x&&y||true&&!z

2．求下列表达式的值。

（1）已知 x=20、y=60、z=50.0，求 x+(int)y/2*z%10 的值。

（2）已知 x=256，求 x/100+x%100/10+x%10 的值。

（3）已知 x=100、y=3.14、z=55，求(byte)x+(int)y+(float)z 的值。

（4）设 int x=100，y=50，执行语句 x%=x++/--y 后求 x 的值。

（5）设 int a=70，b=60，c=50，求(a+b)>(c*c)&&b==c||c>b 的值。

3．利用下表根据销售额计算销售提成。从键盘输入销售额并在控制台显示结果。

销售额	提成
1 元～5000 元	8%
5001 元～10000 元	10%
10001 元以上	12%

4．试编写一个程序，输入 3 条边的长度值，并判断这 3 条边的长度是否能构成三角形，如果能，则给出所构成三角形的形状（一般、等边、等腰）。

【探索】

1．编写 Java Application，要求输出一个如下的菱形。

```
    *
   * * *
  * * * *
   * * *
    *
```

2．编写一个主类 Triangle，要求在它的 main 方法中写一个嵌套的 for 循环，通过这个嵌套的循环在屏幕上打印下列图案。

```
        1
      1 2 1
    1 2 3 2 1
  1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
1 2 3 4 5 6 5 4 3 2 1
```