第 1 章　Java 语言基础

自 1995 年发布以来，Java 语言一直是软件开发的主流编程语言，在 TIOBE 编程语言排行榜中一直位居前列；同时，由于 Java 语言纯粹的面向对象特性，也是学习面向对象程序设计的首选教学用语言。本章首先简述 Java 语言背景，然后从一个 Java 程序实例出发，重点介绍 Java 语言基础内容，主要包括标识符、关键字、基本数据类型、运算符与表达式、流程控制等内容。

Java 语言特点

1.1　Java 语言简介

依据 Java 语言官方的描述，Java 语言共有十个特点，分别为：简单性、面向对象、分布性、编译和解释性、稳健性、安全性、可移植性、高性能、多线索性、动态性。其实，Java 语言最大的优点就是具有与平台无关性，在一个平台上编写软件，然后即可在几乎所有其他平台上运行。

这里对 Java 起源与发展、Java 技术主要平台，以及如何搭建 Java 语言开发环境进行简单介绍。

Java 语言简介

1.1.1　Java 发展

Java 是美国 Sun Microsystems 公司所开发的一种面向对象程序设计语言。1991 年，Sun 公司成立了一个称为 Green 的项目，开发消费性电子产品的控制软件。由于当时所使用的 C++程序语言过于复杂且缺乏安全性，James Gosling 领导的团队以 C++为基础，重新开发了一套新的程序语言，命名为 Oak。1993 年，Sun 将 Oak 作为独立产品注册商标，但未能通过商标名称测试，于是将 Oak 取名为 Java。1994 年，Green 项目小组将开发转向了 Internet，用 Java 编写了支持跨平台交互式图形界面的浏览器 HotJava，并于 1995 年 5 月 23 日与 Java 语言开发工具一起发布，在产业界引起了巨大的轰动，从此 Java 迅速成长为重要的网络编程语言。1996 年，Sun 公司成立 Javasoft 分公司，并正式发表 Java 开发者版本 JDK 1.0。

1998 年 JDK 1.2 版本发布，Java 2 平台诞生。Java 开始向企业应用、桌面应用和移动设备应用 3 大领域挺进。2000 年发布 JDK 1.3，2002 年发布 JDK 1.4。2004 年 Sun 公司将版本号 1.5 改为 5.0，JDK 5.0 引入了泛型、自动装箱、枚举等现代编程特性。2006 年发布 Java 6。

2010 年 Sun 公司被 Oracle 公司收购。Oracle 在 2011 年、2014 年和 2017 年，分别发布了 Java 7、Java 8 和 Java 9。之后，Oracle 公司修订了 Java SE 发布模型，每 6 个月发布一次新的版本；每 3 个月发布一次更新；每 3 年发布一次长期支持（long-term support，LTS）版本，Java 17 之后 LTS 版本发布时间又修改为 2 年。于是，2018 年 3 月、9 月发布了 Java SE 10、11，之后每年 3 月和 9 月都会发布新的版本。截至 2025 年 9 月，Java SE 最新版本为 Java 25，最新的 LTS 版本为 Java 25。而 Java EE 和 Java ME 分别发展为 Jakarta EE 和继续服务于嵌入式

设备的 Java ME。

OpenJDK 是 Java SE 平台的开源实现，通常是免费且由社区驱动的。与 Oracle JDK 的主要区别在于许可、成本和一些高级工具的可用性。国产 OpenJDK 如华为的毕昇 JDK、腾讯的 Kona、阿里巴巴的 Dragonwell 等，实现了针对特定的性能和安全性需求进行优化和定制，推动了国内 Java 生态系统的建设，体现了国内 IT 企业的创新能力，也为全球 Java 社区做出了贡献。

对于 Java 开发人员来说，建议使用 JDK 8 及以上的 LTS 版本或者对应的 OpenJDK 版本。

1.1.2 Java 平台

Java 工作原理

平台是程序运行的硬件或软件环境。Java 平台是一个运行在某种硬件平台之上的软件平台，包括两个部分：Java 虚拟机（Java virtual machine，JVM）和 Java 应用编程接口（application programming interface，API）。其中，Java 虚拟机可以移植到各种硬件平台上，屏蔽具体底层硬件差异，是 Java 平台的基础；API 是相关类和接口库的软件组件的集合。API 和 Java 虚拟机将程序与底层硬件隔离，实现了 Java 程序的“一次编写，到处运行”目标。Java 平台的示意图如图 1-1 所示。

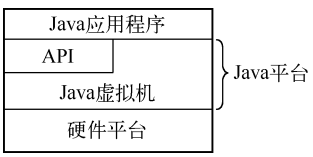

图 1-1 Java 平台位于硬件和应用程序之间

Java 平台根据不同的应用场景细分为三个主要版本：Java SE（Java Platform, Standard Edition）主要服务于桌面应用的开发；Java EE（Java Platform, Enterprise Edition）为企业级应用支持；以及 Java ME（Java Platform, Micro Edition）专为移动设备和嵌入式系统提供轻量级的解决方案。依据实际应用场景不同，Oracle 将 Java 应用平台具体分为 Java SE、Java EE 和 Java Embedded。

Java 平台

Java SE 用于开发和部署台式机和服务器上的 Java 应用程序。Java SE 和对应组件技术能提供应用程序所需的丰富的用户界面、性能、多功能性、可移植性和安全性。

Java EE 为开发和运行大型、多层、可靠和安全的企业应用程序提供 API 和运行时环境，这些应用程序具有可移植性和可扩展性，并且可以轻松地与原来的应用程序和数据集成。2017 年，Java EE 交给开源组织 Eclipse Foundation，更名为 Jakarta EE。

Java Embedded 包括 Java ME Embedded、Java SE Embedded 和 Java Card 三个面向嵌入式开发的方面。Java ME Embedded 是为资源受限的设备而设计的，如用于 M2M 无线模块、工业控制、智能电网基础设施、环境传感器和跟踪等。Java SE Embedded 为基于网络的设备提供安全、优化的运行时环境。Java Card 则是为运行在智能卡和其他内存及处理能力非常有限的设备上的应用程序提供了一个安全的环境。

具体应用平台的描述参见官方文档：https://docs.oracle.com/en/java/index.html。

1.1.3 Java 开发环境

Java 开发环境

Java 开发环境指的是 JDK（Java SE Development Kit，Java 标准开发包），提供了编译、运行 Java 程序所需的各种工具和资源，包括 Java 编译器等工具、

Java 运行时环境（Java runtime environment，JRE）以及常用的 Java 类库等。Java 虚拟机（JVM）是 JRE 的一部分，负责解释及执行 Java 程序，是运行 Java 字节码文件的虚拟计算机。

下面以 Windows 平台下 JDK 的安装和设置为例描述 Java 开发环境的安装和配置，基本步骤包括：下载、安装、配置和测试开发环境。

1．下载

下载软件安装包，网址为：https://www.oracle.com/java/technologies/javase-downloads.html。

选择对应版本的链接进行下载。例如，选择 Windows x64 Installer 对应的可执行程序下载，典型文件名为：jdk-*version*_windows-x64_bin.exe。

2．安装

以管理员身份运行安装程序，按照提示信息完成安装。

3．配置

为了便于使用 JDK 中的相关工具（如编译工具 javac 等），需要设置系统的 PATH 环境变量。方法是在系统“控制面板”中选择“高级系统设置”，单击“环境变量”按钮，在弹出的“环境变量”对话框中双击“系统变量”下的 PATH，将 JDK 安装目录下的 bin 目录加入 到系统的 PATH 环境变量中。例如，如果 JDK 安装目录为 D:\JDK21\，则 PATH 环境变量设置如图 1-2 所示。

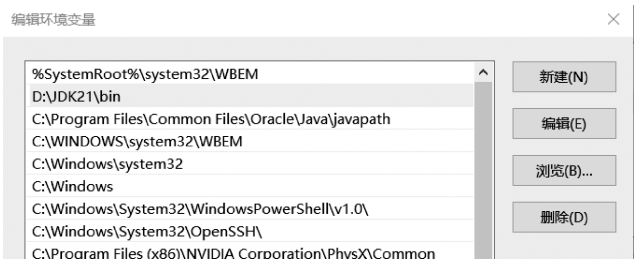

图 1-2　配置 PATH 环境变量，包含 JDK 安装位置的 bin 目录

4．测试开发环境

在命令行窗口中，输入 java -version 可以查看当前有效 JDK 版本号；输入 javac 会显示使用指南，表明 JDK 安装成功。典型运行结果如图 1-3 所示。

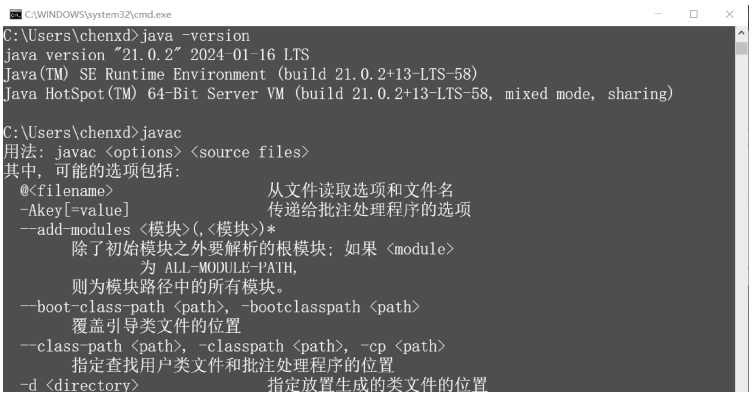

图 1-3　测试 JDK 安装和配置是否成功

其他操作系统平台如 Linux、MacOS 的 JDK 的安装请参考 https://docs.oracle.com/en/java/javase/ 对应的链接。

编写 Java 程序

1.2 简单的 Java 程序

Java 程序是由 JVM 中的 Java 解释器来解释运行的字节码（bytecode）程序。编写 Java 程序的步骤如下：

① 编辑 Java 源文件，扩展名为.java；

② 编译 Java 源文件，生成字节码指令集的二进制文件，扩展名为.class；

③ 由 Java 解释器运行字节码程序。

下面以最简单的输出“Hello, World!”为例，说明 Java 应用程序的建立及运行过程。

1.2.1 编辑 Java 源文件

Java 程序的源代码是由符合 Java 语言规范的语句构成的文本文件，文件的扩展名必须为 .java。可以使用任何文本编辑器来编写源文件，当然也可以用集成开发环境编写。

例 1-1 程序的功能是在屏幕上显示“Hello, World!”信息。在文本编辑器（如记事本）中完成文本编辑后，将其保存为 HelloWorld.java。需要注意的是，如果文件中定义的类为 public 类，保存的文件名就必须和其类名 HelloWorld 一致，也就只能是 HelloWorld.java。

例 1-1 第一个 Java 程序。

文件名：HelloWorld.java

```
/*  本程序功能是在屏幕上输出"Hello, World!" */
public class HelloWorld {
        /**
          * main()方法是任何 Java 应用程序运行的入口
          * @param args  为输入参数，  main()方法参数是用于接收命令行参数
          */
        public static void main(String[] args) {
                //输出"Hello, World!"
                System.out.println("Hello, World!");
        }
}
```

Java 语言是大小写敏感的编程语言，在编写代码时要严格区分大小写；空白符可以是空格、Tab 或者回车符。

1.2.2 编译源程序

在 Windows 中按下 Win+R 组合键可以打开“运行”对话框，输入 cmd 命令可以进入系统的命令行窗口。

在命令行窗口进入待编译文件所在目录之后，输入如下命令编译源代码：

```
E:\JavaBook\1> javac HelloWorld.java
```

如果命令行窗口没有出现任何提示信息，则说明 Java 源程序已经编译成功，当前目录下

会产生一个扩展名为.class 的字节码文件，可以在 Java 虚拟机内运行。

1.2.3　运行 Java 应用程序

在命令提示符下输入如下命令，并按回车键。

```
E:\JavaBook\1>java HelloWorld
```

屏幕上则显示程序运行结果，输出以下信息：

```
Hello, World!
```

其中，命令 java 是 JDK 中所提供的 Java 解释器，其作用是调用 Java 虚拟机加载、解释和执行.class 文件。

图 1-4 为编译和运行 Java 程序 HelloWorld.java 的操作过程示意图。

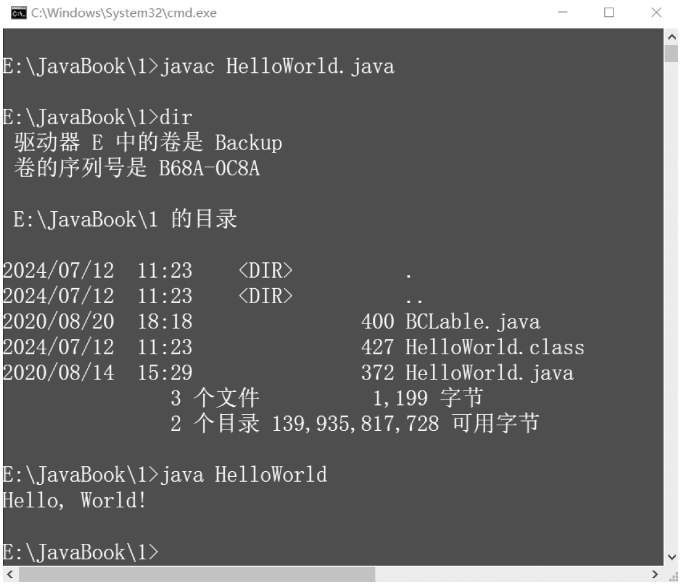

图 1-4　编译和运行 Java 程序

1.2.4　程序分析

通过例 1-1 可以看出，Java 程序的基本结构为：

```
public class 类名{
        public static void main ( String [ ] args ){//main()方法
                … //语句
        }
}
```

所有的 Java 程序必须定义在类中才可以编译执行。Java 语言使用 class 关键词来定义一个类。在类里面可以定义各种方法，方法由方法头和方法体构成。方法头说明方法的访问约束、返回类型、方法名及参数列表；方法体则由一对大括号{ }中放置若干语句来构成，完成具体的任务。需要强调的是，main()方法是 Java 程序运行的入口。

在下面的分析中，为了叙述方便，为 Hello World 程序的源代码增加上行编号：

```
1.    /*  本程序功能是在屏幕上输出"Hello, World!" */
2.  public class HelloWorld {
3.          /**
4.           * main()方法是任何 Java 应用程序运行的入口
5.           * @param args  为输入参数，  main()方法参数是用于接收命令行参数
6.           */
7.          public static void main(String[] args) {
8.                  //输出"Hello, World!"
9.                  System.out.println("Hello, World!");
10.         }
11.  }
```

编写 Java 程序时应该遵循如下的 Java 语言编程规范。

1．注释

为了方便软件以后的维护和升级，通常要在源代码中加入注释，帮助自己或者他人能更好地理解和维护源代码。Java 语言提供了以下 3 种注释方法。

（1）行注释，也称单行注释，以“//”开始。表示从这个符号开始到这行结束的所有内容都是注释。如 HelloWorld.java 程序代码中第 8 行：//输出“Hello, World!”。

（2）段落注释，也称多行注释，以“/*”开始，“*/”结束。使用方式为：/*注释文字*/。一般当注释的内容比较长需要换行时，使用段落注释方法，当然也可以用于单行内容的注释。如程序代码中的第 1 行。

（3）文档注释，也称 Javadoc 注释，以“/**”开始，“*/”结束。如程序代码中的第 3 行到第 6 行的内容就是文档注释。采用这种方法的注释，可以使用 javadoc 命令从 Java 源文件里自动生成 Javadoc 文档，生成组织有序的帮助文档。文档注释用于说明紧接其后的类、属性或者方法等。文档注释的一般形式如下：

```
/**
*  注释文字 1
*  注释文字 2
…
*/
```

在这 3 种注释方法中，前两种与 C 和 C++的注释方法一样，第 3 种文档注释则是 Java 语言专门引入的一种注释方式。

2．声明类

第 2 行是类的定义。关键字 class 用于定义了一个类，class 后面的 HelloWolrd 是类的名字，也就是本例中定义了一个名为 HelloWorld 的类。public 关键字表示这个类的访问约束是公共的，也就是其他任何类都可以使用该类。

整个类定义由第 2 行的左大括号“{”及第 11 行的右大括号“}”括起来。括号的内部是类体。类体中可以声明类的属性和方法等内容，它们也称为类的成员。

在 Java 中，程序都是以类的方式组织的。一个 Java 源文件可以定义多个类，但仅允许有一个公共的类，源程序的文件名要与公共类的名称相同（包括大小写），其扩展名为.java。因此，HelloWolrd 程序的源程序文件名必须是 HelloWorld.java。

3．main( )方法

只有含有 main()方法的 Java 程序才能执行。与 C 和 C++语言中的 main 函数类似，main()方法是执行程序的入口点。

所有 Java 应用程序必须含有一个 main()方法。public 关键字表示这个方法是公共的，即可以从其他类中调用它；static 关键字表示这个方法是静态的，表明不存在类的实例（也称对象）就可以运行；关键字 void 表示 main()方法无须返回任何值。

main()方法后的小括号中的 String [ ] args 是 main()方法的参数列表，它可以接收从外部环境向方法中传递的参数，对于 main()方法实际就是命令行参数。

第 7 行最后到第 10 行的一对大括号中的内容为方法体。方法体由语句构成，完成具体任务，分号是语句结束的标识。本例方法体中为调用语句，调用 System.out 对象的 println()方法，输出“Hello World!”。

除 main()方法外，依据需要，一个 Java 类可以定义很多其他方法。

4．输出语句

第 9 行是 main()方法唯一的一个语句，其作用是在标准输出设备即屏幕上输出一行字符“Hello, World!”。字符串常量需要使用双引号。

Java 语言使用模块（module）和包（package）的方式提供了实现诸多功能的标准类库。在 Java 程序中使用类库中的类，需要先使用 import 语句导入相应的类。import 语句必须位于类定义之前，按需要可以多次使用以导入多个类。由于 java.lang 包（位于 java.base 模块中）是 Java 最基本的类库，其中所有类会由 Java 环境自动导入，本例使用的 System 类就位于 java.lang 包中，因此本例没有使用 import 语句。

5．分隔符

在程序设计语言中，分隔符是用来分隔不同字符串的标记字符，以便编译器能够确认源代码在何处分隔。Java 语言的分隔符除了注释分隔符还有空白分隔符和普通分隔符。

（1）空白分隔符。Java 是一种形式自由的语言。在源代码元素之间可插入若干空白分隔符以改善源代码的可读性。在 Java 中，空白分隔符可以是空格、Tab、回车符或换行符。

（2）普通分隔符。Java 语言中的普通分隔符包括：( ) { } [ ] ; , ... @ :: 等。实际是语言中有特定含义的符号。下面简单介绍常用的几种分隔符。

- 圆括号“()”：在定义和调用方法时用来容纳参数表；用于条件和循环语句；用于强制类型转换；用于表达式中改变计算的优先顺序等。
- 大括号“{}”：用来定义语句块、类体、方法体以及初始化数组的值。
- 方括号“[]”：用来声明数组的类型，访问数组元素。
- 分号“;”：用于语句的结束。在 for 循环语句中可用于分隔初始表达式、条件表达式和迭代表达式，如，for(int i=0; i<10;i++){}。
- 逗号“,”：用于分隔变量表中的各个变量和方法说明中的各个参数；也可作为顺序运算符使用，连接多个表达式。
- 句号“.”：用来将软件包的名字与它的子包或类分隔；也用于对象引用变量访问自己的成员，例如，System.out.println()。

6．代码编排

为了使源程序更加容易阅读，在编写 Java 程序时，一般会使用缩排和留白的方式来编排

代码，反映出程序代码的逻辑和结构。Java 源程序在编译时，编译器会忽略所有空白。

综上所述，例 1-1 虽然很短，但包含了 Java 应用程序应具备的几个关键特性，以此为基础，就可以编写简单的 Java 应用程序了。

1.3 关键字与标识符

关键字与标识符

任何一种高级程序设计语言，都会包含语言必需的基本元素，如关键字（keywords）、标识符（identifier）、数据类型、运算符与表达式以及基本流程控制语句等内容。

关键字是语言自用的具备特定语法含义的特殊标识符，不允许用于其他用途。标识符用于为变量、参数、常量、方法、类型（类、接口、枚举、注解等）命名。

1.3.1 关键字

关键字，也称为保留字，是指程序设计语言中事先定义的、有特定语法含义的标识符，不允许编程人员用作其他用途。

Java 语言总共有 51 个关键字，见表 1-1。

表 1-1 Java 语言的关键字

关键字类型	关 键 字
基本数据类型（8 个）	int double long byte short float char boolean
流程控制（11 个）	for do while continue break if else switch case default return
异常处理（6 个）	throw throws try catch finally assert
类型定义（5 个）	class extends implements interface enum
修饰符和访问约束（8 个）	public private protected abstract static final transient native
其他（13 个）	new void this super import package synchronized volatile strictfp instanceof（运算符） goto（未用）const（未用） _（下划线）

Java 中有三个字面常量值可以看成是关键词，分别是 true、false 及 null。

Java 中还有 17 个上下文关键词（contextual keyword），分别是 exports，opens，requires，uses，yield，module，permits，sealed，var，non-sealed，provides，to，when，open， record，transitive 和 with，这些单词仅在语法适当上下文（如 Java 模块等相关功能）中使用时是关键字。

需要说明的是，friendly，then 和 sizeof 等都不是 Java 语言的关键字，goto 和 const 虽然是关键字，但是在 Java 语言中却并未使用。

1.3.2 标识符

标识符是指程序中使用的各种数据对象如类型、变量、参数、常量、方法等的名称的有效字符序列。简单地说，标识符就是一个名字。

Java 语言标识符遵循以下规则：

① 标识符必须以 Unicode 字符、数字（0～9）、下划线（_）和美元符（$）组成。

② 标识符的首字符不能为数字。

例如，下面声明的变量名都是符合 Java 规定的有效标识符：

```
int MAX_VALUE, isLetterOrDigit, i3, $1x, _1x;
int 北交大, κυκλοφορία, 교통, こうつう; //Unicode 字符
```

Java 标识符在实际使用中，还应注意：

① 大小写敏感：myID、MyID、MyId、myid 是 4 个不同的标识符。

② 长度限制：标识符的长度没有限制，但是不宜过长。

③ 避免使用关键字：自定义标识符不能使用关键字。

④ 不以$或_开始和结束：例如$在 Java 中用于匿名内部类名，由编译器自动生成。

⑤ 有意义的命名：建议使用英文单词或者词组，尽可能做到“见名知意”。

⑥ 命名约定：变量名、方法名用小写字母开头；类型名用大写字母开头。由多个单词构成的标识符采用驼峰式命名法，变量和方法使用小驼峰式命名法（lowerCamelCase，即第一个单词的首字母小写，后续单词的首字母大写）命名，类型使用大驼峰式命名法（UpperCamelCase，即所有单词的首字母都大写）命名。

这些人为制定的规则是否遵守并不会影响 Java 编译器的工作，但是养成良好的标识符定义习惯可以使程序易于理解和维护。

1.4　数据类型


数据类型

Java 语言的数据类型分为基本数据类型（primitive type）和引用类型（reference type）两大类。

Java 语言基本数据类型总共有 8 种，可以分为 4 大类：整数类型、浮点类型、字符类型和布尔类型。其中，整数类型包含 byte，short，int 和 long 4 种；浮点类型包括 float 和 double 两种；字符类型为 char；布尔类型为 boolean。

8 种基本数据类型的简单说明见表 1-2。

表 1-2　Java 语言基本数据类型

整数类型	byte	最小的数据类型，在内存中占 8 位，取值范围-128～127，默认值 0
	short	短整型，在内存中占 16 位，取值范围-32 768～32 767，默认值 0
	int	整型，用于存储整数，在内存中占 32 位，取值范围-2 147 483 648～2 147 483 647，默认值 0
	long	长整型，在内存中占 64 位，取值范围 -2^{63}～$2^{63}-1$，−9 223 372 036 854 775 808～9 223 372 036 854 775 807，默认值 0L
浮点类型	float	浮点型，在内存中占 32 位，有效小数点只有 6～7 位，默认值 0
	double	双精度浮点型，在内存中占 64 位，有效小数点 16 位，默认值 0
字符类型	char	字符型，用于存储单个字符表示，采用 UTF-16 编码格式，占 16 位 2 个字节，取值范围为'\u0000' to '\uffff'，即整数值 0～65 535，默认值为'\0'(或 0)
布尔类型	boolean	布尔类型，占 1 个字节，用于判断真或假（仅有两个值，即 true、false），默认值 false

除了基本数据类型外的所有其他类型都是引用类型，如类、接口、数组、枚举、注解等类型。另外，null 类型也可以看成是一种特殊的引用类型，可以将 null 类型强制转换为任何其他引用类型。

本节先介绍 Java 语言的基本数据类型。

1.4.1 整数类型

整型数据是最普通的数据类型，定义一个整数。Java 有 4 种整数类型，byte、short、int 和 long，分别表示 8 位、16 位、32 位和 64 位有符号的整数值，可表示数据的范围从小到大，在程序设计中应该选择最合适的类型来定义整数。

Java 中整数字面常量值（literal）表现方式有：二进制、八进制、十进制和十六进制。默认类型为 int。对于 long 类型的常量字面值，需要使用 L 后缀来表示。十六进制整数必须以 0X 或 0x 作为开头，八进制整数以数字 0 开头，二进制以 0B 或 0b 开头。对于字面值由比较长的数字串构成的整数，为了便于理解，可以用下划线（_）作为分隔符。

整数字面值是赋值兼容的，也就是说，尽管字面值默认为 int，但是可以同时赋值给 byte、short、int 和 long。如下为整数字面值使用的示例：

```
byte b = 022;       //八进制值
short s = 277;      //十进制值
int bytes = 0b11010010_01101001_10010100_10010010; //二进制值
long creditCardNumber =1234_5678_9012_3456L; //十进制 long 类型数值
long hexWords = 0xCAFE_BABE;   //十六进制值
```

1.4.2 浮点类型

浮点类型也称为实数类型。在 Java 语言中有 float 和 double 两种浮点类型。其中 float 是单精度型，double 是双精度型，分别表示 32 位和 64 位的 IEEE 754 浮点数。

浮点数字面常量可以用一个带小数的十进制数表示，如 1.35 或 23.6；也可以用科学记数法表示，如 1.35E2，其中 E 或 e 之前必须为一个有效数字，其后必须为整数。

浮点数字面常量默认为 double 类型，不是赋值兼容，因此对于 float 类型变量，赋值的字面常量需要在其后加“F”或“f”后缀，否则会出现编译错误。对于 double 型的值则不需要加后缀，当然也可以加“D”或“d”后缀。以下为浮点数常量使用的示例：

```
float pi = 3.14_15F;
double w = 3.1415;
double d = +306D;
```

1.4.3 字符类型

在 Java 语言中，字符类型 char 表示的是 Unicode 字符，一个 char 类型数据实际是一个 16 位无符号整数所表示的 UTF-16 编码单元，从“\u0000”到“\uffff”，对应从 0～65 535 整数值，并且支持整数运算，所以 char 类型也可以归类到整数类型中。

字符字面常量是由一对单引号括起来的单个字符。大多数字符字面值都可以用单引号方式直接表示，例如：'A' '京' '@'等。但是，对于一些特殊字符，则需要使用转义序列来表示，以反斜杠（ \ ）开头来转变字符所代表的意义，给其指定新的含义，如 '\n' 表示换行。

表 1-3 列出了 Java 中的常用的转义符。

表 1-3　常用的转义字符及含义

转 义 字 符	含　义
\'	单引号
\"	双引号
\\	反斜杠
\n	换行
\r	回车
\t	Tab 制表符
\f	换页
\b	退格
\ddd	3 位八进制数据所表示的字符
\udddd	4 位十六进制 Unicode 编码所表示的字符

相比较于字符类型数据，字符串是由若干字符构成的连续字符数据，字面常量值为一对双引号括起来字符序列，其中也可以包括转义字符，如"Good Morning !! \n"。Java 中，字符串是由 String 类所实现的，是引用类型而不是基本数据类型。

1.4.4　布尔类型

布尔类型也称为逻辑数据类型，在 Java 语言中用 boolean 来表示。布尔类型是最简单的一种数据类型，布尔数据只有真和假两种状态，分别用字面值 true 和 false 来表示。与 C 语言不同，Java 中的布尔型数据不对应于任何整数值，不能使用整数来对布尔变量赋值。下面的语句对变量 truth 初始化为 true：

```
boolean truth = true;
```

1.5　变量声明与赋值

变量声明与赋值

与字面常量不同，变量是指程序运行期间其值可以改变的量。Java 是一种强类型语言，变量必须要先声明后才能使用。

Java 中变量声明语句的语法为：

```
数据类型  变量名  [=变量初始值][,变量名[=变量初始值]...];
```

数据类型表示该变量所存储的数据的类型，它可以是 Java 语言中的任意一种数据类型，包括用户自定义的类型；变量名是一个合法的标识符；方括号表示内容可选，此处方括号内的内容为对变量初始化操作；相同数据类型可以同时声明多个变量，以逗号作为分隔。

例如，以下语句声明了一个双精度浮型变量 pi，并进行了初始化，其初始值为 3.1415926。

```
double pi=3.1415926;
```

变量在使用（如参与表达式运算）前，必须有确定的值。变量的值可以通过两种方法获得：一种是声明变量的同时进行初始化操作；另一种就是在使用赋值语句赋值。

变量通过赋值运算来修改其值，赋值运算的基本语法为：

```
变量名 = 表达式;
```

变量的赋值操作要求变量的类型和表达式类型要一致，如果不一致则需要进行类型转换。

从 Java 11 开始，Java 语言支持局部变量的类型推定。可以使用 var 来声明变量，由编译器根据上下文自动推定变量的实际类型。例如，下面语句中，声明的变量 i 初始化为 2025（默认类型为 int），i 会被自动推定为 int 类型；变量 s 由于初始化为字符串，s 会被自动推定为字符串（String）类型：

```
var i = 2025;
var s = "北京交通大学";
```

等同于：

```
int i = 2020;
String s = "北京交通大学";
```

1.6 运算符与表达式


运算符与表达式

Java 运算符主要分为 4 类：算术运算符、关系运算符、布尔运算符和位运算符。由运算符和操作数构成表达式。表达式中运算符参与运算时遵循一定的优先级与结合性。当参与运算的操作数类型不同时，需要进行类型转换。

1.6.1 算术运算

算术运算符用来完成整型数据和浮点型数据的算术运算。Java 的算术运算符分为一元运算符和二元运算符。一元运算符只有一个操作数，而二元运算符有两个操作数参加运算。

1．一元运算符

一元运算符有：+/−（相当于符号位），++（自增）和−−（自减）。

自增、自减运算符既可放在操作数之前（如++i），也可放在操作数之后（如 i++），两者的运算方式不同。对++i 来言，操作数先加 1，表达式结果为加 1 后的值；对 i++，表达式结果为操作数未增加的值先参加其他的运算，然后再对操作数进行自增 1 运算。自减运算的过程类似。

自增、自减运算符不能用于表达式，只能用于简单变量。

2．二元运算符

二元运算符有：加（+）、减（−）；乘（*）、除（/）和取余（%）。

其中，+、−、*、/完成加、减、乘、除四则运算，%则是求两个操作数相除的余数。这五种运算符均适用于整型数据和浮点型数据。当在不同数据类型的操作数之间进行算术运算时，表达式结果的类型与表示范围最高的操作数类型一致。

对于运算符%，运算结果的符号与被除数相同。例如，5%(−3) 的结果为 2，(−5)%3 的结果为−2，(−5)%(−3)的结果为−2。

1.6.2 关系运算

关系运算符也称为比较运算符（comparison operators），用于比较两个值之间的大小，结果返回为布尔值。

关系运算符有：等于（==）、不等于（！=）；大于（>）、大于等于（>=）、小于（<）和

小于等于（<=）。例如，3<=2 的结果为 false。

1.6.3　布尔运算

布尔运算符是对布尔值进行运算，参与运算的操作数必须为布尔类型（boolean 或者 Boolean），运算的结果为 ture 或 false。

Java 语言的布尔运算符见表 1-4。

表 1-4　Java 语言的布尔运算符

运　算　符	含　　义
!	逻辑非，取反
&	逻辑与，两个操作数值都为 true 时返回 true，否则返回 false
^	逻辑异或，两个操作数值相同时返回 false，不同时返回 true
|	逻辑或，两个操作数值都为 false 时为 false，否则为 true
&&	条件与，含义与 & 同，但只有当左操作数为 true 时才计算右操作数
||	条件或，含义与 | 同，但只有当左操作数为 false 时才计算右操作数

例如，要判断 n 是否为一个三位数的布尔表达式是（n>=100）&&（n<=999）。在进行&&和||运算时，有“短路”现象，只要当左操作数的值可以用于确切判断整个表达式的真伪时，计算过程便直接结束，右操作数不再计算。例如，要计算表达式（10<5）&&（5>1）；当计算（10<5）后便可以知道整个表达式值是 false，所以无须对（5>1）进行求值操作。

1.6.4　位运算

位运算符只能应用于整数类型（包括 char 类型），不支持浮点数据类型。位运算是对数据进行按位操作，用于对整数中的位进行测试、置位或移位处理。Java 语言的位运算符见表 1-5。

表 1-5　Java 语言的位运算符

运　算　符	含　　义
&	按位进行与运算。可用于检查某个二进制位是否为 1，如 a & 1 如果不是 0，则最低位必为 1
~	按位进行取反运算，即把 1 变为 0，把 0 变为 1
|	按位进行或运算。可用于对某个二进制位置 1
^	按位进行异或运算。异或运算两个操作数对应二进制位相同则结果值为 0，不同则为 1
<<	按位左移。运算符左侧对象左移由右侧指定的位数，低位补 0，最高位抛弃。在未溢出的情况下，左移位运算相当于对左操作数进行乘 2 运算
>>	按位右移，也称算术右移。右移过程中保持符号位不变，也就是说，若值为正，高位补 0，若值为负，高位补 1。算术右移相当于对左边操作数进行除 2 运算
>>>	按位进行无符号右移，也称逻辑右移。无论运算符左边的运算对象值正负，都在高位补 0

例如，3<<2 操作数 1 的值 3 左移 2 位，二进制运算为 00000011 左移 2 位，结果为 00001100，相当于 3 乘以 2 的 2 次方。

区别&、|、^ 是布尔运算还是位运算，只需要判定操作数类型即可，如果操作数都是布尔类型，则为布尔运算；如果都是整数类型，则为位运算。

1.6.5 其他运算

除了上述的 4 类基本运算外，Java 提供了若干运算符来支持其他运算，包括 ?:（条件运算）、复合赋值运算、instanceof（类型检查）、new（创建对象或数组）、.（成员访问）、[]（数组索引访问）、..（表示范围）、->（用于 Lambda 表达式）等。

1．条件运算 ?:

条件运算符是 Java 语言中唯一的三元运算符，基本语法格式为：

<条件> ? <表达式 1> : <表达式 2>

其中，<条件>必须为布尔表达式。当<条件>为 true，运算返回<表达式 1>的值；当<条件>为 false，运算返回<表达式 2>的值。

2．复合赋值运算

除了基本的赋值运算之外，Java 语言还支持复合赋值运算。复合赋值运算就是一种将赋值运算符与其他算术运算符组合在一起使用的一种简捷使用方式。Java 语言的复合赋值运算符及使用方式见表 1-6。

表 1-6 Java 语言的复合赋值运算符

运 算 符	用 法	等 价 于
+=	s += i	s = s+i
-=	s -= i	s = s-i
*=	s *= i	s = s*i
/=	s /= i	s = s/i
%=	s %= i	s = s%i
&=	a &= b	a = a&b
|=	a |= b	a = a|b
^=	a ^= b	a = a^b
<<=	s <<= i	s = s<<i
>>=	s >>= i	s = s>>i
>>>=	s >>>= i	s = s>>>i

3．instanceof 运算符

instanceof 运算符用来测试一个指定对象是否为指定类型的对象实例，若是则返回 true，否则返回 false。例如 e instanceof Employee，如果变量 e 引用的对象类型是 Employee，则返回 true。

4．[ ] 和 ()

方括号 [] 是数组元素访问运算符，方括号中的数值是数组的下标，整个表达式代表数组中该下标所在位置的元素。

括号（）用于改变表达式中运算符的优先级。

5．点运算符

点运算符“.”主要用于访问类中成员。

1.6.6　运算符的优先级与结合性

Java 语言主要的运算符优先级及结合性见表 1-7。

表 1-7　运算符优先级与结合性

优　先　级	运　算　符	结　合　性
1	() [] .	从左到右
2	! +(正) -(负) ~ ++ --	从右向左
3	* / %	从左向右
4	+(加) -(减)	从左向右
5	<< >> >>>	从左向右
6	< <= > >= instanceof	从左向右
7	== !=	从左向右
8	&	从左向右
9	^	从左向右
10	|	从左向右
11	&&	从左向右
12	||	从左向右
13	? :	从右向左
14	= += -= *= /= %= &= |= ^= ~= <<= >>= >>>=	从右向左

表 1-7 中的优先级按照从高到低的顺序组织，即优先级 1 的优先级最高，优先级 14 的优先级最低。一元运算符、赋值（含复合赋值）运算符及条件运算符为右结合，其他都是左结合。

1.6.7　类型转换

Java 程序中的每一个数据（包括变量、常量、表达式、方法返回数据等）都必须有且只有一个数据类型。当不同类型的数据进行运算（包括赋值运算）操作时，必须先进行类型转换，数据类型一致才能运算。

Java 中的数据类型转换分为自动类型转换和强制类型转换两种情况。

1．自动类型转换

Java 语言支持不同类型数据的混合运算。不同类型的数据会先自动转化为同一类型，然后再进行运算。基本数据类型的转换规则是将参与运算的操作数自动转化为运算表达式中表示范围更大的操作数类型的数据类型，因此，自动类型转换也称为类型自动提升。对于引用类型，子类的实例可以自动转换为父类类型，这种转换也称为向上转型（upcasting）。

Java 语言中的基本数据类型表示范围从“小”到“大”的顺序为：byte→short→char→int→long→float→double。

例如，表达式 12 + 62d + 40f 运算的结果是 double 类型。

不同类型的操作数进行赋值运算，如果下列两个条件都能满足，将执行自动类型转换。

① 这两种类型是兼容的。数字类型彼此兼容；数字类型和字符类型或布尔类型不兼容；

字符类型和布尔类型也互不兼容。

② 目标类型数据（左值）比源类型数据（右值）所占内存空间大。

例如，两个变量分别为 byte 类型和 short 类型，在进行数学运算 + 时，会进行自动类型转换，都会自动转换为 int，因此运算的结果类型为 int 而不是 short，再赋值给 short 类型变量时，就会出现“不兼容的类型”的编译错误。

```
byte numOranges = 123;
short numApples = 5;
short num = numOranges + numFruit;//编译错，因为运算后表达式类型为 int
```

2．强制类型转换

为了实现两种数据类型兼容，但表示范围由“大”到“小”的转换，必须进行强制类型转换。强制类型转换的语法为：

```
(目标类型)表达式;
```

在强制转换中，数据不可以超出转换后类型的范围，否则可能导致溢出或损失。当把 int 型的值赋给 byte 型变量时，如果整数的值超出 byte 类型的取值范围，实际是截取最低字节数据赋给 byte 型变量。例如，执行下面 3 条语句的结果是 9：

```
byte b;
int i=265;
b=(byte)i;
```

当把浮点数的值强制类型转换为 int 类型时，会截去小数部分。如果浮点数的值已经超出 int 类型的取值范围，则将去掉小数点后的数值对 int 型的值域取模，然后再赋给 int 型变量。

需要注意，boolean 和整数类型不能相互转换，boolean 和浮点数类型也不能相互转换。

使用引用类型时，从父类类型转换为子类类型，如果对象实际上是子类的实例，可以使用强制类型转换。

1.6.8 表达式

表达式是由操作数和运算符按一定的语法形式组成的符号序列。每个表达式运算后都会得到一个确定的值，该值被称为表达式的值。所有表达式都会有一个确定的类型，表达式类型就是表达式运算结果的类型。

Java 规定了表达式的运算规则，对操作数类型、运算符性质、运算结果类型及运算次序都做了严格的规定，程序员使用时必须严格遵循系统的规定。

对于整数类型数据参与运算时，如果有一个或多个操作数的类型为 long，整个表达式按 64 位精度计算，其他非 long 类型的操作数自动提升到 64 位的 long 类型参与运算，表达式结果类型为 long；否则，表达式默认按 32 位精度计算，即使所有操作数的类型都不是 int，也都会被自动提升为 32 位的 int 类型再参与运算，表达式结果类型为 int。

只要有操作数为浮点数据类型，运算就成为浮点运算，参与运算的整数都会自动转换为浮点数据类型。如果有操作数类型为 double，那么运算就会按 64 位浮点数方式计算，参与运算的数都会自动转换为 double 类型，表达式结果类型也为 double；否则运算按 32 位浮点数方式计算，表达式结果类型也为 float。

1.7　枚举类型

Java 中的枚举（Enum）类型是一种特殊的类，它可以用来定义一个固定数量的常量。枚举类型在 Java 语言中使用 enum 关键字来声明。

Java 中使用 enum 关键字声明枚举类型，语法如下：

```
enum 枚举类型名称 {枚举常量 1, 枚举常量 2,…, 枚举常量 n}
```

作为常量，枚举类型中表示枚举常量通常都是大写。枚举常量之间用逗号“,”分隔。声明后的枚举类型是一个新的数据类型，枚举名称就是该数据类型的名称。

例如，下面的语句就声明了季节的枚举类型 Season 和表示颜色的枚举类型 Color：

```
enum Season { WINTER, SPRING, SUMMER, FALL } ;
enum Color { RED, WHITE, BLUE };
```

下面的语句使用枚举类型 Color 声明了一个变量 c，赋值为枚举常量值 WHITE 并输出：

```
Color c = Color.WHITE
System.out.println(c);
```

1.8　流程控制

流程控制

程序的流程控制分为顺序、分支、循环 3 种结构。

顺序控制结构是指将程序要执行的各种语句按出现的先后顺序排列起来的程序结构。顺序结构是最简单的一种基本结构。

分支结构是根据给定条件进行判定，以决定执行某个分支程序段。

循环控制结构的特点是在给定条件成立时，反复执行某段程序，直到条件不成立为止。给定的条件称为循环条件，反复执行的程序段称为循环体。

Java 语言中，有 if 语句和 switch 语句 2 种分支控制语句；有 3 种循环语句 for、while、do…while 实现循环结构；还有 break、continue、return 等流程转移语句。

Java 条件和循环语句中作为判断条件的表达式，必须为布尔类型的表达式。

1.8.1　if 语句

Java 语言的 if 语句有两种形式：基本 if 语句和嵌套 if 语句。

基本 if 语句的语法格式为：

```
if ( <布尔表达式> )
    <语句块 1>
[else
    <语句块 2>
]
```

其中，方括号内的内容是可选项。其含义是，如果<布尔表达式>的值为 true，则执行<语句块 1>；否则，如果 else 子句存在，执行<语句块 2>；结束 if 语句。If 语句执行结束后，会执行后续的其他语句。

当 if 语句中的语句块中又出现了 if 语句，可以看成是嵌套 if 语句。以下为典型的嵌套的

if 语句形式：

```
if ( <布尔表达式 1> )
    <语句块 1>
else if ( <布尔表达式 2> )
    <语句块 2>
[[...
else if ( <布尔表达式 n> )
    <语句块 n>]
else
    <语句块 n+1>]
```

Java 语言的语法规定 else 总是与它前面最近的 if 配对。

1.8.2 switch 语句

if 语句解决了依据条件从两种方案中选择其中之一的情况。在实际应用中还会经常遇到多个分支的情况，使用嵌套的 if 语句来解决会比较麻烦。Java 提供 switch 语句支持多分支流程控制。

1．基本 switch 语句

switch 语句的基本语法格式为：

```
switch（<表达式>）{
    case 常量 1:              //case 标号（Label）
            <语句块 1>;
            [break;]          //中止当前 switch 语句
    case 常量 2:
            <语句块 2>;
            [break;]

    …
    case 常量 n:
            <语句块 n>;
            [break;]

     [default: 语句块;]
}
```

其含义是，首先计算<表达式>的值，然后依次与 case 标号中的常量值作比较。如果表达式的值与某个 case 标号中的常量值相等，控制流就会转到该 case 标号之后的语句块中的第 1 条语句，执行对应语句块中的语句；如果找不到与表达式的值相等的常量值，控制流就会转到 default 标号之后的语句块中；如果没有 default，switch 语句会结束执行。

break 语句在 switch 语句中为可选语句，用于在 switch 语句的某处结束当前的 switch 语句的执行，程序流程跳转到 switch 语句之后的第 1 条语句。一般情况下，应该统一在每个 case 子句的最后，增加 break 语句，表示当某个 case 处理完成后，退出 switch 语句，之后再继续执行 switch 之后的其他语句。

在使用 switch 语句时，需注意以下几点：

① switch 语句中的表达式也称为选择表达式，类型支持整数（byte、short、int，但不支持 long）、字符（char）、枚举及字符串（String）类型，不支持布尔类型（boolean）。选择表达式也支持对应的包装数据类型：Byte, Short, Integer 和 Character。

② case 标号中只能使用常量或者字面常量，同一个常量值不能重复出现在多个 case 中。

③ case 标号之后的语句（包括 break 语句）实际上整体是一个语句块，但是可以不使用{}括起来。

④ 多个 case 标号可以共用一组执行语句。其格式为：

```
case 常量 1: case 常量 2:  … case 常量 n: 语句块;
```

⑤ default 标号最多只能有一个，一般应该放在所有 case 之后。

⑥ case 语句块中需要使用 break 语句来结束 switch 语句，不然会按语句顺序连续执行下去，直到 switch 结束。

2．switch 表达式

从 JDK 14 开始正式支持 switch 表达式，具体包括：在 switch 语句中使用箭头标号语法；case 条件中多个常量可以写在一行，用逗号分开；可以省略 break 关键字；switch 可以作为表达式直接返回一个值。具体标准参见 https://openjdk.java.net/jeps/361。

例如，下面定义的 rangeOf( )方法中，switch 语句中使用了箭头标签（arrow labels），case 条件中使用多个常量，并且省略了 break 语句：

```
static void rangeOf(int k) {
    switch (k) {
        case 1,2,3          -> System.out.println("Ranking one");
        case 4,5,6,7,8,9    -> System.out.println("Ranking two");
        default             -> System.out.println("Not Ranked");
    }
}
```

连续使用以下的方法调用：

```
rangeOf(1);
rangeOf(19);
rangeOf(8);
```

会得到以下的输出结果：

```
Ranking one
Not Ranked
Ranking two
```

switch 语句扩展成为 switch 表达式的基本语法为：

```
<目标类型> result = switch (<参数>) {
    case Label1 -> <表达式 1>;
    case Label2 -> <表达式 2>;
    …
    default -> <表达式 3>;
};
```

如果<目标类型>确定，整个 switch 表达式的类型就是目标类型；如果<目标类型>不确定，那么 switch 表达式类型就是各 case 分支的表达式计算出来的独立类型。

大多数情况下，case 标号箭头（"case Label ->"）后面是单个表达式，对于后面由多条语句构成的语句块，可以在最后使用 yield 语句来返回 switch 表达式值。例如：

```
int j = switch (s) {
    case "BJTU" -> 1;
```

```
        case "SSE"    -> 2;
        default        -> {
            System.out.println("Neither BJTU nor SSE, hmmm...");
            yield 0;
        }
};
```

yield 语句也可以用于传统的 case 标号（"case Label :"）下，返回一个值并结束 switch 表达式。例如：

```
int result = switch (s) {
    case "BJTU":
        yield 1;
    case "SSE":
        yield 2;
    default:
        System.out.println("Neither BJTU nor SSE, hmmm...");
        yield 0;
};
```

1.8.3 while 语句

while 语句的定义形式为:

```
while (<布尔表达式>)
    <语句块>;
```

while 语句是前判定型循环，首先计算<布尔表达式>的值，当值为 true 时，才执行循环体<语句块>，然后继续判定<布尔表达式>的值，直到其值为 false 才终止循环。

如果循环<布尔表达式>的值永远为 true，则循环会一直执行，不会停止，此情况常称为死循环。

1.8.4 do…while 语句

do…while 循环语句的定义形式为：

```
do
    <语句块>
while (<布尔表达式>);
```

do…while 语句是后判定型循环，先执行循环体<语句块>，再计算<布尔表达式>判定循环条件。当循环条件为 true 时，反复执行循环体，直到循环条件为 false 终止循环。

do…while 语句的循环体将至少被执行一次，而 while 语句的循环体有可能一次都不被执行。

1.8.5 for 语句

for 语句是最灵活也是最常用的循环结构的流程控制语句。

1．基本型 for 循环

基本型 for 循环的定义形式为：

```
for(<表达式 1>;<表达式 2>;<表达式 3>)
```

```
    <语句块>;
```

其中，<表达式 1>和<表达式 3>可以是任意表达式；<表达式 2>必须为布尔表达式，表示循环条件。在使用基本型 for 循环时，常把<表达式 1>用于初始化循环变量，<表达式 2>是循环的判定条件，<表达式 3>常用作循环增量表达式。三个表达式都可以被省略，但是两个分号不能省略。如果作为循环判定条件的<表达式 2>不存在，则默认条件一直为 true。

for 语句的执行过程为：

① 计算<表达式 1>，一般用于对循环变量赋初值；

② 计算<表达式 2>，若其值为 true，则执行循环体的<语句块>，然后转步骤③；否则转步骤⑤；

③ 计算<表达式 3>的值，一般用于修改循环变量；

④ 转步骤②，继续执行；

⑤ 退出循环，执行 for 语句后面的语句。

<表达式 1>和<表达式 3>可以是逗号表达式。逗号表达式是指通过逗号运算符连接起来的两个表达式。逗号运算符也称为顺序运算符，是所有运算符中级别最低的。逗号表达式的一般形式为：

```
<表达式 1>，<表达式 2>
```

逗号表达式的求解过程是：先求解<表达式 1>，再求解<表达式 2>。整个逗号表达式的值和类型就是<表达式 2>的值及类型。

2．增强型 for 循环

增强型 for 循环主要是用于枚举、数组和集合对象的元素遍历，其定义形式为：

```
for (<类型>  <变量>  :  <集合对象>)
    <语句块>;
```

for 后的括号中的两个元素之间用一个冒号“：”分隔开。第一个元素用于声明一个具有给定<类型>的临时<变量>；第二个元素用于指定一个可迭代的<集合对象>，如数组、枚举、集合数据对象等。第一个元素声明的<变量>用于依次获取<集合对象>元素的值，因此，其类型需要与<集合对象>中存放的元素类型兼容。

例如，以下的程序就使用增强型 for 循环来遍历枚举类型 Season 定义的常量值。

```
public class Test {
    enum Season { WINTER, SPRING, SUMMER, FALL }

    public static void main(String[] args) {
        for (Season s : Season.values()){
            System.out.println(s);
        }
    }
}
```

其中方法 values 返回枚举类型 Season 中的所有枚举值。程序输出结果为：

```
WINTER
SPRING
SUMMER
FALL
```

使用增强型 for 循环，在遍历集合对象时能使程序更加简洁。

如果循环控制语句的循环体中又包含了循环控制语句，这样就构成了嵌套循环。这 3 种循环语句之间可相互嵌套，构成复杂的逻辑嵌套结构。同 if 语句的嵌套一样，Java 支持无限级循环嵌套。通过循环和分支语句的嵌套可以实现任意复杂的业务逻辑。

1.8.6 流程转移语句

实现 Java 语言的流程转移，可以使用 break、continue、return、yield 和 throw 语句。其中，yield 语句用于 switch 表达式中，参考之前的 switch 语句的说明； throw 语句用于异常抛出，在后面介绍异常处理时再讲解。与 C / C++语言不同，在 Java 中尽管有 goto 关键字，但是不存在 goto 语句。这里仅对 break、continue 和 return 进行介绍。

1．标号语句

Java 允许在语句前加上标号，构成标号语句。标号语句的定义形式为：

```
<标号标识符>： <语句>
```

其中的标号标识符应是 Java 语言中合法的标识符，语句需要为 Java 的有效语句。其含义是，为冒号“：”后的语句指定名为“标识符”的标号。

在 Java 程序中，用到标号的地方基本是在循环语句之前，一般用于多重循环的外循环之前，用于与 break、continue 结合使用，进行循环的跳转控制。

为了便于代码的理解和维护，正常代码尽量不要使用标号来进行流程控制。

2．break 语句

break 语句用于跳出外部语句，语法格式为：

```
break [标号标识符] ;
```

break 语句有以下 3 种主要用法：

（1）在 switch 语句中，用于中止 case 语句序列，跳出 switch 语句；

（2）用在循环结构中，用于中止循环语句，跳出当前循环结构；

（3）与标签语句配合使用，跳出标号指定的循环语句。带标号的 break 语句的定义形式为：

```
break 标号标识符;
```

3．continue 语句

与 break 语句不同，continue 语句并不终止当前循环。在循环体中遇到 continue 语句时，本次循环结束，回到循环条件判断是否执行下一次循环，所以 continue 语句仅跳过当前循环体内 continue 之后的剩余语句。

在循环语句中，continue 语句一般与 if 语句一起使用，即当满足某种条件时，跳过本次循环剩余的语句，强行检测判定条件以决定是否进行下一次循环。

带标号的 continue 语句用于结束当前循环的其他语句，跳转到标号指定循环的下一次循环。仅允许程序从内循环退出到外循环。带标号的 continue 语句格式如下：

```
continue 标号标识符;
```

break 与 continue 语句示例如下。

例 1-2　带标号的 break 与 continue 语句演示。

文件名：BCLable.java

```
public class BCLable {
   public static void main(String[] args) {
      OuterLoop:
      for (int i = 2;; i++) {
         System.out.println("i=" + i);
         for (int j = 2; j < i; j++) {
            if (i % j == 0) {
               continue OuterLoop;
            }
            System.out.println("\t j=" + j);
         }
         if (i == 5) {
            break OuterLoop;
         }
      }
   }
}
```

程序中，语句 continue OuterLoop 的作用是当 i % j 为 0 时，控制流程转移到标号 OuterLoop 的位置，继续外循环的下一次循环；语句 break OuterLoop 控制程序的结束，当 i 等于 5 时，带标号的 break 语句会到达标号 OuterLoop 处，中止标号语句对应的外循环，从而结束程序的执行。程序输出结果为：

```
i=2
i=3
        j=2
i=4
i=5
        j=2
        j=3
        j=4
```

4．return 语句

return 语句的当前方法将程序控制流程返回方法的调用者。语法格式如下：

```
return [<表达式>];
```

当含有 return 语句的方法被调用时，执行 return 语句并从当前方法中退出，返回到调用该方法的语句处。

如果包含有<表达式>，会先计算表达式的值，结束当前方法，并返回表达式的值给调用者。在这种情形下，要求当前方法定义时，方法返回类型要和表达式类型兼容。

如果不包含表达式，实际上不返回任何值，仅在返回类型为 void 的方法中使用。

习题

1．简述一个 Java 应用程序的主要组成。

2．简述 Java 标识符的组成规则。

3．Java 中的基本数据类型有哪几种？不同类型对应常量的表示方法及取值范围是什么？
4．Java 中有哪些运算符？这些运算符的优先关系是怎样的？
5．Java 提供了几类控制语句，每类中有哪些语句？其基本的功能含义是什么？
6．举例说明 switch 语句与 if 语句的异同。
7．举例说明 break 语句、continue 语句和 return 语句三者之间的区别。
8．举例说明&和&&之间的区别。
9．对例 1-2 进行修改，使它能输出任意两个整数之间的所有素数。