高等院校计算机应用系列教材

Java 程序设计

孙　沛　陈珍珍　邓晓林　主　编

黄旭义　王晓涵　谭　淞　熊诗颜　副主编

清华大学出版社

北　京

内 容 简 介

本书系统地构建了 Java 程序设计的完整知识体系，全面涵盖了核心语法、面向对象思想及实战应用技术三大模块。本书内容结构分为四个层次：基础语法篇介绍变量控制、流程结构等过程式编程基础；面向对象篇介绍封装、继承、多态三大特性及其相关应用；技术支撑篇讲授异常处理机制与集合框架等系统级组件；应用实践篇讲授 IO 流操作、并发编程、网络通信以及 JDBC 数据库开发等工程技能。本书以构建知识框架和培养面向对象思维为核心目标，帮助学习者建立系统化的技术认知。全书采用理论与实践相结合的方式，精选典型代码案例辅助理解，确保技术原理的准确传达与实践能力的同步提升。

本书兼具学术深度与实践应用价值，既可作为高等院校计算机专业 Java 程序设计课程的教材，也适合 IT 培训机构用作高级开发课程的教材。同时，本书也适合具有编程基础的自学者作为技术提升的指南。随书提供的在线资源平台包括教学课件、习题答案及扩展案例，构建了一个多维度的学习支持体系。

图书在版编目(CIP)数据

Java 程序设计 / 孙沛, 陈珍珍, 邓晓林主编.
北京 : 清华大学出版社, 2025. 10. -- (高等院校计算机应用系列教材). -- ISBN 978-7-302-70362-4

Ⅰ. TP312.8

中国国家版本馆 CIP 数据核字第 2025RK7758 号

责任编辑：刘金喜
封面设计：高娟妮
版式设计：思创景点
责任校对：成凤进
责任印制：曹婉颖

出版发行：清华大学出版社

网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市龙大印装有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**15.75　　**字　　数：**423 千字
版　　次：2025 年 10 月第 1 版　　**印　　次：**2025 年 10 月第 1 次印刷
定　　价：68.00 元

产品编号：111531-01

前　言

Java 作为当下主流的通用编程语言，广泛应用于桌面应用、企业级应用、Web 应用、移动应用以及大数据处理等多个领域的软件开发中。Java 具有面向对象、跨平台、安全性高、易于使用等显著优势，长期稳居受欢迎编程语言排行榜的前列。与同为主流的 C 语言或 C++语言相比，Java 不仅功能强大，而且应用更为简便。它既适合作为学习程序设计的入门语言，也是学习面向对象程序设计思想的推荐选择。

本书专为广大 Java 初学者及爱好者精心打造，旨在帮助读者构建完整的 Java 知识框架，并掌握出色的面向对象编程技能。其特色体现在内容与形式两个方面。在内容上，全面覆盖了 Java 编程的关键知识点，呈现出完整的 Java 知识体系，包括 Java 核心技术和应用扩展内容。整体结构层次清晰，既方便学习，又便于灵活组合，适配不同的教学学时安排。在形式上，每章都明确设定了学习目标、能力目标与素质目标，重点阐述基本原理，表述简洁明了，易于阅读与理解，同时所举案例经典且具备良好的扩展性。

本书共分为 12 章，具体如下：第 1 章为 Java 概述，第 2 章介绍 Java 基础语法，第 3 章讲解数组，第 4 章为面向对象(上)，第 5 章为面向对象(下)，第 6 章讲解异常处理，第 7 章介绍常用类库，第 8 章讲解集合，第 9 章介绍多线程编程，第 10 章介绍 I/O，第 11 章介绍网络编程，第 12 章介绍数据库编程。

本书第 1 至 7 章属于技术核心模块，建议读者系统学习。其中，第 1 至 6 章内容紧密相连，建议按顺序逐章学习。第 8 至 12 章则属于应用扩展模块，各章内容相对独立，读者可以根据需求自主选择学习。本书条理清晰，授课教师可根据学时安排灵活调整教学内容。

为方便教学，本书配套提供了丰富的教学资源，包括教学大纲、教案、教学课件、案例代码和习题参考答案等，可通过扫描右侧二维码获取。

教学资源

本书的编写得到了成都文理学院校领导的大力支持，同时也非常感谢成都文理学院人工智能与大数据学院老师们的辛勤付出。此外，清华大学出版社编校人员的努力工作，确保了本书的顺利出版。在此，向所有给予帮助的人员致以诚挚的感谢！

由于作者水平有限，书中难免存在不足之处，恳请广大读者与同行批评指正，提出宝贵意见和建议。

编　者

2025 年 4 月

目　录

第 1 章

Java概述

学习目标

1. 了解什么是 Java
2. 掌握 Java 开发环境(JDK)的搭建
3. 掌握系统环境变量的配置，能够独立完成 PATH 和 CLASSPATH 环境变量的配置
4. 掌握 IntelliJ IDEA 开发工具的基本用法，能够独立安装 IntelliJ IDEA 并使用它开发和调试代码

能力目标

1. 能够独立安装 JDK
2. 掌握 Java 程序的基本操作，能够独立编写第一个 Java 程序

素质目标

1. 培养学生良好的专业素养和精益求精的工匠精神
2. 培养学生协同合作的团队精神
3. 培养学生的自主学习能力和可持续发展能力

1.1 Java 简介

1.1.1 什么是 Java

Java 是一种广泛使用的面向对象程序设计语言，能够编写跨平台应用软件。它由 Sun Microsystems 公司于 1995 年正式推出，最初设计用于消费类电子产品中的嵌入式芯片，被命名为 Oak，但为了更好地体现其特性，后来更名为 Java。Java 具备简单性、面向对象、分布式、解释型、健壮性、安全性、体系结构中立、可移植性、高性能、多线程和动态性等诸多特性。除此之外，Java 还拥有一个功能丰富的应用程序接口(API)，其中包含大量的类和方法，用于执行各种常见的编程任务，如文件操作、网络通信、数据库连接等。Java 的设计哲学强调“一次编写，到处运行”的理念，这使得它在不同的操作系统和硬件平台上都能够保持一致的性能和行为。

Java 的应用领域非常广泛，涵盖了从桌面应用到 Web 应用、从移动应用到企业级应用等多个领域，其身影随处可见。特别是在企业级应用开发方面，Java 凭借其稳定、高效、安全的特点，成为许多大型企业和政府机构的首选技术栈。Java 的这些优势使得它在构建复杂的系统时，能够提供可

靠且可扩展的解决方案，从而在竞争激烈的市场中保持了其技术领先地位。Java 的生态系统非常庞大，拥有丰富的开发工具、框架和库，为开发者提供了极大的便利。不仅如此，Java 社区活跃，拥有大量的资源和文档，为学习和解决问题提供了强有力的支持。

1.1.2 Java 的特点

1. 简洁性

Java 语言的语法结构与 C 语言和 C++语言非常相似，这使得大多数程序员能够轻松学习和应用。同时，Java 剔除了 C++中那些较少使用、容易引起混淆的特性，例如操作符重载、多继承、自动强制类型转换等，使其更易于学习和掌握。

2. 面向对象

Java 语言提供了类、接口和继承等面向对象的特性，并全面支持动态绑定。然而，它只支持类之间的单继承。Java 语言是一种纯粹的面向对象程序设计语言，其所有代码必须位于类中。

3. 分布式

Java 是一种面向网络的语言，提供了丰富的 API，支持 TCP/IP 协议，便于进行网络编程和分布式计算。通过 URL，Java 可以访问网络上的对象，并实现远程方法调用等功能。

4. 编译与解释性

Java 程序首先将源代码编译成字节码文件(.class)，然后在 Java 虚拟机(JVM)上对字节码进行解释执行。通过这种编译与解释相结合的方式，Java 既确保了程序的运行效率，又提升了程序的可移植性。

5. 稳健性

Java 在编译和运行程序的过程中会对可能出现的问题进行检查，以防止错误的发生。同时，Java 不支持指针操作，显著减少了出错的可能性。此外，Java 还具备异常处理机制，能够有效地处理程序中的异常情况。

6. 安全性

Java 通常用于网络环境中，为了防止恶意代码的攻击，Java 提供了一套安全机制。除了 Java 语言本身具备的多种安全特性外，Java 还对通过网络下载的类实施了严格的安全防范措施(如类 ClassLoader)。例如，它通过分配不同的命名空间以防止本地同名类被替代，进行字节码检查，并提供安全管理机制(如 SecurityManager 类)，使 Java 应用能够设置安全哨兵。

7. 体系结构中立

Java 程序编译后的字节码可以在任何安装了 Java 虚拟机(JVM)的计算机上运行，从而实现了“一次编写，到处运行”的特性，这显著提高了软件的可移植性。

8. 可移植性

Java 的可移植性源自其体系结构中立性。此外，Java 还严格规定了各个基本数据类型的长度，并且 Java 编译器是用 Java 语言实现的，Java 的运行环境是用 ANSI C 语言实现的，这些设计确保了 Java 程序可以在不同平台上稳定运行。

9. 解释型

Java 程序在运行时通过 JVM 解释执行字节码，这赋予了 Java 强大的跨平台能力，并支持即时(JIT)编译，从而提升了执行性能。

10. 高性能

尽管 Java 是解释型语言，但通过 JIT 编译技术，Java 程序的执行速度接近于原生编译的 C++程序，兼具灵活性和高性能。

11. 多线程

Java 原生支持多线程编程，提供了一套简便的 API 来创建和管理线程。这简化了并发编程，提升了程序的响应性和整体性能。

12. 动态性

Java 是一种动态语言，可以在运行时加载新类和对象，支持动态扩展和模块化编程，从而适应不断变化的运行环境。

1.2 Java 平台与 JVM

1.2.1 Java 平台与 JVM 简介

Java 平台与 JVM(Java Virtual Machine)是 Java 语言运行环境中的两个核心组件，它们相互紧密关联，彼此依存。Java 平台是一个完整的软件开发和部署环境，包含了 Java 编程语言、Java 虚拟机(JVM)、Java API，以及一组用于开发和测试 Java 应用程序的工具。Java 平台的设计目标是提供一套简单、一致、动态、安全且可用于各种计算环境的开发平台。

JVM 是 Java 平台的核心组件之一，它充当一个抽象的计算机，提供了一种独立于具体硬件和操作系统的平台，用于执行 Java 字节码。JVM 负责将 Java 字节码转换为特定平台的机器码，并确保 Java 程序在不同的硬件和操作系统平台上都能够稳定一致地运行。此外，JVM 还提供了内存管理、垃圾回收、安全控制等功能，为 Java 程序的执行提供了强有力的支持。

Java 平台通过 JVM 实现了“一次编写，到处运行”的理念，使得 Java 程序可以在不同的操作系统和硬件平台上无缝运行，无须针对每个平台进行单独的编译和测试。这种跨平台能力极大地提高了 Java 程序的可移植性和灵活性，使得 Java 成为广泛应用于企业级应用开发、Web 应用开发、移动应用开发等领域的强大编程语言。

1.2.2 JDK 的安装

Oracle 公司提供了针对多种操作系统的 JDK，不同操作系统的 JDK 在使用上类似。初学者可以根据自己使用的操作系统，从 Oracle 官方网站下载相应的 JDK 安装文件。下面以 64 位的 Windows 11 操作系统为例，介绍 JDK 21 的安装过程。

1. 下载 JDK

访问 Oracle 官网(http://www.oracle.com)，选择适用于 64 位系统的 JDK 21 版本，如图 1-1 所示。

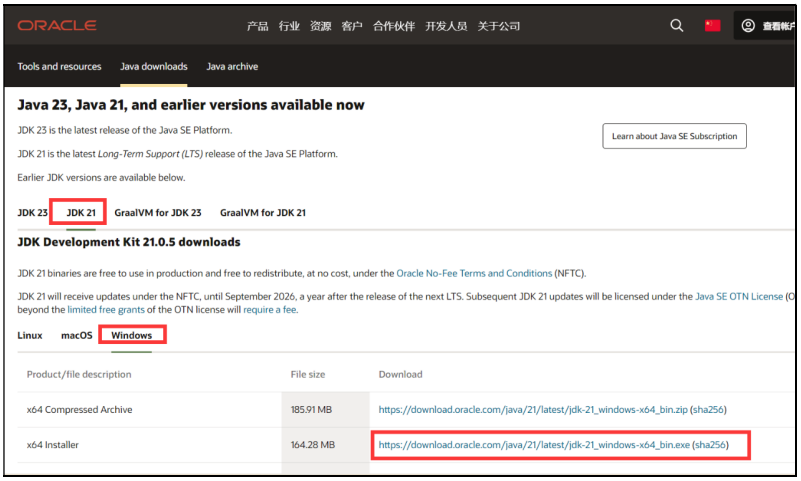

图 1-1　Oracle 官网

2. 自定义安装过程和路径

下载完成后，进入图 1-2 所示的安装界面，单击“下一步”按钮开始 JDK 的安装过程。在安装过程中可以选择自定义安装目录，如图 1-3 所示。安装完成后，将显示如图 1-4 所示的界面。

图 1-2　JDK 21 安装界面

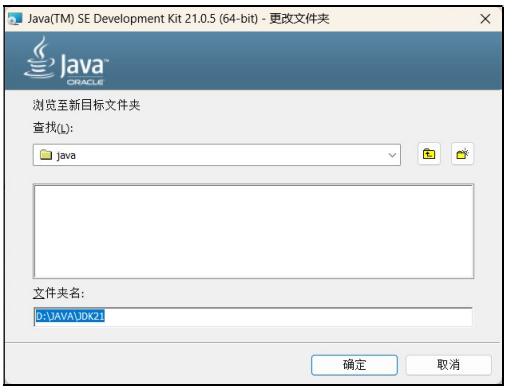

图 1-3　自定义安装目录

图 1-4　安装完成界面

JDK 安装完成后，系统会在磁盘上生成一个 JDK 安装文件夹，如图 1-5 所示。

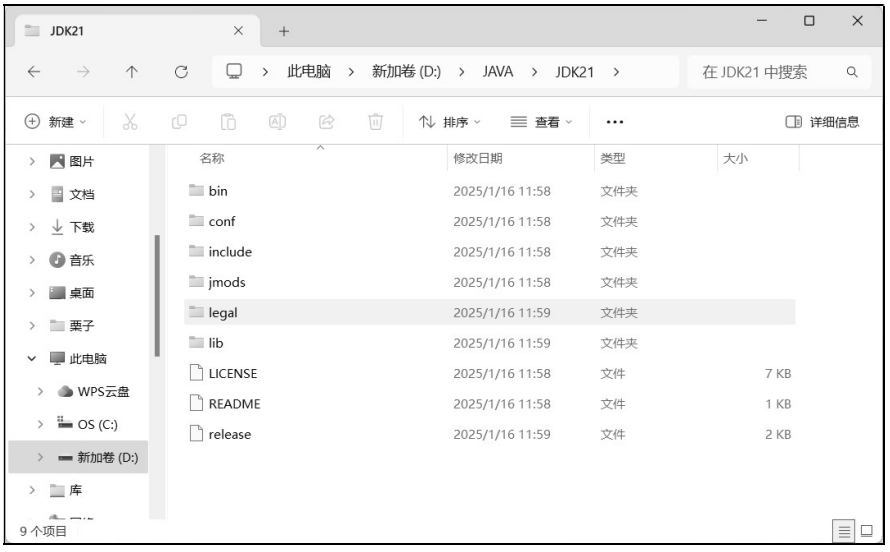

图 1-5　JDK 安装文件夹

为了更好地学习，下面对 JDK 安装文件夹中的子文件夹进行介绍。

(1) bin 文件夹：包含了 JDK 的各种工具命令和可执行文件，如 javac(Java 编译器)、java(Java 运行环境)、javadoc(Java 文档生成器)等。这些工具是 Java 开发过程中必不可少的。

(2) lib 文件夹：存放 Java 的类库文件，包括 JDK 自带的工具类和第三方库。这些类库为 Java 程序提供了丰富的功能和接口。

(3) include 文件夹：包含了 Java 和 JVM 的头文件，这些文件主要用于 C/C++程序员在开发本地代码(Native Code)时，与 Java 代码进行交互。

(4) jmods 文件夹：存放 Java 模块文件。自 Java 9 起，Java 引入了模块系统，将 Java 平台划分为多个模块，从而提高了 Java 平台的模块化和可维护性。

(5) legal 文件夹：包含了 JDK 的使用许可和法律声明。对于使用 JDK 进行开发的用户来说，了解这些许可和声明是非常重要的。

(6) conf 文件夹：包含了 JDK 的配置文件，这些文件用于配置 JDK 的运行环境，如默认的区域设置、安全策略等。

此外，JDK 安装文件夹还可能包含其他子文件夹和文件，这些内容会根据 JDK 的版本和发行版有所不同。例如，man 文件夹包含了 JDK 工具的 Unix 手册页，提供了 JDK 工具的详细使用说明和选项；db 文件夹包含了 Java 的调试数据库，用于存储 Java 程序的调试信息。了解这些子文件夹和文件的作用，有助于更好地使用 JDK 进行 Java 开发。

1.2.3　JDK 环境变量配置

配置 JDK 环境变量是为了使操作系统能够正确找到 JDK 的可执行文件和类库文件，从而编译和运行 Java 程序。以下是配置 JDK 环境变量的步骤。

(1) 右击“我的电脑”图标，在弹出的快捷菜单中选择“属性”命令，在打开的对话框中单击“高级系统设置”按钮，打开“系统属性”对话框，选择“高级”选项卡，单击“环境变量”按钮，如

图 1-6 所示。

(2) 配置 JAVA_HOME 环境变量。在打开的“环境变量”对话框中单击“新建”按钮(如图 1-7 所示)。打开图 1-8 所示的“新建系统变量”对话框，创建一个新的环境变量，设置变量名为 JAVA_HOME，变量值为 JDK 的安装路径(如“D:\JAVA\JDK21”)。

图 1-6 “系统属性”对话框

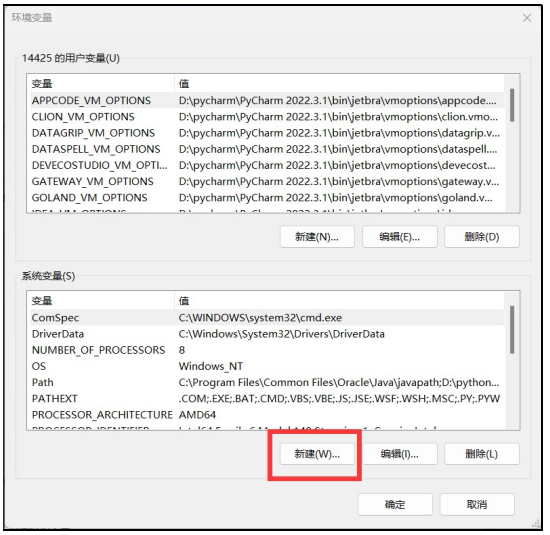

图 1-7 “环境变量”对话框

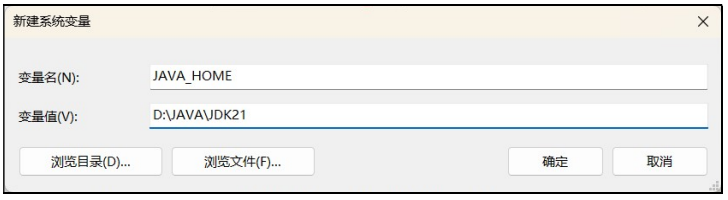

图 1-8 新建系统变量

(3) 配置 PATH 环境变量。PATH 环境变量用于指定操作系统搜索可执行文件的目录。为了让操作系统能够找到 JDK 的可执行文件，需要将 JDK 的 bin 目录添加到 PATH 环境变量中。在“系统变量”区域，找到名为 Path 的变量，将其选中后单击“编辑”按钮(如图 1-9 所示)。在打开的“编辑环境变量”对话框中，单击“新建”按钮，将 JDK 的 bin 目录添加到 PATH 变量中，输入“%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin”，如图 1-10 所示(如果 Path 变量已经包含其他目录，需要用分号(;)将它们分隔开)。

(4) 新建 CLASSPATH 变量，变量值填写“.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\ tools.jar”，如图 1-11 所示。

(5) 验证配置。配置完成后，可以通过命令提示符(CMD)验证 JDK 环境变量是否配置成功。打开 CMD，输入“java –version”命令，如图 1-12 所示。此时，如果能够显示版本信息，则表示安装和配置成功。

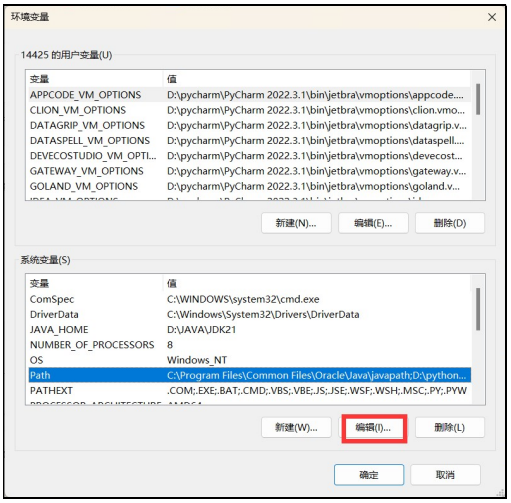

图 1-9　配置 PATH 环境变量

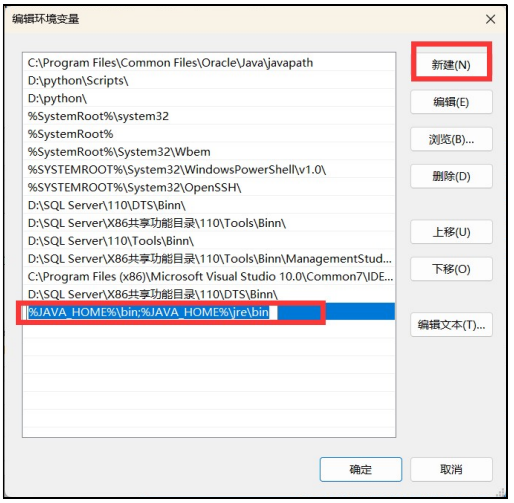

图 1-10　编辑环境变量

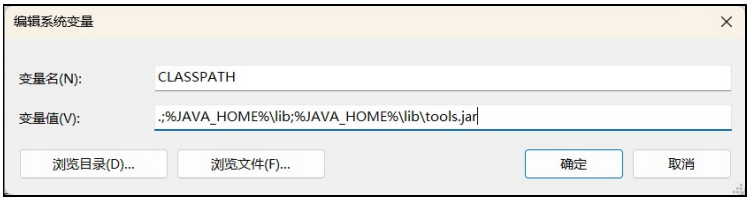

图 1-11　新建 CLASSPATH 变量

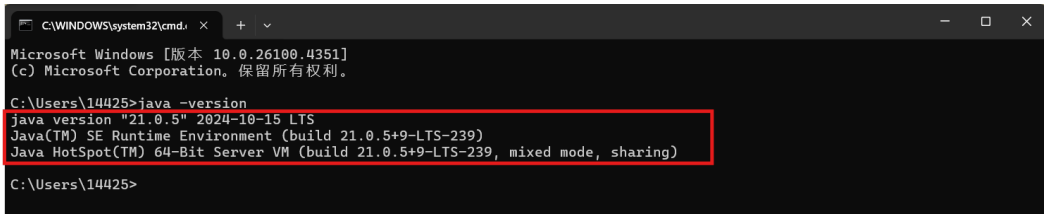

图 1-12　验证配置

1.3　集成开发环境

集成开发环境(IDE)是软件开发过程中不可或缺的工具，它提供了代码编辑、编译、调试、版本控制等一系列功能，极大地提高了开发效率。对于 Java 开发来说，选择合适的 IDE 尤为重要。目前，市面上有许多优秀的 Java IDE，如 Eclipse、IntelliJ IDEA 和 NetBeans 等。这些 IDE 不仅提供了丰富的插件和扩展功能，还支持多种编程语言和框架，能够满足不同开发者的需求。

1. Eclipse

Eclipse 是一个开源的 Java IDE，具有良好的可扩展性和灵活性，支持多种插件和扩展点，用户可以根据需要定制和扩展功能。Eclipse 还提供了强大的调试工具，支持断点调试、表达式求值、内存检查等功能，可以帮助开发者快速定位和解决问题。

2. IntelliJ IDEA

IntelliJ IDEA 是 JetBrains 公司开发的一款商业 Java IDE，以其强大的代码分析和重构功能而闻名。它支持智能代码补全、代码导航、代码审查等功能，极大地提高了开发效率和代码质量。IntelliJ IDEA 还提供了丰富的插件和扩展功能，支持多种编程语言和框架，例如 Spring、Hibernate、Maven 等，为开发者提供了极大的便利。

3. NetBeans

NetBeans 是一款广受欢迎的 Java IDE，具有简洁易用的用户界面和丰富的功能，支持代码编辑、调试、版本控制等功能。NetBeans 还提供了良好的集成支持，可以与多种数据库和服务器进行连接和交互，为开发者提供更多选择和灵活性。

1.4 第一个 Java 程序

通过安装 JDK，我们已经搭建了 Java 的开发环境。接下来，让我们体验一下如何编写第一个 Java 程序。我们将创建一个简单的 Java 程序，用于在控制台上输出“HelloWorld!”这一经典的入门信息。

1. 编写 Java 源文件

在 JDK 安装文件夹的 bin 文件夹下新建文本文档，命名为 HelloWorld.java。用记事本打开 HelloWorld.java 文件，并编写以下 Java 代码：

```
public class HelloWorld{
    public static void main(String[]args){
        System.out.println("HelloWorld!");
    }
}
```

代码解析：

- public class HelloWorld：这一行定义了一个名为 HelloWorld 的公共类。在 Java 中，每个源文件通常包含一个公共类，且类名应与文件名保持一致(文件名应为 HelloWorld.java)。
- public static void main(String[] args)：这是 Java 程序的入口点。main 方法是程序开始执行的起始位置。String[] args 是传递给程序的命令行参数的数组。
- System.out.println("HelloWorld!");：这一行代码将字符串“HelloWorld!”输出到控制台。System.out 是标准输出流，而 println 方法用于打印一行文本。

2. 编译 Java 源文件

在运行 Java 源文件之前，需要先进行编译，将其转换为 Java 虚拟机(JVM)能够理解的字节码。编译过程是通过 JDK 提供的 javac 命令完成的。

打开命令提示符(CMD)，导航到保存 HelloWorld.java 文件的目录。输入以下命令进行编译：

```
javac HelloWorld.java
```

如果编译成功，命令提示符将不会显示任何错误消息，并且在同一目录下生成一个名为

HelloWorld.class 的字节码文件，如图 1-13 所示。

3. 运行 Java 程序

编译成功后，用户可以运行生成的 HelloWorld.class 文件。在命令提示符中输入以下命令：

```
java HelloWorld
```

此时，控制台将输出：“HelloWorld!”。这表明第一个 Java 程序已经成功编写、编译并运行，如图 1-14 所示。

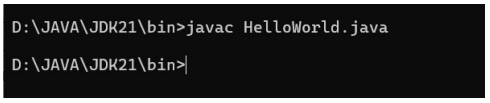

图 1-13　编译成功

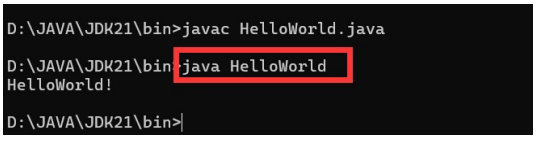

图 1-14　HelloWorld 程序运行结果

1.5 IntelliJ IDEA 的安装与启动

IntelliJ IDEA(简称 IDEA)是一款功能强大的 Java 集成开发环境(IDE)，以其出色的代码分析、重构和智能提示功能而闻名。IntelliJ IDEA 提供了便捷的类和方法导航功能，通过简单的快捷键或鼠标操作，开发者可以快速定位到代码中的任何位置。此外，它还支持代码折叠、书签、代码模板等高级编辑功能，进一步提升了开发体验。

除了基本的代码编辑和调试功能，IntelliJ IDEA 还提供了丰富的插件和扩展功能。通过安装这些插件，开发者可以轻松集成版本控制系统(如 Git、SVN 等)、构建工具(如 Maven、Gradle 等)以及其他有用的工具和服务。这些插件的引入，使得 IntelliJ IDEA 成为一个功能齐全、易于扩展的 Java 开发平台。

1.5.1　安装 IDEA 开发工具

访问 IDEA 官网(http://www.jetbrains.com/idea)，下载适合当前操作系统的安装包。IDEA 提供旗舰版和社区版两种选择，这里以下载 Windows 操作系统的社区版为例，如图 1-15 所示。

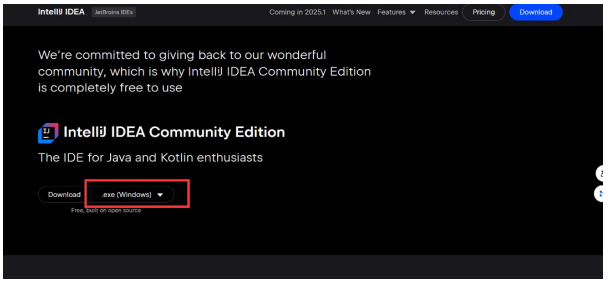

图 1-15　IDEA 官网下载

(1) 下载完成后，启动安装程序，在打开的安装界面中单击“下一步”按钮，如图 1-16 所示。

图 1-16　IntelliJ IDEA 安装界面

(2) 选择安装路径后，单击“下一步”按钮，进入基本安装选项配置界面，选中图 1-17 所示的复选框，然后单击“下一步”按钮。

图 1-17　基本安装选项配置界面

(3) 打开“选择开始菜单”界面，保持默认设置，单击“安装”按钮，进入图 1-18 所示的安装界面。

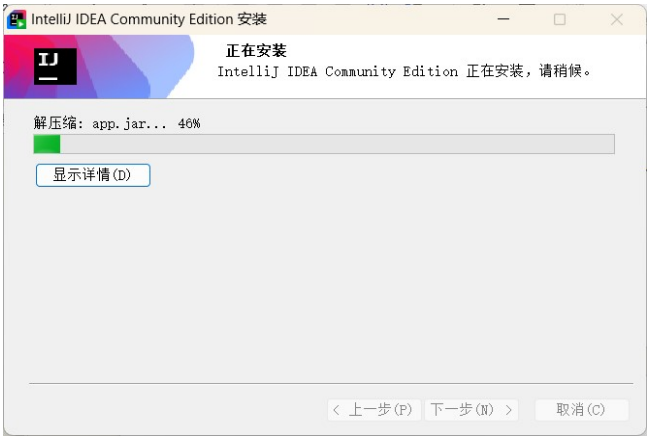

图 1-18　安装界面

(4) 安装完成后，单击“完成”按钮重启系统，如图 1-19 所示。

图 1-19　安装完成界面

1.5.2　启动 IDEA

双击 Windows 系统桌面上的 IDEA 快捷方式，将启动 IDEA 并打开图 1-20 所示的 IDEA 欢迎界面。

图 1-20　IDEA 欢迎界面

1.6　使用 IntelliJ IDEA 进行开发

在上一节中，我们完成了 IDEA 的安装与启动。接下来，将使用 IDEA 创建一个 Java 程序，实现控制台输出“Hello Java!”的功能。

1. 创建 Java 项目

在图 1-20 所示的 IDEA 欢迎界面中单击“新建项目”按钮，在打开的对话框中选择新建 Java 程

序，并选择已下载的 JDK，然后单击“创建”按钮，如图 1-21 所示。

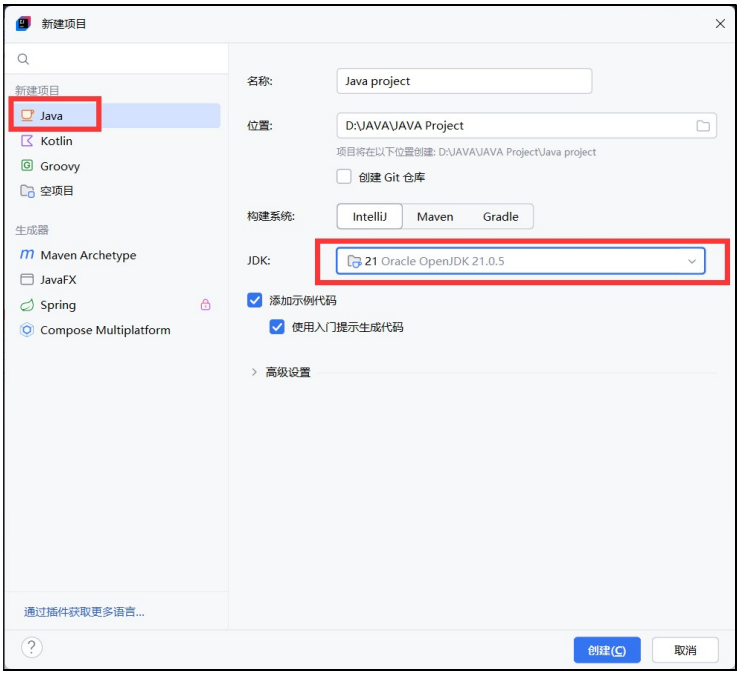

图 1-21　新建项目

项目创建完成后的界面如图 1-22 所示。

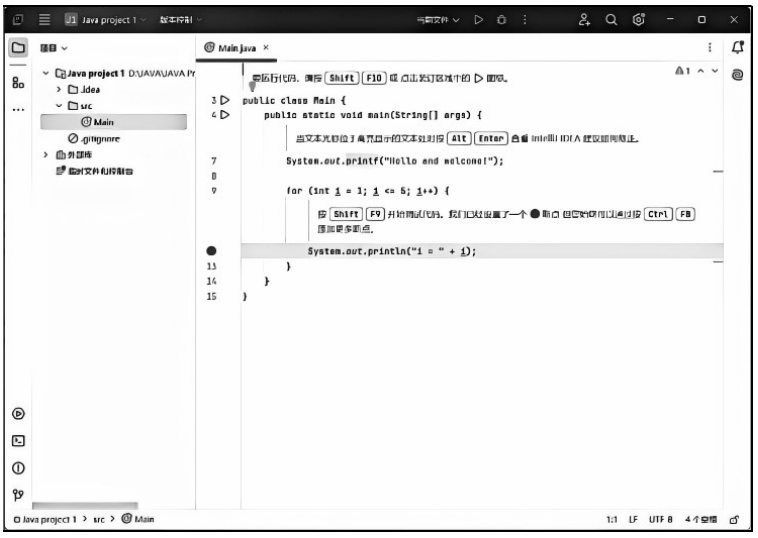

图 1-22　IDEA 项目结构界面

2. 创建 Java 类

在图 1-22 所示的项目结构界面中，右击 src 文件夹，在弹出的快捷菜单中选择“新建”|“Java 类”命令，如图 1-23 所示。在打开的界面中输入类名(如“HelloJava”)，然后按 Enter 键完成 Java 类的创建。Java 类创建完成之后，src 文件夹中会生成 HelloJava.java 文件，该文件会自动在界面的

右侧区域打开，如图 1-24 所示。

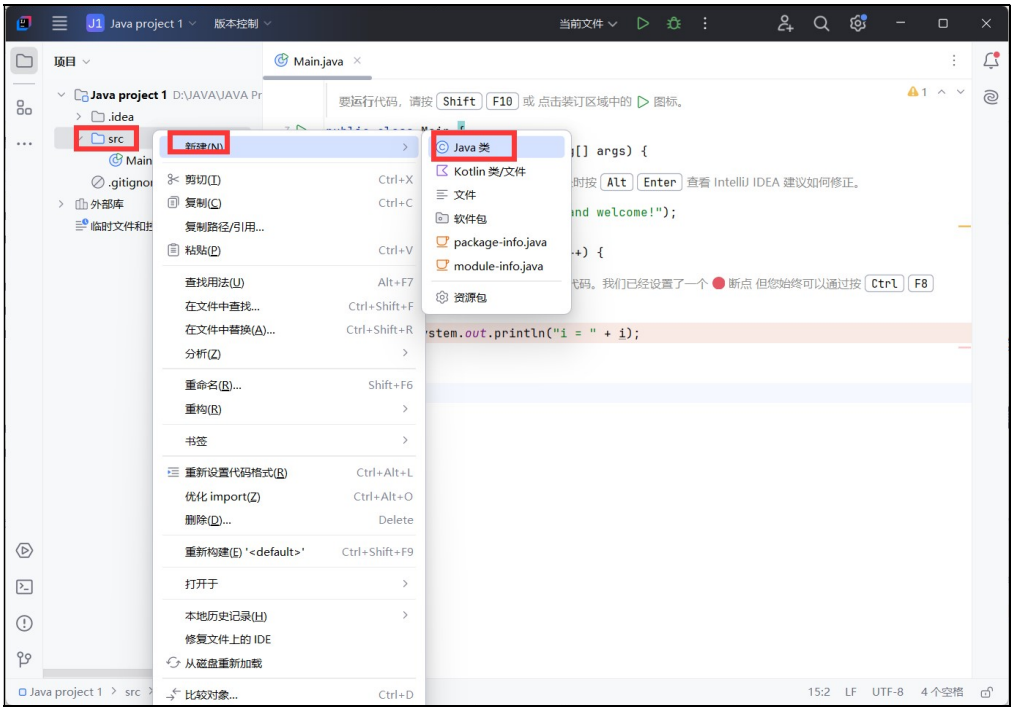

图 1-23　新建 Java 类

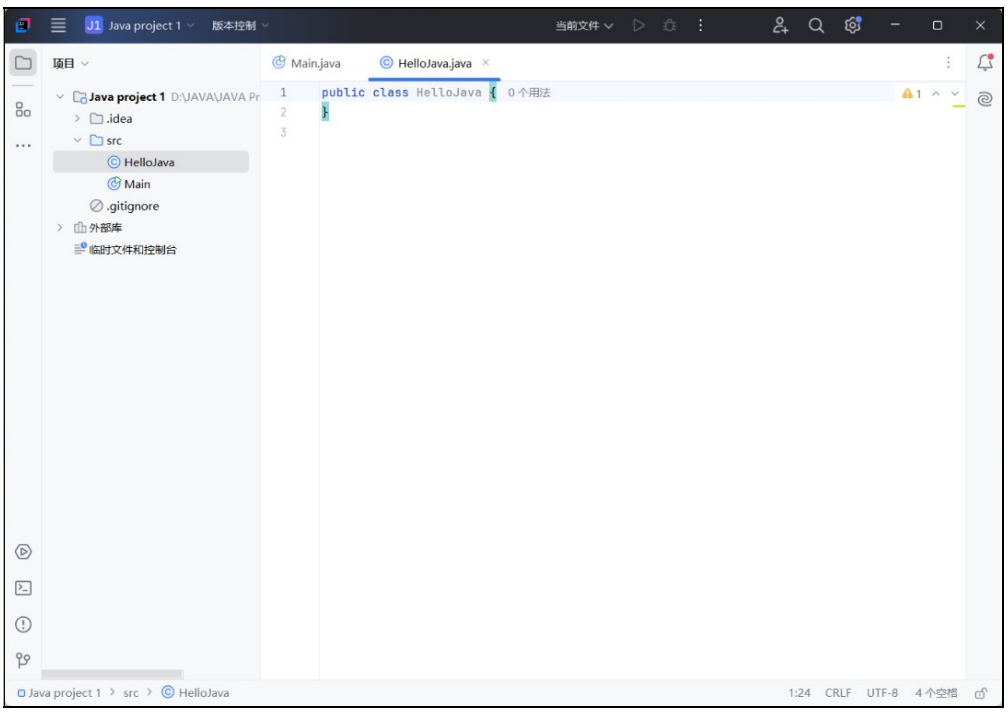

图 1-24　HelloJava.java 文件

从图 1-24 可以看出，HelloJava 文件以.java 为扩展名，右侧显示的是 HelloJava.java 文件创建时

的默认代码。这里，HelloJava 为类的名称，class 是定义类的关键字，public 是类的权限修饰符，表示该类是共有类。在 HelloJava 后面的一对大括号中，可以编写类的程序代码。

3. 编写程序并运行

在 HelloJava.java 文件中，编写以下代码：

```
public class HelloJava {
    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

上述代码定义了一个名为 HelloJava 的类，并在其中编写了一个 main 方法。main 方法是 Java 程序的入口点，当运行程序时，将执行 main 方法中的代码。在 main 方法中，通过 System.out.println 语句在控制台上输出“Hello Java!”。

程序编写完成后，单击 IDEA 界面右上角的“运行”按钮，或者右击 HelloJava.java 文件，在弹出的快捷菜单中选择“运行”命令，即可运行程序。程序运行成功后，会在 IDEA 界面下方的控制台窗口中显示“Hello Java!”，如图 1-25 所示。

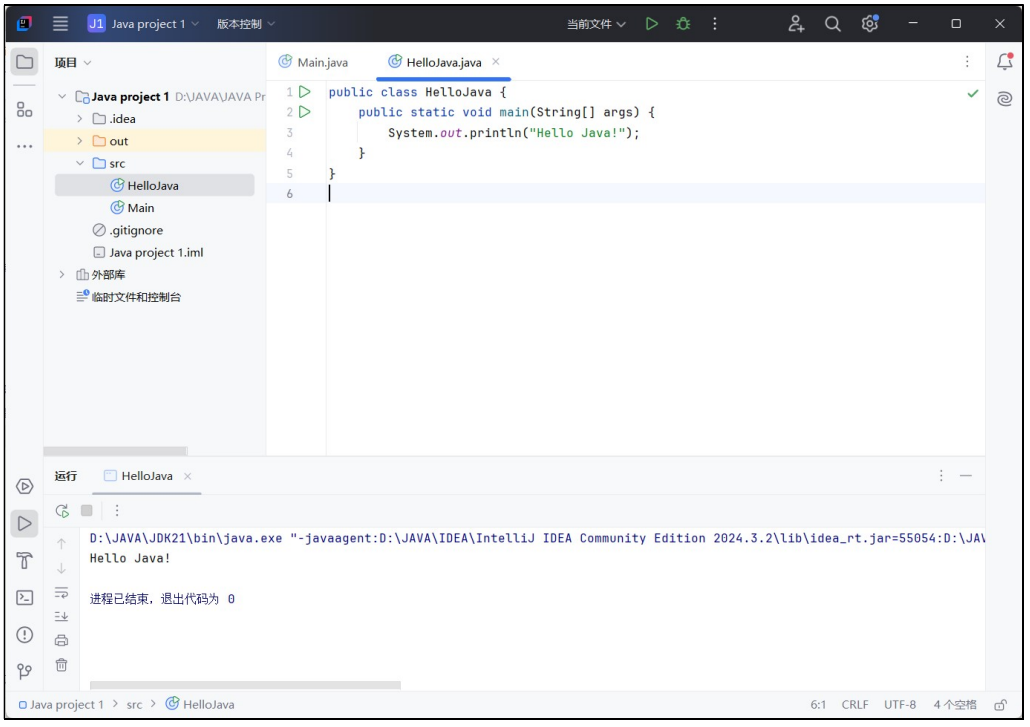

图 1-25　控制台输出信息

至此，用户已经通过 IntelliJ IDEA 成功创建并运行了一个简单的 Java 程序。通过这个例子，用户可以初步了解如何使用 IDEA 进行 Java 开发，包括项目创建、类创建、代码编写和程序运行等基本操作。随着学习的深入，用户将逐渐掌握 IDEA 的其他高级功能和技巧，从而提高开发效率和代码质量。

1.7 小结

本章主要介绍了 Java 开发环境的搭建，包括 JDK 的安装与环境变量的配置，以及如何在命令提示符中验证配置是否成功。同时，我们简要介绍了集成开发环境(IDE)的概念，并列举了 Eclipse、IntelliJ IDEA 和 NetBeans 等几款流行的 Java IDE，以及它们各自的特点和优势。通过安装和配置 JDK，我们体验了如何编写、编译和运行一个简单的 Java 程序“HelloWorld!”。此外，我们还以 IntelliJ IDEA 为例，详细介绍了如何安装和启动 IDEA，以及如何使用 IDEA 创建一个 Java 项目，并实现控制台输出“Hello Java!”的功能。这些内容为我们后续深入学习 Java 语言奠定了坚实的基础。

1.8 习题

一、选择题

1. Java 是一种(　　)的程序设计语言。

A. 面向过程　　B. 面向对象　　C. 函数式　　D. 命令式

2. Java 语言的特点包括(　　)。

A. 不支持继承　　B. 支持多继承　　C. 支持单继承　　D. 支持多重继承

3. Java 程序首先被编译成(　　)文件。

A. exe　　B. class　　C. obj　　D. o

4. Java 的“一次编写，到处运行”特性主要依赖于(　　)技术。

A. JVM　　B. JIT　　C. JDK　　D. API

5. 在 Java 中，(　　)关键字用于定义公共类。

A. private　　B. protected　　C. public　　D. package

二、简答题

1. 解释什么是 Java 虚拟机(JVM)。
2. 简述 Java 语言的主要特点。

三、编程题

1. 编写一个简单的 Java 程序，计算两个整数的和并输出结果。
2. 编写一个 Java 程序，判断一个整数是否是偶数。

第 2 章

Java基础语法

学习目标

1. 掌握 Java 的基本语法，能够根据基本格式要求编写 Java 程序，熟练使用 Java 中的注释、关键字和常量，并能够准确定义标识符

2. 掌握变量的定义和使用，能够熟练定义各种数据类型的变量，并独立实现变量之间的类型转换

3. 掌握运算符的使用

4. 掌握选择结构和循环结构语句的应用

能力目标

1. 能够正确使用运算符解决程序中的运算问题

2. 能够熟练使用 if 条件语句、三元运算符和 switch 条件语句解决程序中的选择问题

3. 能够熟练使用 while 循环语句、do-while 循环语句、for 循环语句以及循环嵌套和跳转语句解决程序中的循环问题

素质目标

1. 培养学生良好的专业素养和团队精神

2. 培养学生自主学习能力和可持续发展的能力

2.1 标识符与关键字

2.1.1 标识符

在编程过程中，常常需要在程序中定义一些符号，用来标记一些名称，例如包名、类名、方法名、参数名、变量名等，这些符号被称为标识符。标识符的命名需要遵循一定的规则，以确保程序的正确性和可读性。

以下是一些关于标识符命名的规则。

(1) 标识符必须以字母(A~Z 或 a~z)、下画线(_)或美元符号($)开头，后续字符可以是字母、数字(0~9)、下画线或美元符号。

(2) 标识符区分大小写，例如变量名 myVariable 和 MyVariable 在 Java 中被视为两个不同的标识符。

(3) 避免使用 Java 的关键字和保留字作为标识符。关键字是 Java 语言中具有特殊含义的单词，(本章 2.1.2 节列举了 Java 关键字)，使用这些单词作为标识符会导致编译错误。

(4) 遵循一定的命名约定，具体如下。

- 类名和接口名中的每个单词的首字母都大写(如 MyClass)。
- 方法名和变量名的第一个单词的首字母小写，后续单词的首字母大写(如 myMethod 和 myVariable)。
- 对于常量，通常使用全大写字母和下画线进行命名(如 MAX_VALUE 和 PI)。
- 包名中的所有字母一律小写。

2.1.2　关键字

Java 语言中定义了一系列具有特殊含义的单词，这些单词被称为关键字。关键字在 Java 程序中具有固定的用途，不能用作标识符(例如变量名、方法名等)。Java 的关键字随着版本的更新而有所变化，一些常用的 Java 关键字如表 2-1 所示。

表 2-1　Java 关键字

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

了解并掌握这些关键字，对于正确编写和理解 Java 程序至关重要。在编写 Java 程序时，关键字的使用需要注意以下几点。

(1) 所有的关键字都小写。

(2) 不能使用关键字命名标识符。

(3) const 和 goto 是保留的关键字，虽然在 Java 中没有任何意义，但在程序中不能使用它们作为自定义的标识符。

(4) true、false 和 null 虽然不属于关键字，但它们具有特殊的意义，也不能作为标识符使用。

2.2　基本数据类型

2.2.1　常量

在 Java 编程语言中，常量指的是在程序执行过程中值不能被改变的量。常量在定义时就需要被

初始化，并且在程序的后续执行过程中，它们的值不能被修改。常量的使用有助于提升代码的可读性和可维护性，因为它们提供了一种表示固定值的方式，使得这些值在代码中更容易被理解和识别。

在 Java 中，常量包括整型常量、浮点数常量、字符常量、字符串常量、布尔常量等。

1. 整型常量

整型常量是整数类型的数据，它有二进制、八进制、十进制和十六进制 4 种表示形式，具体如下。

- 二进制：由数字 0 和 1 组成的数字序列。从 JDK 7 开始，允许使用字面值表示二进制数，前面要以 0b 或 0B 开头
- 八进制：以 0 开头并且其后由 0~7 范围内(包括 0 和 7)的整数组成的数字序列。
- 十进制：由数字 0~9 范围内(包括 0 和 9)的整数组成的数字序列，第一位不能是 0，0 本身除外。
- 十六进制：以 0x 或 0X 开头并且其后由 0~9、A~F(包括 0 和 9、A 和 F，字母不区分大小写)组成的数字序列。

不同进制的示例及说明如表 2-2 所示。

表 2-2　整型变量的不同进制

类型	示例	说明
二进制	0b0110, 0B10101	以 0b 或 0B 开头
八进制	0432	以 0 开头
十进制	198, 0	首位不能为 0(0 除外)
十六进制	0x25AF, 0xaf3, 0Xff	以 0x 或 0X 开头

2. 浮点数常量

浮点数常量用于表示带有小数部分的数值。在 Java 语言中，浮点数分为单精度(float)和双精度(double)两种类型。单精度浮点数以字母 F 或 f 结尾，而双精度浮点数则以字母 D 或 d 结尾。若在浮点数后未添加任何后缀，Java 虚拟机(JVM)将默认其为 double 类型的浮点数。

浮点数常量的具体示例如表 2-3 所示。

表 2-3　浮点数常量示例

类型	示例	说明
单精度	3.14F, 2.718f	以 F 或 f 结尾
双精度	3.14D, 2.718d	以 D 或 d 结尾
默认类型	3.14	默认为 double 类型

3. 字符常量

字符常量是指在程序中用于表示单个字符的符号。在 Java 中，字符常量被单引号(' ')所包围，并且仅能包含一个字符。例如，字符常量'A'代表大写字母 A，'1'代表数字 1，'#'代表井号符号等。

字符常量除了能表示普通字符以外，还可通过转义序列来表示特殊字符。转义序列以反斜杠(\)开头，用于表示那些无法直接在字符集中表示的字符，或具有特殊意义的字符。例如，换行符可用'\n'表示，回车符可用'\r'表示，制表符可用'\t'表示。

Java 中的字符常量以 Unicode 编码形式存储于内存中，每个字符占用 2 个字节。Unicode 编码是一种国际标准化的字符编码系统，支持多种语言字符，可以确保 Java 程序能够跨平台运行，并正确处理不同语言的字符。

在编写 Java 程序时，合理运用字符常量有助于更准确地描述和处理文本数据。例如，字符常量可用于表示用户输入的密码字符，或在处理字符串时逐个字符地进行操作。在 Java 编程中，正确区分字符常量与字符串常量至关重要，后者使用双引号(" ")包围，并可以包含多个字符。

4. 字符串常量

字符串常量是指在 Java 程序中用于表示一串字符的符号。与字符常量不同，字符串常量使用双引号(" ")包围，并可以包含零个或多个字符。例如，"Hello, World!"和""(空字符串)均为合法的字符串常量。

在 Java 中，字符串常量是不可变的，这意味着一旦字符串被创建，其内容便无法更改。若需修改字符串内容，实际上并不是直接对原字符串进行修改，而是创建了一个新的字符串对象，并废弃了原来的对象。这种不可变性使得字符串成为 Java 中一种安全且易于管理的数据结构。

字符串常量在 Java 程序中具有广泛的应用。它们可用于表示文本信息，如用户输入提示、程序输出结果等。此外，字符串常量还可用于执行各种字符串操作，如拼接、比较、查找、替换等。Java 提供了丰富的字符串处理类和方法，便于开发者处理和操作字符串数据。

值得注意的是，尽管字符串常量在内存中以对象形式存在，Java 仍通过字符串常量池来优化其存储和管理。字符串常量池是一个特殊的存储区域，用于存储字符串字面量。创建字符串常量时，Java 会先检查字符串常量池中是否存在相同的字符串字面量。若存在，则返回该字符串字面量的引用；若不存在，则在字符串常量池中创建新的字符串字面量，并返回其引用。这种机制避免了重复创建相同的字符串字面量，从而提高了内存使用效率。

5. 布尔常量

布尔常量在 Java 中表示逻辑上的真(true)和假(false)。与整型常量、浮点数常量、字符常量、字符串常量不同，布尔常量只包含两个值：true 和 false，它们分别表示逻辑上的真和假。

布尔常量常用于条件判断语句中(如 if 语句和 while 语句)，以控制程序的执行流程。

2.2.2　变量

在 Java 程序中，变量用于存储数据值。这些数据值可以是整数、浮点数、字符、字符串或布尔值等。变量在使用前必须先声明，即在代码中明确指出变量的类型和名称。

1. 变量的数据类型

Java 支持多种数据类型，每种数据类型都有其特定的用途和特点。主要的数据类型如下。

1) 基本数据类型

- byte：8 位有符号整数。
- short：16 位有符号整数。
- int：32 位有符号整数。
- long：64 位有符号整数。
- float：单精度浮点数。
- double：双精度浮点数。

- char：Unicode 字符。
- boolean：布尔值(true 或 false)

2) 引用数据类型

- class：类类型。
- interface：接口类型。
- void：无类型。
- String：字符串类型。

2. 变量的声明与初始化

变量的声明是指在程序中指定变量的类型和名称。例如：

```
int age;                                    // 声明一个整型变量 age
String name;                                // 声明一个字符串变量 name
```

变量的初始化是为变量分配初始值的过程。可以在声明时进行初始化，也可以在之后通过赋值语句完成。

在声明时初始化：

```
int age = 25;                               // 声明并初始化整型变量
String name = "John";                       // 声明并初始化字符串变量 name
```

在声明后初始化(赋值)：

```
int age;                                    // 声明整型变量 age
age = 25;                                   // 为 age 赋值 25(初始化)
String name;                                // 声明字符串变量 name
name = "John";                              // 为 name 赋值"John"(初始化)
```

3. 变量的作用域

变量的作用域决定了变量在程序中的可访问范围。Java 中的变量作用域主要分为以下几种。

1) 局部变量

局部变量在方法、构造函数或语句块内部声明。它们仅在声明的方法、构造函数或语句块内可见和可用。

```
public class Main {
    public static void main(String[] args) {
        int localVar = 10;                  // 声明局部变量
        System.out.println(localVar);       // 可以访问局部变量
    }
}
```

2) 成员变量(实例变量)

成员变量在类的内部，但在方法外部声明。它们属于类的每个实例，可以通过对象访问。

```
public class Person {
    int age;                                // 实例变量
// ...
}
```

3) 静态变量(类变量)

静态变量在类的内部，使用 static 关键字声明。它们属于类本身，而不是类的任何实例，可以通过类名直接访问。

```
public class Person {
    static int population = 100;            // 类变量
// ...
}
```

2.3 运算符与表达式

运算符是用于执行特定操作的符号。Java 中的运算符可以分为以下几类。

- 算术运算符：用于执行基本的数学运算。
- 关系运算符：用于比较两个值的关系。
- 逻辑运算符：用于执行布尔逻辑运算。
- 位运算符：用于执行位级别的运算。
- 赋值运算符：用于给变量赋值。
- 其他运算符：如条件运算符、三元运算符等。

1. 算术运算符

算术运算符用于执行基本的数学运算，如表 2-4 所示。

表 2-4　算术运算符

运算符	描述	示例	结果
+	加法	5 + 3	8
-	减法	10 - 4	6
*	乘法	7 * 6	42
/	除法	8 / 2	4
%	取模运算	9 % 2	1
++	自增(前)	a=2;b=++a;	a=3,b=3
++	自增(后)	a=2;b=a++;	a=3,b=2
--	自减(前)	a=2;b=--a;	a=1,b=1
--	自减(后)	a=2;b=a--;	a=1,b=2

2. 关系运算符

关系运算符用于比较两个值的大小或关系，返回布尔值 true 或 false，如表 2-5 所示。

表 2-5　关系运算符

运算符	描述	示例	结果
>	大于	5 > 3	true
<	小于	2 < 4	true
>=	大于等于	5 >= 3	true
<=	小于等于	2 <= 4	true
==	等于	5 == 4	false
!=	不等于	5 != 3	true

3. 逻辑运算符

逻辑运算符用于组合多个条件，返回一个布尔值，如表 2-6 所示。

表 2-6　逻辑运算符

运算符	描述	示例	结果
&&	逻辑与	5 > 3 &&4 < 10	true
‖	逻辑或	5>3‖2>4	true
!	逻辑非	!(5 > 3)	false

4. 赋值运算符

赋值运算符用于将值赋给变量，如表 2-7 所示。

表 2-7　赋值运算符

运算符	描述	示例
=	简单赋值	int a = 5;
+=	加法赋值	a += 3;
-=	减法赋值	a -= 3;
*=	乘法赋值	a *= 3;
/=	除法赋值	a /= 3;
%=	取模赋值	a %= 3;

5. 运算符中的优先级

运算符中的优先级如表 2-8 所示。

表 2-8　运算符中的优先级

优先级	运算符
1	. [] ()
2	++ -- ～ ! (数据类型)
3	* / %
4	+ -

(续表)

优先级	运算符
5	<< >> >>>
6	< > <= >=
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?:
14	= *= /= %= += -= <<= >>= >>>= &= ^= |=

2.4 顺序结构

在 Java 程序中，顺序结构是最基本、最简单的程序结构。顺序结构意味着程序中的语句将按照它们在代码中出现的顺序逐一执行。换句话说，程序会从上到下依次执行每一行代码，直到遇到循环结构、选择结构或程序结束。

顺序结构是构建复杂程序的基础。在顺序结构中，每条语句的执行都依赖于前一条语句的完成。如图 2-1 所示，如果程序中有两条输出语句 A 和 B，那么第二条输出语句 B 将在第一条输出语句 A 执行完毕后才会执行。

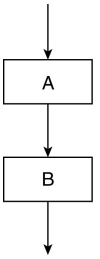

图 2-1　顺序结构流程图

【例 2-1】两个整数相加并输出结果。

```
public class SequentialStructureExample {
    public static  main(String[] args) {
        // 声明变量
        int num1;
        int num2;
        int sum;

        // 赋值操作
        num1 = 5;
        num2 = 10;

        // 计算和输出结果
        sum = num1 + num2;
        System.out.println("The sum of " + num1 + " and " + num2 + " is: " + sum);
    }
}
```

以上代码首先声明 3 个整数变量 num1、num2 和 sum。然后，将 num1 赋值为 5，将 num2 赋值

为 10。接下来，计算 num1 和 num2 的和，并将结果存储在 sum 中。最后，使用 System.out.println 方法输出结果。当运行这个程序时，控制台会输出以下内容。

```
The sum of 5 and 10 is: 15
```

通过这个简单的示例，我们可以看到顺序结构是如何按照代码的书写顺序依次执行的(每个步骤都是按顺序进行的，没有任何跳转或条件判断)。

2.5 选择结构

在 Java 编程中，选择结构是控制程序流程的重要手段之一。它允许我们根据不同的条件执行不同的代码块。选择结构主要包括 if 语句和 switch 语句。

2.5.1 if 条件语句

1. if 语句

if 语句用于在满足特定条件时执行某段代码，其基本语法如下：

```
if (判断条件) {
    执行语句
}
```

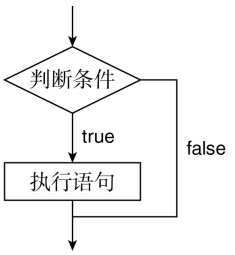

图 2-2 if 语句的执行流程

if 语句的执行流程如图 2-2 所示。

【例 2-2】通过一个案例学习 if 语句的具体用法。

```
public class IfExample {
    public static void main(String[] args) {
        int a = 10;
        if (a > 5) {
            System.out.println("a is greater than 5");


        }
    }
}
```

以上代码定义了一个名为 IfExample 的公共类，并包含一个 main 方法，它是 Java 应用程序的入口点。在 main 方法中，声明并初始化了一个整型变量 a，其值为 10。if(a > 5)是条件表达式，当 a 的值大于 5 时，条件为真。本题中 a=10，10>5，条件为真，程序将执行大括号{}内的代码。在{}内，程序调用 System.out.println("a is greater than 5")输出一行文本到控制台。程序运行结果如图 2-3 所示。

图 2-3 程序运行结果

2. if-else 语句

if-else 语句在 if 的基础上增加了一个备选的代码块，当条件不满足时执行该代码块，其基本语法如下：

```
if (判断条件) {
    执行语句 1
} else {
    执行语句 2
}
```

在上述格式中，判断条件是一个布尔值。当判断条件为 true 时，if 后面{}中的执行语句 1 会执行。当判断条件为 false 时，else 后面{}中的执行语句 2 会执行。执行流程如图 2-4 所示。

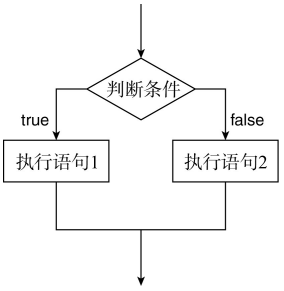

图 2-4　if-else 语句执行流程

【例 2-3】通过一个案例学习 if-else 语句的具体用法。

```
public class IfElseExample {
    public static void main(String[] args) {
        int b = 4;
        if (b > 5) {
            System.out.println("b 大于 5");
        } else {
            System.out.println("b 不大于 5");
        }
    }
}
```

以上代码定义了一个名为 IfElseExample 的公共类，并包含一个 main 方法。在 main 方法中，声明并初始化了一个整型变量 b，其值为 4。if (b > 5)是条件表达式，当 b 的值大于 5 时，条件为真，程序将执行语句块 1；当 b 的值不大于 5 时，程序将执行语句块 2。本题中因为 b 的值是 4，不大于 5，所以条件为假，程序将执行 else 块中的代码，程序调用 System.out.println("b 不大于 5")输出一行文本到控制台。程序运行结果如图 2-5 所示。

图 2-5　程序运行结果

3. if-else if-else 语句

if-else if-else 语句可以处理多个条件，根据不同条件执行相应的代码块，其基本语法如下：

```
if (判断条件1)
{
    执行语句1
}
else if (判断条件2)
{
    执行语句2
}
...
else if (判断条件n)
{
    执行语句n
}
else
{
    执行语句n+1
}
```

根据以上格式，判断条件是一个布尔值。当判断条件 1 为 true 时，if 后面{}中的语句 1 会执行。当判断条件 1 为 false 时，会继续执行判断条件 2，如果判断条件 2 为 true 则执行语句 2，以此类推。如果所有的判断条件都为 false，则意味着所有条件均不满足，else 后面{}中的语句 n+1 会执行。if-else if-else 语句的执行流程如图 2-6 所示。

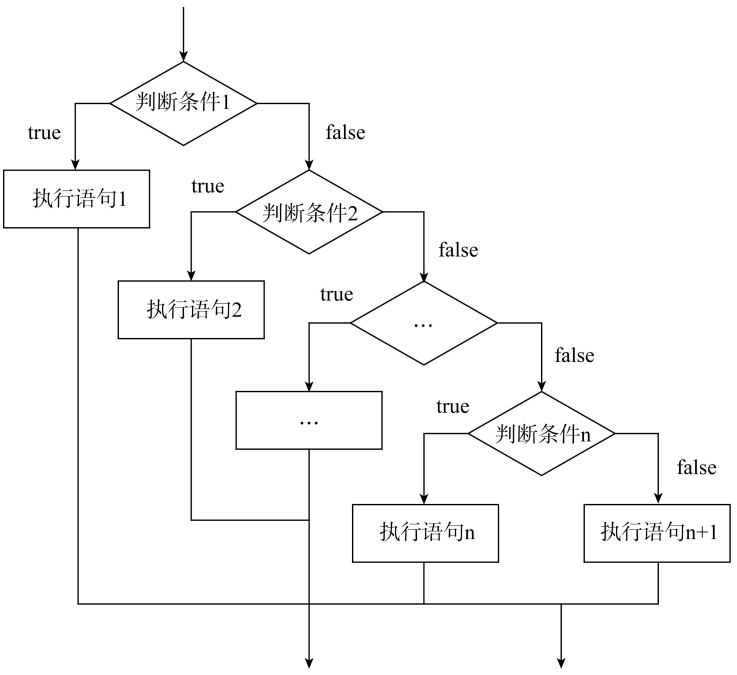

图 2-6　if-else if-else 语句执行流程

【例 2-4】通过一个实现对学生考试成绩进行等级划分的程序演示 if-else if-else 语句。

```
public class IfElseifElseExample {
    public static void main(String[] args) {
        int grade = 75;              // 定义学生成绩
        if (grade>80){
            System.out.println("该成绩的等级为优");
        }
        else if(grade>70) {
            System.out.println("该成绩的等级为良");
        }
        else if(grade > 60) {
            System.out.println("该成绩的等级为中");
        }
        else {
            System.out.println("该成绩的等级为差");
        }
    }
}
```

以上代码定义了一个名为 IfElseifElseExample 的公共类，并包含一个 main 方法。在 main 方法中，声明并初始化了一个整型变量 grade，其值为 75，表示学生的成绩。代码使用多层 if-else if-else 语句来根据成绩判断等级。首先检查 grade 是否大于 80，如果 grade>80 为真，执行 if 块内的代码，输出该成绩的等级为优；如果 grade>80 为假，继续检查下一个条件 grade > 70，如果 grade> 70 为真，执行第一个 else if 块内的代码，输出 "该成绩的等级为良"；如果 grade > 70 为假则继续检查下一个条件 grade > 60，如果 grade > 60 为真则执行第二个 else if 块内的代码，输出"该成绩的等级为中"；如果 grade > 60 为假则执行最后的 else 块内的代码，输出"该成绩的等级为差"。程序中 grade 的值是 75，满足 grade > 70 但不满足 grade > 80，因此程序将执行第一个 else if 块内的代码。在第一个 else if 块内，程序调用 System.out.println("该成绩的等级为良")输出一行文本到控制台。

程序运行结果如图 2-7 所示。

图 2-7　程序运行结果

2.5.2　switch 条件语句

switch 语句用于多分支选择结构，根据变量的值匹配相应的 case 并执行对应的代码块，其基本语法如下：

```
switch (表达式){
    case 目标值1:
        执行语句1
```

```
        break;
    case 目标值2:
        执行语句2
        break;
    ...
    case 目标值n:
        执行语句n
        break;
    default:
        执行语句n+1
        break;
}
```

【例 2-5】通过一个案例学习 switch 条件语句的用法。

```
public class SwitchExample {
    public static void main(String[] args) {
        char grade = 'B';
        switch (grade) {
            case 'A':
                System.out.println("Grade is A");
                break;
            case 'B':
                System.out.println("Grade is B");
                break;
            case 'C':
                System.out.println("Grade is C");
                break;
            default:
                System.out.println("Grade is unknown");
                break;
        }
    }
}
```

以上代码定义了一个名为 SwitchExample 的公共类，并包含一个 main 方法。在 main 方法中，声明并初始化了一个字符变量 grade，其值为 B。接下来，代码使用 switch 语句来根据 grade 的值执行不同的操作。

- case 'A'：检查 grade 是否等于'A'。如果相等，执行大括号{}内的代码，然后执行 break 语句跳出 switch 结构。
- case 'B'：检查 grade 是否等于'B'。如果相等，执行大括号{}内的代码，然后执行 break 语句跳出 switch 结构。
- case 'C'：检查 grade 是否等于'C'。如果相等，执行大括号{}内的代码，然后执行 break 语句跳出 switch 结构。
- default：是一个默认分支，当 grade 的值不匹配任何 case 时执行。

在这个例子中，由于 grade 的值是'B'，所以程序会执行 case 'B'块内的代码。程序调用 System.out.println("Grade is B")输出一行文本到控制台，输出结果如图 2-8 所示。

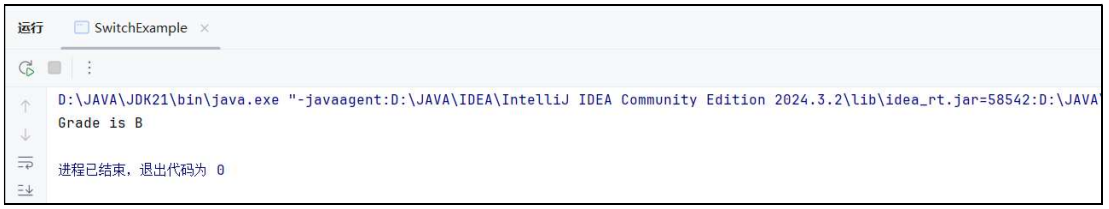

图 2-8　输出结果

2.6 循环结构

循环结构是编程中最基本也是最常用的控制流结构之一，用于重复执行一段代码，直到满足某个条件为止。在 Java 中，主要提供了以下几种循环结构。

- while 循环：适用于未知迭代次数，直到某个条件不再满足时停止循环。
- do-while 循环：类似于 while 循环，其至少会执行一次循环体，即使条件一开始就不满足。
- for 循环：适用于已知迭代次数的情况。

以上循环结构各有其独特的用途和适用场景，通过合理选择和使用它们，可以编写出更加简洁、高效的代码。

2.6.1 while 循环语句

while 循环用于在不知道循环次数的情况下重复执行一段代码，直到条件不再满足为止，其基本语法如下：

```
while (条件表达式) {
    // 循环体
}
```

在以上语法结构中，{}中的执行语句被称作循环体，循环体是否执行取决于循环条件。

当循环条件为 true 时，循环体就会执行。循环体执行完毕，程序继续判断循环条件，如果条件仍为 true 则继续执行循环体，直到循环条件为 false 时，整个循环过程才会结束。

while 循环的执行流程如图 2-9 所示。

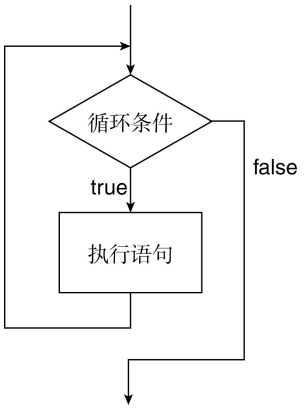

图 2-9　while 循环流程图

【例 2-6】使用 while 循环打印数字 1~10。

```
public class WhileLoopExample {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 10) {
            System.out.println(i);
            i++;        // 更新循环控制变量
        }
```

```
        }
    }
```

以上程序首先初始化变量 i 为 1，然后进入循环体。在每次循环中，先判断 i 是否小于等于 10，如果是，则打印当前 i 的值，并将 i 自动增加 1。这个过程一直持续到 i 大于 10 为止，此时循环结束。程序运行结果如图 2-10 所示。

图 2-10　程序运行结果

2.6.2　do-while 循环语句

do-while 循环与 while 循环类似，但它会至少执行一次循环体，然后再检查条件表达式，其基本语法如下：

```
do {
    // 循环体
} while (条件表达式);
```

在以上语法结构中，关键字 do 后面{}中的执行语句是循环体。do-while 循环语句将循环条件放在了循环体的后面。这意味着循环体会无条件执行一次，然后再根据循环条件决定是否继续执行。

do-while 循环的执行流程如图 2-11 所示。

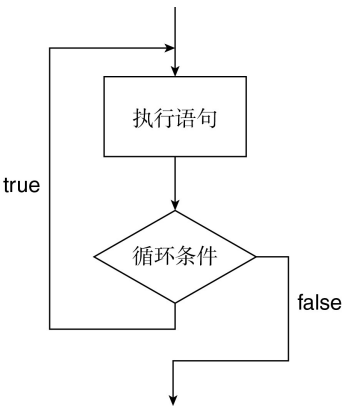

图 2-11　do-while 循环的执行流程

【例 2-7】使用 do-while 循环打印数字 1~10。

```
public class DoWhileLoopExample {
    public static void main(String[] args) {
        int i = 1;
        do {
            System.out.println(i);
            i++;        // 更新循环控制变量
        } while (i <= 10);
    }
}
```

以上程序首先初始化变量 i 为 1，然后进入循环体并打印当前 i 的值。接着，将 i 自动增加 1。

由于 do-while 循环在执行完一次循环体后才会检查条件，所以即使 i 的初始值为 11(即条件为假)，也会打印 11 并继续执行。这个过程一直持续到 i 大于 10 为止，此时循环结束。程序运行结果如图 2-12 所示。

图 2-12　程序运行结果

2.6.3　for 循环语句

for 循环是 Java 中最常用的循环结构之一，特别适合在已知迭代次数的情况下使用，其基本语法如下：

```
for (初始化; 条件表达式; 更新表达式) {
    // 循环体
}
```

- 初始化：设置循环变量的初始值。
- 条件表达式：每次循环开始前都会检查这个条件，如果为真(true)，则执行循环体；如果为假(false)，则退出循环。
- 更新表达式：通常用于更新循环变量的值。

【例 2-8】使用 for 循环打印数字 1~10。

```
public class ForLoopExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

以上程序首先初始化变量 i 为 1，然后进入循环体。在每次循环中，先判断 i 是否小于等于 10，如果是，则打印当前 i 的值，并将 i 自动增加 1。这个过程一直持续到 i 大 10 为止，此时循环结束。程序运行结果如图 2-13 所示。

图 2-13　程序运行结果

2.6.4　循环嵌套

在 Java 中，循环结构可以嵌套使用，即在一个循环体内部再定义另一个循环。嵌套循环常用于处理二维数组、矩阵等问题。

【例 2-9】使用嵌套 for 循环打印一个 5×5 的星号矩阵。

```
public class NestedForLoopExample {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                System.out.print("* ");
            }
            System.out.println();        // 换行
        }
    }
}
```

以上程序首先进入外层循环，变量 i 从 0 开始递增至 4。在外层循环的每次迭代中，进入内层循环，变量 j 从 0 开始递增至 4。在内层循环中，打印星号 "* "，并在内层循环结束后换行，以形成 5×5 的矩阵形式。当外层循环结束后，程序结束，输出结果如图 2-14 所示。

图 2-14　输出结果

2.7 break 与 continue 语句

2.7.1 break 语句

在 switch 条件语句和循环语句中都可以使用 break 语句。当 break 语句出现在 switch 条件语句中时，其作用是终止某个 case 并跳出 switch 结构。当 break 语句出现在循环语句中时，其作用是跳出循环语句，执行循环后面的代码。

【例 2-10】通过一个例子演示如何使用 break 语句来提前退出循环。在这个例子中，当变量 i 的值等于 5 时，break 语句会终止 for 循环，并继续执行循环后面的代码。

```
public class BreakExample {
    public static void main(String[] args) {
        // 定义一个 for 循环，从 i=0 开始，每次循环后 i 递增 1，直到 i=10 为止
        for (int i = 0; i < 10; i++) {
            // 如果 i 等于 5，则执行 break 语句，跳出循环
            if (i == 5) {
                break; // 当 i 等于 5 时，退出循环
            }
            // 打印当前的 i 值
            System.out.println("i 的值是：" + i);
        }
        // 循环结束后，打印"循环结束"
        System.out.println("循环结束");
    }
}
```

程序说明如下。

- 程序首先进入 for 循环，变量 i 从 0 开始递增至 9。
- 在每次循环中，先判断 i 是否等于 5。
- 当 i 等于 5 时，执行 break 语句，退出 for 循环。
- 循环结束后，打印"循环结束"。

程序运行结果如图 2-15 所示。

图 2-15　程序运行结果

注意：当 break 语句出现在嵌套循环中的内层循环时，它只能跳出内层循环，如果想使用 break 语句跳出外层循环，则需要在外层循环中使用 break 语句。

2.7.2 continue 语句

在循环结构中，continue 语句用于跳过当前迭代的剩余代码，直接进入下一次迭代。它通常与 if 条件语句结合使用，当满足某个条件时，通过 continue 语句跳过当前的剩余代码。

【例 2-11】本例用于演示 continue 语句的作用。该 Java 程序的功能是遍历从 0 到 10 的整数，并打印出所有的奇数。

```
public class ContinueExample {
    public static void main(String[] args) {
        // 定义一个 for 循环，从 i=0 开始，每次循环后 i 递增 1，直到 i=10 为止
        for (int i = 0; i < 10; i++) {
            // 如果 i 是偶数，则执行 continue 语句，跳过当前迭代的剩余代码
            if (i % 2 == 0) {
                continue; // 跳过当前迭代的剩余代码
            }
            // 打印当前的 i 值(奇数)
            System.out.println("奇数：" + i);
        }
    }
}
```

程序说明如下。

- 初始化：程序开始时，进入 main 方法。
- 进入 for 循环：for 循环从 i=0 开始，每次循环后 i 递增 1，直到 i=10 为止。
- 条件判断：在每次循环中，首先判断 i 是否是偶数(即 i % 2 == 0)。
- 如果 i 是偶数，执行 continue 语句，跳过当前迭代的剩余代码，直接进入下一次循环。
- 如果 i 不是偶数，打印当前的 i 值。
- 循环结束：当 i 等于或大于 10 时，for 循环结束。

程序运行结果如图 2-16 所示。

图 2-16　程序运行结果

2.8 小结

在本章中，我们深入探讨了 Java 语言的基础知识，包括标识符与关键字、基本数据类型、常量、

变量、运算符与表达式、顺序结构、选择结构和循环结构等重要概念。

(1) 标识符与关键字。我们学习了如何命名标识符，并理解了 Java 中的关键字，这些关键字在 Java 程序中具有固定的用途，不能用作自定义标识符。

(2) 基本数据类型。我们系统地学习了常量和变量的概念，并全面了解了 Java 中的基本数据类型(如整型、浮点型、字符型和布尔型)。

(3) 运算符与表达式。我们全面掌握了各种运算符(如算术、关系、逻辑、赋值等)的使用。同时，深入理解了运算符的优先级规则。

(4) 顺序结构。通过具体示例，我们学习了顺序结构的程序执行方式。顺序结构严格按照代码书写的顺序依次执行。

(5) 选择结构。本章详细介绍了 if 语句、if-else 语句、if-else if-else 语句和 switch 语句。这些结构能够根据条件执行不同的代码块。

(6) 循环结构。我们学习了 while 循环、do-while 循环和 for 循环的使用场景和语法。此外，还讨论了循环嵌套和控制循环流程的 break 与 continue 语句。

通过本章的学习，读者应能够编写简单的 Java 程序，掌握基本的编程技巧和逻辑结构，为后续更复杂的编程任务打下坚实的基础。

2.9 习题

一、选择题

1. 在 Java 中，(　　)是合法的标识符。

A. intValue　　B. _myVariable　　C. my-variable　　D. 9myVariable

2. 下面(　　)是 Java 的关键字。

A. package　　B. class　　C. variable　　D. constant

3. 在 Java 中，整数常量可以使用(　　)表示形式。

A. 二进制　　B. 八进制　　C. 十进制　　D. 以上都是

二、填空题

1. 在 Java 中，布尔常量 true 和 false 分别表示_____和_____。

2. 在 Java 中，字符常量被单引号(' ')所包围，并且仅能包含_____个字符。

3. 在 Java 中，字符串常量是不可变的，这意味着一旦字符串被创建，其内容便_____。

三、编程题

1. 编写一个程序，使用 while 循环计算并输出 1～100 的整数之和。

2. 编写一个程序，使用 for 循环打印一个 5×5 的星号矩阵。

3. 编写一个程序，使用 do-while 循环实现用户输入一个整数，并判断该整数是正数、负数还是零。

4. 编写一个程序，使用 if-else if-else 语句根据用户输入的成绩(整数)输出对应的等级(如 A、B、C、D、F)。