高等学校计算机应用规划教材

Java 程序设计案例教程

秦　军　编著

清华大学出版社

北　京

内 容 简 介

本书是关于 Java 程序设计的案例教程。全书共分为 11 章，包括 Java 语言基础、Java 语言的基本语法、面向对象程序设计、Java 的类型封装器与注解、Java 异常处理机制、常用 Java API、Java I/O 流、Java 多线程编程、Java 传统 GUI 编程、JavaFX GUI 编程以及 JavaFX 绘图与动画等技术。

本教程内容丰富、结构合理、语言简练流畅、案例新颖、针对性强，主要面向 Java 程序设计语言的初学者，适合作为高等院校的 Java 语言基础教材，还可作为 Java 程序设计与开发人员的参考书。

本书对应的电子课件、习题答案和实例源文件可以到 http://www.tupwk.com.cn/downpage 网站下载。

图书在版编目(CIP)数据

Java 程序设计案例教程 / 秦军　编著. —北京：清华大学出版社，2018
(高等学校计算机应用规划教材)
ISBN 978-7-302-48718-0

Ⅰ. ①J… Ⅱ. ①秦… Ⅲ. ①JAVA 语言－程序设计－高等学校－教材 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字(2017)第 272650 号

责任编辑：胡辰浩　李维杰
装帧设计：孔祥峰
责任校对：曹　阳
责任印制：李红英

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　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：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：22.25　　**字　　数**：542 千字
版　　次：2018 年 1 月第 1 版　　**印　　次**：2018 年 1 月第 1 次印刷
印　　数：1~4000
定　　价：56.00 元

产品编号：075902-01

前　言

Java 是当今世界最重要，也是使用最广泛的计算机语言之一。与其他一些计算机语言随着时间的流逝，影响也逐渐减弱不同，Java 随着时间的推移反而变得更加强大。从首次发布开始，Java 就跃到了 Internet 编程的前沿，Java 不断地进行完善以适应编程环境和开发人员编程方式的变化。Java 不仅仅是在跟随潮流，更是在帮助创造潮流。在现实世界中，很多应用都是使用 Java 开发的。本书使用其最新版本 Java SE 8(JDK 8)，从开发环境的搭建讲起，遵循“从简单到复杂”、“从抽象到具体”的原则，结合大量案例，介绍了 Java 程序设计语言的基本语法和编程技巧。

本书共 11 章，第 1 章是 Java 语言概述，主要介绍 Java 语言的发展史、Java 语言的特点、Java 虚拟机的工作原理以及 Java 开发环境的搭建；第 2 章是 Java 编程基础，主要讲述 Java 的基本语法，包括程序的基本元素、变量的作用域、基本数据类型、运算符与表达式、流程控制语句以及数组等内容；第 3 章和第 4 章是学习 Java 的重点，全面介绍面向对象编程的基本思想和关键特性，包括类与对象、继承与多态、内部类与静态类、接口与包、枚举、注解、装箱与拆箱等，理解面向对象是学好 Java 最关键的一点；第 5 章介绍 Java 的异常处理机制，包括异常的捕获与处理、JDK 7 新增的异常特性等；第 6 章简要介绍一些常用的 Java API 类库，重点介绍字符串的处理、java.lang 包和 java.uti 包中的集合框架；第 7 章介绍 Java 的 I/O 流，包括文件操作、字节流和字符流，以及对象序列化等；第 8 章讲述 Java 的多线程编程，包括 Java 中创建线程的方法、线程同步与线程间通信等；第 9 章介绍传统的 Java GUI 编程，包括 AWT 和 Swing 两种框架；第 10 章介绍 Java 的新 GUI 框架——JavaFX，包括 JavaFX 应用程序框架、JavaFX 常用控件、效果与变换以及 JavaFX 菜单等，JavaFX 将在 Java 应用程序的设计方式上扮演重要角色，所以 Java 程序员需要具备 JavaFX 的使用经验；第 11 章介绍 JavaFX 绘图与动画，包括传统的 AWT 绘图以及新的 JavaFX 对绘图和动画的支持等内容。

本书内容丰富、结构合理、思路清晰、语言简练流畅、案例新颖、针对性强。每一章的开始部分概述该章的作用和内容，指出该章的学习目标。正文部分，结合每章的知识点和关键技术，穿插大量极富实用价值的程序案例，所有实例都在 Eclipse+JDK 1.8 环境下调试运行通过。每一章的末尾有本章小结，总结该章的内容和重点与难点；同时安排了有针对性的思考和练习，帮助读者巩固该章所学内容，提高读者的实际动手能力。

本书主要面向 Java 程序设计语言的初学者，适合作为高等院校 Java 语言基础相关课程的教材以及 Java 应用程序开发人员的参考书。

除封面署名的作者外，参加本书编写的人员还有蔡小爱、刘训星、于中海、张小奇、苏文明、龚勇、胡敏、何学成、张海民、袁婷婷、徐宏进、刘钊颖、王玉、薛琛、刘煜、李泽峰、陈华东、王田田、李健男、艾欣和林桂妃等。由于作者水平有限，本书难免有不足之处，欢迎广大读者批评指正。我们的信箱是 huchenhao@263.net，电话是 010-62796045。

本书对应的电子课件、习题答案和实例源文件可以到 http://www.tupwk.com.cn/downpage 网站下载。

作　者

2017 年 8 月

目　　录

第1章　Java语言概述

Java 是一种跨平台的面向对象程序设计语言，自问世以来，受到越来越多开发者的喜爱。它不仅吸收了 C++语言的各种优点，而且摒弃了 C++里难以理解的多继承、指针等概念，因此 Java 语言具有功能强大和简单易用等特征。本章将从 Java 的起源讲起，详细介绍 Java 的发展历程、Java 的特点以及开发环境的搭建，并创建一个简单的 Hello World 程序。

本章学习目标：

- 了解 Java 的发展史
- 理解 Java 语言的特点
- 掌握 Java 程序的运行机制
- 掌握 JDK 的安装与配置
- 掌握在 Eclipse 中新建 Java 工程
- 熟悉 Eclipse 的常用操作

1.1　Java 的起源与发展史

Java 由 Sun 公司于 1995 年推出，在 Java 语言出现以前，很难想象在 Window 环境下编写的程序可以不加修改就在 Linux 系统中运行，因为计算机硬件只识别机器指令，而不同操作系统中的机器指令是有所不同的。所以，要把一种平台下的程序迁移到另一个平台，必须针对目标平台进行修改。如果想要程序能够运行在不同的操作系统下，就要求程序设计语言能够跨平台，可以跨越不同的硬件、软件环境，而 Java 语言就能够满足这种要求。

1.1.1　程序设计语言

程序设计语言(Program Design Language，PDL)又称为编程语言，是一组用来定义计算机程序的语法规则。语言的基础是一组记号和一组规则，根据规则由记号构成的记号串的总体就是语言。在程序设计语言中，这些记号串就是程序。

正如人类交流所使用的自然语言(如汉语、英语、俄语等)一样，人与计算机之间的交流必须使用人和计算机都能理解的程序设计语言。自 20 世纪 60 年代以来，世界上公布的程序设计语言已有上千种之多，但是只有很小一部分得到了广泛应用。从发展历程来看，程序设计语言可以分为 4 类。

1. 机器语言

机器语言由二进制 0、1 代码指令构成，不同的 CPU 具有不同的指令系统。机器语

言程序难编写、难修改、难维护，需要用户直接对存储空间进行分配，编程效率极低。这种语言已经渐渐被淘汰了。

2. 汇编语言

汇编语言指令是机器指令的符号化，与机器指令存在着直接的对应关系，所以汇编语言同样存在难学难用、容易出错、维护困难等缺点。但是汇编语言也有自己的优点：可直接访问系统接口，用汇编程序翻译成的机器语言程序的效率高。从软件工程角度来看，只有在高级语言不能满足设计要求，或不具备支持某种特定功能的技术性能(如特殊的输入/输出)时，汇编语言才被使用。

3. 高级语言

高级语言是面向用户的、基本上独立于计算机种类和结构的语言。其最大的优点是：形式上接近于算术语言和自然语言，概念上接近于人们通常使用的概念。高级语言的一个命令可以代替几条、几十条甚至几百条汇编语言的指令。因此，高级语言易学易用，通用性强，应用广泛。高级语言种类繁多，可以从应用特点和对客观系统的描述两个方面对其进一步分类。

(1) 从应用角度分类

从应用角度来看，高级语言可以分为基础语言、结构化语言和专用语言。

- 基础语言：基础语言也称通用语言。它历史悠久，流传很广，有大量的已开发的软件库，拥有众多的用户，为人们所熟悉和接受。属于这类语言的有 FORTRAN、COBOL、BASIC、ALGOL 等。
- 结构化语言：20世纪70年代以来，结构化程序设计和软件工程的思想日益为人们所接受和欣赏。在它们的影响下，先后出现了一些很有影响的结构化语言，这些结构化语言直接支持结构化的控制结构，具有很强的过程结构和数据结构能力。Pascal、C、Ada 语言就是它们的突出代表。
- 专用语言：是为某种特殊应用而专门设计的语言，通常具有特殊的语法形式。一般来说，这种语言的应用范围狭窄，移植性和可维护性不如结构化程序设计语言。随着时间的流逝，被使用的专业语言已有数百种，应用比较广泛的有 APL 语言、Forth 语言、LISP 语言。

(2) 从对客观系统的描述分类

从描述客观系统来看，程序设计语言可以分为面向过程语言和面向对象语言。

- 面向过程语言：以“数据结构+算法”程序设计范式构成的程序设计语言，称为面向过程语言。前面介绍的程序设计语言大多为面向过程语言。
- 面向对象语言：以“对象+消息”程序设计范式构成的程序设计语言，称为面向对象语言。比较流行的面向对象语言有 Delphi、Visual Basic、Java、C++等。

Java 语言是一种面向对象的、不依赖于特定平台的程序设计语言，简单、可靠、可编译、可扩展、多线程、结构中立、类型显式声明、动态存储管理、易于理解，是一种理想的、用于开发 Internet 应用软件的程序设计语言。

4. 非过程化语言

非过程化语言是面向应用，为最终用户设计的一类程序设计语言，具有缩短应用开发过程、降低维护代价、最大限度地减少调试过程中出现的问题以及对用户友好等优点。

非过程化语言在编码时只需说明“做什么”，不需描述算法细节。数据库查询和应用程序生成器是这类语言的两个典型应用。

- 用户可以用数据库查询语言(SQL)对数据库中的信息进行复杂的操作。用户只需将要查找的内容在什么地方、 根据什么条件进行查找等信息告诉 SQL，SQL 将自动完成查找过程。
- 应用程序生成器则是根据用户的需求“自动生成”满足需求的高级语言程序。

1.1.2　Java 的家世

Java 继承了 C 和 C++语言的许多优点。为了更好地理解 Java 语言的特点，我们将从 C 语言的出现讲起，看一下 Java 产生的背景和原因。

1. 现代编程语言的诞生：C 语言

C 语言的诞生震惊了计算机界，因为它从根本上改变了编程的方式和思想。

在 C 语言出现以前，程序员通常需要在品质不同的各种计算机语言之间进行选择。例如，FORTRAN 语言适于编写科学计算应用程序，但是对于编写系统代码则不是很好；BASIC 语言易于学习，但它的功能不是很强大，并且由于缺少结构化设计，很难将其应用于大型程序；汇编语言可以生成非常高效的代码，但是它不易于学习，且调试困难，所以使用效率低。所以，C 语言的产生是人们对结构化、高效率、高级语言需求的直接结果。

C 语言是由 Dennis Ritchie 在运行 UNIX 操作系统的 DEC PDP-11 机器上发明并首次实现的，它是老式 BCPL 语言不断发展的结果，BCPL 语言是由 Martin Richards 开发的。BCPL 语言对 Ken Thompson 发明的 B(取 BCPL 的首字母)语言产生了影响，1972 年，Dennis Ritchie 又在 B 语言的基础上设计出了 C(取 BCPL 的第二个字母)语言。

C 语言具有较强的生命力，其主要特点有：

- 语言简洁、紧凑，使用方便、灵活；
- 运算符丰富，数据类型丰富，具有现代语言的各种数据结构；
- 具有结构化的控制语句(如 if…else 语句等)；
- 语法限制不太严格，程序设计自由度大；
- C 语言允许直接访问物理地址，能执行位(bit)操作，能实现汇编语言的大部分功能，可以直接对硬件进行操作；
- 生成的目标代码质量高，程序执行效率高；
- 与汇编语言相比，用 C 语言编写的程序的可移植性好。

从 20 世纪 70 年代晚期到 80 年代早期，C 语言成为主要的计算机编程语言，并且在今天仍然被广泛使用。

2. C++的出现

高级语言的出现为程序员提供了更多用于处理复杂性的工具。20 世纪 60 年代诞生了结构化编程(structured programming)。这种编程方法被 C 语言这类语言采用。通过使用结构化编程语言，程序员第一次能够比较容易地编写出相对复杂的程序。但是，使用结构化编程方法，一旦项目达到一定的规模，它的复杂性就会超出程序能够管理的范围。到了 20 世纪 80 年代早期，许多项目超出了结构化方法的极限。为了解决这一问题，发明了一种新的编程方法，称为面向对象编程(Object-Oriented Programming，OOP)。

说明：

面向对象编程将在本书后面详细讨论，但是在此先给出它的简短定义：OOP 是一种编程方法论，通过使用继承、封装和多态来帮助组织复杂的程序。

随着面向对象编程思想的出现，C 语言作为一种面向过程的编程语言，已经不能满足运用面向对象方法开发软件的需要。C++便是在 C 语言的基础之上为支持面向对象的程序设计而设计的。

最初，C++被称为“C with Class”，1983 年正式取名为 C++，其标准化工作从 1989 年开始。1994 年，ISO 制定了 ANSI C++标准草案。1998 年 11 月，ISO 正式批准了 C++的国际标准。C++通过添加面向对象特征对 C 语言进行了扩展。因为 C++构建于 C 语言的基础之上，所以它包含了 C 语言的全部特征以及优点，这是 C++作为一种语言能够成功的关键原因。发明 C++语言不是试图创建一种全新的编程语言，而是对已经取得极大成功的 C 语言的改进。

C++语言的主要特点表现在两个方面：一是全面兼容 C，二是支持面向对象的方法。C++既支持面向过程的程序设计，又支持面向对象的程序设计。

3. Java 出现的时机已经成熟

到了 20 世纪 80 年代末 90 年代初，使用面向对象编程的 C++语言占据了主导地位。确实，程序员好像一度找到了完美的语言。因为 C++既支持面向对象编程模式，又具有 C 语言的高效率以及风格优点，它确实是一种可以用于创建各种程序的语言。然而，就像过去一样，推动计算机语言向前演变的力量又一次在酝酿。在短短的几年中，万维网(World Wide Web)和 Internet 达到了临界规模。这一事件又将会促成编程的另一场革命。

1.1.3　Java 的诞生

Java 是由 James Gosling、Patrick Naughton、Chris Warth、Ed Frank 和 Mike Sheridan 于 1991 年在 Sun 公司构想出来的。开发第一个版本花费了 18 个月。这种语言最初被称为 OAK，是 Sun 公司为一些消费类电子产品而设计的通用环境。他们最初的目的只是为了开发一种独立于平台的软件技术，而且在网络出现之前，OAK 可以说是默默无闻，甚至差点夭折。但是，网络的出现改变了 OAK 的命运。在互联网出现的早期，Internet 上的信息内容都是一些乏味死板的 HTML 文档。这对于那些迷恋于 Web 浏览的人们来说简直不可容忍。他们迫切希望能在 Web 中看到一些交互式的内容，开发人员也极希望能够在 Web 中

创建一类无须考虑软硬件平台就可以执行的应用程序，当然这些程序还要有极大的安全保障。对于用户的这种要求，传统的编程语言显得无能为力，而 Sun 的工程师敏锐地察觉到了这一点，从 1994 年起，他们开始将 OAK 技术应用于 Web 上，并且开发出了 HotJava 的第一个版本。当 Sun 公司 1995 年正式以 Java 这个名字推出的时候，几乎所有的 Web 开发人员都料到：噢，这正是我想要的。于是 Java 成了一颗耀眼的明星，随着互联网的发展，Java 被推到计算机语言设计的最前沿。

Java 从 C 和 C++继承了许多特性，这是有意而为之。使用与 C 语言类似的语法以及模仿 C++的面向对象特性，可以使 Java 语言对于众多经验丰富的 C/C++程序员更具吸引力。

虽然 Java 深受 C++的影响，但它不是 C++的增强版，也不是“Internet 版的 C++”。无论是在实践上，还是在理论上，Java 都与 C++有着很大的区别。例如，Java 与 C++既不向上兼容，也不向下兼容。设计 Java 的目的不是取代 C++，Java 是针对解决特定的一系列问题而设计的，Java 和 C++将会长期共存。

计算机语言的发展取决于两个因素：适应环境的变化以及实现编程艺术的提高。促使 Java 发展的环境变化是对平台独立程序的需求，Internet 上的分布式系统天生就需要平台独立的程序。同时，Java 也体现了编程方式的变化。例如，Java 增强并改进了 C++的面向对象编程，增加了对多线程的支持，提供了简化 Internet 访问的库。Java 对于 Internet 编程的意义，就如同 C 语言对于系统编程一样：它们都是改变世界的革命性力量。

1.1.4　Java 的发展历程

Java 自问世以来，持续以爆炸性的步伐向前发展，本节将介绍其发展历程中几个比较重要的里程碑。

1. Java 1.x

1996 年 1 月，Sun 公司发布了 Java 的第一个开发工具包(JDK 1.0)，这是 Java 发展历程中的重要里程碑，标志着 Java 成为一种独立的开发工具。当年 9 月，约 8.3 万个网页应用了 Java 技术来制作。当年 10 月，Sun 公司发布了 Java 平台的第一个即时(Just-In-Time，JIT)编译器。

在 JDK 1.0 发布不久，Java 的设计人员就着手创建下一个版本。1997 年 2 月，JDK 1.1 面世，新版本添加了许多新的库元素，改进了事件处理方式，并且重新配置了 JDK 1.0 中库的许多特性，也去掉了最初版本中的一些特性。

2. Java 2

1998 年 12 月 8 日，第二代 Java 平台的企业版 J2EE 发布。1999 年 6 月，Sun 公司发布了第二代 Java 平台(简称为 Java 2)的 3 个版本：J2ME(Java 2 Micro Edition，Java2 平台的微型版)，应用于移动、无线及有限资源的环境；J2SE(Java 2 Standard Edition，Java 2 平台的标准版)，应用于桌面环境；J2EE(Java 2 Enterprise Edition，Java 2 平台的企业版)，应用于基于 Java 的应用服务器。Java 2 平台的发布，是 Java 发展过程中最重要的一个里程碑，标志着 Java 的应用开始普及。Java 2 添加了大量新特性，例如 Swing 和集合框架，并且改进了 Java 虚拟机和各种编程工具。Java 2 也建议不再使用某些特性。最重要的影响是 Thread

类，建议不再使用该类的 suspend( )、resume( )和 stop( )等方法。

1999 年 4 月 27 日，HotSpot 虚拟机发布。HotSpot 虚拟机发布时是作为 JDK 1.2 的附加程序提供的，后来它成为 JDK 1.3 及之后所有版本的 Sun JDK 的默认虚拟机。

2000 年 5 月，JDK 1.3、JDK 1.4 和 J2SE 1.3 相继发布，几周后获得了 Apple 公司 Mac OS X 的工业标准的支持。J2SE 1.3 是对 Java 2 原始版本的第一次重要升级。这次升级主要是更新 Java 的现有功能以及“限制”开发环境。2001 年 9 月 24 日，J2EE 1.3 发布。

2002 年 2 月 26 日，J2SE 1.4 发布。J2SE 1.4 进一步增强了 Java，这个发布版本包含了一些重要的升级、改进和新增功能。自此 Java 的计算能力有了大幅提升，与 J2SE 1.3 相比，其多了近 62%的类和接口，添加了新的关键字 assert、链式异常(chained exception)以及基于通道的 I/O 子系统。该版本还提供广泛的 XML 支持、安全套接字(Socket)支持(通过 SSL 与 TLS 协议)、全新的 I/O API、正则表达式等。

3. J2SE 5

J2SE 1.4 之后的下一个发布版本是 J2SE 5(内部版本号 1.5.0)，该版本也是革命性的，于 2004 年 9 月 30 日发布，成为 Java 语言发展史上的又一里程碑。它与先前的大多数 Java 升级不同，因为那些升级提供了重要但却有规律的改进，而 J2SE 5 从根本上扩展了 Java 语言的应用领域、功能和范围。J2SE 5 的新特性如下：

- 泛型
- 注解(annotation)
- 自动装箱和自动拆箱
- 枚举
- 增强的 for-each 风格的 for 循环
- 可变长度参数(varargs)
- 静态导入
- 格式化的 I/O
- 并发实用工具

上述新特性不是细枝末节的改动或增量式的升级，列表中的每一项都表示对 Java 语言的重大弥补。某些新特性，比如泛型、增强的 for-each 风格的 for 循环以及可变长度参数，引入了新的语法元素；自动装箱和自动拆箱，改变了 Java 语言的语义；注解为编程增加了一个全新的维度。所有这些新增特性的影响都超出了它们的直接效果。

4. Java SE 6

2005 年 6 月，在 Java One 大会上，Sun 公司发布了 Java SE 6，内部的开发版本号是 1.6。此时，Java 的各种版本已经更名，已取消其中的数字 2，如 J2EE 更名为 Java EE，J2SE 更名为 Java SE，J2ME 更名为 Java ME。Java 开发工具包叫作 JDK 6。

2006 年 11 月 13 日，Java 技术的发明者 Sun 公司宣布，将 Java 技术作为免费软件对外发布。Sun 公司正式发布有关 Java 平台标准版的第一批源代码，以及 Java 迷你版的可执行源代码。从 2007 年 3 月起，全世界所有的开发人员均可对 Java 源代码进行修改。

Java SE 6 建立在 J2SE 5 的基础之上，进行了一些增量式的改进。Java SE 6 没有为 Java

语言添加真正重要的新特性，但它确实增强了 API 库，添加了几个新的包，并且对运行时进行了改进。随着几次升级，在漫长的生命周期中，Java SE 6 还进行了几次更新。总之，Java SE 6 进一步巩固了 J2SE 5 的发展成果。

5. Java SE 7

2009 年，Oracle 公司宣布收购 Sun 公司。2010 年，Java 编程语言的共同创始人之一詹姆斯 • 高斯林从 Oracle 公司辞职。2011 年，Oracle 公司举行了全球性的活动，以庆祝 Java 7 的推出，随后 Java SE 7 正式发布，内部版本号为 1.7。Java SE 7 是自从 Sun 被 Oracle 公司收购之后第一个重要的发布版本。Java SE 7 包含许多新特性，包括为 Java 语言增加的重要特性和 API 库，并且对 Java 运行时系统进行了升级，升级的内容包括对非 Java 语言的支持。不过对于 Java 开发人员来说，他们最感兴趣的还是为语言和 API 增加的特性。

尽管这些新特性被集中描述为“小的”修改，但就它们对代码的影响而言，这些修改产生的影响却相当大。对于许多开发人员，这些修改可能是 Java SE 7 中最重要的新特性：

- String 能够控制 switch 语句。
- 二进制整型字面值。
- 数值字面值中的下画线。
- 扩展的 try 语句，称为带资源的 try(try-with-resources)语句，这种 try 语句支持自动资源管理(例如，当流(stream)不再需要时，现在能够自动关闭它们)。
- 构造泛型实例时的类型推断(借助菱形运算符“<>”)。
- 对异常处理进行了增强，单条 catch 子句能够捕获两个或更多个异常(multi-catch)，并且对重新抛出的异常提供了更好的类型检查。
- 对与某些方法(参数的长度可变)类型关联的编译器警告进行了改进，并且对警告具有更大的控制权。

Java SE 7 为 Java API 库新增了一些内容。其中最重要的两个方面是对 NIO 框架进行了增强并且增加了 Fork/Join 框架。NIO(New I/O)是在 1.4 版本中被添加到 Java 中的。然而，Java SE 7 对 NIO 的增强从根本上扩展了它的功能；Fork/Join 框架对并行编程(parallel programming)提供了重要支持。并行编程通常是指有效使用具有多个处理器(包括多核系统)的计算机的技术。多核环境提供的优点是可以在相当大的程度上提高程序的性能。

6. Java SE 8

2014 年 3 月，Oracle 发布 Java SE 8。这也是最新的 Java 发布版本，对应的 Java 开发工具包称为 JDK 8，内部版本号为 1.8。JDK 8 是 Java 语言的重要升级，包含了一个影响深远的新语言特性：lambda 表达式。

lambda表达式为Java添加了函数式编程特性。在这个过程中，lambda表达式可以简化并减少创建特定结构(如某些类型的匿名类)所需的源代码量。lambda表达式的引入对Java库也产生了广泛的影响，Java库中有一些新功能就是为了使用lambda表达式而添加的。其中最重要的是新的流API，包含在java.util.stream包中。流API支持对数据执行管道操作，并针对lambda表达式做了优化。另外一个重要的新包是java.util.function，它定义了许多函数式接口，为lambda表达式提供了额外的支持。在整个API库中，还可以找到许多与lambda

表达式相关的新功能。

从 JDK 8 开始，可以为接口指定的方法定义默认实现。如果没有为默认方法创建实现，就使用接口定义的默认实现。这种特性允许接口随着时间优雅地演化，因为在向接口添加新方法时，不会破坏现有代码。在默认实现更加合适时，这也有助于简化接口的实现。

JDK 8 中的其他新特性包括新的时间和日期 API、类型注解，以及在对数组进行排序时使用并行处理等。JDK 8 也捆绑了对 JavaFX 8 的支持，JavaFX 8 是 Java 新的 GUI 应用框架的最新版本。预计 JavaFX 很快会在几乎全部 Java 应用程序中扮演重要的角色，并最终取代 Swing，成为大多数基于 GUI 的项目的首选。

最后，Java SE 8 大大扩展了 Java 语言的功能，并改变了编写 Java 代码的方式。其影响将在随后多年的 Java 世界中持续存在。本书的内容将使用最新版本 Java SE 8，并体现 Java SE 8 的新特性。

1.2　Java 的特点

促使 Java 诞生的基本动力是可移植性和安全性，但是在 Java 语言最终成型的过程中，其他因素也扮演了重要角色。下面就来看一下 Java 语言的一些特点。

1.2.1　简单性

Java 的设计目标之一是让专业程序员能够高效地学习和使用。Java 继承了 C/C++的语法以及许多面向对象特性，设计者们把 C++语言中一些复杂且容易出错的特征去掉了，例如，Java 不支持 goto 语句，代之以提供 break 和 continue 语句以及异常处理：Java 还剔除了 C++的操作符重载和多继承特征；另外，因为 Java 没有结构，数组和字符串都是对象，所以不需要指针；Java 能够自动处理对象的引用和间接引用，实现自动的无用单元收集，使用户不必为存储管理问题烦恼，将更多的时间和精力花在研发上。

对于一位有经验的 C++程序员，只需要非常少的努力就可以使用 Java 进行程序开发。对于初学者，只要理解了面向对象编程的基本概念，学习 Java 也会变得非常容易。

1.2.2　面向对象

C++是一种经典的面向对象程序设计语言，Java 继承了 C++中面向对象的理论，但是简化了这种面向对象的技术，去掉了一些复杂的技术，例如多继承、运算符重载等。经过这样的处理，Java 中的面向对象技术变得更简单。

通过大量借鉴过去几十年中的诸多对象软件环境，Java 设法在纯进化论者的“任何事物都是对象”模式和实用主义者的“够用就好”模式之间找到了平衡。Java 中的对象模型既简单又易于扩展，而基本类型(例如整型)仍然是高性能的非对象类型。

在一个面向对象的系统中，类(class)是数据和操作数据的方法的集合。数据和方法一起描述对象(Object)的状态和行为。每一个对象是其状态和行为的封装。类是按一定体系和层次安排的，子类可以从父类继承行为。

Java 还包括一个类的扩展集合，分别组成各种程序包(package)，用户可以在自己的程

序中使用。例如，Java 处理输入/输出的类通常位于 java.io 包中。

1.2.3　可移植性

可移植性(跨平台)是 Java 语言最大的优势，在 Java 中，并不是直接把源文件编译成硬件可以识别的机器指令，Java 的编译器把 Java 源代码编译为字节码文件，这种字节码文件就是编译 Java 源程序时得到的 class 文件，Java 语言的跨平台主要是指字节码文件可以在任何软硬件平台上运行，而执行这种 class 文件的就是 Java 虚拟机，Java 虚拟机是软件模拟出的计算机，可以执行编译 Java 源文件后得到的字节码文件，而各种平台的差异就是由 Java 虚拟机来处理的，由 Java 虚拟机把字节码文件解释成目标平台可以识别的机器指令，从而实现了 Java 的“一次编译，多处运行”的优势。

1.2.4　安全性

在 C/C++中，指针的使用是一个高级话题，熟练掌握指针可以给程序的开发带来很大的方便，但是如果使用不当，就有可能造成系统资源泄漏。更严重的是，错误的指针操作有可能非法访问系统的地址空间，从而给系统带来灾难性的破坏。所以，在 C/C++中，在使用指针的时候，需要非常小心。

Java 语言放弃了指针操作，不提供对存储空间直接访问的方法，所有的存取过程都由 Java 语言自身处理，这样就可以保证系统的地址空间不会被有意或无意破坏。而且经过这样的处理，也可以避免系统资源的泄漏。例如在 C\C++中，如果指针不及时释放，就会占用系统内存空间，大量的指针不及时释放就有可能耗尽可用的内存空间。在 Java 中就不用担心这样的问题，Java 提供了一套有效的资源回收机制，会自动回收不再使用的系统资源，从而保证了系统的安全性和稳定性。

另外，Java 虚拟机在运行字节码文件时，会把 Java 程序的代码和数据限制在具体的内存空间内，不允许 Java 程序占用指定内存地址之外的空间，这样就可以保证 Java 程序不会破坏系统的内存空间，从而保证系统的安全性。

1.2.5　分布式

Java 是针对 Internet 的分布式环境而设计的，因为它能处理 TCP/IP 协议。实际上，使用 URL 访问资源与访问文件没有多大区别。Java 既支持各种层次的网络连接，又以 Socket 类支持可靠的流(stream)网络连接，所以用户可以产生分布式的客户机和服务器。

Java 还支持远程方法调用(Remote Method Invocation，RMI)。这个特性允许程序通过网络调用方法。

1.2.6　解释执行和高性能

Java 通过将源代码编译成 Java 字节码的中间表示形式，可以创建跨平台的程序。Java 字节码提供对体系结构中性的目标文件格式，代码被设计成可有效地传送到多个平台，可以在所有实现了 Java 虚拟机的系统上运行。

在 Java 以前，大部分对跨平台解决方案的尝试对性能的影响太大；而 Java 字节码经过了仔细设计，通过使用即时编译器，可以很容易地将字节码直接转换为高性能的本机代

码。Java 运行时系统提供了这个特性，并且没有丢失平台独立代码的优点。

1.2.7　健壮性

Web 的多平台环境对程序有特别的要求，因为程序必须在各种系统中可靠地执行。因此，在设计 Java 时，使其具备创建健壮程序的能力被提到了高优先级的地位。为了获得可靠性，Java 在一些关键领域进行了限制，从而迫使程序员在程序开发中及早地发现错误。Java 是一门强类型语言，具有允许扩展编译时检查潜在类型不匹配问题的功能。Java 要求显式的方法声明，不支持 C 语言风格的隐式声明。这些严格的要求保证编译程序能捕捉调用错误，这就导致更可靠的程序。

在传统的编程环境中，内存管理是一项困难、乏味的工作。例如，在 C/C++中，程序员必须手动分配和释放所有动态内存，这就可能导致一些问题，因为程序员可能会忘记释放以前分配的内存，或者更糟糕的是，试图释放程序其他部分仍然在使用的内存。而 Java 的垃圾回收机制从根本上消除了这些问题，因为 Java 为不再使用的对象提供了垃圾回收功能，释放内存完全是自动的。

异常处理是 Java 使得程序更稳健的另一个特征。异常是某种类似于错误的异常条件出现的信号。在 Java 程序设计中，可以使用 try/catch/finally 语句，对各种异常和错误进行处理，例如空指针异常、数组越界异常、类型错误等。这种异常机制，可以捕捉到程序中的所有异常，针对不同的异常，可以采取具体的处理方法，从而保证应用程序在用户的控制下运行，继而保证程序的稳定性和健壮性。

1.2.8　多线程

Java 的设计目标之一是满足对创建交互式、网络化程序的现实需求。为了满足这一目标，Java 支持多线程编程，允许编写同时执行许多工作的程序。Java 运行时系统为多线程同步提供了完善的解决方案，能够创造出运行平稳的交互式系统。Java 提供了易用的多线程方法，使得程序开发人员只需要考虑程序的特定行为，而不需要考虑多任务子系统。

1.2.9　动态性

Java 语言被设计成适应于变化的环境，是一门动态语言。Java 程序本身带有大量的运行时类型信息，这些信息可以用于在运行时验证和解决对象访问问题。对于那些可以在运行的系统中动态更新小段字节码的 Java 环境的健壮性来说，这一特性也是很关键的。

1.3　Java 虚拟机

Java 语言的一个非常重要的特点就是平台无关性，使用 Java 虚拟机(Java Virtual Machine，JVM)是实现这一特点的关键。下面就来看一下什么是 JVM 以及 JVM 的工作原理。

1.3.1　什么是 JVM

JVM 是一种用于计算设备的规范，是一台虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。Java 虚拟机包括一套字节码指令集、一组寄存器、一个栈、一个垃圾回收堆和一个存储方法域。JVM 屏蔽了与具体操作系统平台相关的信息，使 Java 程序只需生成在 Java 虚拟机上运行的目标代码(字节码)，就可以在多种平台上不加修改地运行，这就是 Java 能够实现“一次编译，到处运行”的原因。JVM 在执行字节码时，实际上最终还是把字节码解释成具体平台上的机器指令来执行。

JVM 是 JRE 的一部分，它有自己完善的硬件架构，如处理器、堆栈、寄存器等，还具有相应的指令系统；而 JRE(Java Runtime Environment，Java 运行环境)，也就是 Java 平台，所有的 Java 程序都要在 JRE 下才能运行。普通用户要运行已开发好的 Java 程序，只需要安装 JRE 即可。

JDK(Java Development Kit)是程序开发者用来编译、调试 Java 源程序的开发工具包。JDK 中的工具也是 Java 程序，也需要 JRE 才能运行。为了保持 JDK 的独立性和完整性，在 JDK 的安装过程中，JRE 也是安装的一部分。所以，在 JDK 的安装目录中会有一个名为 jre 的子目录，用于存放 JRE 文件。

1.3.2　JVM 的工作原理

JVM 是 Java 的核心与基础，是一种利用软件方法实现的、抽象的计算机基于底层的操作系统和硬件平台，可以在上面执行 Java 的字节码程序。

Java 程序的编译和运行过程如图 1-1 所示。

Java 编译器只要面向 JVM，生成 JVM 能理解的字节码文件，然后通过 JVM 将每一条指令翻译成不同平台的机器码，通过特定平台来运行。

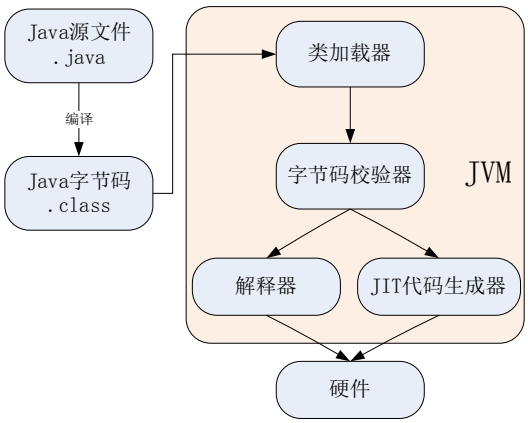

图 1-1　Java 程序的编译和运行过程

1. JVM 执行程序的过程

JVM 是 Java 程序的运行环境，但是它同时也是操作系统的一个应用程序。因此，它也有自己运行的生命周期，并且有自己的代码和数据空间。JVM 在整个 JDK 中处于最底层，负责与操作系统交互，用来屏蔽操作系统环境，提供完整的 Java 运行环境，因此也叫作虚拟计算机。

为操作系统装入 JVM 是通过 JDK 中的 java.exe 来完成的，通过下面 4 个步骤来完成 JVM 环境的搭建：

(1) 创建 JVM 装载环境和配置。

(2) 装载 JVM.dll。

(3) 初始化 JVM.dll 并挂接到 JNIEnv(JNI 调用接口)实例。

(4) 调用 JNIEnv 实例，装载并处理 class 类。

JVM 执行程序的过程如下：

(1) 加载.class 文件。

(2) 管理并分配内存。

(3) 执行垃圾收集。

2. JVM 的生命周期

一个运行中的 JVM 有着一个清晰的任务：执行 Java 程序。程序开始执行时它才运行，程序结束时它就停止。如果在同一台机器上运行 3 个 Java 程序，就会有 3 个运行中的 JVM。

一个 JVM 实例对应一个独立运行的 Java 程序，它是进程级别的；JVM 执行引擎实例则对应属于用户运行程序的线程，它是线程级别的。

1) JVM 实例的诞生

当启动一个 Java 程序时，一个 JVM 实例就产生了，任何一个拥有 public static void main(String[] args)函数的类都可以作为 JVM 实例运行的起点。在程序执行时，必须给 JVM 指明这个包含 main( )方法的类的名称。

2) JVM 实例的运行

main( )方法作为程序初始线程的起点，任何其他线程均由该线程启动。JVM 内部有两种线程：守护线程和非守护线程。main( )方法属于非守护线程，守护线程通常由 JVM 自己使用，Java 程序也可以标明自己创建的线程是守护线程。只要 Java 虚拟机中还有普通的线程在执行，JVM 就不会停止。

3) JVM 实例的消亡

当程序中的所有非守护线程都终止时，JVM 才退出；若安全管理器允许，程序也可以使用 Runtime 类或 System.exit( )来退出。

3. JVM 的体系结构

JVM 的内部体系结构分为三部分：类装载器(Class Loader)、执行引擎和运行时数据区。如图 1-2 所示。

- 类装载器

类装载器的作用是用来装载.class 文件。JVM 将整个类加载过程分为 3 个步骤：

(1) 装载。装载过程负责找到二进制字节码并加载至 JVM 中，JVM 根据类名、类所在的包名通过 Class Loader 来完成类的加载。同样，也采用以上 3 个元素来标识一个被加载的类：类名+包名+Class Loader 实例 ID。

(2) 链接。链接过程负责对二进制字节码的格式进行校验、初始化装载类中的静态变量以及解析类中调用的接口、类。在完成校验后，JVM 初始化类中的静态变量，并将其值赋为默认值。

(3) 初始化。初始化过程即执行类中的静态初始化代码、构造器代码以及静态属性的初始化，在如下 4 种情况下初始化过程会被触发执行：调用了 new；反射调用了类中的方法；子类调用了初始化类；在 JVM 启动过程中指定了初始化类。

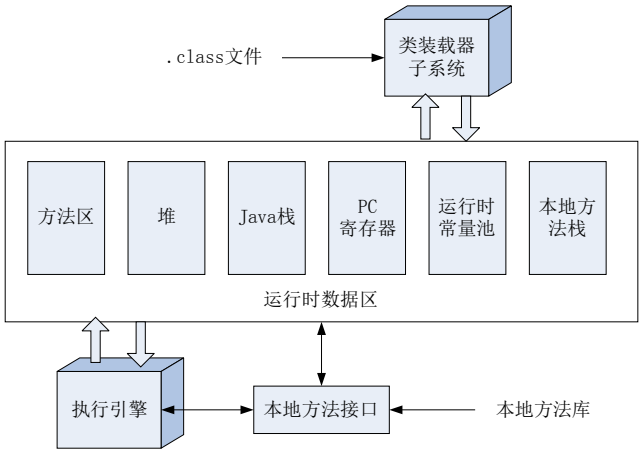

图 1-2　JVM 的内部体系结构

- 执行引擎

执行引擎的作用是执行字节码或本地方法。在执行过程中，JVM 采用的是自己的一套指令系统，每个线程在创建后，都会产生程序计数器(PC)和栈(Stack)。其中，在程序计数器中存放了下一条将要执行的指令，在栈中存放 Stack Frame，表示当前正在执行的方法，每个方法的执行都会产生 Stack Frame，Stack Frame 中存放了传递给方法的参数、方法内的局部变量以及操作数栈。操作数栈用于存放指令运算的中间结果，指令负责从操作数栈中弹出参与运算的操作数，指令执行完毕后再将计算结果压回到操作数栈。当方法执行完毕后则从栈中弹出，继续其他方法的执行。

- 运行时数据区

JVM 在运行时将数据划分为 6 个区域来存储，而不仅仅是大家熟知的 Heap 区域，如图 1-3 所示。

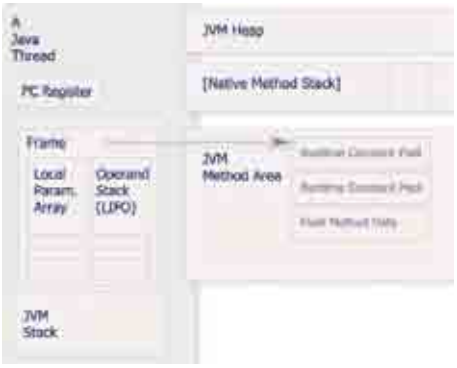

图 1-3　JVM 内存结构

第一块：PC 寄存器(PC Register)

PC 寄存器用于存储每个线程下一步将执行的 JVM 指令，如果方法为本地的，则 PC 寄存器中不存储任何信息。

第二块：JVM 栈(JVM Stack)

JVM 栈是线程私有的，每个线程创建的同时都会创建 JVM 栈。JVM 栈中存放的为当前线程中局部基本类型的变量、部分返回结果以及 Stack Frame。非基本类型的对象在 JVM 栈中仅存放一个指向堆的地址。

第三块：堆(Heap)

堆是 JVM 用来存储对象实例以及数组值的区域，可以认为 Java 中所有通过 new 创建的对象的内存都在此分配。Heap 中对象的内存需要等待垃圾回收机制进行回收。

第四块：方法区域(Method Area)

方法区域存放了所加载的类的信息(名称、修饰符等)、类中的静态变量、类中定义为 final 类型的常量、类中的 Field 信息、方法信息。当开发人员在程序中通过 Class 对象中的

getName( )、isInterface( )等方法来获取信息时，这些数据都来源于方法区域。

第五块：运行时常量池(Runtime Constant Pool)

运行时常量池中存放的是类中固定的常量信息、方法和 Field 的引用信息等，其空间从方法区域中分配。

第六块：本地方法栈(Native Method Stacks)

JVM 采用本地方法栈来支持本地方法的执行，此区域用于存储每个本地方法调用的状态。

1.3.3　JVM 垃圾回收

JVM 中自动的对象内存回收机制称为垃圾回收(Garbage Collection，GC)。

GC 是对内存中不再使用的对象进行回收。GC 中用于回收的方法称为收集器，由于 GC 需要消耗一些资源和时间，Java 在对对象的生命周期特征进行分析后，按照新生代、旧生代的方式来对对象进行收集，以尽可能缩短 GC 对应用造成的暂停。

对新生代对象的收集称为 minor GC；对旧生代对象的收集称为 Full GC；程序中主动调用 System.gc( );语句以强制执行的 GC 为 Full GC。

不同的对象引用类型，GC 会采用不同的方法进行回收。JVM 对象的引用分为 4 种类型：

- 强引用：默认情况下，对象采用的均为强引用(只有在这个对象的实例没有其他对象引用时，GC 时才会被回收)。
- 软引用：软引用是 Java 中提供的一种比较适合于缓存场景的应用(只有在内存不够用的情况下才会被 GC)。
- 弱引用：在 GC 时一定会被 GC 回收。
- 虚引用：虚引用只用来得知对象是否被 GC。

1.4　下载并安装 JDK

JDK 是整个 Java 的核心，其中包括了 Java 运行时环境 JRE、一些 Java 工具和 Java 基础类库(rt.jar)。本节将介绍如何下载并安装 JDK，以及环境变量的配置。

1.4.1　安装 JDK

JDK 的安装文件可以从 http://www.oracle.com/technetwork/java/javase/downloads/index.html 下载，目前的 JDK 版本为 8.0。JDK 的安装文件有 3 个不同操作系统版本，Windows 版本的 JDK 下载下来的文件为 jdk-8u111-windows-i586.exe。如果需要之前的版本，也可以在前面的下载页面中寻找相应的链接地址。

下面以在 Windows 7 环境中安装 JDK 8 为例，介绍 JDK 的安装过程。

(1) 双击运行下载的安装文件，启动安装向导，如图 1-4 所示。

(2) 单击“下一步”按钮，进入如图 1-5 所示的安装功能选择界面。在此可以选择需要安装的功能，接受默认的安装即可，默认的安装中已经提供了基本的 Java 开发和运行环境。

图 1-4　JDK 安装向导第一步

图 1-5　选择要安装的功能

(3) 单击“下一步”按钮，开始安装 JDK。在安装过程中会自动弹出 JRE 的安装窗口，如图 1-6 所示，在该界面中选择所需的 JRE 功能，然后设置其安装路径。

(4) 单击“下一步”按钮，开始 JRE 的安装。几分钟后出现如图 1-7 所示的成功安装界面，单击“完成”按钮，完成 JDK 的安装。

图 1-6　安装 JRE

图 1-7　成功安装 JDK

此时，JDK 和 JRE 的安装工作已经全部结束，但是现在还不能马上使用 JDK 中提供的开发工具。JDK 安装结束之后，必须设置必要的环境变量，然后才能正常使用。在接下来的章节中将介绍 JDK 环境变量的设置方法。

1.4.2　配置环境变量

在上面的章节中，介绍了 JDK 的安装方法，但是在 JDK 安装结束之后，必须进行环境变量的设置，然后才可以使用 JDK 提供的开发工具。下面对环境变量的设置步骤进行详细介绍：

(1) 在桌面上右击“我的电脑”，从弹出的快捷菜单中选择“属性”命令，打开“系统属性”对话框，然后切换到“高级”选项卡，如图 1-8 所示。

(2) 单击“环境变量”按钮，打开“环境变量”对话框，在该对话框中可以对系统的环境变量进行设置。如图 1-9 所示，该对话框分为两部分，上半部分用于设置用户环境变量，下半部分则用于设置系统环境变量。

(3) 将 JDK 的环境变量配置到系统环境变量中，单击下半部分的“新建”按钮，打开“新建系统变量”对话框。在“变量名”文本框中输入需要新建变量的名称“JAVA_HOME”，它用于指明 JDK 的安装路径；在“变量值”文本框中输入变量的值，即 JDK 在系统中的安装路径“C:\Program Files (x86)\Java\jdk1.8.0_111”，如图 1-10 所示，单击“确定”按钮，完成系统变量“JAVA_HOME”的新建。

图 1-8　“系统属性”对话框

图 1-9　“环境变量”对话框

(4) 用同样的方法，新建系统变量 classpath，该变量用于设置类的路径。在“新建系统变量”对话框的“变量名”文本框中输入 classpath，然后在“变量值”文本框中输入“.;%JAVA_HOME%\lib\dt.jar; %JAVA_HOME%\lib\tools.jar”，如图 1-11 所示，单击“确定”按钮，完成系统变量 classpath 的新建。

图 1-10　“新建系统变量”对话框

图 1-11　新建 classpath 系统变量

(5) 最后，将 JDK 的 bin 目录配置到环境变量 Path 中。在底部的“系统变量”列表框中，找到 Path 变量，如图 1-12 所示。然后单击“编辑”按钮，打开“编辑系统变量”对话框，在“变量值”文本框中的最前面加入如下内容：“%JAVA_HOME%\bin;”，如图 1-13 所示。

注意：

在引号包围的内容中，最后有一个分号，这个分号一定不能缺少。

图 1-12　找到系统变量 Path

图 1-13　编辑 Path 变量的值

配置完 JDK 的环境变量后，可以在 DOS 命令行中测试 JDK 是否安装并配置成功。单击“开始”菜单，选择“运行”命令，打开“运行”窗口，在“打开”文本框中输入 cmd

后按回车键，打开命令行窗口。在这个界面中输入 java -version，如果可以得到如图 1-14 所示的界面，就说明 JDK 的安装及配置已经成功。

图 1-14　测试 JDK 的安装及配置是否成功

1.4.3　Hello World 程序

安装完 JDK，配置好环境变量，此时就可以开始编写 Java 程序了，我们从最简单的 Hello World 程序开始。

学习一门新的编程语言时，编写的第一个程序通常都是在屏幕上输出“Hello World”，我们也从 Hello World 程序开始学习。

【案例 1-1】使用 Java 语言编写 Hello World 程序，在屏幕上输出“Hello World”。

Java 语言的 Hello World 程序的代码如下：

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

1. 输入程序

对于大多数计算机语言来说，包含程序源代码的文件的名称是任意的。但是，在 Java 中，源文件的名称非常重要。Java 编译器要求源文件的后缀名为.java，并且文件的名称要与该文件中包含的公有类(使用 public 修饰的类)的名称一致(包括大小写也要一致，因为 Java 是大小写敏感的)。对于上面的 Hello World 程序，public 类名为 Hello，所以源文件的名称应当是 Hello.java。

在 Java 中，源文件的正式称谓是编译单元(compilation unit)，是包含一个或多个类定义(本例中只包含一个类)的纯文本文件。我们可以使用任意文本编辑器来输入上述代码，如记事本、NotePad++、UltraEdit、EditPlus、gVim 等。此时我们可以使用记事本来输入上述代码，在保存文件时，需要在弹出的“另存为”对话框中选择“保存类型”为“所有文件(*.*)”，然后输入文件名 Hello.java，如图 1-15 所示。

2. 编译并运行程序

为了编译 Hello 程序，执行编译器 javac，在命令行上指定源文件的名称。例如，本例中 Hello.java 位于 C 盘根目录，可以将命令行目录切换至 C 盘根目录，然后输入如下命令：

```
C:\>javac Hello.java
```

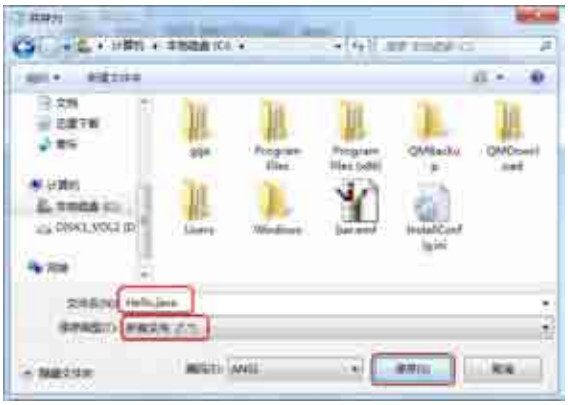

图 1-15　使用记事本保存 Java 源文件

javac 编译器会根据源文件生成字节码文件 Hello.class(名称为类名，后面扩展名为.class)。前面已经讨论过，Java 字节码是程序的中间表示形式，其中包含了 Java 虚拟机将要执行的指令。为了实际运行程序，必须使用 JDK 中的 java.exe 来装载 JVM 环境并加载.class 文件，命令行语句如下：

```
C:\>java Hello
```

此时，将输出“Hello World”信息，如图 1-16 所示。

图 1-16　程序运行结果

3. 分析程序

尽管 Hello.java 相当短，但是它包含了几个所有 Java 程序都具有的关键特征。第一行使用 class 关键字定义了一个新类，类名为 Hello。在 class 前的 public 关键字是一个修饰符，将这个词放在这儿是为了让 Hello 类可以被外界和其他类访问。public 类总是保存在与类同名的文件中，本例中就保存在 Hello.java 文件中。

整个类的定义(包括类的所有成员)都位于一对大括号“{”和“}”之间。接下来的一行代码如下：

```
public static void main(String[] args)
```

初学者可能会对这行代码比较陌生，学习过 C 语言的读者可能容易理解。这是 main( )方法的定义，与 C 语言类似。Java 应用程序都是通过调用 main( )方法开始执行的，以后我们编写的每一个程序都会包含一个带有这行代码的类。

注意：

Java 应用程序中必须包含一个具有 main( )方法的类，这是程序开始运行的位置。程序由 main( )方法的第一条指令开始执行，然后按顺序执行每条指令(除非指令本身让其跳转到程序的某个其他位置)。计算机执行完 main( )方法中的最后一条指令后，程序终止运行。

关键字 public 用于控制类成员的可见性，表示可以在声明该成员的类的外部访问它。在 Java 中，必须将 main( )方法声明为 public，因为当程序启动时，必须从声明 main( )方法的类的外部调用它。关键字 static 表示不必先实例化类的特定实例就可以调用 main( )方法，这也是必需的，因为 Java 虚拟机要在创建任何对象之前调用 main( )方法。关键字 void 表示 main( )方法不返回任何值。后面将会介绍，方法也可以返回值。

在 main( )方法中有一个参数，String args[ ]声明了一个名为 args 的参数，该参数是 String 类的实例数组(数组的概念将在后面详细介绍)。

该行的最后一个字符是“{”，它表示 main( )方法体的开始。构成方法的所有代码也用一对大括号括起来。在 main( )方法中，有一条语句用于在屏幕上输出“Hello World”。这行代码如下：

```
System.out.println("Hello World");
```

输出实际上是通过内置的 println( )方法完成的，println 是“print line”的缩写。在本例中，println( )方法显示传递给它的字符串。println( )方法也可用于显示其他类型的信息。在 println( )方法的前面有 System.out。其中，System 是一个预定义类，提供了访问系统的功能，out 是连接到控制台的输出流。控制台 I/O 主要用于简单的实用程序、演示程序以及服务器端程序，本书很多案例都会使用 println( )方法来输出一些信息。该行代码的最后是一个分号“;”，这是 Java 中的语句结束标识。

接下来的两行都只有“}”，第一个“}”结束了 main( )方法，而最后一个“}”结束了 Hello 类的定义。

4. 为程序添加注释

与大多数其他编程语言一样，Java 也允许在程序的源代码中输入注释。注释只是为了帮助阅读和维护源代码，帮助其他人理解程序代码的功能。编译器会忽略注释的内容。

Java 支持 3 种风格的注释：

1) 对于较短的注释，注释内容可以放在一行中，称为单行注释。单行注释以“//”开头，并在行的末尾结束，编译器会忽略双斜线后到本行结束之间的内容。单行注释可以是一个完整的代码行，也可以出现在普通语句的后面。比如下面的注释语句：

```
//这一行都是注释；下面一行的前边是普通语句，后面才是注释
    System.out.println("Hello World");   //输出 Hello World
```

2) 对于较长的注释(也就是跨越多行的注释)通常采用另一种方法。这种注释类型必须以“/*”开头，并以“*/”结束。编译器会忽略这两个注释符号之间的所有内容。正如名称所暗示的，多行注释可能有若干行。

3) 第三种类型的注释是文档注释，这种类型的注释用于制作程序文档。文档注释以“/**”开头，并以“*/”结束。文档注释其实是多行注释的一种特例，也可以包含多行。

1.5　使用 Eclipse 开发 Java 程序

前面我们使用记事本编写了一个简单的 Java 程序，但是对于较复杂的应用程序，通常需要借助集成开发环境(Integrated Development Environment，IDE)来简化开发工作，从而提高开发效率。Eclipse 是一款主要用 Java 编写的免费 Java IDE，使用 Eclipse 可以创建各种跨平台的可用于手机、网络、桌面和企业领域的 Java 应用程序。本节将简要介绍 Eclipse 的安装与使用。

1.5.1　Java IDE 简介

IDE 是一种用于辅助开发人员开发应用程序的应用软件，一般包括代码编辑器、编译器、调试器和图形用户界面工具，有的还包括版本控制系统、性能分析器等更多辅助工具，因此 IDE 都具有编写、编译、调试等多种功能。正是基于这些功能，使用 IDE 开发应用程序才能够大大减轻程序员的工作，减少项目的开发周期，从而提高应用程序的开发效率。

IDE 的种类非常多，有的 IDE 能同时支持多种应用程序的开发，例如，Eclipse 能用于 Java、PHP、C++等多种语言的开发；有的 IDE 只针对特定语言的开发，如 JSource 只能用于 Java 的开发，Zend Studio 只能用于 PHP 的开发。本节将介绍几种常用的 Java IDE。

1. IntelliJ IDEA

IntelliJ IDEA 是 JetBrains 公司的产品，是 Java 语言开发的集成环境，IntelliJ 在业界被公认为最好的 Java 开发工具之一，尤其在智能代码助手、代码自动提示、重构、J2EE 支持、各类版本工具(git、svn、github 等)、JUnit、CVS 整合、代码分析、创新的 GUI 设计等方面的功能可以说是超常的。

该软件的官方下载地址为 http://www.jetbrains.com/idea/download/index.html。它的旗舰版本还支持 HTML、CSS、PHP、MySQL、Python 等。免费版只支持 Java 等少数语言。IntelliJ IDEA 的开发界面如图 1-17 所示。

图 1-17　IntelliJ IDEA 开发界面

2. Eclipse

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境，它的最大特点就是其扩展性，几乎能够集成开发人员编写的任何开发源代码的插件。Eclipse 最早是由 IBM 开发的，后来

IBM 将 Eclipse 作为一个开放源代码的项目，献给了开源组织 Eclipse.org，但仍由 IBM 的子公司 OTI(主要从事 Eclipse 开发的人员)继续 Eclipse 的开发。其官方下载地址为 http://www.eclipse.org/downloads/，本书所有的案例都是使用 Eclipse 来开发的。

3. NetBeans

NetBeans 是一款屡获殊荣的 Java IDE，可以方便地在 Windows、Mac、Linux 和 Solaris 中运行。

NetBeans IDE 支持所有 Java 应用类型(Java SE、JavaFX、Java ME、网页、EJB 和移动 App)标准开箱即用式的开发。NetBeans 的模块化设计意味着它可以由第三方创建功能提升的插件来扩展 NetBeans。

NetBeans功能是基于Ant的项目系统，支持Maven、重构、版本控制。NetBeans项目由一个活跃的开发者社区提供支持，NetBeans开发环境提供了丰富的产品文档和培训资源以及大量的第三方插件。该软件的官方下载地址为https://netbeans.org/downloads/index.html，其开发界面如图 1-18 所示。

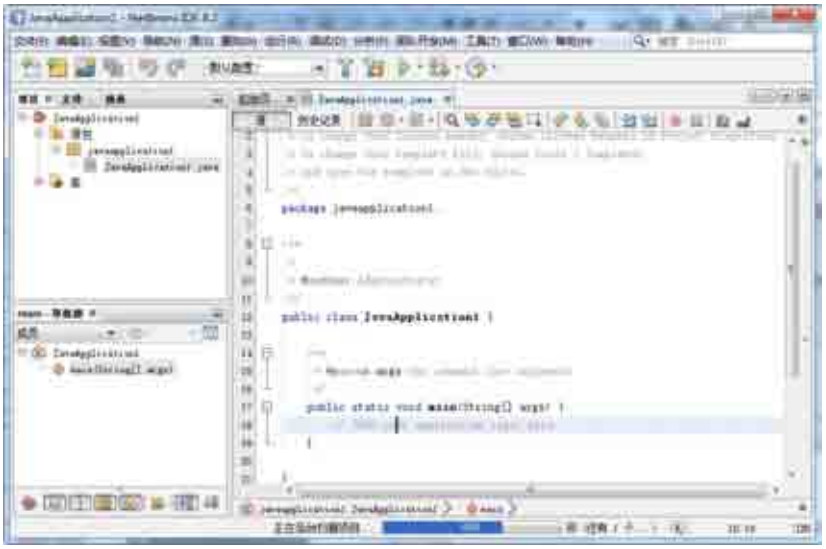

图 1-18　NetBeans 开发界面

4. Android Studio

谷歌的 Android Studio 主要被设计用于 Android 平台上的开发，并且还可以运行和编辑一些 Java 代码。

起初 Android Studio 是 JetBrains 公司在 IntelliJ IDEA Community Edition(社区版)基础上创建的。同时它也基于 Gradle 的编译系统、变量设置以及多个 APK 的生成系统，另外还支持可扩展的模板和多种设备类型。其丰富的布局编辑器还可以满足对不同主题的布局编辑，它提供的 Android Lint 工具可用来对 Android 项目源代码进行扫描和检查，发现潜在的问题。

Android Studio 可以在 Apache 2.0 协议下免费使用，也可以通过 Windows、Mac OS X 和 Linux 下载，它取代 Eclipse 成为谷歌用于原生 Android 应用开发的主要 IDE。其官方下

载地址为 http://developer.android.com/sdk/index.html。

Java 的 IDE 还有很多，读者可根据自己喜好选择其中一个，本书使用的是 Eclipse 的用于 Java 开发的最新版本 Eclipse 4.6(代号 Neon)。

1.5.2　使用 Eclipse 新建 Java 工程

Eclipse 是一个开放源代码、基于 Java 的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件构建开发环境。但是，Eclipse 附带了一个标准的插件集，包括 Java 开发工具(Java Development Tools，JDT)，这就使其功能变得非常强大。

1. 下载并安装 Eclipse

Eclipse 的官方下载地址为 http://www.eclipse.org/downloads，从该网站上可下载最新版本的 Eclipse，最新版本代号为 Eclipse Neno。在 Get Eclipse Neon 中，有用于不同开发的多个版本，我们选择其中的“Eclipse IDE for Java Developers”，该版本适合 Java 开发者，集成了 CVS、Git、XML 编辑器、Mylyn、Maven Integration 和 WindowBuilder 等插件。可根据需要选择 Windows 平台的 64 位或 32 位版本，下载成功后会得到一个名如“eclipse-java-neon-3-win32.zip”的压缩包，将其解压后得到 eclipse 文件夹，这样就完成了 Eclipse 的安装。

2. 新建 Java 工程

本节将使用 Eclipse 新建一个 Java 工程，然后在该工程中输入 Hello World 程序，通过该应用来学习如何使用 Eclipse 来创建 Java 工程，以及工程的运行与调试。

(1) 双击 Eclipse 安装目录下的 eclipse.exe 文件即可启动 Eclipse，此时会加载所需文件，之后会显示工作空间选择界面，该界面用于设置应用程序的默认存储位置，如图 1-19 所示。

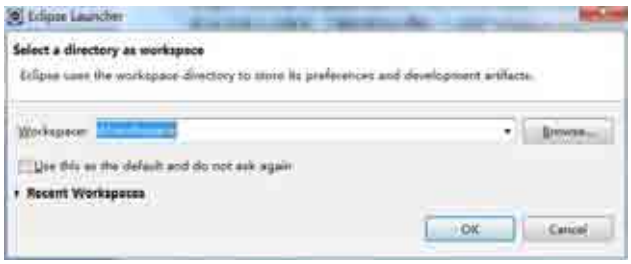

图 1-19　选择工作空间

提示：

如果选中了“Use this as the default and do not ask again”复选框，则在下次启动 Eclipse 时默认使用本次设置的工作空间，并且不再出现选择工作空间的对话框。

(2) 设置好工作空间后，单击 OK 按钮进入 Eclipse 欢迎界面，如图 1-20 所示。

(3) 选择 File | New | Java Project 命令，打开如图 2-23 所示的 New Java Project 对话框，如图 1-21 所示。

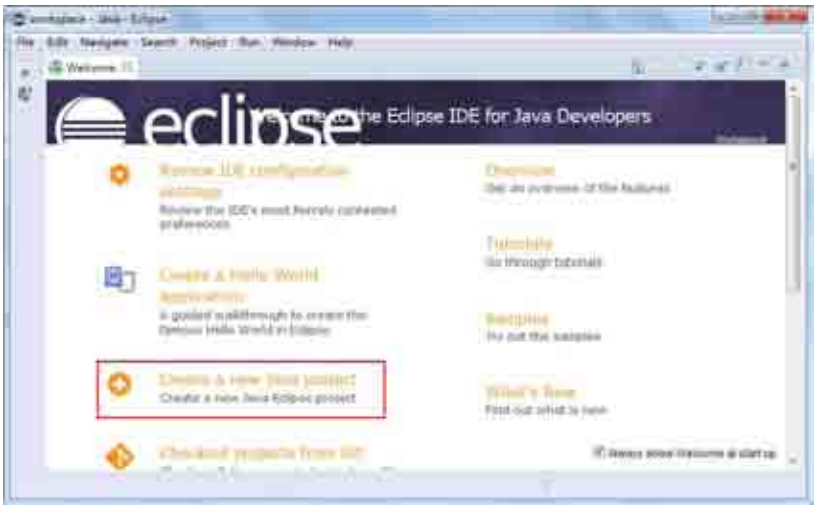

图 1-20　Eclipse 欢迎界面

提示：

单击欢迎界面中的“Create a new Java Project”链接，可以直接打开图 1-21 所示界面。

(4) 在“Project name”文本框中输入工程名 ch01，在 JRE 区域选中“Use an execution environment JRE”单选按钮，然后从后面的下拉列表中选择“Java SE-1.8”。

(5) 单击 Finish 按钮，完成工程的创建。

(6) 接下来，我们可以在该工程内新建一个 Java 类，用来输出 Hello World。选择 File | New | Class 命令，打开“New Java Class”对话框。新建类的文件默认存放的路径为“{工程名}/src”，默认的包名(Package name)为“{工程名}”，如图 1-22 所示。

图 1-21　“New Java Project”对话框

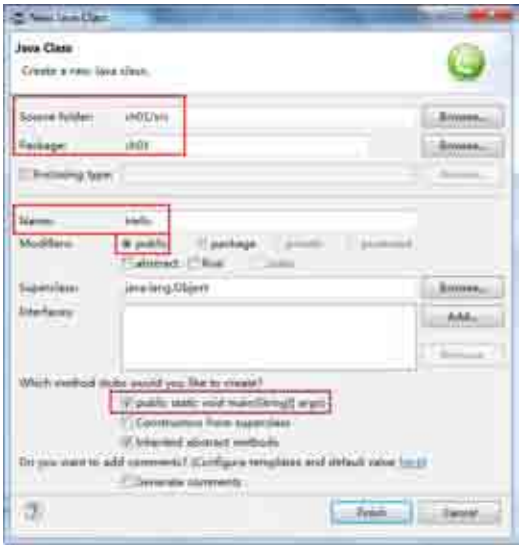

图 1-22　“New Java Class”对话框

(7) 本例中，在 Name 文本框中输入新建的类名 Hello，使用默认修饰符 public，因为我们的类中需要 main( )方法，所以可以选中下面的“public static void main(String[] args)”复选框。这样，系统会默认生成一个 main( )方法。

(8) 单击 Finish 按钮，完成类的创建，从 Package Explorer(包浏览器)窗格中可以看到，生成的 Hello.java 文件位于 src 目录的 ch01 包中，在该文件中默认生成了一个空的 main( )方法，如图 1-23 所示。

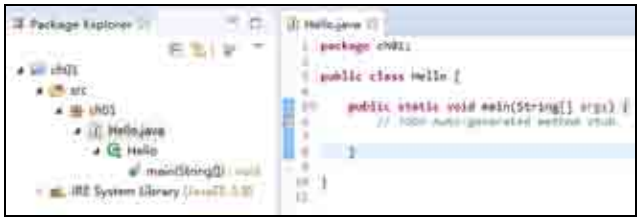

图 1-23　新建 Java 类中的代码

(9) 与前面创建的 Hello.java 不同的是，这里的第一行有一条包声明语句，有关包的相关知识将在第 4 章详细介绍，此时我们只需了解即可。我们需要编写的代码只有 main( )方法中用于输出 Hello World 的一行语句：

```
System.out.println("Hello World");
```

(10) 输入完成后，单击工具栏中的“保存”按钮即可。

使用 Eclipse 编写 Java 程序时，系统默认会自动编译已保存的 Java 源代码。如果要取消自动编译，可以在 Project 菜单中选择“Build Automatically”菜单命令，如图 1-24 所示。自动编译时，该菜单命令的前面会有一个复选框，再次选择该菜单命令，其前面的复选框将消失。对于初学者建议保持自动编译，这样当我们的代码输入有误时，系统会及时给出错误提示。例如，我们在输入 println 时，漏掉了最后一个字母 n，此时系统就会立刻出现红色的错误提示，如图 1-25 所示。

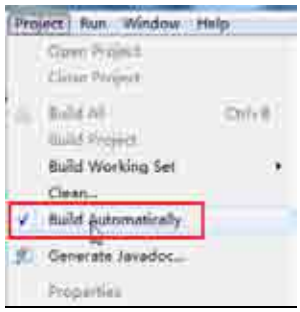

图 1-24　Project 菜单

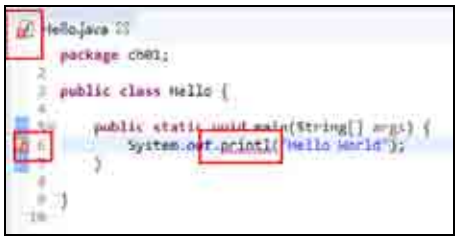

图 1-25　代码输入有误时的错误提示

将光标停留在错误代码处，会出现该错误的详细信息，以及推荐的一些快速修改方案。比如本例中给出的修改方案有 3 个，如图 1-26 所示，根据提示，我们发现原来是少输入了一个 n，所以可以通过选择下面的修改方案 2“Change to 'pringln(...)'”来快速修改代码。

3. 运行与调试程序

代码输入正确后，系统会自动完成编译工作，此时将在工程的bin/ch01目录下生成

Hello.class文件。接下来将介绍如何运行或调试程序。

图 1-26　错误的详细信息及推荐的快速修改方案

选择 Run 菜单中的 Run、“Run As”或“Run History”菜单命令可以直接运行指定的程序，比如在本例中选择 Run｜“Run As”｜“Java Application”菜单命令即可运行 Hello 程序，将在控制台(Console)窗口中输出“Hello World”信息，如图 1-27 所示。

除了直接运行程序，还可以在程序中添加断点，进行调试。具体步骤如下：

(1) 在要设置断点的代码行的最左侧双击鼠标，即可添加一个断点，该行左侧会出现一个实心圆点。如图 1-28 所示，在 Hello.java 的第 6 行添加了一个断点。

图 2-27　在 Console 窗口中输出程序运行结果

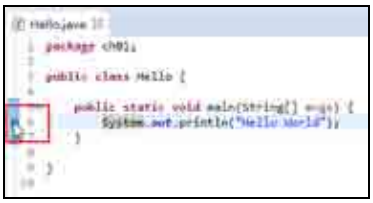

图 1-28　添加断点

提示：

再次双击可以删除断点。

(2) 选择 Run 菜单中的 Debug、“Debug As”或“Debug History”菜单命令即可启动程序调试，在运行到断点位置的语句时，程序将被挂起。

因为 Eclipse 有专门的调试视图，调试视图更方便程序员在调试程序时，监视和查看一些信息，所以会弹出对话框询问是否切换到调试视图，如图 1-29 所示。

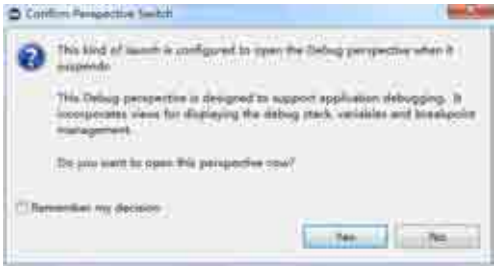

图 1-29　询问是否切换视图

(3) 在调试视图中，各窗口的布局会发生一些变化，读者可根据自己的喜好单击 Yes 或 No 按钮决定是否切换到调试视图。启动调试模式后，程序将在第一个断点处停止，如图 1-30 所示。

图 1-30　程序在断点处挂起

(4) 进入调试模式后，可以单击工具栏中的按钮继续执行程序到下一个断点，也可以单步执行程序(图标)或进入某个方法的内部(图标)，或者单击按钮终止调试。

(5) 终止调试后，可单击工具栏最右侧的 Java 按钮，切换回原来的 Java 视图。

说明：

直接单击工具栏中的 Debug 按钮也可以直接启动调试模式；类似地，Run 按钮用于直接运行程序。

1.5.3　Eclipse 的常用快捷键

Eclipse 提供了丰富的辅助开发功能，而且很多常用的功能都提供了快捷键，使用这些快捷键可以帮助我们更熟练地使用 Eclipse，提高开发效率，对于今后走向工作岗位也大有裨益。

编辑功能的快捷键如表 1-1 所示。

表 1-1　编辑功能的快捷键

快捷键	功能	快捷键	功能
Ctrl+F	查找、替换	Ctrl+A	全部选中
Ctrl+C	复制	Ctrl+V	粘贴
Ctrl+Shift+K	查找上一个	Ctrl+K	查找下一个
Ctrl+Z	撤销	Ctrl+Y	重做
Ctrl+X	剪切	Delete	删除
Alt+/	内容辅助	Ctrl+1	快速修正
Alt+Shift+↓	恢复上一个选择	Alt+?	上下文信息
Ctrl+Shift+J	增量逆向查找	Ctrl+J	增量查找
F2	显示工具提示描述	Alt+Shift+↑	选择封装元素
Alt+Shift+←	选择上一个元素	Alt+Shift+→	选择下一个元素
Ctrl+S	保存文件	Ctrl+Shift+S	全部保存
Ctrl+F4	关闭文件	Ctrl+Shift+F4	全部关闭

Eclipse 提供了强大的搜索功能，这些搜索功能的快捷键如表 1-2 所示。

表 1-2　搜索功能的快捷键

快捷键	功能	快捷键	功能
Ctrl+Shift+U	出现在文件中	Ctrl+H	打开搜索对话框
Ctrl+G	工作空间中的声明	Ctrl+ Shift+G	工作空间中的引用

Eclipse 是一个多窗口的编辑器，在操作每个窗口的时候也提供了对应的快捷键，如表 1-3 所示。

表 1-3　Eclipse 中的窗口快捷键

快捷键	功能	快捷键	功能
F12	激活编辑器	Ctrl+Shift+W	切换编辑器
Ctrl+Shift+F6	上一个编辑器	Ctrl+F6	下一个编辑器
Ctrl+Shift+F7	上一个视图	Ctrl+F7	下一个视图
Ctrl+Shift+F8	上一个透视图	Ctrl+F8	下一个透视图
Ctrl+W	显示标尺上下文菜单	Ctrl+F10	显示视图菜单

Eclipse 提供的导航相关操作的快捷键如表 1-4 所示。

表 1-4　Eclipse 中的导航快捷键

快捷键	功能	快捷键	功能
Ctrl+F3	打开结构	Ctrl+Shift+T	打开类型
F4	打开类型层次结构	F3	打开声明
Shift+F2	打开外部 javadoc	Ctrl+Shift+R	打开资源
Alt+←	后退历史记录	Alt+→	前进历史记录
Ctrl+,	上一个	Ctrl+.	下一个
Ctrl+O	显示大纲	Ctrl+Shift+H	在层次结构中打开类型
Ctrl+Shift+P	转至匹配的括号	Ctrl+Q	转至上一个编辑位置
Ctrl+Shift+↑	转至上一个成员	Ctrl+Shift+↓	转至下一个成员
Ctrl+L	转至指定行		

Eclipse 中对 Java 源代码的操作也提供了一系列的快捷键，如表 1-5 所示。

表 1-5　Eclipse 中的源代码快捷键

快捷键	功能	快捷键	功能
Ctrl+Shift+F	格式化	Ctrl+Shift+O	组织导入
Ctrl+/	注释	Ctrl+Shift+M	添加导入

Eclipse 中对调试应用程序的操作也提供了一系列的快捷键，如表 1-6 所示。

表 1-6　Eclipse 中的调试快捷键

快捷键	功能	快捷键	功能
F7	单步返回	F6	单步跳过
F5	单步进入	Ctrl+F5	单步跳入选择
F11	调试上次启动	F8	继续
Shift+F5	使用过滤器单步执行	Ctrl+Shift+B	添加/去除断点
Ctrl+F11	运行上次启动	Ctrl+R	运行至指定行
Ctrl+U	执行		

1.6　本章小结

本章全面讲述了 Java 语言的起源与发展历程，以及如何使用 Eclipse 开发 Java 应用程序。首先，介绍了 Java 的起源与发展史，包括程序语言的发展阶段、Java 的家世、Java 的发展历程等。接下来，分析了 Java 语言的特点以及 Java 虚拟机的工作原理。之后，讲述了 JDK 的安装与配置，并通过 Hello World 案例介绍 Java 程序的开发、编译和运行过程。最后，重点介绍了使用 Eclipse 开发 Java 应用程序的基本步骤，以及 Eclipse 的常用操作。

1.7　思考和练习

1. 简述 Java 的发展历程。

2. Java 语言有哪些特点？

3. 什么是 Java 虚拟机，简述其工作原理？

4. 配置 JDK 环境变量时，需要新建哪两个环境变量？

5 上机练习：使用 Eclipse 新建一个 Java 工程，然后新建 Hello 类，输出“Hello World”信息。

6. 练习在 Eclipse 中运行和调试 Java 应用程序。