宝典丛书 300万

Java JDK 7 实例宝典

本书实例丰富，共有19章169个实例，每个实例都紧贴实际应用。

采用了多种设计模式，如Singleton、Adapter、MVC等模式，同时在实例中强调应避免的不良编程习惯。

基于J2SE 7.0，用一章专门介绍J2SE 7.0的新特性，精心准备19个实例，帮助读者轻松掌握新特性，并实际应用于项目开发中。

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

韩 雪 郭天娇 编著

宝 典 丛 书

Java JDK 7 实例宝典

韩 雪 郭天娇 编著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书以 J2SE 7.0 为开发环境，选取 Java 应用的典型实例，循序渐进地介绍 Java 语言的各种开发方法和技巧。全书共有 19 个章节，169 个实例，内容涉及 Java 语言基础、面向对象程序设计、数字处理、数组与集合、字符串、异常处理、文件操作、多线程、Swing 编程、图形和多媒体编程、反射机制、网络程序设计、数据库编程、Applet、Java 与 XML、Java Mail、JSP 与 Servlet，并专门用一章介绍 J2SE 7.0 的新特性。本书内容丰富，结构清晰，选择的实例紧贴实际应用，具有很强的实用性和针对性，力求让读者通过实例学到更多、更好的编程方法和技术。

本书适用于 Java 初级、中级和高级开发人员。

图书在版编目（CIP）数据

Java JDK 7 实例宝典 / 韩雪，郭天娇编著. —北京：电子工业出版社，2014.1
（宝典丛书）
ISBN 978-7-121-21707-4

Ⅰ. ①J… Ⅱ. ①韩… ②郭… Ⅲ. ①JAVA 语言—程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2013）第 247218 号

策划编辑：张月萍
责任编辑：葛　娜
印　　刷：北京天宇星印刷厂
装　　订：三河市皇庄路通装订厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　　邮编：100036
开　　本：787×1092　1/16　　印张：39.75　　字数：1175
印　　次：2014年1月第1次印刷
定　　价：89.00元

前　言

Java 语言是一种新型的网络编程语言，其卓越的功能和特性为无数开发人员所推崇，越来越多的应用开发采用了基于 Java 技术的解决方案。Java 语言也是一种真正面向对象的编程语言，它提升了应用程序的编程概念和开发思维。Java 语言提供了丰富的开发类库，为用户编程提供了极大的支持和方便。

目前最新的 Java 标准开发环境是 J2SE 7.0，它在 1.4 版本的基础上增加了自动装箱和拆箱、泛型编程、枚举类型、可变长参数、静态导入等功能，使编写程序更加方便；强化了 1.4 版本中的线程编程，可以很方便地进行多线程编程。

为了帮助更多的 Java 开发人员提高开发技巧，掌握最新的 Java 特性，笔者精心编著了本书。本书以实例为主，Java 的基本理论部分介绍相对较少，编程技巧和方法介绍很多，读者在阅读完本书后可以提高自身的编程技巧，并掌握 J2SE 7.0 的新特性。

本书在内容编排和目录组织上都十分讲究，争取让读者能够快速掌握实例的实现方法。统一按照实例功能来为章节命名，让读者明确知道每一节将实现什么实例。具体讲解实例时，分为 4 个步骤，首先介绍实例的功能，然后介绍实现实例时的关键技术点，接着介绍实例的实现代码和运行效果，最后对程序的关键部分进行解读。本书的代码具备良好的编程风格和详尽的注释，使读者很容易理解程序代码，并养成良好的编程风格。

本书特色

1. 实例经典，内容丰富

本书实例丰富，共有 19 章 169 个实例，每个实例都紧贴实际应用，如用 Servlet 生成图形验证码，具有很强的示范性和实用价值，读者可以直接使用书中的实例解决实际问题。

2. 侧重编程技巧

本书所有的实例在 Java 的基本理论方面介绍不多，重点是讲述常用、实用的编程技术。采用了多种设计模式，如 Singleton、Adapter、MVC 等模式，同时在实例中强调应避免的不良编程习惯。

3. 以 J2SE 7.0 为开发环境

目前介绍 J2SE 7.0 新特性的书籍很少，很多开发人员对这些新特性不了解，在实际的项目开发中也很少使用这些新特性。本书用一章专门介绍 J2SE 7.0 的新特性，精心准备了 19 个实例，通过这些实例可以帮助读者轻松掌握这些新特性的特点，并实际应用于项目开发中。读者会发现，J2SE 7.0 的新特性的确有利于提高开发效率。

4. 层次清晰，主次分明

全书分为 19 个章节，包括 Java 技术的方方面面，章节之间的先后顺序很重要。本书

在章节安排上采用了由浅入深的策略：先介绍基本技术，为后面章节打好基础；再介绍高级技术，并在实例中使用前面章节中介绍的技术，起到巩固的作用。

在讲解实例时，首先介绍实例的目标，让读者了解该实例要做什么，在脑海中形成一个实现思路；然后介绍实例中使用的关键技术点，帮助读者解决实现思路中的问题；最后对代码的关键部分进行解读，帮助读者掌握关键技术点。

5. 编程风格良好，注释详细

每个实例的程序都是经过精心设计的，在实现实例功能的同时，注重程序运行的效率。程序代码具备良好的编程风格，结构清晰，注释非常详细，能帮助读者轻松地掌握实例的实现过程。

学习 Java 的心得体会

1. 熟练掌握 Java 的基本语法

如果读者会 C 编程，那么学习 Java 将会非常容易，学习基本语法时，重点理解各关键字的功能与用法，以及功能相似的关键字之间的区别，如 continue 和 break。

2. 理解 Java 面向对象的相关概念

面向对象是当今最成熟的程序设计方法，它有 3 个非常重要的特性：继承、封装和多态。Java 是一种面向对象的程序设计语言，在 Java 中，一切都是对象，体现了面向对象的封装性；Java 类之间允许单重继承，体现了继承性；多态性主要体现在允许定义接口类、抽象类，子类能够覆盖和隐藏父类的变量和方法，也属于多态性。读者要理解面向对象的概念，掌握如何在 Java 中进行面向对象编程。

3. 善于使用 JDK 自带的帮助文档

Java 提供了非常丰富的类库，要想掌握常用类的用法，最好的方法是查阅 JDK 自带的帮助文档，尽量看英文原版的，不要依赖于中文版的帮助文档或 API 参考书籍。

4. 多动手写 Java 程序

学习 Java 的最终目标就是为了写程序，解决问题。掌握了 Java 的基本语法、面向对象的概念，能够读懂简单的例子后，接下来就要亲手写程序。第一步，把书上的例子在本机上运行通过；第二步，敢于修改书上的程序；第三步，合上书本，独立编写程序，即使程序的功能与书上的例子一样。

5. 多阅读开源项目的源代码

要想具备良好的编程风格，掌握好的编程技术、阅读优秀的源代码是一条捷径。当前很多开源项目的源代码都值得学习（如 Apache 组织的项目，以及 JDK 的源代码）。在阅读源代码时，要注意两点：第一，学习项目的整体框架，结合项目文档，一步步地调试程序是最常用的手段；第二，学习代码中的技巧、算法，这是一个积累的过程。

6. 理解 Java 虚拟机的运行机理

Java 之所以能够跨平台，主要是因为存在 Java 虚拟机。Java 虚拟机运行在操作系统中，不同操作系统下的 Java 虚拟机不同。将 Java 类编译成字节码，在 Java 虚拟机中运行字节码，使 Java 程序独立于操作系统，具备跨平台的能力。为了更好地理解 Java 程序的运行机理，编写出更优秀、更高效的程序，读者需要理解 Java 虚拟机的类加载机制和垃圾回收机制等。

本书包括的内容

第 1 章介绍 Java 基础知识，实例包括：Java 的基本数据类型、Java 的各种运算符、Java 程序的流程控制，以及如何在 Java 程序中使用命令行参数。

第 2 章介绍如何用 Java 进行面向对象的程序设计，实例包括：类的设计、对象的克隆、类的继承、抽象类、变量和方法的覆盖与隐藏、Java 的参数传递、类的加载顺序、接口的设计，以及程序设计中常用的设计模式，如单例模式、工厂模式和适配器模式。

第 3 章介绍 Java 中的数字，实例包括：数字的封装类、数字的舍入、格式化数字、数字进制的转换、生成随机数，以及大数字的处理。

第 4 章介绍 Java 中的数组和集合，实例包括：使用 Arrays 类操作数组、利用数组求质数、动态调整数组的大小、利用二维数组实现矩阵、Java 中各种 List 实现类的区别、结合 Random 和 List 实现一个不重复的随机数序列、利用 LinkedList 实现一个先进先出的队列、对 List 中的元素进行排序、Java 中各种 Set 实现类的区别、集合与数组的相互转化、Java 中各种 Map 实现类的区别、对 Map 中的元素进行排序，以及常用于解析配置文件的 Properties 类。

第 5 章介绍 Java 的字符串操作，实例包括：判断一个字符串是否是合法的 Java 标识符、18 位身份证号码格式的验证、实现一个简单的表达式解析器、对密码进行加密和验证、制作命令行程序，以及使用正则表达式验证电话号码的格式。

第 6 章介绍 Java 的异常处理，实例包括：用于声明和抛出异常的 throws 和 throw 语句的用法、用于捕获处理异常的 try、catch 和 finally 语句的用法、自定义异常类，以及使用异常的几个原则。

第 7 章介绍 Java 的输入输出流，重点介绍文件输入输出流的处理，实例包括：获取文件的描述信息、操作文件和目录、读写文件、搜索文件、序列化和反序列化对象、压缩和解压缩文件、处理 Excel 和 PDF 文件，以及一个自定义的写日志文件的类。

第 8 章介绍 Java 的线程，实例包括：线程的定义、线程的启动与停止、线程的同步、线程 join、线程的优先级、守护线程、线程的死锁、定时器、用线程实现生产者和消费者例子，以及一个简单的线程池。

第 9 章介绍利用 Java 的 AWT 和 Swing 工具包生成图形用户界面（GUI），实例包括：实现图形日历、开窗户小游戏、标准型计算器、更改组件的外观、自定义对话框、制作程序的欢迎画面、一个简单的文本编辑器，以及 Swing 控件的 Drag 和 Drop。

第 10 章介绍 Java 的图形处理，实例包括：实现一个圆形按钮、捕捉屏幕、缩放图片、

画 2D 和 3D 图形，以及实现一个圆形的时钟。

第 11 章介绍用 Java 处理多媒体，包括动画、音频和视频的处理。实例包括：滚动的消息、三维弹球和贪吃蛇游戏、3 种播放音频的方法，以及实现一个媒体播放器。

第 12 章介绍 Java 的反射机制，实例包括：使用 instanceof 操作符判断对象的类型、通过类名查看类的各种信息（如类声明的属性、构造方法、公有方法等），以及动态调用类的方法。

第 13 章介绍 Java 的网络编程，实例包括：从 URL 中提取信息、Web 浏览器、获取 IP 地址和域名、访问 HTTP 服务器的客户端、实现 HTTP 服务器、基本的 Socket 编程、一个支持多线程的服务器框架、基于服务器框架的代理服务、访问 Telnet 服务的客户端、UDP 编程、聊天室服务器和客户端，以及访问 FTP 站点的客户端。

第 14 章介绍 Java 的数据库编程，实例包括：连接各种数据库、获得数据库和数据表的元数据、查询和更新数据库、SQL 语句的批处理、事务的提交和回滚、使用 PreparedStatement 执行 SQL 语句、读写二进制数据到数据库、读写 Blob 数据到数据库、使用 ResultSet 更新数据库、使用 RowSet 操作数据库、调用存储过程，以及实现一个数据库连接池。

第 15 章介绍 Applet 编程，实例包括：实现一个 Applet 时钟、在 Applet 中处理键盘和鼠标事件、英文打字游戏、两个 Applet 间的通信，以及用 Applet 实现一个汉诺塔的游戏。

第 16 章介绍用 Java 操作 XML 文档，实例包括：用 DOM 处理 XML 文档、用 SAX 处理 XML 文档、用 XSLT 转换 XML，以及 XML 文档与对象之间的相互转换。

第 17 章介绍用 Java 收发邮件，实例包括：用 SMTP 协议发送简单邮件、发送带附件的邮件、发送邮件给多人，以及用 POP3 协议接收邮件。

第 18 章介绍 JSP（Java Server Page）和 Servlet 技术，实例包括：获取访问 JSP 页面的客户端的真实 IP 地址、在 JSP 中读取和设置 Cookie、无刷新的 JSP 聊天室、在 JSP 中上传文件、用 Servlet 生成图形验证码，以及用 Servlet 实现分页查看数据库。

第 19 章介绍 J2SE 7.0 版本对 Java 语法更新的关键技术点，实例包括：Switch 处理字符串变量，用二进制形式表示整数，Catch 可以捕获多个异常，JDK 可以自动关闭相关资源，以及新版本 JDBC 的功能演示。

本书相关资源请到 http://www.broadview.com.cn/21707 下载。

适合阅读本书的读者

本书由韩雪、郭天娇编写，其中河北工业大学廊坊分校的韩雪老师负责编写第 1~10 章，吉林工程技术师范学院的郭天娇老师负责编写第 11~19 章，本书具有知识全面、实例精彩、指导性强的特点，力求以全面的知识性及丰富的实例来指导读者透彻学习 Java 各方面的技术。本书可以帮助 Java 初级、中级开发人员提高开发技能，掌握 J2SE 7.0 的新特性，书中的实例对高级开发人员也有一定的启发意义。

作者

目　录

第 1 章 Java 基础

本章包括

- 转换基本数据类型
- Java 的运算符
- 控制程序的流程
- 计算整数的阶乘
- 实现命令行程序

为了表示程序中的基本数据，Java 语言定义了若干基本数据类型，如整型、浮点型等；同时 Java 还定义了若干运算符，专门用来对这些基本类型的数据进行运算，如加法运算、减法运算等；在编程过程中，需要控制程序的流程，根据运行状态决定下一步执行什么，如循环语句、条件语句等。以上这些是编写一个简单的 Java 程序必须掌握的知识，是 Java 语言的基础。

1.1 转换基本数据类型

Java 的基本类型很多，如整型（int）、长整型（long）、浮点型（float）、字符型（char）等，这些类型之间可以相互转换。本节实例主要介绍基本类型的转换，最主要的是转换规则。

关键技术剖析

有两种转换基本类型的方法：自动提升和强制转换。在运算表达式中，基本类型的数据的类型会自动提升，提升规则如下：

- 所有的 byte 类型和 short 类型的值被提升到 int 类型。
- 整数运算时，如果有一个操作数是 long 类型，则其他操作数的值都被提升到 long 类型。
- 浮点运算时，如果有一个操作数是 float 类型，则其他整数值都被提升到 float 类型。
- 浮点运算时，如果有一个操作数是 double 类型，则其他操作数的值都被提升到 double 类型，计算结果也是 double 类型。

类型的自动提升发生的条件是：两种类型是兼容的，或者目标类型的范围比源类型的范围大。

在其他情况下，如果要转换数据的类型，则需要进行强制类型转换。强制类型转换的方法是：在数据前放一对包括新的类型名的括号，如语句“int i = (int)55555L;”表示将 long 类型的 55555L 转换成 int 类型，然后赋值给变量 i。

实例演示

```
/**---------文件名:ChangeBasicType.java-------------*/
package book.basic;

public class ChangeBasicType {

    /**
```

```
     * 基本类型的自动提升
     */
    private void typeAutoUpgrade() {
        byte b = 44;
        char c = 'b';
        short s = 1024;
        int i = 40000;
        long l = 124631;
        float f = 35.67f;
        double d = 3.1234d;
        // (f * b)时，b 自动提升为 float 类型，(l * f)时，l 自动提升为 float 类型
        // (i / c)时，c 自动提升为 int 类型，(d * s)时，s 自动提升为 double 类型
        // 再相加时，中间结果都变为 double 类型。这里 result 只能声明为 double 类型
        // result 声明为其他类型会出错，除非进行强制类型转换
        double result = (f * b) + +(l * f) + (i / c) - (d * s);
        System.out.print((f * b) + " + " + (l * f) + " + " + (i / c) + " - "
                + (d * s));
        System.out.println(" = " + result);
    }

    /**
     * 基本类型的自动转换
     * 自动转换发生的条件是：(1)两种类型是兼容的；(2)目标类型的范围比源类型的范围大
     * 在其他情况下，必须进行强制类型转换
     */
    private void autoChange() {
        char c = 'a';
        byte b = 44;
        short s0 = b;
        int i0 = s0;
        int i1 = c;
        long l = i0;
        float f = l;
        double d = f;
        System.out.print("c = " + c + "; b = " + b + "; s0 = " + s0);
        System.out.print("; i0 = " + i0 + "; i1 = " + i1 + "; l = " + l);
        System.out.println("; f = " + f + "; d = " + d);

        // 当 float 类型转换成 double 类型时，会出现不相等的情况，这是因为，float 类型的
        // 范围是 32 位，而 double 类型的范围是 64 位，在进行类型转换时，操作的实际上都是
        // 二进制，有些实数用二进制能够精确表示，而有些实数无论用多少位二进制都无法表示
        // 与有理数和无理数的概念一样
        // 对不能够精确表示的实数进行类型转换时，会出现不相等的情况
        float fl = 1.7f;
        double dou = fl;
        System.out.println("fl = " + fl + "; dou = " + dou);
        // 当把一个数从一种类型转换成另一种类型，再转换回来时，值还是一样的
        fl = (float)dou;
        System.out.println("fl = " + fl + "; dou = " + dou);
    }

    /**
     * 强制类型转换
     * 当两种类型不兼容，或者目标类型的范围比源类型的范围小时，必须进行强制类型转换
     * 具体转换时会进行截断、取模操作
     */
    private void forceChange() {
        double d = 123.456d;
        float f = (float) d;
        long l = (long) d;
        int i = (int) d;
        short s = (short) d;
        byte b = (byte) d;
```

```
        System.out.print("d = " + d + "; f = " + f + "; l = " + l);
        System.out.println("; i = " + i + "; s = " + s + "; b = " + b);

        d = 567.89d;
        // 下面的转换首先进行截断操作，将 d 的值变为 567，因为 567 比 byte 类型的范围 256 还大
        // 于是进行取模操作，567 对 256 取模后的值为 55
        b = (byte) d;
        System.out.println("d = " + d + "; b = " + b);
    }

    public static void main(String[] args) {
        ChangeBasicType test = new ChangeBasicType();
        test.typeAutoUpgrade();
        test.autoChange();
        test.forceChange();
    }
}
```

运行该程序，输出如下：

```
1569.48 + 444555.2 + 408 - 3198.3616 = 443334.29465
c = a; b = 44; s0 = 44; i0 = 44; i1 = 97; l = 44; f = 44.0; d = 44.0
fl = 1.7; dou = 1.7000000476837158
fl = 1.7; dou = 1.7000000476837158
d = 123.456; f = 123.456; l = 123; i = 123; s = 123; b = 123
d = 567.89; b = 55
```

源码分析

（1）typeAutoUpgrade 方法演示了基本数据类型的数据在进行运算时，其类型会自动进行提升，并对自动提升规则进行了说明。

（2）autoChange 方法演示了基本数据类型的自动转换，以及自动转换发生的条件。当某些 float 类型的数自动转换成 double 类型时，会出现前后不相等的情况，这是由于该数不能够用有限的二进制位精确表示造成的。

（3）forceChange 方法演示了何时进行强制类型转换，以及如何进行强制转换。在强制类型转换过程中会损失一定的精度。

1.2 Java 的运算符

本节实例主要介绍 Java 的运算符，将 Java 的运算符分成多类，每一类运算符的用法都封装在一个方法内。重点讲述运算符的用法和差别。

关键技术剖析

Java 的运算符主要包括如下几类。

- 算术运算符。如：加（+）、减（-）、乘（*）、除（/）。
- 比较运算符。如：等于（==）、小于（<）、大于（>）、小于或等于（<=）、大于或等于（>=）。
- 布尔运算符。如：短路与（&&）、短路或（||）、非（!）、逻辑与（&）、逻辑或（|）、异或（^）。

- 赋值运算符。如：等于（=）、加等（+=）、减等（-=）、乘等（*=）、除等（/=）、与等（&=）、或等（|=）、异或等（^=）。
- 位运算符。如：与（&）、或（|）、异或（^）、取反（~）、右移（>>，>>>）、左移（<<）。
- 其他运算符。如：三目运算符（?:）、加加（++）、减减（--）。

实例演示

```
/**--------文件名:Operator.java-------------*/
package book.basic;

public class Operator {
    /**
     * 算术运算符: +; -; *; /;
     */
    private void computeOperator() {
        int a = 8;
        int b = 5;
        // 对于除法运算，根据基本类型的自动转换规则，当除数和被除数都是整数时
        // 得到的结果也是整数。因此，8/5 得到的是 1，而不是 1.6
        int f = a / b;
        double g = a / b;
        System.out.println("(f = a / b) = " + f + "; (g = a / b) = " + g);
        // 只要除数和被除数中有一个为 double 类型或者 float 类型，结果就不同了
        // 8/5.0 得到的是 1.6
        double h = a / (b * 1.0d);
        float i = a / (b * 1.0f);
        System.out.println("(h = a / (b * 1.0d)) = " + h + "; (i = a / (b * 1.0f)) = " + i);
    }

    /**
     * 比较运算符:==; <; >; !=; <=; >=;
     */
    private void compareOperator() {
        int a = 5;
        int b = 10;
        System.out.println("(a == b) = " + (a == b));
        System.out.println("(a < b) = " + (a < b));
        System.out.println("(a > b) = " + (a > b));
        System.out.println("(a != b) = " + (a != b));
        System.out.println("(a <= b) = " + (a <= b));
        System.out.println("(a >= b) = " + (a >= b));
        System.out.println("(a = b) = " + (a = b)); // 10
    }

    /**
     * 位运算符:&; |; ^; ~; >>; >>>; <<;
     */
    private void bitOperator() {
        byte a = 23; // "00010111"
        byte b = 26; // "00011010"

        // 按位与，两个运算数都为 1 时，结果为 1，其余结果为 0
        int c = a & b; // "00010010"
        System.out.println("(c = a & b) = " + c);
        // 按位或，两个运算数都为 0 时，结果为 0，其余结果为 1
        int d = a | b; // "00011111" 1
        System.out.println("(d = a | b) = " + d);
        // 按位异或，两个运算数相同时结果为 0，不同时结果为 1
```

```
        int e = a ^ b; // "00001101"
        System.out.println("(e = a ^ b) = " + e);
        // 按位非，0变成1，1变成0
        int f = ~a; // "11101000"
        // 注意:Java中采用二进制补码存储数字，对于整数，原码与补码一致
        // 对于负数，符号位不变，将原码取反然后加1，得到补码
        // 补码“11101000”对应的原码是“10011000”，即-24
        System.out.println("(f = ~a) = " + f);
        // 右移，左边空出位以符号位填充
        int g = a >> 1;// "00001011" = 11
        System.out.println("(g = a >> 1) = " + g);
        // 右移，左边空出位以0填充
        int h = a >>> 1;// "00001011" = 11
        System.out.println("(h = a >>> 1)" + h);
        // 对负数操作时，>>和>>>得到的结果会不一样
        System.out.println("(-24 >> 1) = " + (-24 >> 1) + "\t (-24 >>> 1) = " + (-24 >>> 1));
        // 左移
        int i = a << 1; // "00101110" = 46
        System.out.println("(a << 1) = " + i);
    }

    /**
     * 布尔运算符: &&; ||; !; &; |; ^;
     */
    private void booleanOperator() {
        boolean b1 = true;
        boolean b2 = true;
        boolean b3 = false;

        // &&运算符:对操作数进行与运算，当所有操作数都为true时，结果为true
        // 否则结果为false
        if (b1 && b2 && b3) {
            System.out.println("变量b1, b2, b3的值都为true");
        } else {
            System.out.println("变量b1, b2, b3中至少有一个的值为false");
        }
        // 注意:&&是短路与，意思是，当对操作数的表达式进行从左到右运算时
        // 若遇到有操作数的值为false，则结束运算，将结果置为false
        int i = 2;
        int j = 0;
        if (b3 && ((j = i) == 2)) {
        }
        System.out.println("j = " + j);
        if (b1 && ((j = i) == 2)) {
        }
        System.out.println("j = " + j);

        // ||运算符:对操作数进行或运算，当所有操作数都为false时，结果为false
        // 否则结果为true
        if (b1 || b2 || b3) {
            System.out.println("变量b1, b2, b3中至少有一个的值为true");
        } else {
            System.out.println("变量b1, b2, b3的值都为false");
        }
        // 同样，||是短路或，意思是，当对操作数的表达式进行从左到右运算时
        // 若遇到有操作数的值为true，则结束运算，将结果置为true
        if (b1 || ((j = j - 1) == 1)) {
        }
        System.out.println("j = " + j);
        if (b3 || ((j = j - 1) == 1)) {
        }
        System.out.println("j = " + j);
```

```
        // !运算符:对操作数的值进行取反运算，若操作数为 true，则取反为 false
        if (!b1) {
            System.out.println("变量 b1 为 false，因为 b1 取反后的值为 true");
        } else {
            System.out.println("变量 b1 为 true，因为 b1 取反后的值为 false");
        }

        // &运算符:和&&一样，对操作数进行与操作，不同的是，它不是短路的
        // 它会运算完所有的操作数表达式
        if (b3 & ((j = i) == 2)) {
            System.out.println("b3 & ((j=i) == 2): true");
        } else {
            System.out.println("b3 & ((j=i) == 2): false");
        }
        System.out.println("j = " + j);
        if (b1 & ((j = j - 1) == 0)) {
        }
        System.out.println("j = " + j);

        // |运算符:和||一样，对操作数进行或操作，但它不是短路的
        if (b1 | ((j = i) == 2)) {
            System.out.println("b1 | ((j=i) == 2): true");
        } else {
            System.out.println("b1 | ((j=i) == 2): false");
        }
        System.out.println("j = " + j);
        if (b3 | ((j = j - 1) == 1)) {
        }
        System.out.println("j = " + j);

        // ^运算符:对操作数进行异或操作，相同为 false，不同为 true
        if (b1 ^ b2) {
            System.out.println("变量 b1, b2 的值不同！");
        } else {
            System.out.println("变量 b1, b2 的值一样！");
        }
    }

    /**
     * 赋值运算符: =; +=; -=; *=; /=; &=; |=; ^=;
     */
    private void evaluateOperator() {
        // =是最常用的赋值运算符
        int i = 5; // 将变量 i 的值赋为 5
        // +=运算符是先将运算符左边的操作数的值加上右边的操作数的值
        // 然后将结果赋值给运算符左边的操作数
        i += 3; // 等价于 i = i + 3;
        i -= 3; // 等价于 i = i - 3;
        i *= 3; // 等价于 i = i * 3;
        i /= 3; // 等价于 i = i / 3;
        i &= 3; // 等价于 i = i & 3;
        System.out.println("(i &= 3) = " + i);
        i |= 3; // 等价于 i = i | 3;
        System.out.println("(i |= 3) = " + i);
        i ^= 3; // 等价于 i = i ^ 3;
        System.out.println("(i ^= 3) = " + i);
    }

    /**
     * 其他运算符:三目运算符; ++; --; -;
     */
    private void otherOperator() {
```

```
        int i = 5;
        // ++是将操作数加1
        // ++i 表示将 i 加 1 后再进行运算
        if (i++ == 5) {
            System.out.println("表达式(i++ == 5)的值为 true");
        } else {
            System.out.println("表达式(i++ == 5)的值为 false");
        }
        System.out.println("i = " + i);
        // i++表示先进行运算，再将 i 的值加 1
        i = 5;
        if (++i == 5) {
            System.out.println("表达式(++i == 5)的值为 true");
        } else {
            System.out.println("表达式(++i == 5)的值为 false");
        }
        System.out.println("i = " + i);
        // --是将操作数减 1。同样，--i 表示先将 i 减 1，再进行运算
        // i++表示先进行运算，再将 i 减 1
        i--;
        --i;

        // 可以用三目运算符替代简单的 if 语句
        // 格式是： x = a ? b : c。如果 a 的值为 true，则将 b 的值赋予 x
        // 否则将 c 的值赋予 x
        int x = (i >= 0) ? i : -i; // 将一个减号 "-" 和数字连用，表示取数字的负数
        System.out.println("i 的绝对值为: " + x);
    }

    public static void main(String[] args) {
        Operator test = new Operator();
        System.out.println("算术运算符方法的输出:");
        test.computeOperator();
        System.out.println("比较运算符方法的输出:");
        test.compareOperator();
        System.out.println("位运算符方法的输出:");
        test.bitOperator();
        System.out.println("布尔运算符方法的输出:");
        test.booleanOperator();
        System.out.println("赋值运算符方法的输出:");
        test.evaluateOperator();
        System.out.println("其他运算符方法的输出:");
        test.otherOperator();
    }
}
```

运行该程序，输出如下：

```
算术运算符方法的输出:
(f = a / b) = 1; (g = a / b) = 1.0
(h = a / (b * 1.0d)) = 1.6; (i = a / (b * 1.0f)) = 1.6
比较运算符方法的输出:
(a == b) = false
(a < b) = true
(a > b) = false
(a != b) = true
(a <= b) = true
(a >= b) = false
(a = b) = 10
位运算符方法的输出:
(c = a & b) = 18
(d = a | b) = 31
(e = a ^ b) = 13
```

```
(f = ~a) = -24
(g = a >> 1) = 11
(h = a >>> 1)11
(-24 >> 1) = -12     (-24 >>> 1) = 2147483636
(a << 1) = 46
布尔运算符方法的输出:
变量 b1, b2, b3 中至少有一个的值为 false
j = 0
j = 2
变量 b1, b2, b3 中至少有一个的值为 true
j = 2
j = 1
变量 b1 为 true, 因为 b1 取反后的值为 false
b3 & ((j=i) == 2): false
j = 2
j = 1
b1 | ((j=i) == 2): true
j = 2
j = 1
变量 b1, b2 的值一样!
赋值运算符方法的输出:
(i &= 3) = 1
(i |= 3) = 3
(i ^= 3) = 0
其他运算符方法的输出:
表达式(i++ == 5)的值为 true
i = 6
表达式(++i == 5)的值为 false
i = 6
i 的绝对值为: 4
```

源码分析

（1）computeOperator 方法演示了算术运算符的使用。对于除法运算，当除数和被除数都是整型时，得到的结果也是整型；当除不尽时，得到的结果将是真实结果的整数部分。

（2）compareOperator 方法演示了比较运算符的使用。比较两个数是否相等，用“==”而不是“=”，前者是比较运算符，后者是赋值运算符。

（3）bitOperator 方法演示了位运算符的使用。“>>”是将数字按二进制位右移，左边空出位以符号位填充；“>>>”也是将数字按二进制位右移，左边空出位补 0。对于正数，“>>”和“>>>”得到的结果一样；但是对于负数，结果差别就非常大。

右移 1 位相当于运算数除以 2，左移 1 位相当于运算数乘以 2。事实上，右移 n 位，相当于运算数除以 2 的 n 次方；而左移 n 位，相当于运算数乘以 2 的 n 次方。在进行乘除法时，可以通过移位操作实现。

（4）booleanOperator 方法演示了布尔运算符的使用。“&&”是短路与，即当“&&”左边连接的表达式的值为 false 时，右边的表达式的值就不用计算了，此时整个表达式的值为 false；而“&”运算符不同，无论它左边连接的表达式的值是什么，它都会计算右边的表达式的值，整个表达式最后得到的结果与“&&”得到的一样。“||”与“|”的区别和“&&”与“&”的区别一样。在实际编程过程中一般都使用“&&”和“||”。

（5）evaluateOperator 方法演示了 Java 的赋值运算符的使用。

（6）otherOperator 方法演示了三目运算符“?:”的使用方法，用三目运算符实现简单的条件语句，比 if 条件语句更直观。“++”和“--”运算符完成自动加减 1 的功能，程序演示了++i 和 i++