

技术顾问 李坤

你必须知道的261个 Java语言问题

梁建全 编著

261个编程新手最常遇到的Java语言问题

菜鸟想问不敢开口？

扫除入门者的障碍，开辟成长捷径

 人民邮电出版社
POSTS & TELECOM PRESS

请相信，你并不是第一个遇到问题的人。
发现问题，思考问题，寻找答案，解决问题。

本书内容涵盖：

- Java基本概念及环境配置
- Java线程和序列化
- Java编程基础
- Java网络编程
- Java与面向对象
- Java常用功能
- Java流和文件操作
- Java数据库操作
- Java GUI编程
- Java Web程序设计

封面设计：黄 凯

分类建议：计算机/程序设计/Java

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-21528-4



9 787115 215284 >
ISBN 978-7-115-21528-4

定价：49.00 元

技术顾问 李坤

你必须知道的261个 Java语言问题

梁建全 编著

人民邮电出版社
北京

新华书店
PDG

图书在版编目 (C I P) 数据

你必须知道的261个Java语言问题 / 梁建全编著. —
北京: 人民邮电出版社, 2009.11
ISBN 978-7-115-21528-4

I. ①你… II. ①梁… III. ①JAVA语言—程序设计
IV. ①TP312

中国版本图书馆CIP数据核字(2009)第176504号

内 容 提 要

本书以问答的形式组织内容, 讨论了学习或使用 Java 语言的过程中经常遇到的一些问题。这些问题均是在经过充分调研的基础上, 从实际应用中总结出来的, 是作者和众多 Java 开发者的经验总结。书中精选了 Java 开发人员经常遇到的 261 个典型问题, 涵盖了基本概念、环境配置、基本语法、异常处理、流操作、图形用户界面编程、网络编程、线程、序列化、数据库操作、Java Web 程序设计等各方面的主题, 并分别给出了详细的解答, 而且结合代码示例阐明了技术要点。

本书结构清晰、讲解透彻、实用性强, 是各高校相关专业 Java 语言课程很好的教学参考书, 也是各层次 Java 程序员的优秀实践指南。

你必须知道的 261 个 Java 语言问题

- ◆ 编 著 梁建全
责任编辑 蒋 佳
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京顺义振华印刷厂印刷
- ◆ 开本: 800×1000 1/16
印张: 30
字数: 670 千字
印数: 1-4 000 册
- 2009 年 11 月第 1 版
2009 年 11 月北京第 1 次印刷

ISBN 978-7-115-21528-4

定价: 49.00 元

读者服务热线: (010)67132692 印装质量热线: (010)67129223
反盗版热线: (010)67171154

前言

现今，互联网资源非常丰富，在生活中如果遇到疑难问题，相信有很多人会借助百度或谷歌等搜索引擎寻找问题答案。他们认为自己并不是第一个遇到此问题的人，肯定还会有很多前人遇到过与自己相同或类似的问题。利用前人的经验可以帮助我们快速地解决自己所遇到的问题，从而使问题的解决变得相对容易，节省了大量的时间和精力，这是一种非常明智的做法。

同样，如果你是一个 Java 初学者或开发者，肯定也会遇到很多前人遇到过的相同的困难和问题。为了帮助更多 Java 爱好者顺利地学习和使用 Java 语言，作者以自身多年的 Java 开发和教学经历为基础，组织众多 Java 开发者将这些常见问题收集起来形成了常见问题（FAQ）列表。常见问题列表中的很多问题是作者亲身经历过的，它们也许正是你或本书的其他读者要问的问题。

关于本书

大多数 Java 语言的相关书籍都是从作者的角度写成，它们是用一种作者自己明白的方式讲解知识点，而且书中的重点内容也是作者自认为重点的内容。如果那种方式不适合你，你很可能在读完之后被弄得一头雾水，感觉像是在看天书。

而本书却不一样，它是由 261 个问题组成，所有问题都是开发人员在学习和使用 Java 语言的过程中提出的真实问题，是由众多 Java 爱好者参与整理并精简，并不是由作者一人的主观意志选取。作者在组织和编写本书内容的同时也是站在读者的角度，用自身多年的 Java 开发和工作经历验证本书内容的实用性。

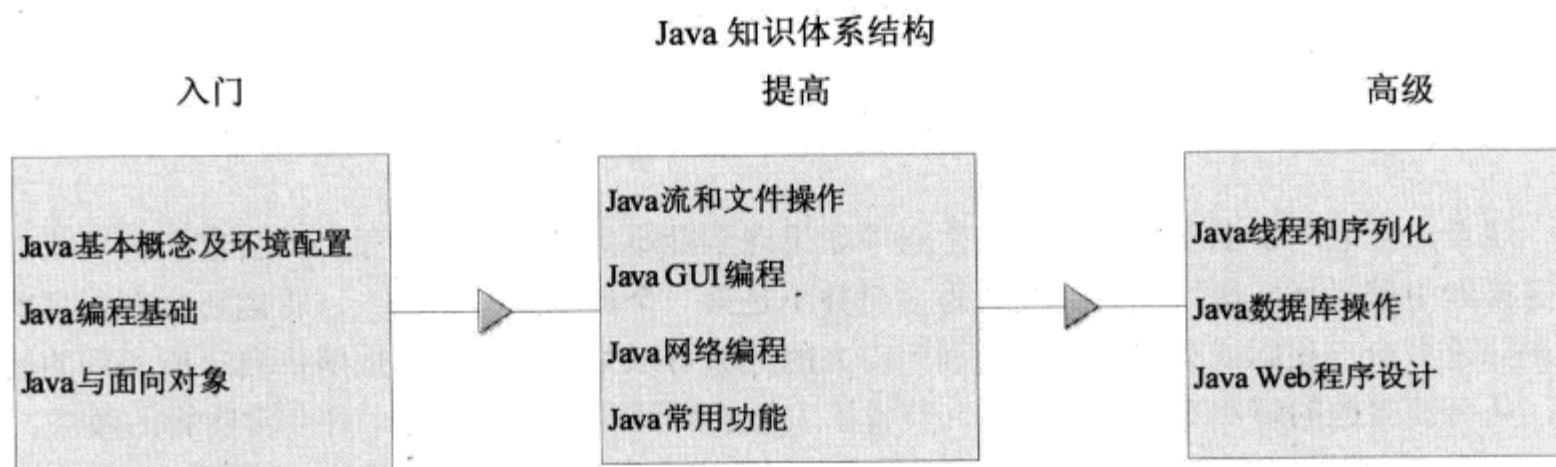
本书所列出的问题是处在各层次 Java 程序员常见问题中提炼出来的，具有典型性和普遍性，因此不能保证解答你在 Java 语言编程中遇到的所有问题。毕竟，本书内容在有限的篇幅内想要涵盖每个人的各种问题，是一件不太现实的事情。本书中的问题是开发爱好者在读完一本 Java 语言入门书或上了一门 Java 语言课之后常常会提到的，因此本书不是一步步教你学 Java 语言，而是重点在于答疑解惑。

在介绍书中问题时，首先会提示该问题被提问的频率和难度系数，然后详细描述该问题及其产生环境，其次介绍问题答案及相关知识点，最后针对该问题做一个总结并给出相关问题的链接。针对每一个问题都做到了全面、细致地剖析，如果你是一个 Java 初学者或开发者，那么本书将成为你今后学习 Java 语言的得力助手。

本书结构

本书的知识构成主要分为入门、提高和高级应用 3 个部分，每一部分都收录了大量经典的

疑难解答和常用功能，你可以根据自身情况选择不同部分进行学习。本书的知识体系结构如下图所示。



在 Java 语言的学习与开发过程中，你可能会遇到各种各样的问题，本书将根据这些问题按照不同的知识点分为各个章节，并且将问题进行深入，用讲解实例代码的方式将问题解决，使你在学习和使用 Java 语言时更加得心应手，做到学以致用。全书共分为 10 章，各章包含的主要内容如下表所示。

章节名	章节内容
第1章 Java 基本概念及环境配置	针对 Java 基本概念、开发环境配置等所产生的疑惑解答
第2章 Java 编程基础	针对 Java 编程中基本语法、数据类型、运算符及数组等操作所产生的疑惑解答
第3章 Java 与面向对象	针对 Java 编程中与面向对象相关的概念、Java 异常处理等所产生的疑惑解答
第4章 Java 流和文件操作	针对 Java 编程中 I/O 流和文件等操作所产生的疑惑解答
第5章 Java GUI 编程	针对 Java 图形用户界面编程技术所产生的疑惑解答
第6章 Java 线程和序列化	针对 Java 线程和序列化等操作所产生的疑惑解答
第7章 Java 网络编程	针对 Java 网络编程技术所产生的疑惑解答
第8章 Java 常用功能	针对 Java 基本功能、常用功能等操作所产生的疑惑解答
第9章 Java 数据库操作	针对 Java 数据库操作所产生的疑惑解答
第10章 Java Web 程序设计	针对 Java Web 程序设计技术所产生的疑惑解答

问题格式

目前市面上与 Java 语言相关的图书很多，但以问答形式介绍 Java 基础知识和关键技术的却非常少。作者在编写每个问题或需求时，都会根据技术难度的不同加以标识，并给出在实际开发中专家的处理意见。

每个问题的内容都包括以下几个部分。

- 核心解答——给出问题的解决办法和满足需求的解决方案并做适当深入，使读者获得更多的知识。
- 疑难点评——根据解答出的内容，将问题的特点进行详细说明，并给出处理此问题的注

意事项。

- 知识链接——指明当前问题与其他问题的相关知识点及原理的链接。

适用对象

- 没有 Java 基础的初学者
- 有少量基础的 Java 读者
- 大中专院校的老师 and 学生
- 对 Java 知识点产生疑问的人群
- Java 编程爱好者

技术支持

本书由启明睿智科技组织编写，参加编写的有梁建全、李坤、卞征辉、孟志勇、田立军、周力、孙强、李想、张亚东、董丽丽、石增娟、张文娟等。在本书编写过程中我们力求精益求精，虽然尽了最大的努力，但由于笔者能力有限，难免存在一些错误及不足之处，敬请读者批评指正。如果在阅读本书的过程中有意见或问题可发送 E-mail 至作者邮箱 ljq_2000@tom.com，或者寄信让出版社转交。感谢您购买本书，希望本书能够成为您的良师益友。

启明睿智科技
2009 年 8 月



目 录

第 1 章	Java 基本概念及环境配置	1
FAQ1.01	什么是面向对象程序设计?	1
FAQ1.02	面向对象程序设计的基本特征有哪些?	2
FAQ1.03	Java 语言是如何产生和发展的?	3
FAQ1.04	Android 与 Java 语言有什么关系?	4
FAQ1.05	Java SE、Java EE 和 Java ME 有什么区别?	5
FAQ1.06	Java 语言的运行机制如何?	6
FAQ1.07	什么是 JVM? 有什么作用? 工作机制如何?	8
FAQ1.08	什么是 JRE (或 J2RE)?	9
FAQ1.09	什么是 JDK?	10
FAQ1.10	JDK、JRE 和 JVM 之间有什么区别?	11
FAQ1.11	什么是 GC? GC 的工作原理如何?	12
FAQ1.12	如何安装 Java 基本开发环境 JDK?	13
FAQ1.13	为何在 JDK 安装路径下存在两个 JRE?	16
FAQ1.14	JDK 安装时设置 PATH 和 CLASSPATH 环境变量有何作用?	17
FAQ1.15	如何编译、运行 Java 应用程序?	18
FAQ1.16	如何将程序中的文档注释提取出来生成说明文档?	20
FAQ1.17	怎样制作鼠标双击就可以运行的 Jar 文件?	23
FAQ1.18	怎样给 main(String[] args)方法的 args 指定参数值?	26
第 2 章	Java 编程基础	28
FAQ2.01	Java 中的标识符如何命名? 可以用中文吗?	28
FAQ2.02	Java 中有哪些关键字?	29
FAQ2.03	用 public、protected 和 private 修饰方法有什么区别?	30
FAQ2.04	this 关键字有什么含义? 在哪些情况下应用?	31
FAQ2.05	super 关键字有什么含义? 在哪些情况下应用?	32
FAQ2.06	static 关键字有什么含义? 具体如何应用? 能修饰构造方法吗?	34
FAQ2.07	final 关键字有什么含义? 具体如何应用?	36

FAQ2.08	instanceof 关键字有什么含义? 如何应用?	37
FAQ2.09	Java 中有哪些数据类型?	38
FAQ2.10	如何解决 double 和 float 精度不准的问题?	40
FAQ2.11	int 和 Integer 都可以作为整数类型, 那么它们有什么区别?	43
FAQ2.12	float f=3.4 语句是否正确?	44
FAQ2.13	成员变量和局部变量有什么区别?	45
FAQ2.14	变量之间传值时可分为值传递和引用传递, 那么它们有何区别?	46
FAQ2.15	Java 中有哪些运算符? 优先级如何?	48
FAQ2.16	在实现 x 和 y 相加时, x+=y 和 x=x+y 两种实现方式有区别吗?	50
FAQ2.17	在执行与运算时, 运算符&和&&有什么区别?	50
FAQ2.18	在实现 x 递增加 1 操作时, x++和++x 有什么区别?	52
FAQ2.19	x?:y:z 格式的语句表示什么意思?	53
FAQ2.20	“+”操作符在 Java 内部是如何实现字符串连接的?	54
FAQ2.21	==和 equals()都可用于比较两个操作数是否相等, 它们有什么区别吗?	55
FAQ2.22	创建 String 对象时, 使用 String s=new String (“abc”)和 String s=“abc”语句有什么区别?	56
FAQ2.23	break 和 continue 语句有什么区别?	56
FAQ2.24	数组如何定义和初始化?	57
FAQ2.25	如何实现一维和二维数组的遍历?	59
FAQ2.26	如何实现数组的复制?	60
FAQ2.27	数组的排序算法有哪些? 如何实现?	61
FAQ2.28	如何解决 ArrayIndexOutOfBoundsException 异常?	64
第 3 章	Java 与面向对象	65
FAQ3.01	什么是类、对象、属性和方法?	65
FAQ3.02	什么是包? 有什么好处?	67
FAQ3.03	什么是抽象类? 有什么好处?	67
FAQ3.04	什么是接口? 有什么好处?	69
FAQ3.05	什么是多态? 有什么好处?	70
FAQ3.06	什么是内部类? 有什么好处?	71
FAQ3.07	什么是匿名内部类? 如何使用?	73
FAQ3.08	什么是封装类? 有什么作用?	74
FAQ3.09	什么是继承? 有什么好处?	75
FAQ3.10	使用 new 关键字创建对象时, 为什么有时候提示找不到无参的	

构造方法?	76
FAQ3.11 抽象类和接口都可以包含抽象方法, 那么它们有什么区别? 使用时 该如何选择?	76
FAQ3.12 什么是方法重写? 为什么需要方法重写?	78
FAQ3.13 什么是方法重载? 为什么需要方法重载?	78
FAQ3.14 构造方法是否可以被重写? 能否被重载?	79
FAQ3.15 static 修饰的方法能否在子类中重写?	80
FAQ3.16 在定义类时, 何时需要重写 Object 类中 toString() 方法?	81
FAQ3.17 在定义类时, 何时需要重写 Object 类中 equals() 方法?	82
FAQ3.18 为什么在重写 equals() 方法时, 一般都会重写 hashCode() 方法?	84
FAQ3.19 如何重写 hashCode() 方法?	85
FAQ3.20 Java 中动态绑定是什么意思?	87
FAQ3.21 Java 中是如何实现多态的? 实现机制是什么?	88
FAQ3.22 创建类的对象时, 类中各成员的执行顺序是什么样的?	89
FAQ3.23 什么是初始化块? 有什么作用?	90
FAQ3.24 静态初始化块与非静态初始化块有什么区别?	92
FAQ3.25 如何调用内部类中的方法?	93
FAQ3.26 当内部类和外部类的成员名称相同时, 如何在内部类中调用 外部类的成员?	94
FAQ3.27 匿名内部类如何访问外部方法的局部变量或参数?	95
FAQ3.28 Java 异常处理机制是什么样的?	96
FAQ3.29 常见的 RuntimeException 异常有哪些?	97
FAQ3.30 Java 中异常处理的方式有哪些?	98
FAQ3.31 try-catch-finally 语句块各部分的执行顺序如何?	100
FAQ3.32 为什么使用自定义异常? 自定义异常如何使用?	101
第 4 章 Java 流和文件操作	103
FAQ4.01 如何获取文件的属性信息?	103
FAQ4.02 如何判断文件是否为空?	105
FAQ4.03 如何实现文件的创建、删除和移动?	105
FAQ4.04 如何创建和删除文件夹?	107
FAQ4.05 如何遍历目录中所有的文件?	109
FAQ4.06 如何获取文件夹大小?	110
FAQ4.07 什么是流? 如何分类? 具体包含哪些类?	111
FAQ4.08 如何实现字节流和字符流之间的转化?	113

FAQ4.09	如何判断要读的文件是否到达末尾?	116
FAQ4.10	如何读文件、写文件?	117
FAQ4.11	如何以追加的方式写文件?	118
FAQ4.12	如何实现文件和文件夹的复制?	119
FAQ4.13	如何在文件的任意位置进行读写?	121
FAQ4.14	使用 Buffered 缓冲流写文件, 为什么内容没有写入?	122
FAQ4.15	如何实现文件的分割与合并?	123
FAQ4.16	什么是 NIO? 与 I/O 有什么区别和联系?	125
FAQ4.17	如何使用 NIO 读写文件?	127
FAQ4.18	什么是字符编码和解码?	129
FAQ4.19	读写文件时为什么中文字符经常产生乱码?	130
FAQ4.20	如何解决 FileReader 读文件乱码的问题?	131
FAQ4.21	为什么 DataInputStream 和 DataOutputStream 读写文件时乱码?	132
FAQ4.22	如何实现文件锁定功能?	134
FAQ4.23	如何实现对文件和字符串加密、解密?	135
FAQ4.24	如何实现对文件和目录的压缩、解压缩?	139
FAQ4.25	如何读写 properties 文件?	143
FAQ4.26	如何读写 XML 文件?	146
FAQ4.27	如何读写 XML 文件中的元素属性?	151
FAQ4.28	如何读写 CSV 格式的文件?	152
FAQ4.29	如何为图片文件生成缩略图?	154
FAQ4.30	如何操作 Excel 文件?	156
FAQ4.31	如何操作 Word 文件?	161
第 5 章	Java GUI 编程	164
FAQ5.01	什么是 Java GUI? Swing 与 AWT 有什么关系?	164
FAQ5.02	什么是布局管理器? 常用的布局管理器有哪些?	165
FAQ5.03	如何在窗体中显示一张图片?	170
FAQ5.04	如何为容器添加滚动条功能?	171
FAQ5.05	如何实现一个打开文件或者是存储文件的对话框?	173
FAQ5.06	如何实现弹出消息框的功能?	175
FAQ5.07	如何使用 Dialog 对话框?	177
FAQ5.08	如何为按钮添加单击事件?	178
FAQ5.09	如何为窗体添加关闭事件?	181
FAQ5.10	如何实现窗体菜单功能?	183

FAQ5.11	如何处理键盘输入事件?	185
FAQ5.12	如何处理鼠标单击事件? 如何区分是左键还是右键?	186
FAQ5.13	如何实现鼠标右键弹出菜单的功能?	189
FAQ5.14	如何使用表格组件?	191
FAQ5.15	如何实现记事本功能?	193
FAQ5.16	如何实现贪吃蛇游戏?	198
第 6 章 Java 线程和序列化		208
FAQ6.01	线程、进程和程序有何区别和联系?	208
FAQ6.02	如何创建和启动一个线程?	209
FAQ6.03	线程的基本状态有哪些?它们之间有何关系?	211
FAQ6.04	什么是线程优先级? 线程依据什么原则调度执行?	212
FAQ6.05	什么是后台线程? 如何创建一个后台线程?	214
FAQ6.06	如何使正在运行的线程在指定时间内休眠?	216
FAQ6.07	如何终止一个正在运行的线程?	218
FAQ6.08	为何 stop()和 suspend()方法不推荐使用?	219
FAQ6.09	如何控制线程的暂停和启动?	220
FAQ6.10	如何实现多个线程同步?	225
FAQ6.11	什么是对象序列化和对象反序列化?	233
FAQ6.12	实现对象序列化的方法有哪些?	234
FAQ6.13	如何实现对象在磁盘中的存取操作?	236
FAQ6.14	使用 ObjectInputStream 读取对象时为什么会发生 StreamCorruptedException 异常?	238
FAQ6.15	对象中的成员哪些参与序列化? 哪些不参与序列化?	241
FAQ6.16	如何自定义序列化和反序列化过程?	243
FAQ6.17	如何使用 Externalizable 接口定制序列化过程?	247
FAQ6.18	在序列化类中添加 serialVersionUID 属性有什么作用?	250
FAQ6.19	当序列化遭遇继承时, 如何正确处理对象序列化过程?	251
第 7 章 Java 网络编程		256
FAQ7.01	什么是 TCP/IP? 什么是 IP?	256
FAQ7.02	TCP 和 UDP 有什么区别?	258
FAQ7.03	什么是 HTTP? HTTP 的工作原理如何?	259
FAQ7.04	在 Socket 通信时如何获取主机和客户机的 IP 地址?	261
FAQ7.05	如何利用 Socket 实现基于 TCP 的通信?	264

FAQ7.06	如何利用 Socket 传输中文字符?	265
FAQ7.07	如何在 Socket 读取数据时使用超时设置?	267
FAQ7.08	如何利用 Socket 传递对象信息?	268
FAQ7.09	如何利用 Socket 实现文件传输?	271
FAQ7.10	如何基于 Socket 实现聊天系统?	274
FAQ7.11	如何利用 Socket 实现基于 UDP 的通信?	278
FAQ7.12	如何利用 UDP Socket 技术实现 IP 多点传送?	280
FAQ7.13	如何获取 Internet 资源的大小?	285
FAQ7.14	如何实现 Internet 资源的单线程下载?	286
FAQ7.15	URL 如何通过 proxy 代理访问 Internet 资源?	287
FAQ7.16	如何实现 Internet 资源下载的断点续传?	288
FAQ7.17	如何实现 Internet 资源的多线程下载?	290
FAQ7.18	如何解析 Internet 网页内容?	293
第 8 章	Java 常用功能	296
FAQ8.01	如何使字符串中包含 “\” 字符	296
FAQ8.02	如何实现字符串和整数之间的转化?	297
FAQ8.03	如何替换字符串中的字符或子字符串?	298
FAQ8.04	如何过滤字符串前后以及中间出现的空格?	299
FAQ8.05	如何对字符串中的子字符或子字符串进行截取?	301
FAQ8.06	如何判断一个字符串是否符合数值格式?	302
FAQ8.07	如何实现字符串的切割和查找?	303
FAQ8.08	如何实现十进制和二进制之间的相互转化?	306
FAQ8.09	如何将字节流转换为指定编码的字符串?	307
FAQ8.10	如何实现日期格式和字符串之间的转化?	308
FAQ8.11	String、StringBuffer 和 StringBuilder 有什么区别?	310
FAQ8.12	如何获得一个随机数?	311
FAQ8.13	List、Set 和 Map 是否继承自 Collection 接口? 有什么区别?	312
FAQ8.14	ArrayList 与 LinkedList、Vector 的区别是什么?	313
FAQ8.15	HashMap 和 Hashtable 有什么区别?	314
FAQ8.16	如何遍历 Map 和 Vector 集合?	315
FAQ8.17	如何获取系统当前时间?	317
FAQ8.18	如何获得系统属性?	318
FAQ8.19	什么是反射机制? 有什么作用?	319
FAQ8.20	如何读取键盘输入的信息?	322

FAQ8.21	如何获取当前工程目录?	323
FAQ8.22	如何使用 Java 调用系统的 exe 文件?	324
FAQ8.23	如何使用 Java 执行 cmd 命令?	325
FAQ8.24	如何使用 Java 程序打开一个 Word 文档?	326
FAQ8.25	如何使用 MD5 和 SHA 算法加密信息?	327
第 9 章 Java 数据库操作		329
FAQ9.01	什么是 JDBC? 有什么作用?	329
FAQ9.02	Java 与数据库的连接方式有哪些?	330
FAQ9.03	如何连接各种类型的数据库?	331
FAQ9.04	如何实现对数据库数据的查询?	335
FAQ9.05	如何实现对数据库数据的增加、删除和修改?	336
FAQ9.06	如何使用 PreparedStatement 对数据库操作?	339
FAQ9.07	Statement 和 PreparedStatement 有什么区别?	340
FAQ9.08	如何调用数据库中的存储过程?	341
FAQ9.09	如何通过 JDBC-ODBC 桥访问 Access 数据库?	344
FAQ9.10	连接 Oracle 数据库时 thin 和 oci 方式有什么区别?	346
FAQ9.11	如何判断 ResultSet 结果集为空?	347
FAQ9.12	如何获取 ResultSet 中含有的记录数量?	348
FAQ9.13	如何获取 ResultSet 中 n~m 位置区间的记录?	350
FAQ9.14	如何利用 ResultSet 更新数据库数据?	351
FAQ9.15	如何使用 LIKE 关键字实现模糊查询?	352
FAQ9.16	如何实现查询的分组统计和排序?	354
FAQ9.17	如何实现多表联合查询?	355
FAQ9.18	如何使用 JDBC 的批处理操作?	357
FAQ9.19	如何实现 Oracle 字段值递增的功能?	358
FAQ9.20	如何处理数据表中 Date 类型的字段?	360
FAQ9.21	如何向表中插入含有特殊字符的信息?	360
FAQ9.22	如何使用 BLOB 类型的字段存取图片?	361
FAQ9.23	如何使用 CLOB 类型的字段存取字符文件?	363
FAQ9.24	如何通过程序创建和删除数据表?	365
FAQ9.25	如何获取数据表的结构信息?	367
FAQ9.26	如何获取数据库中所有表名?	369
FAQ9.27	如何用程序备份和恢复数据库?	370
FAQ9.28	什么是事务? 如何使用 JDBC 事务控制?	372

FAQ9.29	什么是 JTA? JTA 事务与 JDBC 事务有什么区别?	373
FAQ9.30	如何使用 JTA 实现分布式事务控制?	374
FAQ9.31	什么是数据库连接池? 工作原理如何?	376
FAQ9.32	如何提升 SQL 语句的查询性能?	377
FAQ9.33	如何解决 MySQL 数据库插入乱码的问题?	379
第 10 章 Java Web 程序设计		381
FAQ10.01	什么是 JSP? JSP 的工作原理如何?	381
FAQ10.02	JSP、Java 和 JavaScript 有什么区别和联系?	383
FAQ10.03	JSP 程序开发和运行环境是什么? 如何搭建?	383
FAQ10.04	如何开发一款 JSP 程序?	387
FAQ10.05	在 JSP 中有哪些注释格式? 有什么作用?	390
FAQ10.06	JSP 中有哪些内建对象? 分别有什么作用?	391
FAQ10.07	page、request、session 和 application 有什么区别?	395
FAQ10.08	如何解决 request.getParameter()取值乱码问题?	396
FAQ10.09	JSP 中 forward 和 redirect 有什么区别?	397
FAQ10.10	如何在多个 JSP 页面之间传递信息?	399
FAQ10.11	如何解决 URL 传递中文时出现乱码的问题?	400
FAQ10.12	动态 include 与静态 include 有什么区别?	401
FAQ10.13	什么是 JavaBean? 如何使用 JavaBean?	402
FAQ10.14	什么是 Session? 如何使用 Session?	403
FAQ10.15	如何在关闭页面时自动清除 Session?	405
FAQ10.16	什么是 Cookie? 如何使用 Cookie?	406
FAQ10.17	如何在禁用 Cookie 的情况下使用 Session?	408
FAQ10.18	如何在 JSP 中避免表单的重复提交?	409
FAQ10.19	如何实现 JSP 数据和 JavaScript 数据的交互使用?	411
FAQ10.20	什么是 Servlet? Servlet 与 JSP 有什么区别?	412
FAQ10.21	Servlet 容器的工作原理如何?	413
FAQ10.22	如何在 Servlet 中使用 Session 和 Application?	414
FAQ10.23	如何编写多线程安全的 Servlet 程序?	415
FAQ10.24	如何在 Servlet 和 JSP 中获取工程文件的绝对路径?	417
FAQ10.25	如何获取客户端浏览器和操作系统信息?	417
FAQ10.26	如何在 Web 程序中实现定时运行的功能?	418
FAQ10.27	如何实现网站登录记忆跳转的功能?	421
FAQ10.28	如何将 JSP 动态页面转换为 HTML 静态页面?	423

FAQ10.29	如何实现数据分页显示的功能?	424
FAQ10.30	如何将 JSP 内容以 Excel 或 Word 格式输出?	431
FAQ10.31	如何在 JSP 中实现打印功能?	432
FAQ10.32	如何实现图片验证码功能?	433
FAQ10.33	如何实现饼状图、柱状图和曲线图?	436
FAQ10.34	如何实现进度条显示功能?	443
FAQ10.35	如何实现网站计数器功能?	447
FAQ10.36	如何发送 HTML 格式和带附件的邮件?	448
FAQ10.37	如何实现文件的上传和下载?	453
FAQ10.38	如何禁止浏览器缓存页面内容?	457
FAQ10.39	如何在网页中在线播放音乐和视频?	458
FAQ10.40	如何处理 JSP 页面的错误?	460
FAQ10.41	如何利用过滤器实现权限验证功能?	462
FAQ10.42	如何实现 JSP 防盗链功能?	464

第 1 章

Java 基本概念及环境配置

本章重点介绍一些与 Java 语言编程相关的基本概念和开发环境的配置。刚刚开始接触 Java 编程的读者,通常会被一些比较基础的问题所困扰,例如感觉面向对象思想太抽象、不清楚 Java 语言的运行机制、不知道如何开发一个 Java 程序、不知道如何编译运行 Java 程序、不知道如何将 Java 程序打包发布等。本章将针对初学者常见的一些疑难问题进行详细解答,帮助读者快速、顺利地进入 Java 初级阶段的学习。

FAQ1.01 什么是面向对象程序设计?

📖 难度系数: ★★★

📖 问题频率: 70%

核心解答

程序设计的本质是把人们在现实生活中遇到的问题通过抽象处理,利用编程语言转换到计算机能够理解的层面上去。程序设计从开始到现在,大致经历了过程式程序设计、结构化程序设计和面向对象程序设计 3 个阶段。

- ❑ 过程式程序设计需要开发者对程序的每一步进行精确地设计和严格控制。
- ❑ 结构化程序设计需要开发者在编码之前将程序进行完整的规划,设计出各种图表,画出各种数据的流向,指明各个函数之间的相互作用,是一种自顶向下、逐步求精、使程序结构模块化的程序设计方法。
- ❑ 面向对象程序设计(OOP)是将对象作为程序的基本单元,并将程序和数据封装在其中,以提高软件的重用性、灵活性和扩展性,每一个对象都代表现实世界中的一个具体事物(或者称为“实体”)。

面向对象程序设计是现今主流的程序设计思想,当前流行的 Java、C#等都属于面向对象程序设计语言。面向对象的编程思想力图使程序和现实世界中的具体实体完全一致,这样可以使开发者和用户之间能更好地理解和沟通。

疑难点评

面向对象程序设计是现今主流的程序设计思想,Java 语言是属于面向对象的程序设计语言,因此了解面向对象程序设计思想对以后学习 Java 开发至关重要。

知识链接

FAQ1.02 面向对象程序设计的基本特征有哪些?

FAQ1.03 Java 语言是如何产生和发展的?

FAQ1.02 面向对象程序设计的基本特征有哪些?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

面向对象程序设计的基本特征是封装、继承和多态。

(1) 封装

封装是指将对象相关的状态信息和行为捆绑为一个逻辑单元,即将客观事物封装成抽象的类。通过封装可以隐藏一个类的实现细节,使用者可以通过指定的方法来访问该类的对象,而不必关心其内部细节。

(2) 继承

继承是指一个类继承另一个类后,即可以获得另一个类的属性和方法,继承者为子类,被继承者为父类。通过类之间的继承,实现了代码的重复利用,在子类中可以新增属性和方法,并且可以重写父类中方法的具体实现方式。在 Java 中一个子类只能继承一个父类,不能同时继承多个父类, Object 类是所有类的顶级父类。

(3) 多态

多态在 Java 中是指对象变量是多态的,一个类型为 A 的变量既可以指向类型为 A 的对象,又可以指向 A 的任何子类的对象。在方法调用中,以多态的形式来传递参数,可以增强参数类型的灵活性。一个接口类型的变量也可以指向该接口实现类的对象。

疑难点评

封装、继承和多态是面向对象程序设计的 3 个基本特征,这些基本特征在 Java 语言中提供了良好的实现,在 Java 开发过程中会经常涉及。深入理解封装、继承和多态的概念有助于初学者对 Java 语言的学习和应用。

知识链接

- FAQ1.01 什么是面向对象程序设计?
- FAQ3.05 什么是多态?有什么好处?
- FAQ3.08 什么是封装类?有什么作用?
- FAQ3.09 什么是继承?有什么好处?

FAQ1.03 Java 语言是如何产生和发展的?

📖 难度系数: ★★

📖 问题频率: 85%

核心解答

Java 是由 Sun 公司于 1995 年 5 月推出的 Java 程序设计语言和 Java 平台的总称。

1991 年, 美国 Sun 公司的某个研究小组为了能够在消费电子产品上开发应用程序, 积极寻找合适的编程语言。消费电子产品种类繁多, 包括 PDA、机顶盒、手机等, 即使是同一类消费电子产品所采用的处理芯片和操作系统也不相同, 还存在着跨平台的问题。当时最流行的编程语言是 C 和 C++, Sun 公司的研究人员就考虑是否可以采用 C++ 语言来编写消费电子产品的应用程序。但是研究表明, 对于消费电子产品而言 C++ 语言过于复杂和庞大, 并不适用, 安全性也无法令人满意。于是, Bill Joy 领导的研究小组就着手设计和开发出一种语言, 称之为 Oak。该语言采用了许多 C 语言的语法, 提高了安全性, 是一种面向对象的语言, 但是 Oak 语言在当时的商业环境中并未获得成功。

1995 年, 随着互联网的迅速发展, Sun 公司发现 Oak 语言所具有的跨平台、面向对象和安全性高等特点非常符合互联网的需要, 于是改进了该语言的设计, 并取名为 Java。

Java 平台由 Java 虚拟机和 Java 应用编程接口构成。在硬件或操作系统平台上安装一个 Java 平台之后, Java 应用程序即可运行。现在由于 Java 平台可以嵌入几乎所有的操作系统中, 因此具有跨平台的优点。Java 应用编程接口为 Java 应用提供了一个独立于操作系统的 API, 分为基本部分和扩展部分。经过 10 多年的发展, Java API 已经从 1.0 版本发展到最近的 7.0 版本。

疑难点评

Java 语言从产生到现在已经经历了 10 多年的完善和发展, 目前已经非常成熟, 其应用范围也非常广泛, 在应用软件、手机软件、企业应用中都能捕捉到它的身影。Java 语言以其特有的魅力和优势, 赢得了众多 IT 公司和程序开发者的认可和追随, 就连 Google 公司最近推出的 Android 手机开发平台也采用了 Java 开发语言。随着 3G 业务在中国的不断发

台开发会成为 Java 语言应用中的又一大热点,同时也将会为 Java 程序员的发展带来更加广阔的天地。

知识链接

FAQ1.01 什么是面向对象程序设计?

FAQ1.04 Android 与 Java 语言有什么关系?

FAQ1.05 Java SE、Java EE 和 Java ME 有什么区别?

FAQ1.06 Java 语言的运行机制如何?

FAQ1.04 Android 与 Java 语言有什么关系?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

1. Android 介绍

Android 一词的意思是指“机器人”,Android 是 Google 公司于 2007 年 11 月 5 日宣布的基于 Linux 平台的开源手机操作系统的名称,该平台由操作系统、中间件、用户界面和应用软件组成。Google 公司称其是首个为移动终端打造的真正开放和完整的移动软件。

2008 年 9 月 22 日,美国运营商 T-Mobile USA 在纽约正式发布第一款 Google 手机(T-Mobile G1)。该款手机为台湾宏达电(HTC)代工制造,是世界上第一部使用 Android 操作系统的手机,支持 WCDMA/HSPA 网络,理论下载速率 7.2 Mbit/s,并支持 WiFi(WiFi 是一种无线联网的技术)。

Android 平台是 Google 公司和“开放手机联盟”合作开发的,这个联盟由包括中国移动、摩托罗拉、高通、宏达电和 T-Mobile 在内的 30 多家技术和无线应用的领军企业组成。Android 是一个真正意义上的开放性移动设备综合平台,它包括操作系统、用户界面和应用程序等移动电话工作所需的全部软件,而且不存在任何以往阻碍移动产业创新的专有权障碍。

Android 平台的研发队伍阵容强大,包括 Google、HTC、PHILIPS、T-Mobile、高通、魅族、摩托罗拉、三星、LG 以及中国移动在内的 34 家企业都将基于该平台开发手机的新型业务,应用之间的通用性和互联性将在最大程度上得到保持,同时新型手机设备的研发成本也将大大降低。

2. Java 平台与 Java 语言的关系

Java 编程语言与 Java 平台是两个完全不同的概念。前者泛指一系列编程的语法,而后者包括前者,同时又超出前者的范围。一般而言,Java 平台由三部分组成,分别为核心的 Java API(包、框架及类库)、Java 字节码(编译且可执行的形式)以及 Java 虚拟机(JVM,执行字节码

的运行机制)。Java 语言只不过是 Java 平台中的一小部分, 因为其他语言同样可以实现 Java 语言的功能, 例如 Groovy、JRuby 及 JPython 等, 这些语言同样可以编写出运行在 JVM 上执行的字节码。

3. Android 平台与 Java 语言的关系

Google 公司推出的 Android 是一款手机平台, 其功能等价于 Java 平台, 它不仅使用了 Java 编程语言, 而且还使用了核心的 Java API。但是, Android 的可执行形式与 Java 平台的字节码形式是不同的, 同时, Android 使用的虚拟机与 Java 平台的 JVM 也不一样, 因此 Java 环境下生成的 Java 字节码在 Android 平台上是不能执行的。

Google 公司没有使用标准的 JME (Java Monkey Engine, 一款 Java 3D 游戏引擎) 作为运行 Java 应用程序的引擎, 而是为 Android 配备了名为 Dalvik 的虚拟机; 这样做可以避免因使用 JME 所带来的与 Sun 公司之间的纷争问题。与 Java 虚拟机不同, Dalvik 虚拟机执行的是 Dalvik 字节码, 而不再是 Java 字节码, 因此, Android 平台与 Java 平台既存在相似又存在不同。

疑难点评

通过上面的介绍可以看出, Google 公司的 Android 平台与 Sun 公司的 Java 平台非常相似, 却又有不同, 因此从 Android 发布至今, Google 公司和 Sun 公司在某些问题上一直存在一些争论。例如, 在 Google 发布 Android 平台的当天, Sun 公司的 CEO 施瓦兹在一篇博客中将 Android 称为是 Java/Linux 平台。相反, Google 则在避免将 Android 称为是 Java 平台, 它将 Android 软件开发工具包称作是让开发人员开发使用 Java 的应用软件的一系列工具。

知识链接

FAQ1.03 Java 语言是如何产生和发展的。

FAQ1.05 Java SE、Java EE 和 Java ME 有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

Java 现在已不仅仅是一种语言, 从广义上说它代表了一个技术体系, 该体系根据应用方向的不同主要分为 Java SE、Java EE 和 Java ME 3 个部分。

Java SE 全称为 Java Standard Edition, Java EE 全称为 Java Enterprise Edition, Java ME 全称为 Java Micro Edition。

1998 年 12 月份 Sun 公司发布的 Java 1.2 版本, 同时将它的名字改称为“Java 2 软件开发工具箱 1.2” (JDK 1.2), 它和它的后续版本也通常被称为“Java 2 标准版” (J2SE)。在 J2SE 推

出的同时,还推出了“Java 2 微缩版”(J2ME)和“Java 2 企业版”(J2EE)。

- J2SE 为创建和运行 Java 程序提供了最基本的环境,是 Java 技术的核心和基础。J2EE 和 J2ME 都建立在 J2SE 基础之上。
- J2EE 为基于服务器的分布式企业应用提供开发和运行环境,是目前 Java 技术应用最广泛的部分。J2EE 不仅继承了 J2SE 中的许多优点,同时还提供了对 EJB、JSP、Servlet 以及 XML 技术的全面支持,降低了企业级开发的复杂度。
- J2ME 为嵌入式应用提供开发和运行环境,例如手机程序和 PDA 程序等。

在 Java 5.0 (或者称为 1.5) 版本推出后,为了避免版本混淆,便将 J2SE、J2EE 和 J2ME 改称为 Java SE 5、Java EE 5 和 Java ME 5。后续版本只变更相应的版本号,例如 Java EE 6。

疑难点评

目前,Java 平台包括 3 个版本,它们是适用于小型设备和智能卡的 Micro 版 (Java ME)、适用于桌面系统的标准版 (Java SE) 和适用于创建服务器应用程序和服务的企业版 (Java EE)。这 3 个版本的平台具有不同的应用领域,可开发和运行不同需求的应用软件,它们各自都包含了一系列的开发技术和规范。

在学习 Java 开发过程中,不仅要学习 Java 语言及其语法,还要重点学习和掌握各平台所包含的主要技术和规范。比如 Java SE 可以重点学习集合类、文件操作、I/O 流、线程、序列化和 GUI 窗体编程等内容;Java EE 可以重点学习 JSP、Servlet、JDBC、EJB 以及现在流行的 Struts、Spring、Hibernate 等轻量级框架;Java ME 则可以重点学习配置 (Configuration) 和简表 (Profile) 等。

知识链接

FAQ1.03 Java 语言是如何产生和发展的?

FAQ1.04 Android 与 Java 语言有什么关系?

FAQ1.06 Java 语言的运行机制如何?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

计算机高级编程语言按其程序的执行方式可分为编译型语言 and 解释型语言。

编译型语言是指使用专门的编译器,针对特定操作系统将源程序代码一次性翻译成计算机能识别的机器指令。例如 C、C++ 等都属于编译型语言。

解释型语言是指使用专门的解释器,将源程序代码逐条地解释成特定平台的机器指令,解

释一句执行一句，类似于“同声翻译”。例如 ASP、PHP 等都属于解释型语言。

Java 既不是编译型语言也不是解释型语言，它是编译型和解释型语言的结合体。首先采用通用的 Java 编译器将 Java 源程序编译成为与平台无关的字节码文件（class 文件），然后由 Java 虚拟机（JVM）对字节码文件解释执行。Java 代码的具体执行过程如图 1-1 所示。

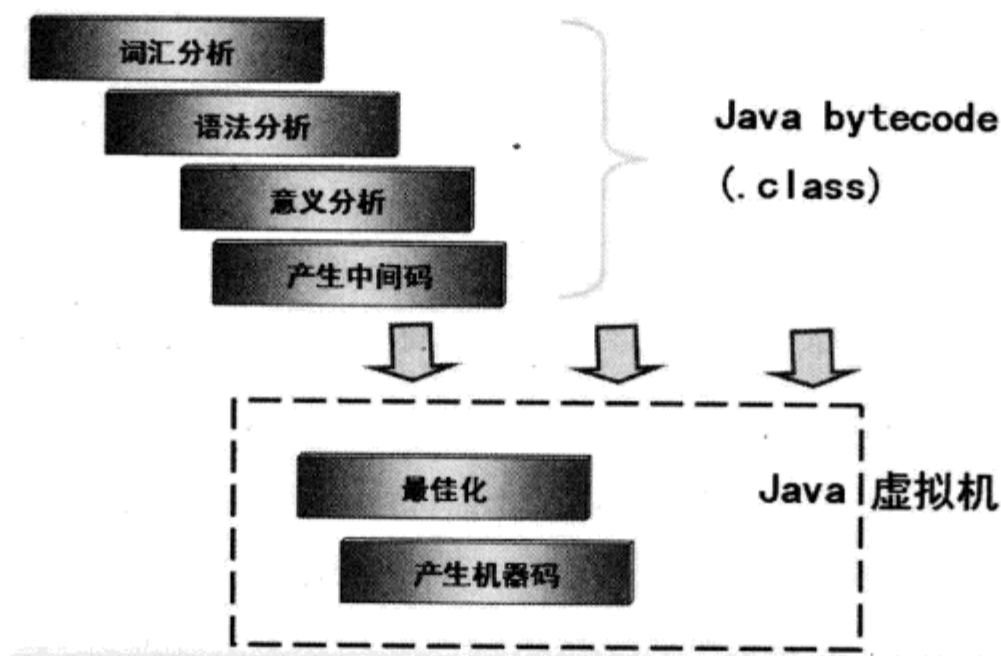


图 1-1 Java 运行机制

疑难点评

编译型语言和解释型语言都有其各自的优缺点，比如编译型语言会在程序第一次执行时将其全部编译成与当前系统平台相对应的机器指令，在后续执行时，直接运行第一次编译的结果，减少了编译次数，提高了程序运行效率，但是程序在第一次编译时与系统平台相对应，因此移植性比较差。而解释型语言在程序每次运行时都要将源程序解释成当前系统平台相对应的机器指令，因此每一次运行都需要解释并执行，运行效率较低，但是移植性强。

Java 语言综合了编译型和解释型语言的优点，采取了一种折中方案。即 Java 语言首次运行时采取编译机制将 Java 源程序编译成 Java 字节码文件，该字节码与系统平台无关，是介于源代码和机器指令之间的一种状态。在后续执行时，采取解释机制将 Java 字节码解释成与系统平台对应的机器指令。这样既减少了编译次数，又增强了程序的可移植性，因此被称为“一次编译，多处运行！”。

提示：Java 字节码具有平台无关性，可以在各种不同系统平台中运行，但是，需要有不同的版本的 Java 虚拟机，不同系统平台的 Java 运行环境其 Java 虚拟机是不一样的。

知识链接

FAQ1.07 什么是 JVM？有什么作用？工作机制如何？

FAQ1.08 什么是 JRE（或 J2RE）？

FAQ1.07 什么是 JVM? 有什么作用? 工作机制如何?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

1. JVM 定义

JVM (Java Virtual Machine) 意思是 Java 虚拟机。它是一个虚构出来的计算机, 可在实际的计算机上模拟各种计算机功能。JVM 有自己完善的硬件架构, 例如处理器、堆栈和寄存器等, 还具有相应的指令系统。

2. JVM 作用

JVM 是 Java 字节码执行的引擎, 为 Java 程序的执行提供必要的支持, 它还能优化 Java 字节码, 使之转换成功率更高的机器指令。程序员编写的程序最终都要在 JVM 上执行, JVM 中类的装载是由类加载器 (ClassLoader) 和它的子类来实现的。ClassLoader 是 Java 运行时一个重要的系统组件, 负责在运行时查找和装入类文件的类。

JVM 屏蔽了与具体操作系统平台相关的信息, 从而实现了 Java 程序只需生成在 JVM 上运行的字节码文件 (class 文件), 就可以在多种平台上不加修改地运行。不同平台对应着不同的 JVM, 在执行字节码时, JVM 负责将每一条要执行的字节码送给解释器, 解释器再将其翻译成特定平台环境的机器指令并执行。Java 语言最重要的特点就是跨平台运行, 使用 JVM 就是为了支持与操作系统无关, 实现跨平台运行。

3. JVM 工作原理

JVM 在整个 JDK 中处于最底层, 负责与操作系统的交互, 用来屏蔽操作系统环境, 提供一个完整的 Java 运行环境, 因此也称为虚拟计算机。操作系统装入 JVM 是通过 JDK 中的 java.exe 来实现, 主要通过以下几个步骤完成。

- (1) 创建 JVM 装载环境和配置。
- (2) 装载 jvm.dll。
- (3) 初始化 jvm.dll。
- (4) 调用 JNIEnv 实例装载并处理 class 类。
- (5) 运行 Java 程序。

疑难点评

JVM 是 Java 运行环境的最核心部分, 是运行 Java 程序的最基本环境, 想了解 Java 运行机制需要对 JVM 的概念和运行机制有所了解。

知识链接

FAQ1.06 Java 语言的运行机制如何?

FAQ1.10 JDK、JRE 和 JVM 之间有什么区别?

FAQ1.08 什么是 JRE (或 J2RE) ?

📖 难度系数: ★★

📖 问题频率: 85%

核心解答

1. JRE 简介

JRE 是 Java Runtime Enviroment 的简称, 即 Java 运行时环境, 它是 Java 程序运行所必须的环境集合, 主要由 Java 虚拟机、Java 平台核心类和若干支持文件组成。JRE 不包含开发工具、编译器、调试器以及其他工具。J2RE 是 Java2 Runtime Environment 的简称, 有时简称为 JRE。

Sun 公司的 JRE 产品, 包括 Java Runtime Environment 和 Java Plug-in Java Runtime Environment 两部分, 是可以运行、测试和传输应用程序的 Java 平台。

如果需要在浏览器中运行 Java Applet 程序, JRE 需要辅助软件——Java Plug-in。Java Plug-in 软件可以使 Java Applet 和 JavaBeans 组件在使用 Sun 公司的 JRE 环境的浏览器中运行, 而不是使用缺省的 Java 运行环境的浏览器中运行。Java Plug-in 可用于 Navigator 和 Internet Explorer 浏览器。

如果只需要运行 Java 程序或 Applet 程序, 下载并安装 JRE 环境即可。如果要自行开发 Java 软件, 需要下载 JDK 软件, JDK 软件中附带有 JRE 环境。

提示: 由于 Microsoft 公司的产品对 Java 环境的支持不完全, 请不要使用 IE 浏览器自带的虚拟机来运行 Applet 程序, 最好安装一个 Sun 公司的 JRE 或 JDK。

2. JRE 的版本管理

很多与 Java 开发相关的产品都会自带一套 JRE 环境, 例如 Weblogic、JBuilder、Oracle 和 Ration Rose 等软件, 因此很容易发生在同一台计算机上安装很多不同版本 JRE 的情况, 但这不会给软件运行带来冲突, 不同版本的 JRE 不会相互影响, 因为在控制台使用 java.exe 工具执行 Java 程序时, 操作系统将按如下顺序寻找 JRE 环境。

(1) 先查找当前目录下有没有 JRE。

(2) 再查找父目录下有没有 JRE。

(3) 接着在环境变量 PATH 指定的路径中查找 JRE。

(4) 注册表 HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment\ 查看 CurrentVersion 的键值指向哪个 JRE。

最常用的是在 PATH 路径中查找 JRE 环境, 一般情况下, 每一种 Java 软件在运行之前都会先在批处理文件里临时设置 PATH 值, 然后把自己使用的 JRE 路径放到 PATH 路径最前面, 所以肯定会使用自己带的 JRE, 不会造成版本混乱。

3. JRE 的基础类库

JRE 自带的基础类库主要在 JRE\lib\rt.jar 文件中, 该文件包括了 Java 2 平台标准版的所有类库, 与 JRE 的版本是一致的。在程序运行时, JRE 由 ClassLoader (类加载器) 负责查找和加载程序引用到的基类库和其他类库。基础类库, ClassLoader 会自动到 rt.jar 中加载, 操作系统通过 PATH 环境变量来查找 JRE 并确定基础类库文件 rt.jar 的位置; 其他的类库, ClassLoader 在环境变量 CLASSPATH 指定的路径中搜索, 按照先来先到的原则, 放在 CLASSPATH 前面的类库优先被搜到, 因此建议在 Java 程序启动之前先把 PATH 和 CLASSPATH 环境变量设置好。

疑难点评

JRE 是 Java 中非常重要的关键词, 很多人只了解它是 Java 的运行时环境, 但是对其内部构成和运行机制一无所知, 并没有作进一步了解。作为一个初学者, 不仅要求了解 JRE 的概念和作用, 最好对 JRE 做一些深入的了解, 这对以后学习 Java 有很大的帮助。

知识链接

FAQ1.06 Java 语言的运行机制如何?

FAQ1.10 JDK、JRE 和 JVM 之间有什么区别?

FAQ1.09 什么是 JDK?

📖 难度系数: ★★

📖 问题频率: 85%

核心解答

JDK 是 Java Development Kit 的简称, 即 Java 开发工具包。JDK 是 Sun 公司针对 Java 开发者的产品, 提供了 Java 的开发环境和运行环境。自从 Java 推出以来, JDK 已经成为使用最广泛的 Java SDK (Software development kit)。

JDK 是整个 Java 的核心, 它包括了 Java 运行环境 (Java Runtime Environment)、一堆 Java 工具和 Java 基础的类库 (rt.jar)。不论什么 Java 应用服务器实质都是内置了某个版本的 JDK, 因此掌握 JDK 是学好 Java 的第一步。最主流的 JDK 是 Sun 公司发布的 JDK, 除此之外, 还有很多公司和组织都开发了自己的 JDK, 例如 IBM 公司开发的 JDK、BEA 公司开发的 Jrocket 以及 GNU 组织开发的 JDK 等。

JDK 中除了包括 JRE 的全部内容外, 它还包含开发者用以编译, 调试和运行 Java 程序的工

具，主要工具及其介绍如表 1-1 所示。

表 1-1 JDK 常用工具介绍

工 具 名 称	功 能 描 述
javac.exe	编译器，用于将 Java 源程序转成字节码
jar.exe	打包工具，用于将相关的类文件打包成一个文件
javadoc.exe	文档生成器，从源码注释中提取文档
jdb.exe	debugger，查错工具
java.exe	运行编译后的 java 程序，即以.class 为后缀的字节码文件
appletviewer.exe	小程序浏览器，一种执行 HTML 文件上的 Java 小程序的 Java 浏览器
javah.exe	产生可以调用 Java 过程的 C 过程，或建立能被 Java 程序调用的 C 过程的头文件
javap.exe	Java 反汇编器，显示编译类文件中的可访问功能和数据，同时显示字节代码含义
jconsole.exe	Java 进行系统调试和监控的工具

疑难点评

JDK 是 Java 开发必备的工具，它包含了很多开发工具，最常使用的是 javac.exe、java.exe、jar.exe 和 javadoc.exe。目前有很多流行的 IDE 集成开发工具，例如 Eclipse、NetBean 和 JBuilder 等，利用这些工具的确可以快速方便地进行 Java 开发，但是作为初学者建议还是通过命令行界面进行学习，后期再采用集成工具。

知识链接

FAQ1.10 JDK、JRE 和 JVM 之间有什么区别？

FAQ1.12 如何安装 Java 基本开发环境 JDK？

FAQ1.10 JDK、JRE 和 JVM 之间有什么区别？

📖 难度系数：★★★

📖 问题频率：90%

核心解答

图 1-2 来源于 Sun 公司的官方帮助文档。从帮助文档中可以看出 JDK、JRE 和 JVM 之间的关系。

JDK、JRE 和 JVM 之间是一种包含关系，范围由大到小依次为 JDK、JRE 和 JVM。JDK 中包含 JRE，JRE 中包含 JVM。

疑难点评

JDK、JRE 和 JVM 是 Java 中非常相似又具有联系的 3 个关键词，很容易混淆，在此将它们

一起比较分析，以帮助读者对它们的概念和作用进行清晰区分。

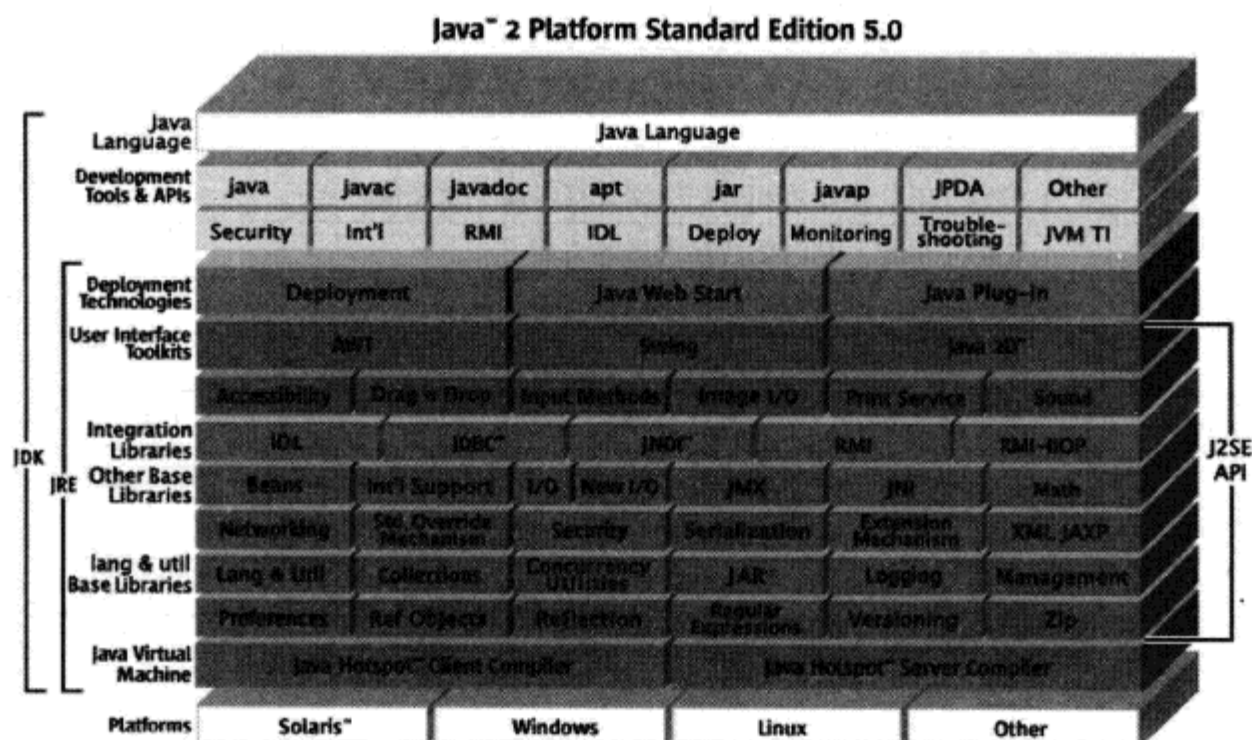


图 1-2 Java 运行机制

知识链接

FAQ1.07 什么是 JVM？有什么作用？工作机制如何？

FAQ1.08 什么是 JRE（或 J2RE）？

FAQ1.09 什么是 JDK？

FAQ1.11 什么是 GC？GC 的工作原理如何？

📖 难度系数：★★★

📖 问题频率：80%

核心解答

Garbage Collection 简称为 GC，是垃圾回收的意思。

内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃。Java 语言提供的 GC 功能可以自动监测对象是否超过作用域，从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法，资源回收工作全部交由 GC 来完成，程序员不能精确控制垃圾回收的时机。

下面简要介绍一下 GC 在实现垃圾回收时的基本工作原理。

Java 的内存管理实际上就是对象的管理，其中包括对象的分配和释放。对于程序员来说，

分配对象使用 `new` 关键字, 释放对象时只要将对象赋值为 `null`, 让程序不能够再访问到这个对象, 该对象被称为“不可达”。GC 将负责回收所有“不可达”对象的内存空间。

对于 GC 来说, 当程序员创建对象时, GC 就开始监控这个对象的地址、大小以及使用情况。通常 GC 采用有向图的方式记录并管理堆中的所有对象, 通过这种方式确定哪些对象是“可达”, 哪些对象是“不可达”。当 GC 确定一些对象为“不可达”时, GC 就有责任回收这些内存空间, 但是为了保证 GC 能够在不同的平台实现, Java 规范对 GC 的很多行为都没有进行严格的规定。例如对于采用什么类型的回收算法、什么时候进行回收等重要问题都没有明确的规定。因此不同的 JVM 实现者通常有不同的实现算法, 这也给 Java 程序员的开发带来很多不确定性。

根据 GC 的工作原理, 可以通过一些技巧和方式让 GC 运行更加合理、高效, 以下是一些 Java 编程时的建议。

- 尽早释放无用对象的引用, 特别注意一些复杂的对象, 例如数组, 队列等。对于此类对象, GC 回收它们的效率一般较低。如果程序允许, 应尽早将不用的引用对象赋为 `null`, 这样可以加速 GC 的工作。
- 尽量少用 `finalize` 函数。`finalize` 是 Java 提供给程序员用来释放对象或资源的函数, 但是它会加大 GC 的工作量, 因此尽量少采用 `finalize` 函数回收资源。

当程序有一定的等待时间, 程序员可以手动执行 `System.gc()`, 通知 GC 运行, 但是 Java 语言规范并不保证 GC 一定会执行。

疑难点评

Java 与 C 和 C++ 相比, 在内存回收方面做了很大改进, 简化了开发者对内存资源的分配和回收。GC 主要负责内存资源的管理, 读者可以深入了解下 GC, 例如 GC 工作原理、回收算法等。

FAQ1.12 如何安装 Java 基本开发环境 JDK?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

JDK 是 Java 开发工具包的简称, 是开发 Java 程序的最基本环境。现在流行的集成开发工具 (例如 Eclipse、Jbuilder 和 NetBean 等) 都是必须基于 JDK 环境, 只不过有些集成工具在安装过程中内置安装了 JDK, 有些则需要使用者事先单独安装。

下面以 JDK 6.0 为例详细介绍 JDK 安装的基本过程。

(1) 下载 JDK 安装文件

下载地址为 <http://java.sun.com/javase/downloads/index.jsp>, 安装文件的名称是 `jdk-6u5-windows-i586-p.exe`。

(2) 安装 JDK

双击 jdk-6u5-windows-i586-p.exe, 进入欢迎安装界面, 然后阅读许可证协议, 如图 1-3 所示。

在图 1-3 中单击【接受】按钮后, 出现“自定义安装”对话框, 如图 1-4 所示。

在图 1-4 中单击【更改】按钮可以改变安装目录。单击【下一步】按钮开始安装, 等待安装完成, 如图 1-5 所示。

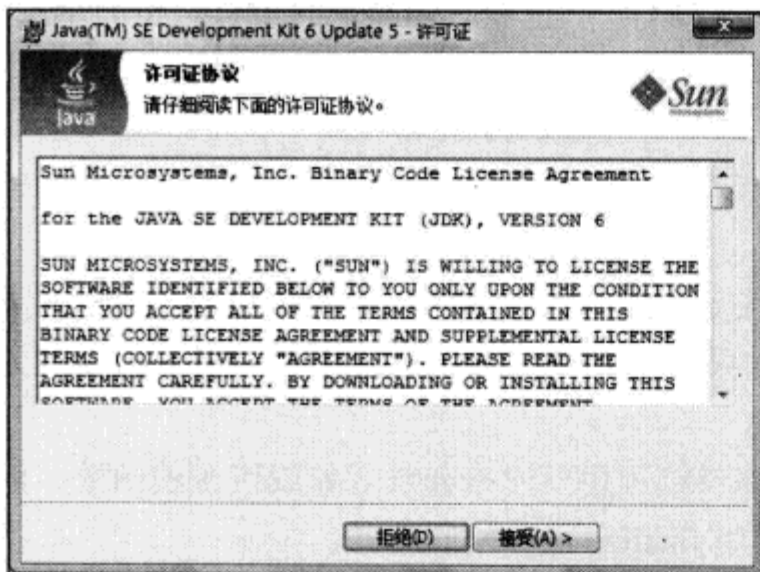


图 1-3 许可证协议

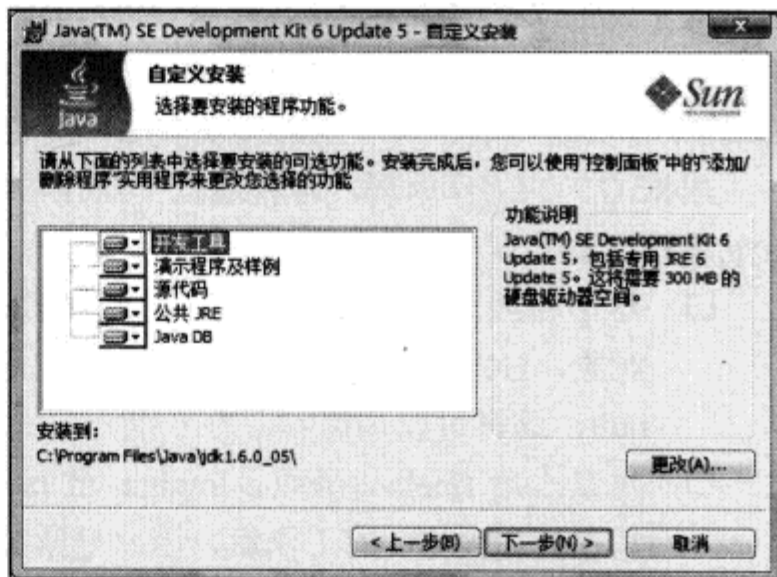


图 1-4 “自定义安装”对话框

(3) 设置环境变量

在 JDK 安装完毕后, 为了以后在命令模式下编译、运行程序方便, 需要进行系统环境配置, 具体操作如下。

选择操作系统桌面的【我的电脑】图标, 单击鼠标右键, 在弹出的菜单中选择【属性】菜单项, 将弹出“系统属性”对话框, 如图 1-6 所示。

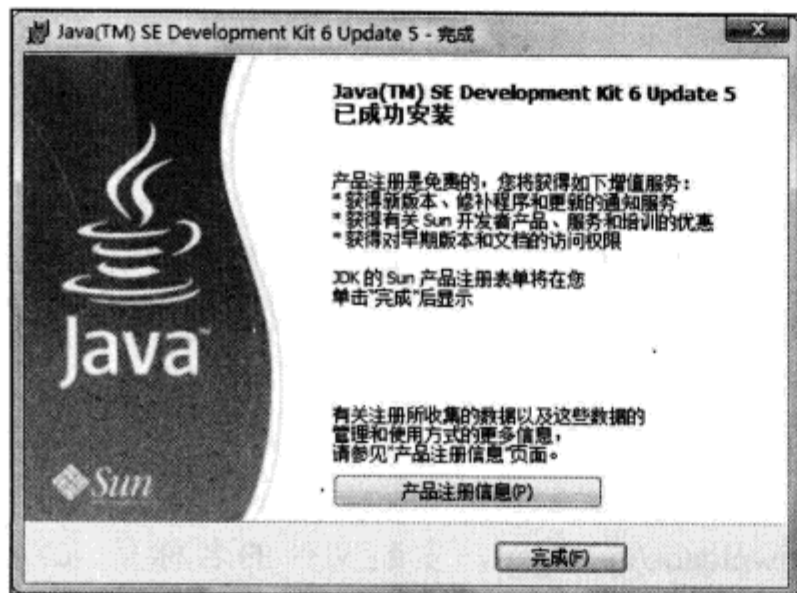


图 1-5 安装完成

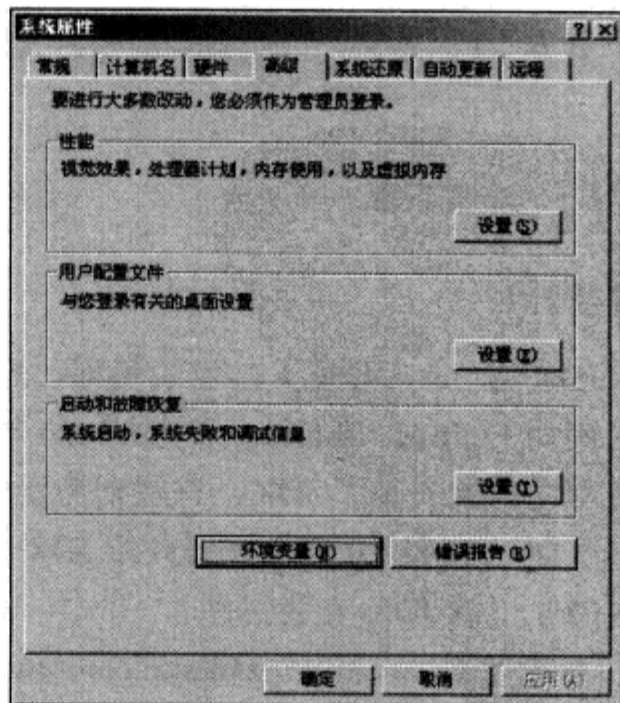


图 1-6 “系统属性”对话框

在图 1-6 中选择【高级】选项卡，单击【环境变量】按钮，将弹出“环境变量”对话框，如图 1-7 所示。

在“系统变量”列表框中设置环境变量，具体设置信息如表 1-2 所示。

注意：环境变量值可以指定多项，项与项之间需要用“;”隔开。

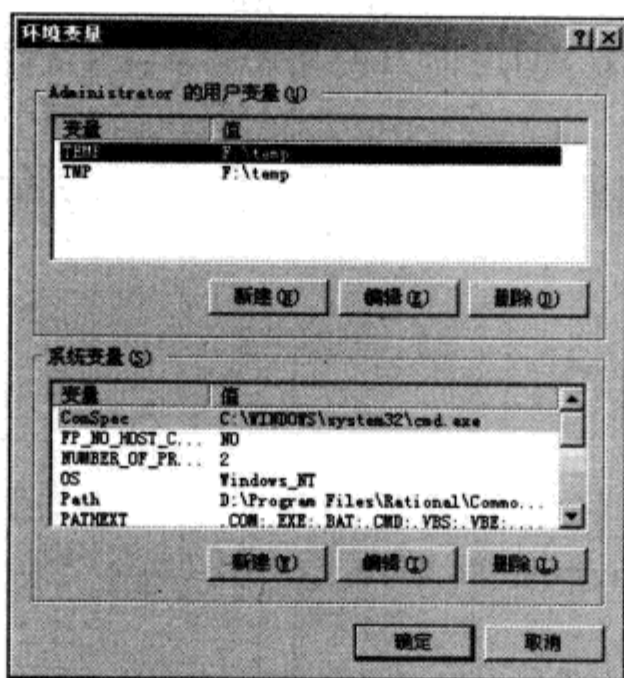


图 1-7 “环境变量”对话框

表 1-2

JDK 环境变量设置

环境变量名称	环境变量值
JAVA_HOME	C:\Program Files\Java\jdk1.6.0_05
CLASSPATH	.;%JAVA_HOME%\lib\dt.jar; %JAVA_HOME%\lib\tools.jar
PATH	%JAVA_HOME%\bin (如果 PATH 环境变量已经存在，可以将该值加到原值的最前)

(4) 了解 JDK 安装目录

在 JDK 安装目录下有很多子文件夹和文件，其中需要重点了解的有以下几项。

- ☐ bin 文件夹：包含了 JDK 提供的工具程序，例如程序编译 (javac.exe)、生成文档 (javadoc.exe)、程序运行 (java.exe) 和程序打包 (jar.exe) 等。
- ☐ demo 文件夹：包含了一些 Java 编写的范例程序。
- ☐ lib 文件夹：工具程序的实现类，例如 javac.exe 实际上使用 tools.jar 中的 com/sun/tools/javac/Main 类。
- ☐ src.zip 文件：Java 提供的 API 类的源代码压缩文件。如果将来需要查看 API 的某些功能是如何实现的，可以查看这个文件中的源代码内容。
- ☐ jre 文件夹：JDK 自带的 Java 运行环境，为 javac.exe 等工具程序提供服务。

疑难点评

JDK 是 Java 开发的基本环境，在学习 Java 编程之前都要安装 JDK，因此读者需要对 JDK

的安装过程熟练掌握。

知识链接

FAQ1.09 什么是 JDK?

FAQ1.13 为何在 JDK 安装路径下存在两个 JRE?

FAQ1.14 JDK 安装时设置 PATH 和 CLASSPATH 环境变量有何作用?

FAQ1.13 为何在 JDK 安装路径下存在两个 JRE?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

在安装 JDK 后, 在 JDK 安装目录下会发现两个 JRE 目录。以 JDK 6.0 为例, 按照默认设置安装后, 一个目录是 “C:\Program Files\Java\jdk1.6.0_01\jre”, 另一个目录是 “C:\Program Files\Java\jre1.6.0_01”。

第 1 个 JRE 用于为 JDK 自带的开发工具提供运行环境, 在 JDK 中有很多用 Java 编写的开发工具 (例如 javac.exe、jar.exe 等), 这些工具的实现代码都放置在 “C:\Program Files\Java\jdk1.6.0_01\lib\tools.jar” 里, 这些代码运行的时候也需要一套 JRE。

第 2 个 JRE 用于为开发者编写的代码提供运行环境。

前面介绍的两个 JRE 都可以作为开发时 Java 程序的运行环境, 但是 JDK 自带工具只能使用第 1 个目录下的 JRE。

既然在计算机中至少有两个 JRE, 那么由谁来决定使用哪一个呢? 这个重任由 java.exe 负责。当使用者在命令行输入 “java xxx” 命令运行某字节码文件时, java.exe 的任务就是在计算机众多的 JRE 中选择合适的 JRE 来执行 xxx。java.exe 依据以下顺序来寻找并使用 JRE。

(1) 自己的目录下有没有 JRE 目录。

(2) 父目录下有没有 JRE 目录。

(3) 查询注册表 “HKEY_LOCAL_MACHINE\Software\JavaSoft\Java Runtime Environment\” 路径。

疑难点评

在 JDK 安装目录下存在两个 JRE, 使很多程序员感到困惑, 这虽然对编写和运行 Java 程序没有影响, 但是如果深入了解 Java 的运行过程, 就有必要追寻原因。

知识链接

FAQ1.08 什么是 JRE (或 J2RE)?

FAQ1.09 什么是 JDK?

FAQ1.12 如何安装 Java 基本开发环境 JDK?

FAQ1.14 JDK 安装时设置 PATH 和 CLASSPATH 环境变量有何作用?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在 JDK 安装之后,需要配置 PATH 和 CLASSPATH 两个环境变量对于习惯图形化接口操作的初学者而言,在命令符模式下编译执行 Java 程序是一件陌生的事,因此不了解设置 PATH 和 CLASSPATH 环境变量的作用。

(1) PATH 环境变量的作用

在安装 JDK 程序之后,在安装目录下的 bin 目录中会提供一些开发 Java 程序时必备的工具程序。对于 Java 的初学者,建议在命令符模式下使用这些工具程序编译运行 Java 程序。在 Windows 2000/XP 操作系统的“开始”菜单中选择“运行”菜单项,输入“cmd”命令来打开命令符模式。

在命令符模式下输入 javac 命令时,会提示错误信息,如图 1-8 所示。

出现上述错误的原因是操作系统在当前目录下找不到 javac.exe 工具程序,因此需要告诉操作系统应该到哪些目录下尝试寻找,设置 PATH 环境变量的目的就是为操作系统指定寻找工具程序的目录。

设置 PATH 变量后,需要重新打开命令符模式才能使设置生效,此时执行 javac 命令获得成功,结果如图 1-9 所示。

当用户在命令符模式输入 javac 命令时,操作系统会尝试在指定的 PATH 变量中寻找指定的工具程序,由于 PATH 变量中设置了 JDK 的 bin 目录的路径,因此操作系统就可以根据这个信息来找到 javac.exe 工具程序。

提示:在使用 javac 命令编译 Java 程序时,如果遇到“javac 不是内部或外部命令,也不是可运行的程序或批处理文件”错误提示,其原因就是 PATH 环境变量未设置或设置错误。

(2) CLASSPATH 环境变量的作用

Java 执行环境本身就是一个平台,用于运行已编译完成的 Java 程序(即 class 字节码文件)。

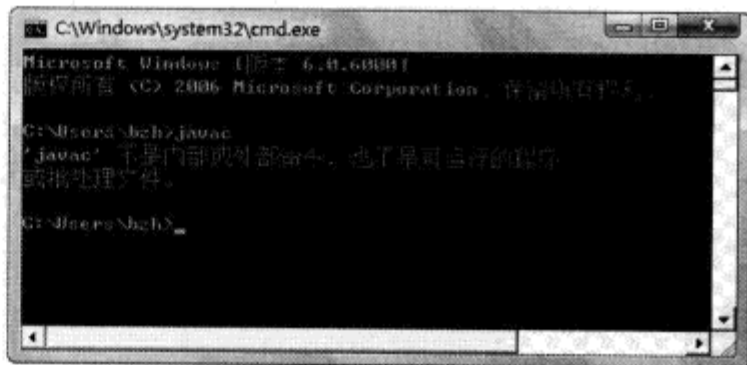


图 1-8 提示找不到 javac 工具程序

如果设置 PATH 变量是为了让操作系统找到指定的工具程序,那么设置 CLASSPATH 变量的目的就是让 Java 执行环境找到指定的 Java 程序对应的 class 文件以及程序中引用的其他 class 文件。



图 1-9 javac 命令效果图

JDK 在默认情况下会到当前工作目录下(变量值用“.”表示)以及 JDK 的 lib 目录下寻找所需的 class 文件,因此如果 Java 程序放在这两个目录中,即使不设置 CLASSPATH 变量执行环境也可以找得到。但是如果 Java 程序放在其他目录下,运行时则需要设置 CLASSPATH 变量。

总之,设置 CLASSPATH 的目的在于通知 Java 执行环境在哪些目录下可以找到所要执行的 Java 程序。

疑难点评

一般在安装 JDK 后,需要设置 PATH 和 CLASSPATH 环境变量,如果不设置会对以后使用造成一定的影响。在此不仅介绍了如何设置两个环境变量,还介绍了 PATH 和 CLASSPATH 环境变量的作用。

知识链接

FAQ1.09 什么是 JDK?

FAQ1.12 如何安装 Java 基本开发环境 JDK?

FAQ1.15 如何编译、运行 Java 应用程序?

难度系数: ★★★★★

问题频率: 90%

核心解答

作为 Java 的初学者,不建议立刻使用那些集成开发工具,例如 Eclipse、NetBean 和 Jbuilder 等。刚开始学习时,一般都是用文本编辑器编写一些简单的 Java 程序,然后从命令符操作界面进行编译和运行。

在命令符界面编译、运行 Java 程序的前提是需要安装 JDK 开发工具包,可参照 JDK 安装介绍进行安装和配置。下面介绍在命令符界面下如何使用 JDK 工具程序编译和运行 Java 程序。

(1) 编写 Java 程序

首先可以使用记事本或其他文本编辑器编写一个 Java 程序文件 Hello.java,代码如下:

```
public class Hello {  
  
    public static void main(String[] args) {  
        //在控制台打印输出 "Hello the world!" 信息  
        System.out.println("Hello the world!");  
    }  
}
```

(2) 编译

程序编写完毕后,将 Hello.java 文件放于系统中某一目录下,例如文件目录为“D:\test\Hello.java”。

在 Windows 2000/XP 操作系统的“开始”菜单中选择“运行”菜单项,输入“cmd”命令打开命令符模式。依次执行“D:”和“cd test”命令改变当前工作目录,设置当前目录为“D:\test”,如图 1-10 所示。

在图 1-10 中执行“javac Hello.java”命令,完成 Hello.java 的编译。此过程调用了 JDK 提供的 javac.exe 工具对 Hello.java 进行编译,编译成功界面如图 1-11 所示。

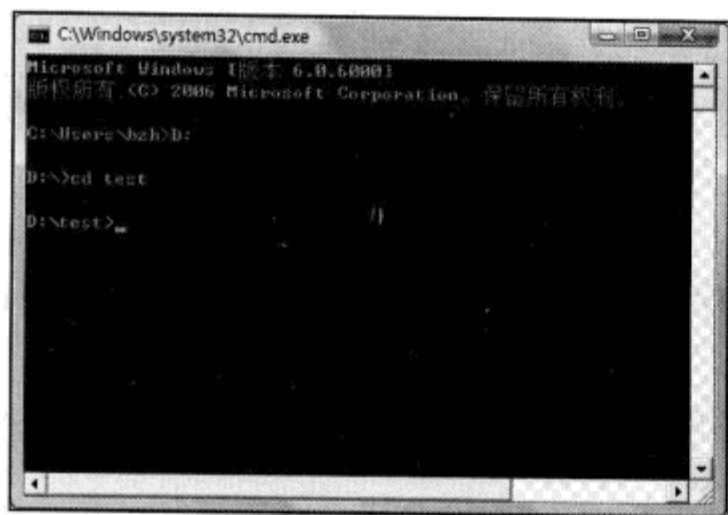


图 1-10 改变工作目录效果图

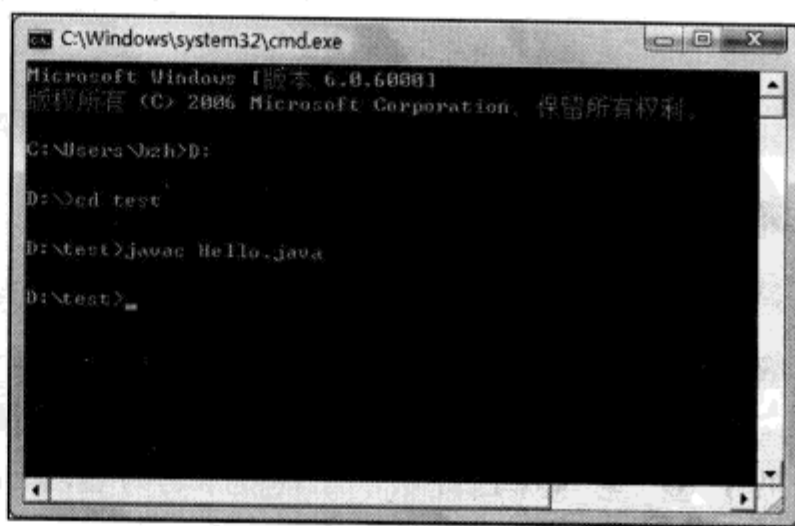


图 1-11 编译成功效果图

编译成功后,在“D:\test”目录下会生成一个字节码文件 Hello.class,该文件将被 Java 执行环境运行解释执行,后期过程与 Hello.java 无关。

如果在编写 Hello.java 时, 添加了包路径的声明, 例如 “package myjava.first”, 那么在编译时应采用 “javac -d . Hello.java” 编译, Hello.class 所存储的路径为 “D:\test\myjava\first\Hello.class”。如果一次编译多个 Java 文件, 可以使用 “javac -d . *.java” 命令。

(3) 运行

编译成功后将产生 class 文件, 此时可以调用解释器解释该文件产生输出结果。运行 Hello.class 文件的命令为 “java Hello”, 执行效果如图 1-12 所示。

注意: 如果 Hello.class 文件具有包路径的定义, 则该文件位置应该按照包路径存放, 并且执行时应使用 “java 包路径.Hello” 命令。

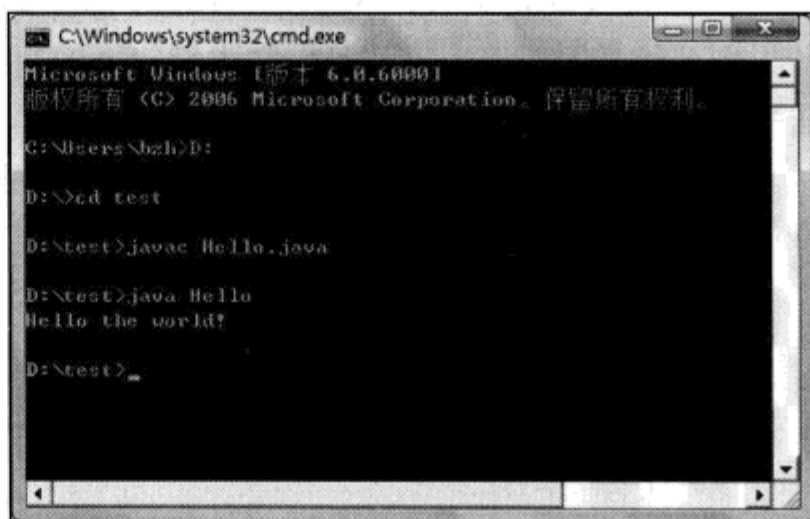


图 1-12 运行成功效果图

疑难点评

在开发 Java 程序时, 为了方便程序的编译和运行, 开发者一般都习惯在一些集成环境中进行。但对于初学者, 如果一上来就用集成开发环境将不利于学习, 因此建议初学者在学习之初使用 DOS 命令编译运行 Java 程序。

知识链接

FAQ1.16 如何将程序中的文档注释提取出来生成说明文档?

FAQ1.16 如何将程序中的文档注释提取出来生成说明文档?

📖 难度系数: ★★★

📖 问题频率: 75%

核心解答

JavaDoc 是 Sun 公司提供的一项技术, 它从程序源代码中抽取类、方法、成员等文档注释形成一个与源代码配套的 API 帮助文档。也就是说, 只要在编写程序时以一套特定的标签作注释, 在程序编写完成后, 通过 JavaDoc 实现工具就可以同时生成程序的开发文档了, JavaDoc 工具在 JDK 环境中已提供, 工具程序名为 javadoc.exe。

Java 中有以下 3 种注释语句。

- ❑ //用于单行注释。
- ❑ /*...*/用于多行注释, 从/*开始, 到*/结束, 不能嵌套。
- ❑ /**...*/用于支持 JavaDoc 工具而特有的文档注释语句。

JavaDoc 工具能从 Java 源文件中读取第 3 种注释,并能识别注释中用@标识的一些特殊变量(见表 1-1),制作成 HTML 格式的类说明文档。JavaDoc 不但能对一个 Java 源文件生成开发文档,而且能对目录和包生成交叉链接的 HTML 格式的类说明文档,十分方便。

JavaDoc 注释中出现的特殊标识都以@开头,常用标识及其作用如表 1-3 所示。

表 1-3 JavaDoc 注释中的特殊标识

关 键 字	作 用
@author	标识作者信息
@version	标识版本信息
@parameter	标识参数名及其意义
@since	标识参数名及其意义
@return	标识返回值
@throws	标识异常类及抛出条件
@deprecated	标识引起不推荐使用的警告
@see	标识交叉参考

使用以上特殊标识,编写的 Java 源程序示例如下:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/**
 * JavadocDemo.java, 一个显示 JavaDoc 注释的 Applet
 * <p>注意这只是 HelloApplet 的一个带注释的版本
 * @see java.applet.Applet
 * @see javax.swing.JApplet
 */

public class JavadocDemo extends Applet{

    /** init()是一个 Applet 方法, 由浏览器调用进行初始化
     * 只调用一次
     * @return 无
     */
    public void init(){
        //创建并添加一个按钮
        //其他什么也不做
        Button b;
        b=new Button("Hello");
        add(b);
        show();
    }

    /** paint() 是一个 AWT 组件方法, 在组件要绘制时调用, 只
     * 是在 Applet 的窗口中画带色的方框
     * 参数 g 是一个 java.awt.Graphics
```



```

    * 用在所有绘制方法中
    */

    public void paint(Graphics g){
        int w=getSize().width,h=getSize().height;
        g.setColor(Color.yellow);
        g.fillRect(0,0,w/2,h);
        g.setColor(Color.green);
        g.fillRect(w/2,0,w,h);
        g.setColor(Color.black);
        g.drawString("Welcome to Java",50,50);
    }

    /** show()用于使组件可见, 此方法在
     * JDK 1.1 中被归入不推荐使用
     * @since 1.0
     * deprecated 换用 setVisible(true)
     */

    public void show(){
        //由于覆盖了 show()方法, 此 Applet 不能显示
    }

    /** Applet 必须有一个公共的无参数构造方法
     * @throws java.lang.IllegalArgumentException
     */

    public JavadocDemo(){
        if(new java.util.Date().getDay()==0){
            throw new IllegalArgumentException("Never on a Sunday");
        }
    }
}

```

将上述 JavadocDemo.java 示例文件放置某一目录下（例如“D:\doc”），进入命令符操作界面，首先进入 D:\doc 目录，然后执行“javadoc JavadocDemo.java”命令，即可生成 HTML 格式的 Java 程序说明文档。

JavaDoc 工具的使用格式如下：

```
javadoc [选项] [软件包名称] [源文件] [@file]
```

上述格式中的@file 指的是包含文件，为了简化 JavaDoc 命令，可以将需要生成文档的软件包名和源文件名放到一个文本文件中。例如，为了简化以下命令：

```
javadoc -d mydoc test.Student test.Teacher
```

可以建立一个名称为 myfile.txt 的文件，内容如下：

```
mydoc test.Student
test.Teacher
```

然后执行如下命令即可。

```
javadoc -d mydoc @myfile.txt
```


使用 Javadoc 生成 API 说明文档时, 主要从以下几项内容中提取 Javadoc 注释信息。

- ☐ 包;
- ☐ 公有 (public) 类和接口;
- ☐ 公有 (public) 方法和受保护 (protected) 方法;
- ☐ 公有 (public) 属性和受保护 (protected) 属性。

注意: 关于 Javadoc 工具详细使用说明, 可以在命令符操作界面输入 “javadoc -help” 命令查阅。

疑难点评

使用 Javadoc 生成 Java 程序的说明文档, 与 JDK API 说明文档相似, 该文档有助于团队合作开发, 具有 Java 代码使用说明的作用。

知识链接

FAQ1.17 怎样制作鼠标双击就可以运行的 Jar 文件?

FAQ1.17 怎样制作鼠标双击就可以运行的 Jar 文件?

📖 难度系数: ★★★★★

📖 问题频率: 89%

核心解答

Java 文件编译后会生成 class 文件, 一个 Java 类对应一个 class 文件, 为了避免 class 文件过多, 方便用户使用, 开发者在为用户发布部署之前需要将 class 文件压缩。Jar 工具的主要用途是可以完成将一系列相关的程序文件压缩为一个文件, 该工具在 JDK 环境中已提供, 工具程序名为 jar.exe。

1. Jar 工具使用格式

打开命令符操作界面, 输入 “jar” 命令后, 将显示详细的使用格式, 具体如图 1-13 所示。

2. 应用示例

(1) 编写 U.java 文件, 源程序代码如下:

```
package demo.com.cn;

import javax.swing.JFrame;

class A {}
class B {}
class C {}
class D {}
```

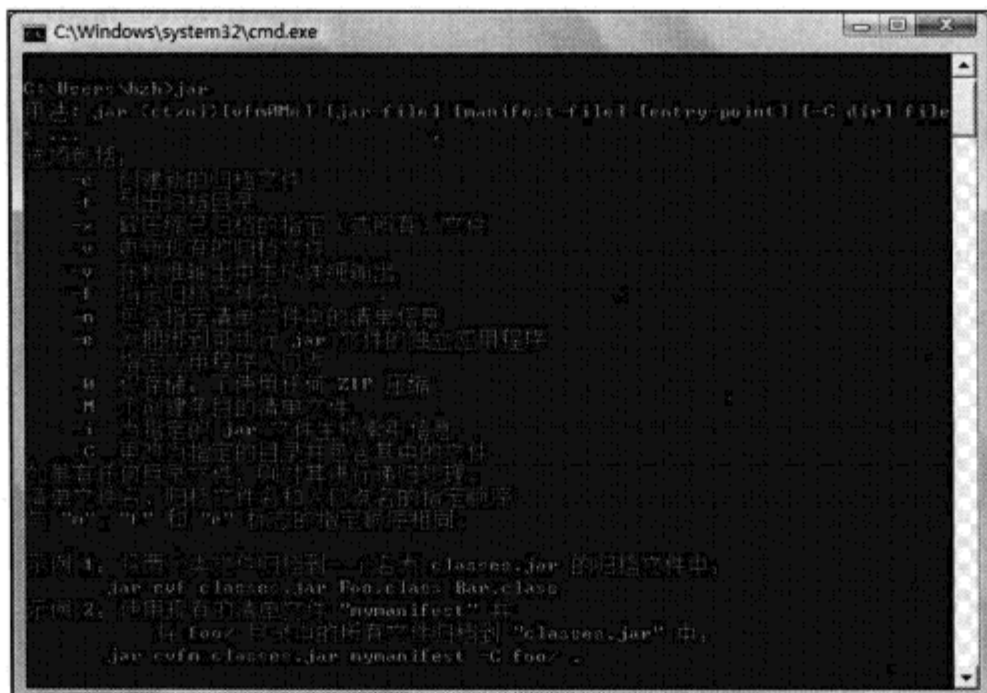


图 1-13 jar 命令格式介绍

```

public class U {
    public static void main(String args[]) {
        System.out.println("Hello java.jar");
        JFrame jf = new JFrame();
        jf.setSize(200, 200);
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setVisible(true);
    }
}

```

(2) 编译 Java 程序

打开命令符操作界面，进入 U.java 所在目录执行“javac -d *.java”命令，进行编译，效果如图 1-14 所示。

上述命令执行后，在 demo\com\cn 目录下将生成 A.class、B.class、C.class、D.class 和 U.class 5 个文件。

(3) 打包 class 文件

在当前目录下执行“jar - cvf hello.jar demo/com/cn/*.*”命令 (*.代表全部文件)，将生成的 class 文件打包，效果如图 1-15 所示。

上述命令执行后，在当前目录中会生成一个 hello.jar 的压缩文件。

(4) 创建可执行文件

经过以上几个步骤之后，即完成了 Java 程序文件的压缩，但还不能实现双击运行的功能。通过解压缩软件可以查看 hello.jar 文件的组成，内部除了包含几个 class 文件之外，在 META-INF 目录下还有一个 manifest.mf 文件，该文件是一个清单文件，可指定 jar 文件的结构信息，例如

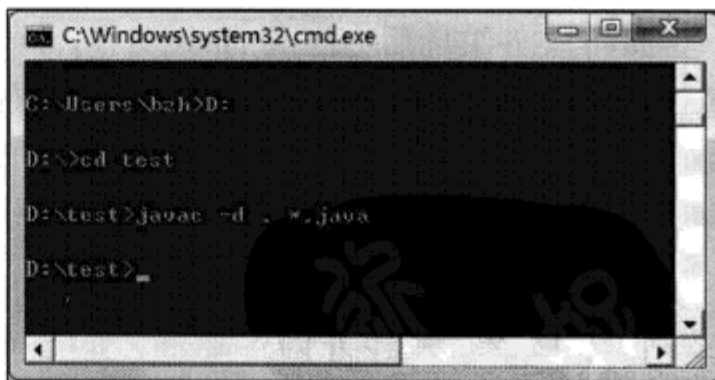


图 1-14 编译 U.java 文件

版本信息和主入口类等。

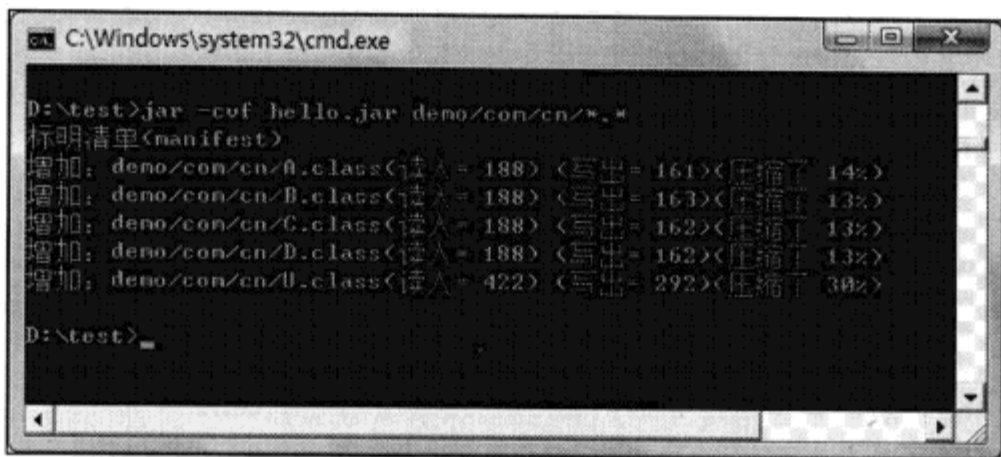


图 1-15 使用 jar 命令打包

在使用 jar 命令打包时, 可以使用 -m 参数, 其功能是在创建 jar 文件时, 定义 jar 文件的结构信息, 例如主入口类。

使用 -m 参数打包的具体过程如下所示。

- 打开记事本编辑一个文件 myfest (文件名随意, 但是不能有后缀), 在文件中输入以下信息。

```
Main-Class: demo.com.cn.U
```

注意: 冒号和 demo.com.cn.U 之间有空格 (一定要有), 另外文件必须以空行结束, 如果忽略的话将失败。

- 执行 “jar -cvfm hello.jar myfest *.class” 命令, 效果如图 1-16 所示。

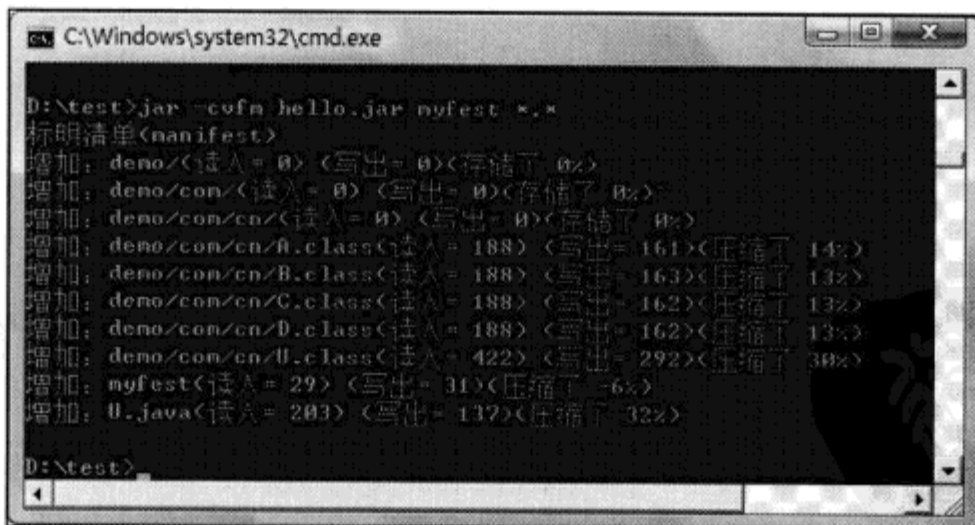


图 1-16 生成可执行 jar 文件

用鼠标双击生成的 hello.jar 文件, 即可自动运行程序。解压缩 hello.jar 文件后, 会发现 manifest.mf 文件的内容有所改变, 如图 1-17 所示。

- 如果需要也可以制作一个 bat 文件, 创建一个 hello.bat 文件 (文件名随意), 内容如图 1-18 所示。

用鼠标双击执行.bat 文件, 也可以运行 Java 程序。

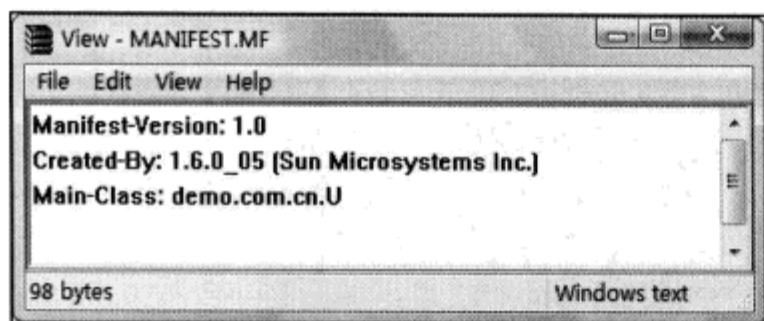


图 1-17 manifest.mf 文件内容



图 1-18 hello.bat 文件内容

疑难点评

为了方便程序的发布，JDK 提供了 Jar 工具，将 Java 代码进行打包。Java 程序打包发布的过程与编译运行过程一样，都属于最基本的操作，需要读者达到熟练使用的程度。

知识链接

FAQ1.15 如何编译、运行 Java 应用程序？

FAQ1.16 如何将程序中的文档注释提取出来生成说明文档？

FAQ1.18 怎样给 main(String[] args)方法的 args 指定参数值？

📖 难度系数：★★★

📖 问题频率：70%

核心解答

Java 应用程序的主入口方法 `main(String[] args)`，表示该方法需要接收一个字符串数组类型的参数值，该参数值如果不指定，`args` 接收的是 `null` 值，当然也可以在 Java 命令执行程序时指定。

例如 Test.java 文件的代码如下：

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println(args[0]);  
        System.out.println(args[1]);  
    }  
}
```

在命令符操作界面，使用“`javac Test.java`”编译后，可使用如下命令执行 Test 程序并为其 `main()` 方法指定参数值。

```
java Test tom rose
```

上述命令为 Test 类 `main()` 方法的字符串数组参数指定了“tom”和“rose”两个元素值，也

可以指定若干个, 参数值之间需要用空格隔开。上述命令的执行结果如图 1-19 所示。

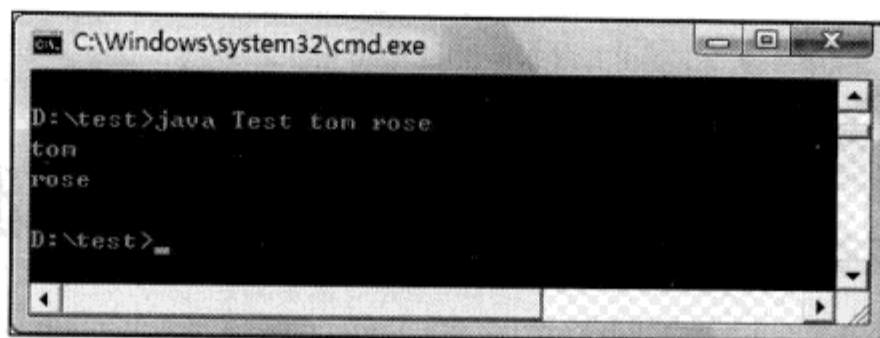


图 1-19 Test 类运行效果图

疑难点评

main()方法是应用程序的主入口方法, 该方法在启动时允许执行环境传入启动参数。为main()方法指定启动参数在实际中使用得不多, 但需要读者了解。

知识链接

FAQ1.15 如何编译、运行 Java 应用程序?



第2章

Java 编程基础

本章重点介绍一些与 Java 编程基础知识相关的疑难问题，内容主要涉及标识符、关键字、数据类型、运算符、流程控制语句以及数组等几个方面。

FAQ2.01 Java 中的标识符如何命名？可以用中文吗？

📖 难度系数：★★

📖 问题频率：90%

核心解答

很多初学者在编写程序时总是习惯任意指定变量名、类名和方法名，结果常常由于不正确的命名，产生一些低级错误。

标识符是用于标识某一事物的符号，Java 程序中的类名、方法名和变量名都属于标识符。

Java 标识符在定义时需要遵循以下几点规则。

- ❑ 可以由中英文字母、下划线“_”、“\$”符和数字字符构成，但不能以数字开头。
- ❑ 注意 Java 是大小写敏感的，标识符也不例外。
- ❑ 注意避开 Java 语言中默认的关键字。
- ❑ 标识符没有长度限制。

通过上述规则，下面给出一些合法和非法的典型示例，具体如下。

合法标识符：TeSt、A1、_boolean、A\$C、姓名等。

非法标识符：Hello World、1A、boolean、A@Ca#、String 等。

在定义标识符时除了必须遵守上述规则以外，通常还有一些编码习惯，具体如下。

- ❑ 类名：每个单词首字母大写，其他小写。
- ❑ 接口：每个单词首字母大写，其他小写。
- ❑ 方法：以小写字母开头，如果方法名由多个单词组成，则从第 2 个单词开始首字母大写。
- ❑ 变量：以小写字母开头，如果变量名由多个单词组成，则从第 2 个单词开始首字母

大写。

- 常量：常量名全部大写，各单词间以“_”连接。
- 包名：全部小写。

疑难点评

类名、变量名和包名等都属于标识符范畴，在编写程序时，一定要遵循特定的命名规范，这样可以使程序更加规范，也可以减少因错误的命名而引起的程序错误。

知识链接

FAQ2.02 Java 中有哪些关键字?

FAQ2.02 Java 中有哪些关键字?

📖 难度系数：★★★

📖 问题频率：95%

核心解答

关键字是 Java 程序中具有特殊含义的字符，不能在程序中当作标识符使用。

Java 语言在其发展的过程中关键字越来越多，甚至有些关键字的出现会直接影响到原有的 Java 程序。例如，当关键字 `enum` 出现后，以前如果写过的某个程序源文件用 `enum` 作为变量将会发生编译错误。甚至有一些知名的开源工具包由于包名为 `enum`，造成与新版本冲突。

Java 中现有的常用关键字并不多，具体如表 2-1 所示。

表 2-1 Java 常用关键字

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

注意：`goto`、`const` 关键字都被当作保留字符保留了下来，为以后使用做准备。

疑难点评

Java 关键字是编程时随时都要使用的，具有特定的含义，读者需要熟悉表 2-1 中提到的每一个关键字及其作用。

知识链接

FAQ2.01 Java 中的标识符如何命名？可以用中文吗？

FAQ2.03 用 public、protected 和 private 修饰方法有什么区别？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

在 Java 中，可以在类、类的属性以及类的方法前面加上一个修饰符，来对其进行访问权限的控制。public、protected 和 private 修饰符是用于定义成员访问权限的，另外还有一种是“default”情况，也就是在成员前不加任何权限修饰符。

上述 4 种定义方式的权限差别如表 2-2 所示。

表 2-2 public、protected、private 和 default 的区别

修 饰 符	类 内 部	同 一 包 中	子 类 中	其 他
public	允许	允许	允许	允许
protected	允许	允许	允许	不允许
default	允许	允许	不允许	不允许
private	允许	不允许	不允许	不允许

如表 2-2 所示，例如用 protected 修饰的成员（变量或方法），在类内部可以调用，同一个包中的其他类也可以调用，子类中也可以调用，其他地方则不可以调用，即在其他包下，并且不是子类的类里，不能使用 protected 修饰的成员。

根据表 2-2 可以总结出访问控制的限制程度由高到低如图 2-1 所示。

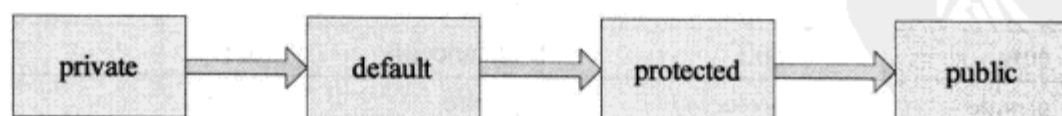


图 2-1 访问控制的限制范围

注意：default 不是修饰符，它表示一种不加任何修饰符的状态。

疑难点评

在定义变量和方法时，都要使用 public、protected 和 private 等修饰符，这些修饰符的使用范围不同，读者需要注意区分各修饰符的使用范围和使用环境。

知识链接

FAQ2.02 Java 中有哪些关键字？

FAQ2.04 this 关键字有什么含义？在哪些情况下应用？

📖 难度系数：★★★★

📖 问题频率：85%

核心解答

Java 中为了解决变量的命名冲突和不确定性问题,引入了关键字 `this`。`this` 代表当前类的一个实例,它经常出现在方法和构造方法中,具体使用情况有以下 3 种。

- ☐ 返回调用当前方法的对象的引用。
- ☐ 在构造方法中调用当前类中的其他构造方法。
- ☐ 当方法参数名和成员变量名相同时,用于区分参数名和成员变量名。

1. 返回调用当前方法的对象的引用

示例代码如下:

```
public class Leaf {  
    private int i = 0;  
    public Leaf increment() {  
        i++;  
        return this;  
    }  
    public void print() {  
        System.out.println("i = " + i);  
    }  
    public static void main(String[] args) {  
        Leaf x = new Leaf();  
        x.increment().increment().increment().print();  
        Leaf y = new Leaf();  
        y.increment().increment().print();  
    }  
}
```

在上述代码中, `this` 表示 `Leaf` 类的实例,当执行“`x.increment()`”时, `this` 表示实例 `x` 的引用,当执行“`y.increment()`”时, `this` 表示实例 `y` 的引用。

2. 在构造方法中调用当前类中的其他构造方法

示例代码如下:

```
public class Person {  
    private String name;  
    private int age;  
    private String sex;
```

```
public Person(){
    sex = "male";
}

public Person(String _name){
    this();
    name = _name;
}

public Person(String _name,int _age){
    this(_name);
    age = _age;
}
}
```

在上述代码中，定义了3个 Person 类的构造方法，分别为无参的、有一个参数的和有两个参数的。为了实现代码重复利用，使用了 this() 和 this(_name) 语句来调用 Person() 和 Person(String _name) 构造方法。

注意：在使用 this 调用其他构造方法时，必须放在构造方法的开始处，否则编译不通过。

3. 当方法参数名和成员变量名相同时，用于区分参数名和成员变量名

示例代码如下：

```
public class Person{
    private String name;
    private int age;
    public void setName(String name){
        this.name = name;
    }
    public void setAge(int age){
        this.age = age;
    }
}
```

在上述代码中，setter 方法的参数名和成员变量名相同，如果不使用 this 关键字，“name=name”语义将变得很模糊。“this.name”代表成员变量名，而“name”代表参数名。

疑难点评

很多 Java 初学者对 this 关键字的含义和使用都很迷惑，通过上述介绍可以使读者十分清晰的理解 this 所代表的含义及其使用方式。

知识链接

FAQ2.02 Java 中有哪些关键字？

FAQ2.05 super 关键字有什么含义？在哪些情况下应用？

📖 难度系数：★★★★

📖 问题频率：85%

核心解答

在 Java 中，this 代指当前类的实例，使用 this 可以调用当前类中的方法、属性和构造方法。而 super 代表父类的实例，在子类中，使用 super 可以调用其父类的方法、属性和构造方法。

super 的具体使用情况有以下 2 种：

- ☐ 调用父类中的构造方法；
- ☐ 调用父类中的方法和属性。

1. 调用父类中的构造方法。

示例代码如下：

```
public class Student extends Person{
    private String stno;
    private int grade;

    public Student(){
        super();
    }

    public Student(String name,String stno,int grade){
        super(name);
        this.stno = stno;
        this.grade = grade;
    }
}
```

在上述代码中，Student 类继承自 Person，在当前类中通过使用“super()”和“super(name)”语句调用 Person 类中无参和有一个参数的构造方法。

注意：如果父类中没有对应的构造方法，将会产生编译错误。

2. 调用父类中的方法和属性

示例代码如下：

```
public class Student extends Person{
    private String stno;
    private int grade;

    public void show(){
        super.show();
        System.out.println("stno="+stno);
        System.out.println("grade="+grade);
    }
}
```

在上述代码中，通过“super.show()”语句调用 Person 中的 show() 方法。属性调用与方法调用类似，因此不再举例。

疑难点评

很多 Java 初学者对 `super` 关键字的含义和使用都很迷惑,通过上述介绍可以使读者十分清晰的理解 `super` 所代表的含义及其使用方法。

知识链接

FAQ2.02 Java 中有哪些关键字?

FAQ2.06 `static` 关键字有什么含义? 具体如何应用? 能修饰构造方法吗?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

在 Java 中, `static` 关键字可以修饰方法、属性、自由块和内部类,使用 `static` 修饰这些成员时,可以理解成这些成员与类相关,通过“类名.成员”的形式调用;没有 `static` 修饰可以理解成这些成员与对象相关,需要通过“对象名.成员”的形式调用。

注意: `static` 关键字不能用于修饰构造方法。

1. `static` 修饰方法

Java 应用程序的主入口方法 `main` 就是适用 `static` 修饰方法的典型示例, `main` 方法的定义如下:

```
public class Hello{
    public static void main(String[] args){
        //TODO 添加代码
    }
}
```

`main()` 是 Java 应用程序的主入口方法,在该方法前添加 `static` 修饰表示该方法是与类相关的,Java 解析器运行时将会寻找该方法。之所以将 `main()` 方法定义为类相关,原因是在解析器调用该方法时,还没有来得及创建当前类对象,因此不能定义成与对象相关的方法。

注意: 在 `static` 修饰的方法中,不能调用没有 `static` 修饰的方法和属性,也不能使用 `this` 和 `super` 关键字。

2. `static` 修饰属性

当 `static` 修饰属性时,除属性调用方式与修饰方法类似外,静态属性还具有一个特性,那就是该属性被多个当前类对象共享,一个对象修改静态属性值后,会影响其他对象。

示例代码如下:

```
public class Test{
    public static int count = 0;
```



```
public Test(){
    count++;
}

public static void main(String[] args){
    Test t1 = new Test();
    System.out.println(t1.count);
    Test t2 = new Test();
    System.out.println(t2.count);
}
```

在上述代码中，由于 `count` 是一个静态变量，因此 `t1` 和 `t2` 对象共享 `count`。当创建 `t1` 对象时，构造方法将 `count` 加 1，当再次创建 `t2` 对象时，`count` 在此基础上继续加 1，程序的输出结果如下：

```
1
2
```

如果将 `count` 变量的修饰符 `static` 去掉，该变量将变成与对象相关，每个对象有一个 `count` 变量，各个对象之间互不影响，此时输出结果如下：

```
1
1
```

3. static 修饰自由块

自由块是 Java 类中使用 `{}` 括起来的代码段，自由块中的代码在构造方法之前执行，因此可以将一部分初始化代码放在自由块中。

当使用 `static` 修饰自由块时，自由块将变成静态自由块，通常用于初始化静态变量。静态自由块与类相关，因此只要类被加载，即使没有创建对象，也将被执行。此外静态自由块无论创建几个对象，仅执行一次。

示例代码如下：

```
public class Test{
    private static int i;

    static{
        System.out.println("自由块被执行");
        i++;
    }

    public void show(){
        System.out.println(i);
    }

    public static void main(String[] args){
        Test t1 = new Test();
        t1.show();
        Test t2 = new Test();
        t2.show();
    }
}
```

```
}
```

上述代码执行结果如下:

自由块被执行

```
1
```

```
1
```

从上述结果中可以看出,即使创建了 t1 和 t2 两个对象,静态自由块也还是执行了一次。

疑难点评

static 关键字可用于修饰属性、方法和自由块,这些元素有 static 修饰和无 static 修饰差别很大,将直接影响程序的结果。static 使用比较频繁,读者应重点关注。

知识链接

FAQ2.02 Java 中有哪些关键字?

FAQ3.15 static 修饰的方法能否在子类中重写?

FAQ2.07 final 关键字有什么含义? 具体如何应用?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

在 Java 中,final 关键字可以在类、成员变量和方法前面修饰,具体使用情况及其含义如下所示。

1. final 修饰类

final 修饰类时,表示该类不能再被其他类继承,例如 String 和 Math。如果不希望自己定义类被继承,可以将类使用 final 修饰。示例代码如下:

```
public final class Person{  
}
```

提示: String 类和 Math 类由于有 final 关键字修饰,因此不能当做父类,被继承使用。

2. final 修饰成员变量

final 修饰成员变量时,表示该变量是一个常量。在需要使用常量的场合,可以使用 final 关键字定义。示例代码如下:

```
final int MALE = 1;  
final int FEMALE = 0;
```

如果 final 修饰的是一个简单类型的变量,那么变量值一旦初始化后,将不能修改。如果 final 修饰的是一个引用类型的变量,那么该变量的引用不可以改变,但可以通过该引用修改引用对象的属性值。

3. final 修饰方法

final 修饰方法时，表示该方法不可以在子类中重写（覆盖）。final 关键字修饰方法的示例代码如下：

```
public final void m1(){
    //TODO 添加代码
}
```

疑难点评

final 关键字常用于修饰类、变量和方法，读者应注意区分 final 在修饰不同元素时所代表的含义。

知识链接

FAQ2.02 Java 中有哪些关键字？

FAQ2.08 instanceof 关键字有什么含义？如何应用？

📖 难度系数：★★★

📖 问题频率：75%

核心解答

instanceof 关键字属于 Java 中的一个二元操作符，和 “==”、“>” 等操作符用法相似，其作用是判断某对象是否为某个类或接口类型。由于 Java 语言的多态性使得可以用一个子类的实例赋值给一个父类的变量，但是在一些情况下需要判断变量的原有类型，此时可以使用 instanceof 操作符实现。

当 instanceof 左操作数是右操作数指定的类型或者子类类型时返回 true，反之则返回 false。instanceof 操作符的具体使用示例如下：

```
/**
 *定义 Animal 类
 */
class Animal {

}

/**
 *定义 Animal 子类 Dog
 */
class Dog extends Animal {
    public void f1() {
        System.out.println("Woof Woof");
    }
}
```



```
/**
 *定义 Animal 子类 Cat
 */
class Cat extends Animal {
    public void f2() {
        System.out.println("miao miao");
    }
}

/**
 *定义测试类 Test
 */
public class Test {
    public static void main(String[] a) {
        Dog d = new Dog();
        show(d);
        Cat c = new Cat();
        show(c);
    }

    //利用多态特性定义 show 方法的参数, 提高参数传递的灵活性
    public static void show(Animal a) {
        if (a instanceof Dog) {
            Dog dog = (Dog) a;
            dog.f1();
        } else if (a instanceof Cat) {
            Cat cat = (Cat) a;
            cat.f2();
        }
    }
}
```

在上述代码中, 测试类 Test 的 show()方法首先利用 instanceof 操作符判断外界传入的对象类型, 然后再根据判定结果做强制类型转换, 调用转型后的对象方法执行不同处理。

疑难点评

instanceof 关键字主要作用是判断一个对象是否为指定类型。在变量使用多态时, 可以使用 instanceof 关键字进行变量的类型判断, 然后执行不同的操作。

知识链接

FAQ2.02 Java 中有哪些关键字?

FAQ2.09 Java 中有哪些数据类型?

📖 难度系数: ★★★

📖 问题频率: 95%

核心解答

Java 语言的数据类型分为两种，一种是简单数据类型，另一种是引用数据类型。简单数据类型都有着固定的长度，其具体取值范围如表 2-3 所示。

表 2-3 Java 简单数据类型的取值范围

数据类型	有效范围 (bit)
boolean	1
byte	8
char	16
short	16
int	32
long	64
float	32
double	64

注意：char 类型为两个字节，采用 Unicode 编码。因此，无论是简单的字母还是汉字，在 Java 中都是占用两个字节。

除了 8 种简单数据类型之外的所有数据类型都被称为引用数据类型，引用数据类型的大小统一为 4 字节，记录的是其引用对象的地址。具体分类如图 2-2 所示。

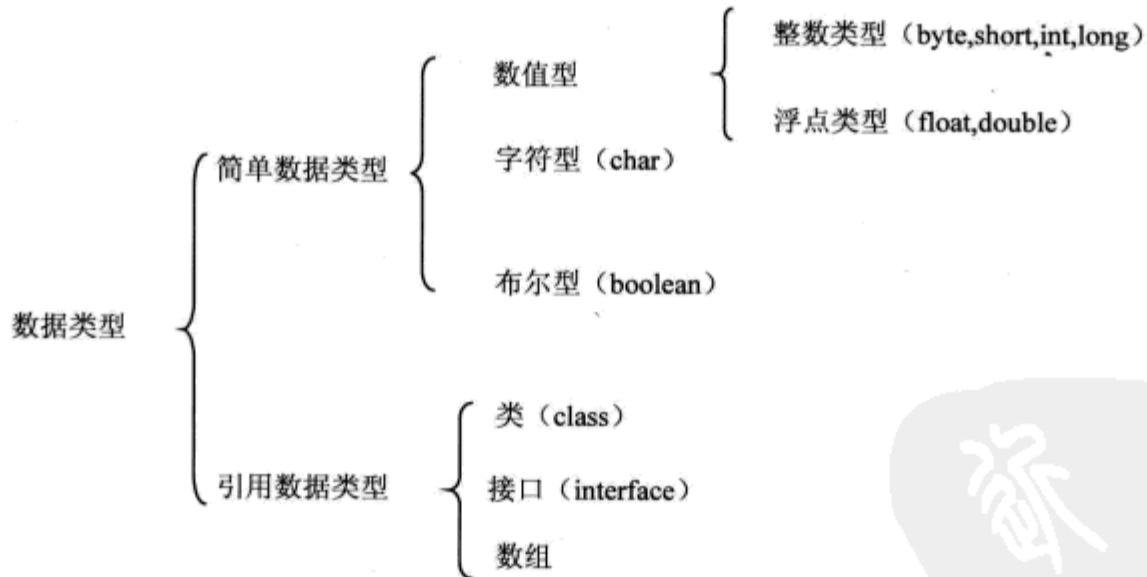


图 2-2 数据类型结构图

疑难点评

Java 属于强类型语言，在定义变量时必须严格指定变量类型。Java 类型主要分为简单类型和引用类型，简单类型变量直接存储变量值，而引用类型变量存储的却是地址，这使得变量之间相互赋值时有很大的区别，具体内容读者可参阅“变量之间传值时可分为值传递和引用传递，那么它们有何区别？”。

由于在 Java 编程时,简单类型变量和引用类型变量的使用区别较大,因此读者需要很清楚地分辨出哪些是简单类型,哪些是引用类型。除了上述 8 种简单类型之外,其他的类型都属于引用类型,例如 `int[]` 和 `String` 等都应属于引用类型,具有引用类型的存储特征。

知识链接

FAQ2.10 如何解决 `double` 和 `float` 精度不准的问题?

FAQ2.11 `int` 和 `Integer` 都可以作为整数类型,那么它们有什么区别?

FAQ2.12 `float f=3.4` 语句是否正确?

FAQ2.10 如何解决 `double` 和 `float` 精度不准的问题?

📖 难度系数: ★★★

📖 问题频率: 85%

核心解答

1. `double` 或 `float` 的精度问题

在进行数字运算时,如果有 `double` 或 `float` 类型的浮点数参与计算,偶尔会出现计算结果不准确的情况。示例代码如下:

```
public class Test{
    public static void main(String args[]){
        System.out.println(0.05+0.01);
        System.out.println(1.0-0.42);
        System.out.println(4.015*100);
        System.out.println(123.3/100);
    }
};
```

上述代码的执行结果如下:

```
0.060000000000000005
0.5800000000000001
401.49999999999994
1.2329999999999999
```

通过上面的例子可以看出, `double` 和 `float` 本身确实存在某种缺陷,不能用于精确运算,不仅是 Java 语言,其他很多语言也都存在相同的问题。在大多数情况下,使用 `double` 或 `float` 计算的结果是准确的,发生上述问题的频率很小,但是在一些精度要求很高的系统中,这种问题是非常严重的,因为如果你有 9.999999999999 元,计算机是不会认为你可以购买 10 元的商品。

2. 解决方法

在《Effective Java》一书中提到了一个原则,那就是 `float` 和 `double` 只能用来做科学计算或者是工程计算,但在商业计算中我们要用 `java.math.BigDecimal`。通过使用 `BigDecimal` 类可以解决上述问题,示例代码如下:

```
public static void main(String[] args) {  
    System.out.println(ArithUtil.add(0.05, 0.01));  
    System.out.println(ArithUtil.sub(1.0, 0.42));  
    System.out.println(ArithUtil.mul(4.015, 100));  
    System.out.println(ArithUtil.div(123.3, 100));  
}
```

ArithUtil 是自定义的一个工具类, 封装了加、减、乘、除操作。实现代码如下:

```
import java.math.BigDecimal;  
  
public class ArithUtil {  
  
    // 默认除法运算精度  
    private static final int DEF_DIV_SCALE = 10;  
    // 这个类不能实例化  
    private ArithUtil() {  
    }  
  
    /**  
     * 提供精确的加法运算  
     * @param v1 被加数  
     * @param v2 加数  
     * @return 两个参数的和  
     */  
    public static double add(double v1, double v2) {  
        BigDecimal b1 = new BigDecimal(Double.toString(v1));  
        BigDecimal b2 = new BigDecimal(Double.toString(v2));  
        return b1.add(b2).doubleValue();  
    }  
  
    /**  
     * 提供精确的减法运算  
     * @param v1 被减数  
     * @param v2 减数  
     * @return 两个参数的差  
     */  
    public static double sub(double v1, double v2) {  
        BigDecimal b1 = new BigDecimal(Double.toString(v1));  
        BigDecimal b2 = new BigDecimal(Double.toString(v2));  
        return b1.subtract(b2).doubleValue();  
    }  
  
    /**  
     * 提供精确的乘法运算  
     * @param v1 被乘数  
     * @param v2 乘数  
     * @return 两个参数的积  
     */  
    public static double mul(double v1, double v2) {  
        BigDecimal b1 = new BigDecimal(Double.toString(v1));  
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
```



```

        return b1.multiply(b2).doubleValue();
    }

    /**
     * 提供（相对）精确的除法运算，当发生除不尽的情况时，精确到小数点以后 10 位，
     * 以后的数字四舍五入
     * @param v1 被除数
     * @param v2 除数
     * @return 两个参数的商
     */
    public static double div(double v1, double v2) {
        return div(v1, v2, DEF_DIV_SCALE);
    }

    /**
     * 提供（相对）精确的除法运算
     * 当发生除不尽的情况时，由 scale 参数指定精度，以后的数字四舍五入
     * @param v1 被除数
     * @param v2 除数
     * @param scale 表示需要精确到小数点以后几位。
     * @return 两个参数的商
     */
    public static double div(double v1, double v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException("The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).doubleValue();
    }
}

```

虽然 `BigDecimal` 类的构造方法很多，但是我们只使用了带 `String` 类型参数的构造方法，具体原因可以从 JDK 帮助文档上查阅到，帮助文档的原文如下。

(1) `BigDecimal(double val)` 构造方法的结果有一定的不可预知性。有人可能认为在 Java 中写入 `new BigDecimal(0.1)` 所创建的 `BigDecimal` 正好等于 0.1（非标度值 1，其标度为 1），但是它实际上等于 0.1000000000000000055511151231257827021181583404541015625。这是因为 0.1 无法准确地表示为 `double`（或者说对于该情况，不能表示为任何有限长度的二进制小数）。这样，传入到构造方法的值不会正好等于 0.1（虽然表面上等于该值）。

(2) `String` 类型参数的构造方法是完全可预知的：写入 `new BigDecimal("0.1")` 将创建一个 `BigDecimal`，它正好等于预期的 0.1。因此，比较而言，通常建议优先使用 `String` 类型参数的构造方法。

(3) 当 `double` 必须用作 `BigDecimal` 的源时，请注意，此构造方法提供了一个准确转换，它不提供与以下操作相同的结果。先使用 `Double.toString(double)` 方法，然后使用 `BigDecimal(String)` 构造方法，将 `double` 转换为 `String`。要获取该结果，请使用 `static valueOf(double)` 方法。

疑难点评

通过 JDK 帮助文档的原文解释，我们可以知道浮点数是无法在计算机中准确表示的，例如 0.1 在计算机中只是表示成了一个近似值，因此，对浮点数运算时结果具有不可预知性。如果两个浮点操作数的误差能相互抵消，计算结果就能保持正确，例如 $0.2+0.5$ ，但如果不能相互抵消就会出现上面示例的情况。

FAQ2.11 int 和 Integer 都可以作为整数类型，那么它们有什么区别？

📖 难度系数：★★★

📖 问题频率：85%

核心解答

int 是一个基本数据类型，为了编码的方便和简化，Java 保留了 int、short、long、byte、float、double、char 和 boolean 等 8 个基本数据类型。基本类型不具备面向对象的基本特征，没有属性和方法。

为了面向对象操作的一致性，Java 为每一种基本类型都提供了相应的封装类型，具体如表 2-4 所示。

表 2-4 基本类型和封装类型对应表

基本类型	封装类型
byte	Byte
short	Short
int	Integer
long	Long
boolean	Boolean
float	Float
double	Double
char	Character

封装类属于引用类型，具有方法和属性，利用这些属性和方法可以很方便地实现一些基本类型难以完成的功能。例如将数值转换成字符串、将字符串转换为数值等。

封装类都提供了 `valueOf(String s)` 方法，可以完成字符串到封装类之间的转化；`toString()` 方法可以将封装类转化为字符串；同时也提供了将封装类转化为基本类型的方法，例如 `intValue()`、`shortValue()` 和 `longValue()` 等；如果需要将基本类型转化为封装类可以使用封装类的构造方法。

注意：从 JDK 5.0 版本开始引入了自动装箱和拆箱特性，主要目的是方便封装类和基本类型之间的相互转化。该特性允许程序员在基本类型和封装类型之间直接相互赋值，不需要经过

任何编码转换。

疑难点评

Java 为每一种简单类型都提供了一个封装类，封装类属于引用类型，具有引用类型的特点。

知识链接

FAQ2.09 Java 中有哪些数据类型？

FAQ2.12 float f=3.4 语句是否正确？

📖 难度系数：★★

📖 问题频率：78%

核心解答

1. 类型转换

Java 中的类型可分为基本类型和引用类型，不同类型之间的变量赋值时，首先需要先进性类型转换，然后再进行赋值。

Java 类型转换可分为自动转换和强制转换两种。自动转换过程不需要程序员使用代码进行干预，而强制转换则需要程序员显示指定要转换的类型。

基本类型之间相互转换，实现自动转换需要满足以下基本条件。

- ❑ 转换双方的类型必须兼容，例如 int 和 long 兼容，int 和 boolean 不兼容。
- ❑ 目标类型比源类型范围要大，例如 long 类型分配内存大小 8 字节，int 类型是 4 字节，因此 long 类型的范围比 int 类型大。

简单类型除了 boolean 类型以外，其他的数据类型之间可以实现自动转换，但需要遵循一定规则，具体如图 2-3 所示。

如图 2-3 所示，按照箭头所示的方向可以实现类型自动转换，例如 byte 转换为 short，short 转换为 int 等。如果按照箭头反

向转换时需要强制转换，例如 int 转换为 short，double 转换为 int 等。

2. 数值常量的默认类型

Java 中整型常量默认为 int 类型，为其分配 4 字节大小的空间。如果需要声明 long 类型的常量，可以在后面加“l”或“L”，例如“3L”。示例代码如下：

```
int i = 3;  
long l = 3L;
```

Java 中浮点型常量默认为 double 型，如要声明一个常量为 float 型，则需在数字后面加“f”或“F”。示例代码如下：

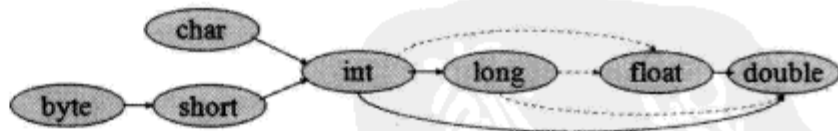


图 2-3 类型转换规则

```
double d = 3.14;  
float f = 3.14f;
```

3. float f = 3.4 语句错误解析

“float f = 3.4”语句的写法是不正确的,因为 3.4 默认为 double 类型,从图 2-2 中可知, double 类型转换为 float 类型是不能自动转换的,需要强制转换,所有该语句正确的写法如下:

```
float f = 3.14f;  
或者  
float f = (float)3.14;
```

疑难点评

整数类型有 byte、short、int 和 long,浮点类型有 float 和 double。数值常量在内存分配时会有其默认的大小空间,例如整数 10 会按 int 类型分配;浮点数 3.15 会按 double 类型分配。如果需要分配指定类型大小的空间,需要在书写时指定,例如 3L 和 3.15f 等。

知识链接

FAQ2.09 Java 中有哪些数据类型?

FAQ2.10 如何解决 double 和 float 精度不准的问题?

FAQ2.13 成员变量和局部变量有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在 Java 中,变量可分为局部变量和成员变量(即属性)两种。

1. 局部变量

局部变量一般是指在方法体内部定义的变量,其作用域是在方法块内部有效。局部变量在使用时,必须先初始化然后才能使用,否则程序将不能顺利通过编译。错误应用的代码如下:

```
public class Test{  
    public void f1(){  
        //变量 i 未初始化,编译将产生错误  
        int i;  
        //int i = 0; //正确使用方式  
        System.out.println(i);  
    }  
}
```

2. 成员变量

成员变量是指在类中定义的变量,也就是指属性,其作用域是在整个类中有效。成员变量在定义时可以不指定初始值,系统可以按默认原则初始化,初始化的具体原则如表 2-5 所示。

表 2-5 成员变量默认初始值

成员变量类型	默认初始值
Byte	0
Short	0
Int	0
Long	0L
Boolean	false
Float	0.0f
Double	0.0d
Char	'\u0000'
其他类型（即引用类型）	null

从上述表可以看出，除了 8 中基本类型之外，其他类型的成员变量都被初始化为 `null` 值。

注意：被 `final` 修饰并且没有被 `static` 修饰的成员变量必须显式赋初始值。

3. 成员变量和局部变量的区别

成员变量和局部变量的区别主要有以下几个方面。

- ☐ `public`, `protect`, `private`, `static` 等修饰符可用于修饰成员变量，但不能修饰局部变量。两者都可以使用 `final` 修饰。
- ☐ 成员变量存储在堆内存中，局部变量存储在栈内存中。
- ☐ 作用域不同，成员变量在整个类中有效，局部变量在方法体中有效，在方法体之外不可见。
- ☐ 成员变量可以默认初始化，局部变量必须显式初始化。

疑难点评

变量根据定义的位置不同可以分为成员变量和局部变量，在使用时需要注意两者差别，例如使用范围、初始化规则以及变量开辟内存的空间大小等。

知识链接

FAQ2.14 变量之间传值时可分为值传递和引用传递，那么它们有何区别？

FAQ2.14 变量之间传值时可分为值传递和引用传递，那么它们有何区别？

📖 难度系数：★★★★

📖 问题频率：95%

核心解答

Java 根据变量的类型不同可分为简单类型变量和引用类型变量，这两种类型的变量主要有

以下几点不同。

1. 存储机制

简单类型变量是直接在栈内存中开辟存储空间存储变量值。

引用类型变量是由引用空间和存储空间两部分构成,引用空间在栈内存中,存储空间在堆内存中,存储空间负责存储变量值,引用空间负责存放存储空间的首地址。引用变量中存放的是地址值,通过地址值可以定义存储位置并修改存储信息。

简单类型和引用类型变量存储机制如图 2-4 所示。

2. 变量传递

当变量与变量之间赋值时,引用类型变量和简单变量都属于值传递,不同的是简单变量传递的是内容本身,而引用变量传递的却是引用地址,具体示例如下。

□ 简单变量赋值

示例代码如下:

```
//定义简单变量 i
int i = 8;
//将 i 变量值赋给 j
int j = i;
//将 j 的值加 1
j = j + 1;
//打印输出 i, j
System.out.println(i);
System.out.println(j);
```

在上述代码中, i 和 j 属于简单类型变量,因此在赋值时,将 i 的值复制一份给了 j,复制完毕后 i 和 j 没有任何关联,修改 j 的值不会影响 i。程序最后输出结果如下:

```
8
9
```

□ 引用变量赋值

示例代码如下:

```
//定义引用变量 p
Person p = new Person();
p.setName("张三");
p.setAge(20);
//将 p 变量的地址赋给 q
Person q = p;
//修改 q 变量的内容
q.setName("李四");
q.setAge(30);
//打印输出 p 和 q 的信息
System.out.println(p.getName()+p.getAge());
System.out.println(q.getName()+q.getAge());
```

在上述代码中, p 和 q 属于引用类型变量,因此在赋值时,将 p 的引用地址复制一份给了 q,

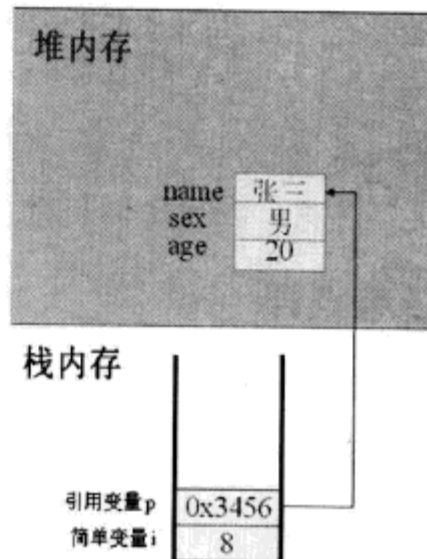


图 2-4 变量存储结构图

复制完毕后 p 和 q 具有相同地址，会共用同一存储空间，因此无论是通过 p 还是 q 修改存储空间信息后，都会相互影响。程序最后输出结果如下：

```
李四 30
李四 30
```

注意：在方法调用时，变量与参数变量的传值机制与上面相同。

疑难点评

由于简单变量和引用变量的存储结构不同，因此，在变量之间相互赋值时，不同类型的变量也有一定的差别。比如简单类型变量采取值传递；引用类型变量采取引用地址传递。

知识链接

FAQ2.13 成员变量和局部变量有什么区别？

FAQ2.15 Java 中有哪些运算符？优先级如何？

📖 难度系数：★★★★ 📖 问题频率：90%

核心解答

Java 中的运算符主要可以分为 5 大类，分别为算术运算符、关系运算符、逻辑运算符、位运算符和其他运算符，具体情况如表 2-6 所示。

表 2-6 Java 运算符

类 型	描 述
算术运算符	加 (+)、减 (-)、乘 (*)、除 (/)、取模 (%)、 递增 1 (++)、递减 1 (--)
关系运算符	大于 (>)、大于等于 (>=)、小于 (<)、小于等于 (<=)、 等于 (==)、不等于 (!=)
逻辑运算符	逻辑非 (!)、逻辑与 (&)、逻辑或 ()、逻辑异或 (^)、 短路与 (&&)、短路或 ()
位运算符	取反 (~)、按位与 (&)、按位或 ()、按位异或 (^)、 右移 (>>)、左移 (<<) 及无符号右移 (>>>)
其他运算符	赋值运算符 (=、+=、-=、*=、/=、>>=、<<=、>>>=)，条件运算符 (x?y:z)， 实例运算符 (instanceof)，创建对象运算符 (new)

在实际的开发中，可能在一个运算符中出现多个运算符，计算时按照优先级级别的高低进行计算，级别高的运算符先运算，级别低的运算符后计算，具体运算符的优先级如表 2-7 所示。

表 2-7 Java 运算符优先级

优 先 级	运 算 符
1、分隔符	. [] () , ;
2、单目运算符	+ (正) - (负) ~ ! ++ --
3、创建对象和类型转换	new (type)强制转型
4、乘法/除法	* / %
5、加法/减法	+ -
6、移位	<< >> >>>
7、关系比较	< <= > >= instanceof
8、等于/不等于	== !=
9、按位与	&
10、按位异或	^
11、按位或	
12、条件与	&&
13、条件或	
14、三目运算	?:
15、赋值	= += -= *= /= %= &= = ^= ~= <<= >>= >>>=

在上述表中，优先级由高到低分为 1~15 个级别。在实际的开发中，不需要去记忆运算符的优先级别，也不要刻意的使用运算符的优先级别，对于不清楚优先级的地方可以使用小括号提高优先级，示例代码如下：

```
int m = 12;
int n = m << 1 + 2;
int n = m << (1 + 2); //这样更直观
```

注意：在 Java 中运算符是不允许重载的。

疑难点评

Java 提供了大量的运算符，例如算数运算符、逻辑运算符、移位运算符等。在书写一个复杂的表达式时，如果包含多个运算符需要注意优先级问题，为了增强表达式的可读性，可以使用()将某些子表达式括起来。

知识链接

FAQ2.16 在实现 x 和 y 相加时，x+=y 和 x=x+y 两种实现方式有区别吗？

FAQ2.17 在执行与运算时，运算符&和&&有什么区别？

FAQ2.18 在实现 x 递增加 1 操作时，x++和++x 有什么区别？

FAQ2.16 在实现 x 和 y 相加时, $x+=y$ 和 $x=x+y$ 两种实现方式有区别吗?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

为了方便程序员进行频繁的赋值操作, Java 提供了很多扩展赋值运算符, 例如 $+=$ 、 $-=$ 、 $/=$ 、 $*=$ 等。具体使用示例如下:

```
int a = 10;
a += 1;
System.out.println(a);
```

上述代码的输出结果为 11。“ $a += 1$ ”的作用与“ $a = a + 1$ ”等价, 在一般情况下两种语句是可以通用的, 但是有些情况却不可通用。代码如下:

```
short s = 3;
s += 1; //编译通过
s = s + 1; //编译发生错误
```

在上述代码中, $s = s + 1$ 语句编译错误的原因是因为右边 $s + 1$ 表达式的计算结果为 int 类型, 而左边变量类型为 short 类型, 违反了自动转换规则, 需要类型强制转换。 $s += 1$ 语句编译正确, 原因是“ $+=$ ”运算符在 Java 环境中会自动根据接收变量的类型进行类型的强制转换。

通过上述示例可以得知, $x+=y$ 语句等价于“ $x=x+y$ ”和类型强制转换两个操作。

注意: $+=$ 、 $-=$ 、 $/=$ 和 $*=$ 等运算符的使用情况相似。

疑难点评

虽然 $x+=y$ 和 $x=x+y$ 两个表达式在平时使用时可以通用, 都可以实现两个变量值相加, 但是这两种表达式在内部运行时存在一些细微差别, 差别在于 $+=$ 除了实现 $+$ 功能之外还要进行一次类型的强制转换。

知识链接

FAQ2.15 Java 中有哪些运算符? 优先级如何?

FAQ2.17 在执行与运算时, 运算符 $\&$ 和 $\&\&$ 有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

在表达式中, 各逻辑运算符的运算规则如表 2-8 所示。

表 2-8 布尔逻辑运算表

操作数 a	操作数 b	! a	a & b	a && b	a b	a b	a ^ b
true	true	false	true	true	true	true	false
true	false	false	false	false	true	true	true
false	true	true	false	false	true	true	true
false	false	true	false	false	false	false	false

从表 2-8 可以看出&和&&, |和||运算符的运算结果相同。虽然运算结果相同但是两个运算符的运算机制有很大差别, 具体测试代码如下:

```
public static boolean f1(){
    System.out.println("f1()执行");
    return false;
}

public static boolean f2(){
    System.out.println("f2()执行");
    return true;
}

public static boolean f3(){
    System.out.println("f3()执行");
    return false;
}
```

在上述代码中, 定义了 f1()、f2()和 f3()方法, 如果使用&操作符计算输出 f1()&f2()&f3()表达式的结果, 控制台输出内容如下:

```
f1()执行
f2()执行
f3()执行
false
```

如果使用&&操作符计算输出 f1()&&f2()&&f3()表达式的结果, 控制台输出内容如下:

```
f1()执行
false
```

通过上述输出内容可以看出, 虽然结果两个表达式的结果一样, 但是&&运算符比&运算符少执行了 f2()和 f3()方法, &&运算符的效率要高些。

&&被称为短路与, 与操作的特点是只要参与运算的操作数有一个是 false, 最后整个表达式的结果就是 false, 因此&&运算表达式从左往右计算时, 如果发现一个 false 值, 就放弃继续运算, 直接将 false 作为整个表达式的计算结果并返回。

&被称为逻辑与, 该运算符在计算表达式结果时, 表达式的每个操作数都要参与计算, 然后才得出整个表达式的计算结果。

注意: |和||运算符的区别与&和&&的情况相似, ||运算符如果遇到 true 操作数就放弃后续运算, 将 true 作为整个表达式的计算结果并返回。

疑难点评

&和&&是 Java 中的布尔逻辑运算符, 可以实现逻辑与操作, 虽然在布尔逻辑运算表达式中使用&或&&都可以得到相同结果, 但是两者的执行效率不同。建议采用&&, 因为该运算符从执行效率上讲要优于&。

知识链接

FAQ2.15 Java 中有哪些运算符? 优先级如何?

FAQ2.18 在实现 x 递增加 1 操作时, x++和++x 有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 89%

核心解答

Java 提供了++和--两个运算符, 目的是方便程序员对数值变量进行加 1 和减 1 操作。

++和--运算符既可以写在变量前面也可以写在变量后面, 如果 x++和++x 作为一条独立语句时没有什么差别, 具体代码如下:

```
int i = 10;
i++;
System.out.println(i);
```

在上述代码中, “i++” 是作为一条独立语句执行的, 因此写成 “++i” 也可以。“i++” 等价于 “i=i+1”, 上述程序运行后的输出结果为 11。

如果 x++和++x 作为某表达式的一部分使用, 就会存在差别。表达式运算时, ++x 表示先将变量 x 的值加 1, 然后 x 再参与表达式运算; x++则表示先用 x 值参与表达式运算, 然后再将变量 x 的值加 1。

具体代码如下:

```
int a = 10;
int b = 10;
int j = 10+(a++); //a 变量值先参与表达式运算, 然后再加 1 运算
int z = 10+(++b); //b 变量值先加 1 运算, 然后再参与表达式运算
System.out.println(j);
System.out.println(z);
```

上述代码执行的输出结果为 20 和 21。

疑难点评

++运算符可实现变量值递增加 1 的功能,在实际使用时,该运算符可以放在变量之前,也可以放在变量之后。如果++用在某表达式中,那么该运算符所处位置不同,代表的含义也不相同。

知识链接

FAQ2.15 Java 中有哪些运算符? 优先级如何?

FAQ2.19 x?y:z 格式的语句表示什么意思?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

在 Java 语言中有一种特殊的运算符,被称为三目运算符。三目运算符表示该运算由 3 部分组成,其格式为“x?y:z”,其中 x 是一个布尔表达式,当 x 表达式的值为 true 时,返回 y 表达式值,否则返回 z 表达式值。

示例代码如下:

```
int age = 20;  
String msg = age>18?"成年":"未成年";  
boolean flag = age>30?true:false;  
System.out.println(msg);  
System.out.println(flag);
```

上述代码的执行结果如下:

```
成年  
false
```

注意: 在“x?y:z”格式的语句中, y 和 z 表达式的值应该是相同的数据类型。

疑难点评

通过上述介绍,我们知道三目运算也是根据条件表达式的结果确定返回值,这与 if...else 和 switch 语句的作用相同。三目运算与 if...else 语句比较的话,三目运算会使代码更加简洁一些,因此可以优先考虑使用三目运算。有很多 Java 爱好者非常关心三目运算和 if...else 语句哪种执行效率高。通过反汇编之后发现 if...else 的效率略高一些,但是,两者并没有十分明显的效率差别,选择哪一种实现方式不会对程序效率产生影响。

知识链接

FAQ2.15 Java 中有哪些运算符? 优先级如何?

FAQ2.20 “+” 操作符在 Java 内部是如何实现字符串连接的?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

Java 不支持运算符重载,但在 Java 内部提供了一些特殊运算符,它们具有重载的特征。例如 “+” 在实现字符串连接时,操作数可以是两个字符串,也可以是一个字符串和一个其他类型。

下面的示例介绍了 “+” 操作符实现字符串连接功能的具体过程。

示例代码如下:

```
String a = "a";  
String b = "b";  
//两个 String 类型连接  
String c = a + b;  
//一个 String 类型和一个数值类型连接  
String d = c + 1;
```

将上述代码编译后再反编译,可以得到的结果如下:

```
String a = "a";  
String b = "b";  
String c = (new StringBuilder(String.valueOf(a))).append(b).toString();  
String d = (new StringBuilder(String.valueOf(c))).append(1).toString();
```

通过上述反编译结果可以看出,“+” 在实现字符串连接过程中,实际上是借助了 `StringBuilder` 类及其 `append()` 方法。

`String` 类代表大小固定的字符串,一旦声明定义后,内容和大小将不可改变。`String` 类中提供的所有字符串操作方法,都是操作结果创建了一个新的 `String` 对象并返回。

为了考虑效率问题,在底层 Java 采用了 `StringBuilder` 类,该类代表大小可变的字符串,利用 `StringBuilder` 类的 `append()` 和 `insert()` 方法可以在原字符串基础上修改。`append()` 和 `insert()` 方法被重载过多次,可以接收多种类型的参数。

疑难点评

在日常编程时,很多 Java 程序员只关注语法格式和程序的输出结果,但是作为一个更高层次的程序设计者,也应该对程序执行时的内部机制有一定了解,例如运行原理、内存分配等。

知识链接

FAQ2.15 Java 中有哪些运算符? 优先级如何?

FAQ2.21 ==和 equals()都可用于比较两个操作数是否相等，它们有什么区别吗？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

==是一个关系运算符，用于判断两个简单变量的值是否相等，或两个引用变量的引用地址是否相等。

equals()是一个方法，用于判断引用变量引用地址指向的存储内容是否相等。

equals()是 Object 类中定义的一个方法，由于其他引用类型默认继承 Object，因此该方法在其他引用类型中都可以使用。代码如下：

```
int a = 10;
int b = 10;
//比较简单变量 a 和 b 的值是否相等
System.out.println(a==b);
String c = new String("tom");
String d = new String("tom");
//比较引用变量 c 和 d 的引用地址是否相等
System.out.println(c == d);
//比较引用变量 c 和 d 的内容是否相等
System.out.println(c.equals(d));
```

上述代码执行结果如下：

```
true
false
true
```

注意：Object 类中定义的 equals()方法是直接使用==操作符实现的，因此，在自定义类型时建议重写 equals()方法，实现自定义的比较规则。例如 String、Integer 等类都已经对 equals()方法进行过重写。

疑难点评

==和 equals()方法可用于比较两个操作数是否相等，一般很多初学者都不知道==和 equals()方法的区别。如果==和 equals()方法两者混用，在某些情况下会造成程序的逻辑错误。在比较两个字符串是否相等时，建议使用 equals()方法实现。

知识链接

FAQ2.15 Java 中有哪些运算符？优先级如何？

FAQ2.22 创建 String 对象时,使用 String s=new String ("abc")和 String s="abc"语句有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

字符串是程序中使用频率最高的一种数据类型,Java 为了加强程序的运行速度,因此设计了两种不同的方法来生成字符串对象,一种是调用 String 类的构造方法,另一种是使用双引号“”。这两种方法生成的字符串对象,在内存中存取机制是不同的。

Java 为 String 类型提供了缓冲池机制,当使用双引号定义对象时,Java 环境首先去字符串缓冲池中寻找相同内容的字符串,如果存在就直接拿出来应用,如果不存在则创建一个新的字符串放在缓冲池中。示例代码如下:

```
String a = "tom";  
String b = "tom";
```

在上述代码中,变量 a 和 b 使用的是缓冲区中同一个存储对象。

在使用 String 构造方法定义对象时,Java 环境会和创建其他类型的对象一样,每次调用时,都会创建一个新的对象。示例代码如下:

```
String a = new String("tom");  
String b = new String("tom");
```

在上述代码中,变量 a 和 b 使用的是两个不同的存储对象,只是对象中的内容相同。

疑难点评

String s=new String("abc")和 String s="abc"都可以定义 String 类型的对象,但需要注意两者的区别。一般情况下,建议使用 String s="abc"方式,因为该方式采用字符串缓冲池机制,效率高。

FAQ2.23 break 和 continue 语句有什么区别?

📖 难度系数: ★★★

📖 问题频率: 85%

核心解答

break 的作用是可以结束其所在的 switch 语句或者循环语句的执行。

continue 的作用是终止本次循环,进入下一次循环。

❑ break 使用示例如下:

```
for(int i=0;i<10;i++){
    if(i == 3){
        break;
    }
    System.out.println(i);
}
System.out.println("Game Over!");
```

在上述代码中, 当 *i* 变量值等于 3 时, 执行 `break` 语句退出 `for` 循环语句。因此程序输出结果如下:

```
0
1
2
Game Over!
```

□ `continue` 使用示例如下:

```
for(int i=0;i<10;i++){
    if(i == 3){
        continue;
    }
    System.out.println(i);
}
System.out.println("Game Over!");
```

在上述代码中, 当 *i* 变量值等于 3 时, 执行 `continue` 语句终止本次循环并进入到下一次。因此程序输出结果如下:

```
0
1
2
4
5
6
7
8
9
Game Over!
```

注意: 如果将上述示例中的 `continue` 关键字改为 `return`, 表示当 *i* 变量值等于 3 时退出当前方法, 因此后续的 “Game Over!” 将不会输出。

疑难点评

`break` 和 `continue` 语句常见于循环语句中, 注意两者的区别。

FAQ2.24 数组如何定义和初始化?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

数组可以用于存储一组相同类型的数据，数组的元素可以是简单类型也可以是引用类型。

1. 数组定义

数组可分为一维数组、二维数组和多维数组。一维数组的定义有以下两种方式：

```
type[] arr_name;  
type arr_name[];
```

二维数组的定义方式如下：

```
type[][] arr_name;  
type arr_name[][];
```

多维数组可根据维数指定相应个数的[]。

在定义数组时，[]可以放在类型后面，也可以放在变量名后面，示例代码如下：

```
int[] arr1;  
String[] arr2;  
float arr3[];  
String[][] arr4;
```

注意：在数组定义中不能指定数组的长度，在数组的创建阶段需要指定大小，用于分配存储空间。

2. 数组初始化

数组初始化有以下两种方式。

□ 静态初始化

静态初始化是指在定义时同时指定数组元素内容，示例代码如下：

```
int[] arr1 = {1,2,3,4,5};  
String[] arr2 = {"tom","rose","sunny"};  
String[][] arr3 = {{ "tom","American"},  
                   { "jack","England"},  
                   { "张三","china"}};
```

在静态初始化时，不需要指定数组的大小，系统会根据指定的内容的数量自动分配大小。

□ 动态初始化

动态初始化是指在定义时首先通过 new 关键字开辟指定大小的存储空间，然后再为存储单元指定内容，示例代码如下：

```
//初始化一维数组  
int[] arr1 = new int[3];  
arr1[0] = 10;  
arr1[1] = 20;  
arr1[2] = 30;  
//初始化二维数组  
String[][] arr2 = new String[3][2];  
arr2[0][0] = "tom";  
arr2[0][1] = "American";  
arr2[1][0] = "jack";  
arr2[1][1] = "England";
```



```
arr2[2][0] = "张三";  
arr2[2][1] = "china";
```

在通过 new 关键字创建多维数组时,不必指定每一维的大小,而只需要指定最左边的维的大小即可。如果指定了某一维的大小,那么处于这一维左边的各维大小都需要指定,否则将编译出错,代码如下:

```
//错误定义格式  
String[][] arr = new String[][2];  
//正确定义格式  
String[][] arr = new String[3][];  
//正确定义格式  
String[][] arr = new String[3][2];
```

注意:使用 new 关键字创建数组对象时,在分配存储空间后,系统会为每一个存储单元默认初始化,其规则遵循成员变量默认初始化规则。例如 int 初始化为 0、boolean 初始化为 false 等。

疑难点评

数组可用于存储一组相同数据类型的元素。数组的定义和初始化方式不是惟一的,可根据不同情况进行选择。

知识链接

FAQ2.25 如何实现一维和二维数组的遍历?

FAQ2.26 如何实现数组的复制?

FAQ2.27 数组的排序算法有哪些?如何实现?

FAQ2.25 如何实现一维和二维数组的遍历?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

可使用 for 语句遍历数组元素,下面提供两种实现方式,一种是原有 for 循环,另一种是从 JDK 5.0 开始提供的新式 for 循环。

❑ 原有 for 循环

```
String[] arr = {"tom","rose","sunny"};  
for(int i=0;i<arr.length;i++){  
    System.out.println(arr[i]);  
}
```

❑ 新式 for 循环

```
String[] arr = {"tom","rose","sunny"};  
for(String s:arr){  
    System.out.println(s);  
}
```

二维数组的遍历需要使用双重循环，具体代码如下：

```
String[][] arr = {{"tom","American"},
                  {"jack","England"},
                  {"张三","china"}};
for(int i=0;i<arr.length;i++){
    for(int j=0;j<arr[i].length;j++){
        System.out.print(arr[i][j]);
    }
    //打印一个元素后，换一行
    System.out.println();
}
```

注意：数组元素的访问索引是从0开始的，因此如果数组大小为n，其访问索引应该为0~n-1。

疑难点评

循环遍历数组的方法有很多，上面只列举了其中的2种，一般在遍历数组时习惯使用for循环语句。

知识链接

FAQ2.24 数组如何定义和初始化？

FAQ2.26 如何实现数组的复制？

📖 难度系数：★★★★

📖 问题频率：80%

核心解答

将一个数组变量复制到另一个数组变量，可以通过多种方式实现，例如采用for循环遍历赋值等。

如果只是需要将数组的值复制到另一个数组，可以使用System类的一个静态方法arraycopy()，该方法的定义如下：

```
static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)
```

arraycopy()方法的参数含义如下。

- ☐ src：源数组。
- ☐ srcPos：源数组中的起始位置。
- ☐ dest：目标数组。
- ☐ destPos：目标数据中的起始位置。
- ☐ length：要复制的数组元素的数量。

示例代码如下所示：

```
int[] a = {10,20,30,40,50};
int[] b = {0,1,2,3,4,5,6,7,8,9};
```

```
System.arraycopy(a,0,b,5,5);
for(int i:b){
    System.out.print(i+" ");
}
```

在上述代码中，实现了从 a 数组中 0 索引开始提取 5 个元素，将其赋值到 b 数组中，从 b 数组的第 5 个索引开始存放。打印输出 b 数组的结果如下：

```
0 1 2 3 4 10 20 30 40 50
```

注意：在使用 arraycopy () 方法时，src 和 dest 参数都必须是同类型或者可以进行转换类型的数组，当类型不一致时，arraycopy() 方法会抛出异常。

疑难点评

实现数组复制的方法有很多，但使用 System 类的 arraycopy() 方法是最简单、最方便的一种。

知识链接

FAQ2.24 数组如何定义和初始化？

FAQ2.27 数组的排序算法有哪些？如何实现？

📖 难度系数：★★★★★

📖 问题频率：90%

核心解答

在实现数组元素排序时，排序算法有很多，例如冒泡排序法、选择排序法、插入排序法以及快速排序法等，下面介绍各种排序法的具体实现。

□ 选择排序法

选择排序法的基本思路是：将要排序的数组分成两部分，一部分是从小到大已经排好序的，一部分是无序的，从无序的部分取出最小的数值，放到已经排好序的部分的最后。选择排序法的实现代码如下：

```
//选择排序法
public int[] xuanze(int[] arr) {
    int t;
    for (int i = 0; i < arr.length; i++) {
        int m = i;
        for (int j = i + 1; j < arr.length; j++) {
            //如果 j 元素比 m 元素小，将 j 赋值给 m
            if (arr[j] < arr[m]) {
                m = j;
            }
        }
        //交换 m 和 i 两个元素的位置
        if (i != m){
```

```
        t=arr[i];
        arr[i]=arr[m];
        arr[m]=t;
    }
}
return arr;
}
```

□ 冒泡排序法

冒泡排序法的基本思路是：从数组开始扫描待排序的元素，在扫描过程中依次对相邻元素进行比较，将数值大的元素后移。每经过一趟排序后，数值最大的元素将移到末尾，此时记下该元素的位置，下一趟排序只需要比较到此位置为止，直到所有元素都已有序排列。冒泡排序法的实现代码如下：

```
//冒泡排序法
public int[] maopao(int[] arr)
{
    int t;
    for (int i=0;i<arr.length;i++)
    {
        //循环比较相邻两个元素大小
        for(int j=0;j<arr.length-i-1;j++)
        {
            //比较相邻元素大小，小的前移，大的后移
            if (arr[j]>arr[j+1])
            {
                t=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=t;
            }
        }
    }
    return arr;
}
```

□ 插入排序法

插入排序法的基本思路是：将要排序的数组分成两部分，每次从后面的数组部分中取出索引最小的数组元素，插入到前面数组部分的适当位置中。通常在开始排序时，将数组的第一个元素作为一组，后面的所有元素被当成另一组。插入排序法的实现代码如下：

```
// 插入排序
public static int[] charu(int[] arr) {
    //把第一个元素看做一部分，第二个元素看做另一部分
    //从第二部分中依次取元素插入到第一部分中
    for (int i = 1; i < arr.length; i++) {
        int tmp = arr[i];
        int j = i - 1;
        //依次和 i 前面元素比较，寻找合适插入位置
        while(tmp < arr[j]){
            arr[j+1] = arr[j];
        }
    }
}
```



```

        j--;
        if(j == -1){
            break;
        }
    }
    //将插入元素插入到合适位置
    arr[j+1] = tmp;
}
return arr;
}

```

□ 快速排序法

快速排序法是当今被认为最好的排序算法之一,它的基本思路是:将一个大的数组的排序问题,分解成2个小的数组的排序,而每个小的数组的排序又可以继续分解成更小的2个数组,这样一直递归分解下去,直到数组的大小最大为2。快速排序法的实现代码如下:

```

//快速排序法
public static int[] kuaisu(int[] arr){
    return quicksort(arr,0,arr.length-1);
}

//递归分组排序
public static int[] quicksort(int[] arr,int left,int right){
    int t;
    if(left < right){
        int s = arr[left];
        int i = left;
        int j = right + 1;
        while(true){
            //向右找大于 s 的数的索引
            while(i+1 < arr.length && arr[++i] < s);
            //向左找小于 s 的数的索引
            while(j-1 > -1 && arr[--j] > s);
            //如果 i>=j,退出循环
            if(i >= j){
                break;
            }else{
                //交换 i 和 j 位置的元素
                t=arr[i];
                arr[i]=arr[j];
                arr[j]=t;
            }
        }

        arr[left] = arr[j];
        arr[j] = s;
        //对左边进行递归
        quicksort(arr,left,j-1);
        //对右边进行递归
        quicksort(arr,j+1,right);
    }
}

```

```
}  
return arr;  
}
```

注意: Arrays 类中提供了 sort()方法并进行了多次重载,可以实现各种类型数组的排序功能。该方法采取的排序算法是一个经过调优的快速排序法,使用示例“Arrays.sort(a)”,a 是一个数组变量。

疑难点评

排序算法很多,各算法的排序效率也不同,很多企业经常拿数组排序功能来考察应聘者的编程能力。上面给出了几种常见的算法实现,可以通过上面的例子来锻炼自己的编程能力。

知识链接

FAQ2.24 数组如何定义和初始化?

FAQ2.25 如何实现一维和二维数组的遍历?

FAQ2.28 如何解决 ArrayIndexOutOfBoundsException 异常?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

ArrayIndexOutOfBoundsException 意思是数组索引越界异常,是在提取数组元素时使用索引不当引起的。例如下列代码将产生 ArrayIndexOutOfBoundsException 异常。

```
int[] a = {1,2,3,4,5};  
int i = a[5];//引发 ArrayIndexOutOfBoundsException 异常
```

在上述代码中,int i = a[5]语句将会产生 ArrayIndexOutOfBoundsException 异常,原因是 a 数组大小为 5,其访问索引只在 0~4 范围内,因此会产生索引越界异常。

注意: 在使用循环语句访问数组时,需要特别小心,因为如果循环边界值如果控制不好,会很容易产生 ArrayIndexOutOfBoundsException 异常。

疑难点评

在数组使用时,如果数组索引越界就会产生 ArrayIndexOutOfBoundsException 异常,注意正确使用数组索引,数组索引从 0 开始计数。

第3章

Java 与面向对象

本章重点介绍了 Java 与面向对象程序设计相关的一些疑难问题，内容主要涉及类、对象、接口、多态、内部类、匿名类、封装类、方法重写、方法重载以及异常处理等几个方面。通过本章的学习读者可以清晰地理解面向对象的基本概念、各技术点的应用及其使用好处等。

FAQ3.01 什么是类、对象、属性和方法？

📖 难度系数：★★★

📖 问题频率：90%

核心解答

类是对一类相同事物的抽象描述。一类事物通常会有很多相同的特征和行为，程序员可以将这些共同特征和行为提取，定义成一个类。

对象是类的一个具体实现，代表一个实实在在的个体。

例如，“地球是人类赖以生存和发展的基础。”中的“人类”代指一类事物，因此表示类。而“张三这个人很实在。”中的“张三”代指具体的一个人，因此表示对象。

在定义类的时候，类中可以包含属性和方法。属性是对一类相同特征的抽象，例如身高、体重、性别等。方法是对一类相同行为的抽象，例如思考、吃饭、学习等。

□ 类的定义

类的定义格式如下：

```
[<修饰符>] class <类名> {  
    [<属性定义>]  
    [<构造方法定义>]  
    [<方法定义>]  
}
```

[]表示该部分可有可无。例如“修饰符”部分可以写 public、protected 或 private，也可以不写，表示 default 状态。这几个修饰符的区别请参考“FAQ2.03 用 public、protected 和 private 修饰方法有什么区别？”的解答。

类定义的示例如下：

```
public class Student{
    //属性定义
    private int age;
    //方法定义
    public void setAge(int _age) {
        age = _age;
    }
}
```

□ 对象的定义

对象的定义格式如下:

类型 对象名 = new 构造方法;

对象定义的示例如下:

```
Person p = new Person();
Student s = new Student("tom","三年级");
```

在上述代码中, Person 类中必须要求有一个无参的构造方法, Student 类中必须要求有一个带有两个 String 类型参数的构造方法。否则编译程序时会提示错误。

□ 属性的定义

属性的定义格式如下:

[<修饰符>] 类型 <属性名> [=属性初始值];

属性定义的示例如下:

```
private int age;
private String name = "tom";
```

在上述代码中, 属性 age 没有指定初始值, 系统编译时将依据默认初始化规则进行初始化。默认初始化规则请参考 FAQ2.13 “成员变量和局部变量有什么区别?” 的解答。

□ 方法的定义

方法定义的格式如下:

```
< 修饰符> <返回类型> <方法名>([<参数列表>]) {
    [< 方法的处理代码>]
}
```

方法定义的示例如下:

```
//有返回值的情况
public int getAge() {
    return age;
}
//无返回值的情况
public void setAge(int _age) {
    age = _age;
}
```

在上述代码中, getAge()方法具有一个 int 返回值, 当方法定义返回类型后, 方法的处理代码部分必须得有一个有效的 return 语句。意思是无论程序怎么执行, 都需要执行到一个 return 语句。方法无返回值时使用 void 关键字, 可以没有 return 语句。

疑难点评

类、对象、属性和方法是 Java 编程入门的最基本概念，需要读者重点理解。

知识链接

FAQ3.09 什么是继承？有什么好处？

FAQ3.02 什么是包？有什么好处？

📖 难度系数：★★★

📖 问题频率：85%

核心解答

大型项目一般需要程序员编写很多的类，有时候几个不同的类可能都想使用的名字相同。为了避免类名重复的冲突，Java 引入了包机制，通过包可以组织程序的类，在不同的包中允许使用相同名字。例如 JDK 提供的 Date 类就存在两个，一个在 java.sql 包下，另一个在 java.util 包下。

package 的定义格式如下：

```
package 包名;  
类的定义;
```

包定义的示例代码如下：

```
package com;  
public class Person{}
```

在上述代码中，表示将 Person 类定义在 com 包下，包名也可以写成“com.bean.qmrz”的形式，包定义的深度没有限制。

注意：两个不同包下的类相互调用时，需要在 package 语句和类定义之间加入 import 语句，用于指定引入类的路径，例如“import java.util.Date;”；也可以不明确指明引入类，例如“import java.util.*;”，表示引入“java.util”包下所有的类。

疑难点评

包是存放 Java 程序的单元，可以将相关的 Java 程序放在同一个包中，便于程序的维护和管理。

FAQ3.03 什么是抽象类？有什么好处？

📖 难度系数：★★★★

📖 问题频率：95%

核心解答

有 `abstract` 关键字修饰, 允许包含未实现方法的类被称为抽象类。

有些情况下, 在定义类时, 可能有些类方法需要具体实现, 有些类方法无法确定具体的实现方式, 这时可以将该类定义成抽象类, 无法具体实现的方法定义成抽象方法。

抽象类的定义格式如下:

```
[<修饰符>] abstract class <类名> {  
    [<属性定义>]  
    [<构造方法定义>]  
    [abstract][<方法声明>]  
}
```

从上述格式可以看出, 抽象类与普通类定义格式相似, 只是在类定义和方法定义中使用了 `abstract` 关键字。

注意: 在抽象类中可以包含抽象方法, 也可以不包含抽象方法。

例如现在需要编写一个图形类, 在该类中定义计算周长和面积的方法时, 由于不同图形有不同的计算方式, 因此不能明确具体如何实现。此时可以将这些方法定义为抽象方法, 将来放到具体子类中去实现。抽象类定义的示例代码如下:

```
public abstract class Shape{  
    //属性定义  
    private String color;  
    //方法定义  
    public void setColor(String _color) {  
        color = _color;  
    }  
    //定义计算周长的抽象方法  
    public abstract double perimeter();  
    //定义计算面积的抽象方法  
    public abstract double area();  
}
```

在上述代码中, 定义了一个抽象类 `Shape`, 其中除了包含一些属性和具体实现的方法之外, 还包含了两个抽象方法, 即 `perimeter()` 方法和 `area()` 方法。

抽象类在使用时, 需要注意以下几点特性。

- ☐ 抽象类不能实例化, 即不能创建对象, 只能作为父类用于被继承。
- ☐ 子类继承一个抽象类后, 必须要实现父类中所有的抽象方法, 否则子类也要定义为抽象类。
- ☐ 抽象类中可以包含抽象方法, 也可以不包含抽象方法。
- ☐ 如果类中包含抽象方法, 那么类必须定义成抽象类。

注意: 由于抽象类可以包含实现方法和抽象方法, 因此非常灵活, 在设计模式中, 有很多模式都是基于抽象类实现的, 例如模板模式等。

疑难点评

抽象类与普通类相比有一些不同的特性，例如不能实例化、可以包含未实现的方法等。读者应重点了解抽象类的特性及其使用环境。

知识链接

FAQ3.11 抽象类和接口都可以包含抽象方法，那么它们有什么区别？使用时该如何选择？

FAQ3.04 什么是接口？有什么好处？

📖 难度系数：★★★★

📖 问题频率：95%

核心解答

接口是方法声明和常量值定义的集合。在有些情况下，如果某个类的所有方法都无法具体实现，此时可以使用接口定义。接口可以理解成一个标准，其他类可以遵守该标准做不同的实现。例如 Java 访问数据库的 API (JDBC)，在定义时基本上都采用了接口定义的方式，具体接口方法的实现交给了各个不同数据库厂商去完成，只要厂商遵循接口标准，Java 就可以进行数据库访问。利用接口使得程序非常灵活，扩展性也变得非常好。

接口的定义格式如下：

```
[<修饰符>] interface <接口名> {  
    [<常量声明>]  
    [<方法声明>]  
}
```

包定义的示例代码如下：

```
public interface Db {  
    public int MAX = 50;  
    public void connection();  
    public void disconnection();  
}
```

在上述代码中，虽然 MAX 没有显示使用 static 和 final 关键字声明，但是编译器在编译时会自动加入。connection()和 disconnection()方法可以根据数据库的不同做不同的实现。

使用示例如下：

```
public class OracleDb implements Db {  
    public void connection() {  
        //添加连接 Oracle 数据库的代码  
    };  
    public void disconnection() {  
        //添加释放 Oracle 数据库的代码  
    };  
}
```

注意：由于接口中的属性属于常量定义，因此在定义属性时必须显示指定初始值，不能使用默认初始化的形式。

接口在使用时，需要注意以下几点特性。

- 接口只包含方法声明和常量定义，即使定义普通属性，该属性在编译后也将变为常量。
- 当其他类实现该接口时，接口中定义的所有方法都要求全部实现，否则需要定义成抽象类。
- 一个类可以实现多个接口。例如 `public class A implements B,C,D{}格式`。
- 定义接口时可以使用继承，接口之间允许多继承。例如 `public interface A extends B,C{}格式`。

疑难点评

接口只包含方法和常量的定义并没有具体实现，一个接口可以有多种不同的实现类。现在经常有人说“面向接口编程”，在程序中使用接口可以提高程序的灵活性，这也是利用大量接口编程的主要原因。

注意：接口是方法声明和常量值定义的集合，不允许包含变量。接口之间允许多继承，类之间只允许单继承。

知识链接

FAQ3.11 抽象类和接口都可以包含抽象方法，那么它们有什么区别？使用时该如何选择？

FAQ3.05 什么是多态？有什么好处？

📖 难度系数：★★★★★

📖 问题频率：98%

核心解答

现实生活也能找到多态关系的原型，例如有一个父亲 F，他有两个孩子 B 和 C，那么 F 在某些场合可以代表孩子 B 和 C 做一些事情，因为父亲对孩子有监护权和抚养权。F 即可以代表 B，又可以代表 C，因此 F 具有一定的多态性。

在 Java 语言中，多态主要是指对象变量的多态，即一个 A 类型的变量既可以指向 A 类型的对象，又可以指向其子类的对象。通过父亲和孩子的示例也可以看出应用多态需要有一定的前提条件，即要求 F 与 B、C 之间需要存在继承关系。

多态的示例代码如下：

```
//A 类的定义
public class A{
    //定义属性和方法
}
//子类 B 的定义
public class B extends A{
```



```
//定义属性和方法  
}
```

在上述代码中，A 类是 B 类的父类，根据多态的定义，“A a = new B()”代码是合法的，并且是多态的一种应用。

注意：除了上述多态形式之外，一个接口类型变量也可以指向其实现类的实例，这也是多态的一种表现。

在定义方法时，如果利用多态的形式定义方法参数和返回类型，可以增强方法的灵活性，这是多态为 Java 程序设计带来的最大好处。例如 ArrayList 类中的 add(Object obj)方法，参数类型为 Object，由于 Object 是 Java 中所有类的基类，因此 add()方法在使用时，可以传入任何一个类的对象，而不用为每一种类型定义一个方法，使程序变得非常灵活。

疑难点评

多态是 Java 编程中最重要、最难理解的一个概念，多态给 Java 编程带来了很多的方便，提高了程序的灵活性，例如利用多态性实现变量赋值、利用多态性实现方法传参等。

知识链接

FAQ3.20 Java 中动态绑定是什么意思？

FAQ3.21 Java 中是如何实现多态的？实现机制是什么？

FAQ3.06 什么是内部类？有什么好处？

📖 难度系数：★★★

📖 问题频率：80%

核心解答

在 Java 中，除了在类中定义属性和方法之外，还可以再定义类。定义在一个类内部的类被称为内部类。

内部类的定义格式如下：

```
//A 类的定义  
public class A {  
    //定义属性  
    private int age;  
  
    //定义内部类  
    class B {  
        //定义属性和方法  
    }  
  
    //定义方法
```

```
        public int getAge(){  
            return age;  
        }  
    }  
}
```

在上述代码中, 类 B 就是一个内部类, B 类与 age 和 getAge()方法是并列关系, 编译后会生成 A.class 和 A\$B.class 两个字节码文件。

例如以下示例代码则不属于内部类形式的定义。

```
//A 类的定义  
public class A{  
    //定义属性  
    private int age;  
  
    //定义方法  
    public void getAge(){  
        return age;  
    }  
}  
  
//B 类的定义  
class B {  
    //定义属性和方法  
}
```

在上述代码中, 类 A 和 B 是并列关系, 编译后会产生 A.class 和 B.class 两个字节码文件, 它们是两个互不相同的类。

内部类的概念是从 JDK 1.1 版本开始引入的, 其优点主要有以下几点。

- ☐ 内部类对象能访问其所处类的私有属性和方法。
- ☐ 内部类能够隐藏起来, 不被同一个包中的其他类访问。如果一个类只对某个类提供使用, 那么可以将其定义为内部类。
- ☐ 匿名内部类可以方便地用在回调方法中, 典型应用是图形编程中的事件处理。

内部类的特征主要有以下几点。

- ☐ 内部类可以声明为抽象类, 因此可以被其他的内部类继承, 也可以声明为 final 的。
- ☐ 和外部类不同, 内部类可以声明为 private 或 protected, 外部类只能用 public 和 default。
- ☐ 内部类可以声明为 static 的, 但此时就不能再使用外层封装类的非 static 的成员变量。
- ☐ 非 static 的内部类中的成员不能声明为 static 的, 只有在顶层类或 static 的内部类中才可以声明 static 成员。

注意: 类除了可以定义在类中与方法并列之外, 还可以定义在方法的内部或者一个自由块中, 此时被称为局部内部类, 只能在方法体或者自由块中使用。

疑难点评

内部类是定义在一个类中的类, 在有些情况下, 使用内部类可以给编程带来方便, 读者应重点掌握内部类如何定义、如何使用的以及使用优点。

知识链接

FAQ3.25 如何调用内部类中的方法？

FAQ3.26 当内部类和外部类的成员名称相同时，如何在内部类中调用外部类的成员？

FAQ3.07 什么是匿名内部类？如何使用？

📖 难度系数：★★★

📖 问题频率：80%

核心解答

内部类可以在类中或方法中定义，匿名内部类是指在定义时没有名称的内部类，一般常见于方法中。如果某个类只需要使用一次，此时可以采用匿名内部类的方式定义。匿名内部类在 GUI 图形用户界面的应用程序中应用非常普遍，例如为某个按钮添加一个处理事件时，有些按钮处理功能只需要使用一次，因此可以采用匿名类为按钮添加处理功能。

匿名内部类的使用格式如下：

```
new <类或接口>(){  
    //定义属性和方法  
}
```

由于匿名内部类没有名字，因此需要在定义的同时使用，这就是在定义时使用 `new` 关键字作为开始的原因。`new` 关键字的后面跟的是匿名类需要继承的父类名或者是要实现的接口名。在匿名类主体中一般需要重写父类中的某些方法或者实现接口中的所有方法。

匿名内部类使用示例的代码如下：

```
Frame f = new Frame();  
f.addWindowListener(new WindowAdapter()  
{  
    public void windowClosing(WindowEvent e)  
    {  
        System.exit(0);  
    }  
});  
...
```

在上述代码中，`addWindowListener()` 方法需要一个事件处理对象，该对象需要是 `WindowAdapter` 类型，因此匿名类继承了 `WindowAdapter` 抽象类，并重写了 `windowClosing()` 方法。

注意：匿名内部类的声明是在编译时进行的，实例化在运行时进行。这意味着在 `for` 循环中使用匿名类时，`new` 语句会创建相同匿名类的几个实例，而不是创建几个不同匿名类的一个实例。

疑难点评

匿名内部类也是内部类的一种，只是没有类名。如果一个类的功能仅使用一次，就可以考虑使用匿名内部类的形式定义。

知识链接

FAQ3.27 匿名内部类如何访问外部方法的局部变量或参数？

FAQ3.08 什么是封装类？有什么作用？

📖 难度系数：★★★

📖 问题频率：75%

核心解答

Java 是完全面向对象编程的语言，为了使用方便 JDK 的设计者保留了 8 种简单类型，例如 int、float 和 double 等。这 8 种简单类型不具备面向对象的基本特征，没有属性和方法。

在某些情况下，JDK 中很多 API 的设计与简单类型相违背，例如 List、Vector 和 Map 等集合类，很多操作方法的参数和返回类型都定义成 Object 类型。如果需要使用这些方法操作简单变量值，首先需要将简单类型转换成引用类型后才能使用。

基于以上原因，JDK 的设计者在提供 8 种简单类型的同时，还为每一种简单类型提供了一个封装类。这些封装类封装了与简单类型相关的各种操作，例如实现简单类型和 String 类型之间的转化等。

简单类型和封装类型之间的对应关系如表 3-1 所示。

表 3-1 简单类型和封装类型对应表

基 本 类 型	封 装 类 型
byte	Byte
short	Short
int	Integer
long	Long
boolean	Boolean
float	Float
double	Double
char	Character

注意：从 JDK 5.0 版本开始引入了自动装箱和自动拆箱机制，可以自动实现简单类型和封装类型之间的相互转化。

疑难点评

封装类可以理解成对 8 种简单类型的封装，为每一种简单类型提供了一个与之对应的封装类，封装类提供了很多关于简单类型操作的方法。

知识链接

FAQ2.11 int 和 Integer 都可以作为整数类型，那么它们有什么区别？

FAQ3.09 什么是继承？有什么好处？

📖 难度系数：★★★★

📖 问题频率：98%

核心解答

面向对象的主要思想是力求使程序设计更贴近于现实，例如对现实中一类事物共同点进行抽象，然后定义成类。但是在某些情况下，有些新定义的类和某个已有类具有很多相同之处，那么还需要将这些相同点再重新定义一遍吗？

在面向对象的世界中，关于上述问题的答案是不用重新定义。因为面向对象程序设计的一个重要原则就是实现代码的重复利用，减少代码冗余。例如父亲和孩子两者之间会有很多相似的行为和特征，如果我们将父亲和孩子两者独立抽象出来类定义，会发现有很多相同的代码，这就造成了代码冗余。

继承是实现类重复利用的重要手段，通过继承，子类可以具有父类中定义的属性和方法，子类还可以根据需求添加新的属性和方法。因此在定义父亲和孩子类型时，只要通过继承关键字指明两者属于父子继承关系，那么孩子就可以通过继承机制拥有父亲的行为和特征。

Java 中继承是通过使用 extends 关键字实现的，使用格式如下：

```
[<修饰符>] class <子类名> extends <父类名>{  
    [<属性定义>]  
    [<构造方法定义>]  
    [<方法声明>]  
}
```

注意：在 Java 中，定义类时只能使用单继承，即 extends 后面只能跟一个类名，而且子类无法继承父类的构造方法。如果是定义接口，则允许在 extends 后面跟多个接口名。

继承的示例代码如下：

```
public class A extends B{  
    //定义属性和方法  
}
```

在上述代码中，A 类继承自 B 类，因此 B 类中定义的很多方法和属性，A 类也可以使用。

注意：在使用继承时，父类中 `private` 关键字修饰的属性和方法，子类是无法继承的。如果子类和父类在不同包下，`default` 状态的属性和方法也无法继承。

疑难点评

继承在 Java 程序中使用非常频繁，子类继承父类后，在子类中可以使用父类中的方法和属性，提高了代码的重复利用率。

FAQ3.10 使用 `new` 关键字创建对象时，为什么有时候提示找不到无参的构造方法？

📖 难度系数：★★★★

📖 问题频率：80%

核心解答

在 Java 类中，程序员如果不显示定义构造方法，那么编译器会在编译时自动添加一个无参的构造方法。这就是即使没显示定义构造方法，“`new 类名()`”也能使用的原因。但是如果程序员显示定义了构造方法后，编译器在编译时就不会自动添加无参构造方法，因此如果定义了带参数的构造方法后，无参的构造方法需要再显示定义，否则无法使用。

疑难点评

在定义一个类时，如果不显示定义一个构造方法，那么编译器会自动生成一个无参的构造方法；但是如果已显示定义了一个构造方法，那么编译器将不会生成一个无参的构造方法。因此，读者在编写程序时需要注意这个特点，否则容易发生上述错误。

FAQ3.11 抽象类和接口都可以包含抽象方法，那么它们有什么区别？使用时该如何选择？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

`abstract` 用于定义抽象类，抽象类不能实例化，抽象类中既可以包含已实现的方法，也可以包含方法定义，而不具体实现。

`interface` 用于定义接口，接口只包含常量定义和方法定义。接口本身也是一种特殊的抽象类。抽象类和接口的区别主要有以下几个方面。

(1) 定义格式不同

抽象类定义格式如下：

```
public abstract class Demo {  
    abstract void method1();  
    abstract void method2();  
    ...  
}
```

接口定义格式如下：

```
public interface Demo {  
    void method1();  
    void method2();  
    ...  
}
```

在抽象类中，可以包含 `abstract` 修饰的未实现的方法，也可以包含没有 `abstract` 修饰的已实现的方法。在接口中，所有的方法都必须是 `abstract` 修饰的，所有成员变量都默认是 `public static final` 修饰的，且必须赋初值，后续不允许改变其值。

(2) 使用方式的不同

抽象类通过继承方式使用，一个子类只能继承一个抽象类，在子类中需要将抽象类的所有抽象方法实现。抽象类使用格式如下：

```
public class A extends 抽象类 {  
    //实现所有抽象方法  
    ...  
}
```

接口通过实现方式使用，一个类可以实现多个接口，在实现类中需要将接口中所有的方法实现。接口的使用格式如下：

```
public class A extends 接口 1,接口 2 ... {  
    //实现所有接口中声明的方法  
    ...  
}
```

(3) 设计理念不同

在一定程度上抽象类和接口很相似，一般情况下，在实现某一功能时，两者可以相互替换，因此程序员对抽象类和接口的选择比较随意。其实抽象类和接口两者的设计理念有很大的不同，抽象类用于继承，表示“is a”的关系，而接口用于实现，表示“like a”的关系。例如具有报警功能的门，可以继承一个抽象的门类，实现一个报警器的接口。

注意：抽象类与接口除了有一些区别之外，它们之间也存在一些使用关系，比如抽象类允许实现一个或多个接口。

疑难点评

由于抽象类和接口都可以包含只有定义、没有实现的方法，因此读者很容易在概念和使用选取上混淆。抽象类和接口都有其各自的优点，而且使用环境也不相同，读者应重点加以区分。

知识链接

FAQ3.03 什么是抽象类？有什么好处？

FAQ3.04 什么是接口？有什么好处？

FAQ3.12 什么是方法重写？为什么需要方法重写？

📖 难度系数：★★★★

📖 问题频率：95%

核心解答

当一个子类继承一个父类时，它同时继承了父类的属性和方法。子类可以直接使用父类的属性和方法，如果父类的方法不能满足子类的需求，则可以在子类中对父类的方法进行重写（或覆盖）。

在方法重写时，如果子类需要引用父类中原有的方法，可以使用 `super` 关键字。当子类重写父类方法后，在子类对象使用该方法时，会执行子类中重写的方法。

在子类重写父类方法时，需要遵守以下几个重写规则。

- ❑ 重写方法名、参数和返回类型必须与父类方法定义一致。
- ❑ 重写方法的修饰符不能比父类方法严格。例如父类方法时用 `public` 修饰，那么重写方法不能使用 `protected` 或 `private` 等修饰。
- ❑ 重写方法如果有 `throws` 定义，那么重写方法 `throws` 的异常类型可以是父类方法 `throws` 的异常类型及其子类类型。

方法在重写时有很多细节需要注意，否则即使定义了方法，也可能不属于重写，不具有方法重写之后的特征。

疑难点评

方法重写在现实中使用的非常频繁，读者除了学会如何实现方法重写之外，还应该知道为什么需要方法重写，即方法重写的优点。

知识链接

FAQ3.16 在定义类时，何时需要重写 `Object` 类中 `toString()` 方法？

FAQ3.17 在定义类时，何时需要重写 `Object` 类中 `equals()` 方法？

FAQ3.19 如何重写 `hashCode()` 方法？

FAQ3.13 什么是方法重载？为什么需要方法重载？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

在 Java 中，同一个类中不允许有两个相同定义的方法，因为编译器无法将方法调用和特定的方法联系起来。但是允许在同一个类中具有多个方法名相同而参数列表不同的方法，这种形式被称为方法的重载。

注意：方法的返回类型不是区分方法相同的标识，因此在方法重载时，同一个类中不能有两个方法名相同、参数相同、返回类型不同的方法。

方法重载在 JDK 提供的 API 中非常普遍，例如 `System.out.println()` 语句中的 `println()` 方法，该方法被重载了很多次。程序员在使用 `println()` 方法时可以传入任意类型的参数值，感觉是在使用同一个方法，实际上 Java 环境却是根据传入参数类型不同调用了不同的方法，此外很多 Java API 的构造方法也进行了多次重载。

方法重载的主要目的是为了简化 API，降低使用复杂度，为用户提供方便。

在进行方法重载时，需要遵守以下几个重载规则。

- ☐ 方法名相同。
- ☐ 参数列表必须不同。
- ☐ 返回值类型可以不同。
- ☐ 重载方法可以通过 `this` 关键字相互调用。

注意：参数列表不同指的是参数类型不一致，参数类型相同、参数名不同的情况被视为参数列表相同。

疑难点评

方法重载在现实中使用的非常频繁，读者除了学会如何实现方法重载之外，还应该知道为什么需要方法重载，即方法重载的优点。

知识链接

FAQ3.14 构造方法是否可被重写？能否被重载？

FAQ3.14 构造方法是否可以被重写？能否被重载？

📖 难度系数：★★★

📖 问题频率：80%

核心解答

方法重载是让类以统一的方式处理不同类型数据的一种手段。Java 的方法重载就是在类中创建多个方法，同时使这些方法具有相同的名字，但具有的参数和实现不同。在方法调用时通

过传递的参数个数和参数类型来决定具体使用哪个方法。构造方法是允许被重载的，而且构造方法重载是一种很普遍的做法，可以方便使用者灵活地创建对象。重载的构造方法之间相互调用可以使用 `this` 关键字。

当一个子类继承一个父类时，它同时继承了父类的属性和方法。子类可以直接使用父类的属性和方法，如果父类的方法不能满足子类的需求，则可以在子类中对父类的方法进行重写（或覆盖）。但是在继承过程中，构造方法是不能被继承的，因此在子类中无法对父类的构造方法重新定义，构造方法也就不能被重写。如果需要在子类中调用父类的方法，可以通过 `super` 关键字引用。

疑难点评

重载和重写从字面上看非常接近，使人容易混淆。其实方法重载和重写是完全两个不同的概念，其使用环境也大不相同，读者应重点加以区分，该问题主要是考察读者对重写、重载以及构造方法相关的知识的理解程度。

FAQ3.15 `static` 修饰的方法能否在子类中重写？

📖 难度系数：★★★

📖 问题频率：70%

核心解答

具有 `static` 修饰的父类方法是无法在子类中进行重写的，因为 `static` 关键字修饰的方法或属性，表示该属性或方法与类相关。当 JVM 运行 `static` 修饰的方法时，解释器不会检查当前变量指向哪个类型实例，而是直接调用当前变量类型的方法。

示例代码如下：

```
public class A {
    public static void show(){
        System.out.println("A");
    }
}
public class B extends A {
    public static void show(){
        System.out.println("B");
    }
}
```

在上述代码中，定义了 A 和 B 两个类，B 是 A 的子类，在 B 中重新定义了 A 中 `static` 修饰的 `show()` 方法。

当运行以下代码时，输出结果为“A”。

```
public static void main(String[] args){
    A a = new B();
    a.show();
}
```

通过上述代码的输出结果可以看出, a 变量虽然指向的是 B 类型实例, 但是运行时解释器由于发现 show()方法是 static 修饰的, 因此没有检测 a 具体指向哪种类型的实例, 而是直接调用了 a 类型的 show()方法。

疑难点评

一个方法如果有 static 关键字修饰, 那么编译器在执行该方法时, 由类型来决定。方法重写特性必须与对象相关, 因此如果一个方法使用 static 修饰, 将不能实现方法重写。

知识链接

FAQ3.12 什么是方法重写? 为什么需要方法重写?

FAQ3.16 在定义类时,何时需要重写 Object 类中 toString()方法?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

Object 类默认是所有引用类型的父类, 所有引用类型都继承了 Object 中的 toString()方法。toString()方法的功能是返回该对象的字符串表示, 使用该方法可以很方便地将引用类型转换为字符串类型。toString()方法的定义格式如下:

```
public String toString()
```

toString()方法返回的字符串格式由类名、“@”符和对象散列码的无符号十六进制表示组成, 具体格式如下:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

如果不满意 toString()方法返回的字符串格式, 可以在子类中重写该方法。在 JDK 帮助文档中, 也建议所有子类根据需要都重写此方法。

重写 toString()方法的示例代码如下:

```
public class Person
{
    //定义 name 属性
    private String name;
    //定义 age 属性
    private int age;
    //定义 sex 属性
    private String sex;
    //获取 name 属性值
    public String getName()
    {
```

```
        return name;
    }
    //设置 name 属性值
    public void setName(String name)
    {
        this.name = name;
    }
    //获取 age 属性值
    public int getAge()
    {
        return age;
    }
    //设置 age 属性值
    public void setAge(int age)
    {
        this.age = age;
    }
    //获取 sex 属性值
    public String getSex()
    {
        return sex;
    }
    //设置 sex 属性值
    public void setSex(String sex)
    {
        this.sex = sex;
    }
    //覆盖 toString()方法
    public String toString()
    {
        return getClass()+"["+name+"age="+age+",sex="+sex+"]";
    }
}
```

疑难点评

toString()方法的功能是将对象转化为字符串,通过该方法可以将任何类型的对象方便地转化为字符串类型。在程序中,如果需要指定转化字符串的格式,就需要重写 toString()方法。

知识链接

FAQ3.12 什么是方法重写?为什么需要方法重写?

FAQ3.17 在定义类时,何时需要重写 Object 类中 equals() 方法?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

在比较两个对象时可以使用 `==` 和 `equals()`。`==` 用于比较两个对象的引用地址是否相等，而 `equals()` 方法主要用于比较两个对象的内容是否相等。`==` 和 `equals()` 的差别请参见“`==` 和 `equals()` 有什么区别？”的解答。

在 `Object` 中已定义了 `equals()` 方法，但是该方法直接采用 `==` 操作符实现，因此子类如果不重写该方法，那么子类对象在比较时将使用 `Object` 中定义的 `equals()`，其结果与 `==` 操作符的比较结果是一样的。

在 Java API 中，有很多常用类都重写过 `equals()` 方法，例如 `String`、`Integer` 和 `Double` 等，使用 `equals()` 方法可以很方便地比较这些对象的内容是否相等。因此为了方便比较类对象内容是否相等，在定义类时一般都建议重写 `equals()` 方法。

重写 `equals()` 方法的示例代码如下：

```
public class Person
{
    //编号
    private String id;
    //姓名
    private String name;

    //覆盖 equals() 方法
    public boolean equals(Object obj)
    {
        //判断 obj 是否为 null，如果为 null，返回 false
        if (obj == null)
        {
            return false;
        }
        //判断测试的是否为同一个对象，如果是同一个对象返回 true
        if (this == obj)
        {
            return true;
        }
        //判断它们的类型是否相等，如果不相等返回 false
        if (this.getClass() != obj.getClass())
        {
            return false;
        }
        //将参数中传入的对象造型为 Person 类型
        Person c = (Person)obj;
        return id.equals(c.id) && name.equals(c.name);
    }
}
```

注意：在重写 `equals()` 方法的同时，一般都相应地重写 `hashCode()` 方法。

疑难点评

`equals()`方法主要用于比较两个对象是否相等，如果需要自定义比较规则就需要重写该方法。

知识链接

FAQ3.12 什么是方法重写？为什么需要方法重写？

FAQ3.18 为什么在重写 `equals()`方法时，一般都会重写 `HashCode()`方法？

📖 难度系数：★★★★★

📖 问题频率：80%

核心解答

重写 `equals()`方法主要是为了方便比较两个对象内容是否相等。`hashCode()`方法用于返回调用该方法的对象的散列码值，此方法将返回整数形式的散列码值。

一个类如果重写了 `equals()`方法，通常也有必要重写 `hashCode()`方法，目的是为了维护 `hashCode()`方法的常规协定，该协定声明相等对象必须具有相等的散列码。`hashCode` 的常规协定主要有以下几点。

- ❑ 在 Java 应用程序执行期间，在同一对象上多次调用 `hashCode()`方法时，必须一致地返回相同的整数，前提是对象上 `equals()`方法比较中所用的信息没有被修改。从某一应用程序的一次执行到同一应用程序的另一次执行，该整数无需保持一致。
- ❑ 如果根据 `equals(Object)` 方法，两个对象是相等的，那么在两个对象中的每个对象上调用 `hashCode()`方法都必须生成相同的整数结果。
- ❑ 以下情况不是必须的：如果根据 `equals(java.lang.Object)` 方法，两个对象不相等，那么在两个对象中的任一对象上调用 `hashCode()`方法必定会生成不同的整数结果。但是，程序员应该知道，为不相等的对象生成不同整数结果可以提高散列表的性能。

实际上，由 `Object` 类定义的 `hashCode()`方法确实会针对不同的对象返回不同的整数。（通常是通过将该对象的内部地址转换成一个整数来实现的，但是 Java 编程语言不需要这种实现技巧。）

注意：相等的对象必须有相同的散列码，反之散列码相同则不一定对象相等，而且不相等的对象并不一定需要有不同的散列码。

基于散列法的集合需要使用 `hashCode()`方法返回的散列码值存储和管理元素，例如 `Hashtable`、`HashMap` 和 `HashSet` 等，在使用这些集合时，首先会根据元素对象的散列码值确定其存储位置，然后再根据 `equals()`方法结果判断元素对象是否已存在，最后根据判断结果执行不

同处理。因此, 实际应用时如果重写了 equals()方法, 那么 hashCode()方法也会被重写。

疑难点评

在 hashCode 的常规协定中已经定义了对对象 hashCode 码和 equals()方法的关系, 为了维护常规协定的规则, 所以一般在重写 equals()方法时, 也重写 hashCode()方法。

知识链接

FAQ3.17 在定义类时, 何时需要重写 Object 类中 equals()方法?

FAQ3.19 如何重写 hashCode()方法?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

hashCode()方法在重写时通常按照以下设计原则实现。

(1) 把某个非零常数值, 例如 17, 保存在 int 型变量 result 中。

(2) 对于对象中每一个关键域 f (指 equals 方法中考虑的每一个域) 参照以下原则处理。

❑ boolean 型, 计算(f?0:1)。

❑ byte、char 和 short 型, 计算(int)。

❑ long 型, 计算(int)(f^(f>>32))。

❑ float 型, 计算 Float.floatToIntBits(afloat)。

❑ double 型, 计算 Double.doubleToLongBits(adouble)得到一个 long, 再执行 long 型的处理。

❑ 对象引用, 递归调用它的 hashCode()方法。

❑ 数组域, 对其中每个元素调用它的 hashCode()方法。

(3) 将上面计算得到的散列码保存到 int 型变量 c, 然后执行 $result = 37 \times result + c$ 。

(4) 返回 result。

重写 hashCode()方法的示例代码如下:

```
import java.util.Arrays;

public class Unit {
    private short ashort;
    private char achar;
    private byte abyte;
    private boolean abool;
    private long along;
    private float afloat;
```



```
private double adouble;
private Unit aObject;
private int[] ints;
private Unit[] units;

//重写 equals()方法
public boolean equals(Object o) {
    if (o == null)
        return false;
    if (this.getClass() != obj.getClass())
        return false;
    Unit unit = (Unit) o;
    return unit.ashort == ashort
        && unit.achar == achar
        && unit.abyte == abyte
        && unit.abool == abool
        && unit.along == along
        && Float.floatToIntBits(unit.afloat) == Float
            .floatToIntBits(afloat)
        && Double.doubleToLongBits(unit.adouble) == Double
            .doubleToLongBits(adouble)
        && unit.aObject.equals(aObject)
    && equalsInts(unit.ints)
    && equalsUnits(unit.units);
}

private boolean equalsInts(int[] aints) {
    return Arrays.equals(ints, aints);
}

private boolean equalsUnits(Unit[] aUnits) {
    return Arrays.equals(units, aUnits);
}

//重写 hashCode()方法
public int hashCode() {
    int result = 17;
    result = 37 * result + (int) ashort;
    result = 37 * result + (int) achar;
    result = 37 * result + (int) abyte;
    result = 37 * result + (abool ? 0 : 1);
    result = 37 * result + (int) (along ^ (along >>> 32));
    result = 37 * result + Float.floatToIntBits(afloat);
    long tolong = Double.doubleToLongBits(adouble);
    result = 37 * result + (int) (tolong ^ (tolong >>> 32));
    result = 37 * result + aObject.hashCode();
}
```



```
        result = 37 * result + intsHashCode(ints);
        result = 37 * result + unitsHashCode(units);
        return result;
    }

    private int intsHashCode(int[] aints) {
        int result = 17;
        for (int i = 0; i < aints.length; i++)
            result = 37 * result + aints[i];
        return result;
    }

    private int unitsHashCode(Unit[] aUnits) {
        int result = 17;
        for (int i = 0; i < aUnits.length; i++)
            result = 37 * result + aUnits[i].hashCode();
        return result;
    }
}
```

疑难点评

为了维护 hashCode 常规协定, 在重写 hashCode() 方法时需要遵循一定的设计原则, 具体如上所述。

知识链接

FAQ3.17 在定义类时, 何时需要重写 Object 类中 equals() 方法?

FAQ3.18 为什么在重写 equals() 方法时, 一般都会重写 hashCode() 方法?

FAQ3.20 Java 中动态绑定是什么意思?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

将一个方法调用同一个方法主体连接到一起称为“绑定”(Binding)。如果在程序运行之前执行绑定, 由编译器决定方法调用的程序, 称为“早期绑定”或“静态绑定”。如果绑定过程在程序运行期间进行, 以对象的类型为基础, 则称为“后期绑定”或“动态绑定”。

如果一种语言实现了后期绑定, 同时必须提供一些机制, 可以在运行期间判断对象的实际类型, 并分别调用适当的方法, 即编译器此时依然不知道对象的类型, 但方法调用机制能够自

己去调查,找到正确的方法主体。Java 方法的执行主要采用动态绑定技术,在程序运行时,虚拟机将调用对象实际类型所限定的方法。

Java 方法在调用过程中主要经历了以下过程。

- (1) 编译器查看对象变量的声明类型和方法名,通过声明类型找到方法列表。
- (2) 编译器查看调用方法时提供的参数类型。
- (3) 如果方法由 `private`、`static` 和 `final` 修饰或者是构造器,编译器就可以确定调用哪一种方法,即采取静态绑定技术。如果不是上述情况,就使用动态绑定技术,执行后续过程。
- (4) 虚拟机提取对象的实际类型的方法表。
- (5) 虚拟机搜索方法签名。
- (6) 调用方法。

疑难点评

Java 方法的执行主要采用动态绑定技术,在程序运行时,虚拟机将调用对象实际类型所限定的方法。

FAQ3.21 Java 中是如何实现多态的? 实现机制是什么?

📖 难度系数: ★★★★★

📖 问题频率: 98%

核心解答

Java 是借助方法的重写和重载实现多态性的。重写是父类与子类之间多态性的一种表现,重载是一个类中多态性的一种表现。

在 JVM 中,对象的引用其实是指向一个句柄 (handle) 的指针,该句柄包含一对指针,一个指针指向一张表格,实际上该表格也有两个指针 (一个指针指向包含了对象的方法表,另外一个指向对象类型,表明该对象所属的类型); 另一个指针指向一块从 Java 堆中分配的内存空间,该空间用于存储对象数据。

Java 通过将子类对象引用赋值给父类对象的引用变量来实现动态方法调用,这种实现方式遵循的原则有以下几点。

- ❑ 如果 `a` 是类 `A` 的一个引用,那么 `a` 可以指向类 `A` 的实例,或者是类 `A` 的子类实例。
- ❑ 如果 `a` 是接口 `A` 的一个引用,那么 `a` 必须指向实现了接口 `A` 的一个类的实例。
- ❑ 当父类对象引用变量引用子类对象时,被引用对象的类型决定了调用的方法,而不是引用变量的类型决定,但是这个被调用的方法必须是在父类中定义过的,即被子类重写的方法。

注意: Java 中除了 `static` 和 `final` 修饰的方法外,其他所有的方法都是在运行时绑定 (即动

态绑定)，是指 JVM 在运行时根据对象的实际类型进行方法的调用。

疑难点评

上面介绍了多态的实现方法和实现机制，这些知识有助于读者对多态更深一步的理解。

知识链接

FAQ3.05 什么是多态？有什么好处？

FAQ3.22 创建类的对象时，类中各成员的执行顺序是什么样的？

📖 难度系数：★★★★

📖 问题频率：85%

核心解答

属性、方法、构造方法和自由块都是类中的成员，在创建对象时，各成员的执行顺序如下。

- (1) 父类静态成员和静态初始化块，按在代码中出现的顺序依次执行。
- (2) 子类静态成员和静态初始化块，按在代码中出现的顺序依次执行。
- (3) 父类实例成员和实例初始化块，按在代码中出现的顺序依次执行。
- (4) 执行父类构造方法。
- (5) 子类实例成员和实例初始化块，按在代码中出现的顺序依次执行。
- (6) 执行子类构造方法。

示例代码如下：

```
public class Test {  
  
    public static void main(String[] args) {  
        Son s = new Son();  
    }  
}  
//父类 Parent  
class Parent {  
    {  
        System.out.println("--Parent 中的初始化块--");  
    }  
    static {  
        System.out.println("--Parent 中的 static 初始化块--");  
    }  
  
    public Parent() {  
        System.out.println("--Parent 中的构造方法--");  
    }  
}
```

```
    }  
}  
//子类 Son  
class Son extends Parent {  
    {  
        System.out.println("--Son 中的初始化块--");  
    }  
    static {  
        System.out.println("--Son 中的 static 初始化块--");  
    }  
  
    public Son() {  
        System.out.println("--Son 中的构造方法--");  
    }  
}
```

上述代码的执行结果如下:

```
--Parent 中的 static 初始化块--  
--Son 中的 static 初始化块--  
--Parent 中的初始化块--  
--Parent 中的构造方法--  
--Son 中的初始化块--  
--Son 中的构造方法--
```

疑难点评

一个类可以定义成员变量、初始化块、方法和构造方法等,还可以使用继承。当创建对象时,这些成员的代码有其特定的执行顺序。了解对象初始化顺序后,可以根据需要在不同的成员中添加相应的初始化代码,读者可以查看知识连接部分的其他问题。另外由于 `static` 修饰的初始化块具有了 `static` 特性,因此与非静态的初始化块相比,它们在执行顺序和执行次数上都有较大区别,读者可以参考 FAQ3.24 “静态初始化块与非静态初始化块有什么区别?”。

知识链接

FAQ3.01 什么是类、对象、属性和方法?

FAQ3.23 什么是初始化块? 有什么作用?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

所谓“块”,就是用“{”和“}”所包含的代码段,它们在逻辑上常常是一个整体。在 Java 程序中,类的定义和方法的定义都必须放在一个“块”中,而条件语句、循环语句中的代码通

常也放在一个“块”中。

除了上述“块”的使用情况外，还有一种特殊的块，它独立于方法体和构造方法之外，可以看成是一个没有参数、没有返回值、没有方法名的特殊的方法，这种块一般称其为“初始化块”、“自由块”或“游离块”。

初始化块主要用于对象的初始化操作，在创建对象时调用，可以用于完成初始化属性值、加载其他类等功能。初始化块和构造方法的功能相似，都可以在创建对象时完成一些初始化操作，一般情况下，构造方法初始化和初始化块初始化可以通用。

构造方法在初始化时可以通过参数接收外界传入的值，而初始化块则不能。初始化块的执行顺序在构造方法之前，如果构造方法多次重载，此时可以考虑将构造方法中共通的代码提到初始化块中定义。

示例代码如下：

```
public class Test {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        A a = new A();  
        A a1 = new A("");  
    }  
  
}  
  
class A {  
    {  
        System.out.println("--初始化块--");  
    }  
  
    public A(){  
        System.out.println("--无参的构造方法--");  
    }  
  
    public A(String name){  
        System.out.println("--带参数的构造方法--");  
    }  
}
```

上述代码输出结果如下：

```
--初始化块--  
--无参的构造方法--  
--初始化块--  
--带参数的构造方法--
```

通过上述示例可以看出，每次创建 A 类对象时，初始化块都会执行一次，因此可以将共同的初始化代码放在初始化块中，而特殊的初始化代码可以放在不同的构造方法中。

疑难点评

初始化块具有初始化功能，在创建对象时先会执行初始化块中的代码，然后再执行相应的

构造方法。

知识链接

FAQ3.24 静态初始化块与非静态初始化块有什么区别？

FAQ3.24 静态初始化块与非静态初始化块有什么区别？

📖 难度系数：★★★★

📖 问题频率：86%

核心解答

非静态初始化块主要用于对象的初始化操作，在每次创建对象时都要调用一次，其执行顺序在构造方法之前。

如果初始化块前有 `static` 关键字修饰，那么该初始化块称为静态初始化块。由于非静态成员不能在静态方法中使用，同样也不能在静态初始化块中，因此，静态初始化块主要用于初始化静态变量和静态方法。静态初始化块只调用一次，是在类被第一次加载到内存时，并非一定要创建对象才执行。

静态初始化块的执行顺序是在非静态初始化块之前，因此比非静态初始化块执行的要早。

示例代码如下：

```
public class Test{

    public static void main(String[] args) {
        T a = new T();
        T a1 = new T();
    }

}

class T {
    {
        System.out.println("--非静态初始化块--");
    }

    static {
        System.out.println("--静态初始化块--");
    }

    public T() {
        System.out.println("--无参的构造方法--");
    }

}
```

上述代码执行结果如下：

```
--静态初始化块--
--非静态初始化块--
--无参的构造方法--
--非静态初始化块--
--无参的构造方法--
```

通过上述示例可以看出静态初始化块仅在一次创建对象时执行了, 第二次就没有执行。

疑难点评

静态初始化块与非静态初始化块有一定的区别, 静态的初始化块比非静态初始化块执行要早, 而且静态初始化块只执行一次, 非静态的初始化块可执行多次。静态初始化块的执行时机需要注意, 它是在类加载器第一次加载该类时调用, 不一定非要创建对象才触发, 如果使用“类名.静态方法(或静态属性)”也会执行静态方法。

知识链接

FAQ3.23 什么是初始化块? 有什么作用?

FAQ3.25 如何调用内部类中的方法?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

内部类是指在一个外部类的内部再定义一个类, 内部类作为外部类的一个成员, 并且依附于外部类而存在的。可以在一个外部类中定义多个内部类, 这些内部类在逻辑上是一个整体, 与外部类之间是从属关系。在内部类中可以访问外部类定义的私有成员。

内部类的示例代码如下:

```
public class Outer {
    private int size;
    /* 定义一个内部类, 名为 "Inner" */
    public class Inner {
        public void doStuff() {
            //内部类可以访问外部类的私有属性
            size++;
        }
    }
    //方法定义
    public void testInner() {
        //调用内部类中的方法
        Inner i = new Inner();
        i.doStuff();
    }
}
```


在上述代码中，在 `Outer` 类中定义了属性 `size`、内部类 `Inner` 和方法 `testInner()`，在方法中调用内部类，直接通过“对象.方法”的形式即可调用。如果在其他类中需要调用 `Inner` 类中的方法，可以使用以下格式：

```
//方法一
Outer.Inner in=new Outer().new Inner();
in.doStuff();
//方法二
Outer o = new Outer();
Outer.Inner in = o.new Inner();
in.doStuff();
```

内部类可以通过 `static` 修饰，称为静态内部类。在静态内部类中调用外部类成员，成员也要求用 `static` 修饰。如果内部类用 `static` 修饰，可以使用以下格式调用其方法。

```
Outer.Inner in=new Outer.Inner();
in.doStuff();
```

疑难点评

上面介绍了内部类的一些具体使用，例如内部类的定义以及内部类方法的调用问题。

知识链接

FAQ3.06 什么是内部类？有什么好处？

FAQ3.26 当内部类和外部类的成员名称相同时，如何在内部类中调用外部类的成员？

FAQ3.26 当内部类和外部类的成员名称相同时，如何在内部类中调用外部类的成员？

📖 难度系数：★★★

📖 问题频率：85%

核心解答

当内部类和外部类成员名称相同时，内部类中调用外部类的成员需要显示指明，否则默认为内部类中的成员。示例代码如下：

```
public class A {
    //外部类成员 i
    private int i = 100;
    //内部类 B
    class B {
        //内部类成员 i
        private int i = 10;
        public void show(){
            //打印内部类成员 i
            System.out.println(i);
        }
    }
}
```



```
//打印外部类成员 i
System.out.println(A.this.i);
}
}
```

在上述代码中, 首先通过“外部类.this”获取外部类对象, 然后再引用 i 成员, 代码执行结果如下:

```
10
100
```

疑难点评

当内部类和外部类的成员名称相同时, 可以通过“外部类.this.成员”的形式引用外部类的成员。

知识链接

FAQ3.06 什么是内部类? 有什么好处?

FAQ3.25 如何调用内部类中的方法?

FAQ3.27 匿名内部类如何访问外部方法的局部变量或参数?

📖 难度系数: ★★★

📖 问题频率: 70%

核心解答

匿名内部类一般定义在一个方法的内部, 如果要访问该方法的参数或者方法中定义的变量, 则这些参数和变量必须使用 final 修饰。

在定义变量时, 如果使用 final 关键字修饰, 那么该变量值一旦被初始化则不可以改变。final 修饰的变量既可以在定义时给其赋值, 又可以在构造方法中赋值。因为匿名内部类没有构造函数, 所以使用 final 变量必须在定义时初始化。

虽然匿名内部类定义在方法的内部, 但在编译时内部类与外部类中的方法属于同一个级别, 外部类中方法的变量或参数只是方法的局部变量, 这些变量或参数的作用域只在当前方法内部有效。但是如果这些变量使用 final 修饰, 内部类就可以保存方法变量的备份, 即使方法销毁也能保证内部类在访问时不会出现访问不到的错误。

示例代码如下:

```
public class Test {
    private int m;

    public void go(int x, final int y) {
        int a;
```

```
final int b = 10;
//定义匿名内部类
new Thread(){
    public void run() {
        //访问外部类的成员变量
        System.out.println("m=" + m);
        // System.out.println("x="+x); //不合法
        //访问 final 修饰的方法参数
        System.out.println("y=" + y);
        // System.out.println("a="+a); //不合法
        //访问 final 修饰的方法变量
        System.out.println("b=" + b);
    }
}.start();
}
```

疑难点评

匿名内部类在访问外部方法的局部变量或参数时，局部变量和参数需要使用 `final` 修饰，否则编译时就会发生错误。

知识链接

FAQ3.07 什么是匿名内部类？如何使用？

FAQ3.28 Java 异常处理机制是什么样的？

📖 难度系数：★★★★★

📖 问题频率：98%

核心解答

例外是在程序运行过程中发生的异常事件，例如除 0 溢出、数组越界和文件找不到等，这些事件的发生将阻止程序的正常运行。为了加强程序的健壮性，在程序设计时必须考虑到可能发生的异常事件并做出相应的处理。

Java 通过面向对象的方法来处理异常事件。一个方法的运行过程中，如果发生了例外，则该方法将生成一个相应类型的异常对象，并将其交给运行时系统，这一过程被称为抛出。运行时系统在接收到异常后，会寻找相应的代码来处理，这一过程被称为捕获。如果运行时系统未找到异常的处理代码，将终止程序运行；若找到异常处理部分则执行处理代码，然后程序继续运行。

Java 程序在运行时遇到的例外情况大致可以分为两类，即错误 (Error) 和异常 (Exception)。

Error 是指 Java 虚拟机内部发生错误, 由虚拟机生成并抛出, 例如资源耗尽等情况, 通常 Java 程序不对这类例外进行处理。Exception 是指因编程错误或偶然的外在因素导致的一般性问题, 例如对负数开平方根、访问的文件找不到和网络连接中断等, 它包含各种不同的子类分别对应于不同类型的例外, 这类例外需要程序员进行处理。

Java 异常类的层次结构如图 3-1 所示。

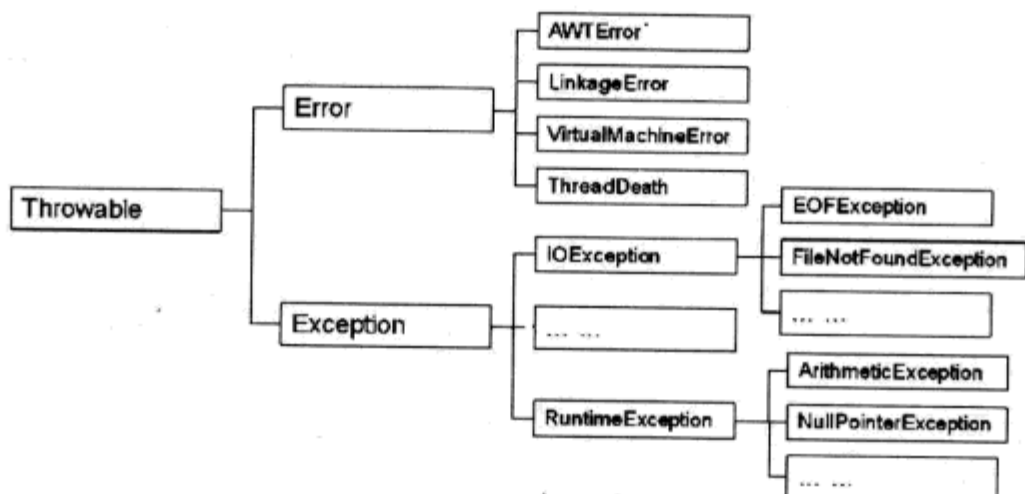


图 3-1 Error 和 Exception 类型层次结构图

如图 3-1 所示, 类 Throwable 位于这一类层次的最顶层, 只有它的子类才可以做为一个例外被抛出, 而 Error 和 Exception 都是 Throwable 的子类。

疑难点评

Java 异常可以分为 Error 和 Exception 两大类型, Error 类型的异常无法使用程序处理, 一般所谓的异常处理是指 Exception 类型异常的处理。Exception 类又可以分为运行时异常 (RuntimeException) 和非运行时异常 (如 IOException 等)。其中类 RuntimeException 代表运行时由 Java 虚拟机生成的异常, 例如算术运算异常 ArithmeticException、数组越界异常 ArrayIndexOutOfBoundsException 等; 非运行时异常, 例如输入输出异常 IOException 等。Java 编译器要求 Java 程序必须捕获或声明所有的非运行时异常, 但对运行时异常可以不做处理, 因此, 在编码时非运行时异常如果不处理, 编译时会出错。

知识链接

FAQ3.29 常见的 Runtime Exception 异常有哪些?

FAQ3.30 Java 中异常处理的方式有哪些?

FAQ3.29 常见的 RuntimeException 异常有哪些?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

`RuntimeException` 是 `Exception` 类的子类, `Exception` 类对象是 Java 程序处理或抛弃的对象, 它有各种不同的子类分别对应于不同类型的例外。其中类 `RuntimeException` 代表运行时由 Java 虚拟机生成的例外, 如算术运算异常 `ArithmeticException`(例如除以 0)、数组索引越界异常 `ArrayIndexOutOfBoundsException` 等; 其他则为非运行时异常, 例如输入输出异常 `IOException` 等。

注意: Java 编译器要求 Java 程序必须捕获或声明所有的非运行时异常, 但对运行时异常可以不做处理。虽然编译器对运行时异常不强制要求处理, 但实际开发中为了程序的健壮性, 还是有必要处理的。

在开发过程中常见的 `RuntimeException` 类型的异常主要有以下几种。

- ☐ `ArithmeticException`: 数学计算异常。
- ☐ `NullPointerException`: 空指针异常。
- ☐ `NegativeArraySizeException`: 负数组长度异常。
- ☐ `ArrayOutOfBoundsException`: 数组索引越界异常。
- ☐ `ClassNotFoundException`: 类文件未找到异常。
- ☐ `ClassCastException`: 类型强制转换异常。
- ☐ `SecurityException`: 违背安全原则异常。

其他非 `RuntimeException` 类型的常见异常主要有以下几种。

- ☐ `NoSuchMethodException`: 方法未找到异常。
- ☐ `IOException`: 输入输出异常。
- ☐ `EOFException`: 文件已结束异常。
- ☐ `FileNotFoundException`: 文件未找到异常。
- ☐ `NumberFormatException`: 字符串转换为数字异常。
- ☐ `SQLException`: 操作数据库异常。

疑难点评

上面列出了属于 `RuntimeException` 类型的所有常见异常, 这些异常在开发时出现的非常频繁, 而且每一种异常产生的原因都是相同的, 因此解决方法都大同小异。

知识链接

FAQ3.28 Java 异常处理机制是什么样的?

FAQ3.30 Java 中异常处理的方式有哪些?

📖 难度系数: ★★★★★

📖 问题频率: 98%

核心解答

在 Java 中处理异常主要有两种方式, 具体如下。

(1) 使用 try-catch-finally

try-catch-finally 语句的语法格式如下:

```
try{
    //可能会抛出特定异常的代码段
}catch(MyExceptionType myException){
    //如果 myException 被抛出, 则执行这段代码
}catch(Exception otherException){
    //如果另外的异常 otherException 被抛出, 则执行这段代码
}finally{
    //无条件执行的语句
}
```

在上述语句块中, try 和 catch 语句是必须的, 而且 catch 语句可以定义多个, finally 语句不是必须的。

(2) 使用 throws

如果在定义一个方法的时候, 并不能确定如何处理其中可能出现的异常, 可以不用在方法中使用 try-catch-finally 结构对异常进行处理, 而是将可能发生的异常抛给该方法的调用者来处理。实现方式就是在定义方法时使用 throws 关键字指定抛出的异常类型, 使用格式如下:

```
public void f1() throws IOException, RuntimeException{
    //业务逻辑代码, 可能会产生 IOException 和 RuntimeException 异常
}
```

上述代码在定义 f1() 方法时, 通过 throws 关键字指定了该方法可能抛出的异常类型, throws 后面可以写一个或多个异常类型, 多个异常类型之间用 “,” 分隔。

(3) 异常处理方式的选择

Java 中提供了 try-catch-finally 和 throws 两种处理异常的方式, 大部分开发者对这两种处理方式的选择比较模糊, 不知道何时使用 try-catch-finally, 何时使用 throws。其实 try-catch 和 throws 两种处理方式有很大的区别, try-catch 方式允许开发者在程序发生异常时, 对异常部分进行补救或处理, 例如尝试通过另一种替代方案或将异常信息保存等, 而 throws 方式仅仅是在发生异常后将异常抛给上一级, 让调用该方法的方法处理异常。

try-catch 和 throws 可以使用以下原则进行选择。

- ☐ 如果方法中前面代码发生异常后, 后面代码需要继续执行, 可以选用 try-catch。
- ☐ 如果方法中的代码发生异常后, 有替代方案或需要特殊处理, 可以选用 try-catch。
- ☐ 如果方法中的代码发生异常后, 无法处理, 后面代码无法正常工作, 可以选用 throws。
- ☐ 根据具体情况而定, 有些厂商规定异常在某一特定层使用 try-catch 处理, 其他层都统一使用 throws 抛出。

注意: 有些异常是由于编程疏忽造成的, 例如 NullPointerException, 如果在编写程序时加

入空值判断，这类异常是可以避免的。在可以预知和可以预防的情况下，建议采用逻辑代码处理，不要过度依赖 try-catch 和 throws，因为过多使用异常处理会降低程序的性能。

疑难点评

异常的处理方法主要有两种，一种是使用 try-catch，另一种是使用 throws。在开发时，一般会采取统一的方式来处理异常，避免一会用 try-catch，一会用 throws。try-catch 和 throws 异常处理方式各有其特点，在使用时，需要根据具体情况选择合适的方式，不能滥用。

知识链接

FAQ3.31 try-catch-finally 语句块各部分的执行顺序如何？

FAQ3.31 try-catch-finally 语句块各部分的执行顺序如何？

📖 难度系数：★★★

📖 问题频率：85%

核心解答

try-catch-finally 语句的语法格式如下：

```
try{
    //可能会抛出特定异常的代码段
}catch(MyExceptionType myException){
    //如果 myException 被抛出，则执行这段代码
}catch(Exception otherException){
    //如果另外的异常 otherException 被抛出，则执行这段代码
}finally{
    //无条件执行的语句
}
```

在上述语句块中，各部分的执行顺序主要有以下几种。

- ☐ 执行 try 语句正常，执行 finally 语句正常，退出语句块执行后续程序。
- ☐ 执行 try 语句正常，执行 finally 语句异常，终止程序。
- ☐ 执行 try 语句异常，执行 catch 语句正常，执行 finally 语句，退出语句块执行后续程序。
- ☐ 执行 try 语句异常，执行 catch 语句异常，执行 finally 语句，终止程序。

注意：finally 部分的语句是必须要执行的，如果在 finally 部分添加 return 语句，那么会严重影响程序的流程。

示例代码如下：

```
public class Test1 {
    public static void main(String[] args){
        System.out.println(getLength("tom"));
    }
}
```

```
        System.out.println(getLength(null));
    }

    public static int getLength(String s){
        try{
            int l = s.length();
            return l;
        }catch(Exception e){
            return 0;
        }finally{
            return -1;
        }
    }
}
```

上述代码的执行结果如下：

```
-1
-1
```

通过上述结果可以看出，无论 try 语句是否发生异常，getLength()方法的返回结果都是 finally 中 return 的值。

疑难点评

try-catch-finally 语句各部分在执行时有特定的顺序，为了保障程序流程的正确性，在使用 try-catch-finally 语句之前必须得清晰的了解该语句各部分的执行流程。

知识链接

FAQ3.30 Java 中异常处理的方式有哪些？

FAQ3.32 为什么使用自定义异常？自定义异常如何使用？

📖 难度系数：★★★★★

📖 问题频率：88%

核心解答

(1) 为什么使用自定义异常

Java 虽然提供了丰富的异常处理类，但是在项目中还会经常使用自定义异常，其主要原因是 Java 提供的异常类在某些情况下还是不能满足实际需求。例如以下几种情况就需要通过自定义异常解决。

- ❑ 系统中有些错误是符合 Java 语法，但不符合业务逻辑的，例如年龄大于 150、用户不存在等，如果这些错误需要用异常的形式处理，就需要使用自定义异常，因为 Java 提供的异常类是无法详细描述业务错误的。
- ❑ 在分层的软件结构中，通常是在表现层统一对系统其他层次的异常进行捕获处理，如

果表现层需要针对系统各层不同的异常执行不同处理,就需要使用自定义异常。因为 Java 提供的异常类在系统每个层次都有可能产生,使得在异常处理时不能区分处理。

(2) 如何自定义异常

自定义异常是通过程序代码显示抛出的,因此自定义异常类需要具有可被抛出的特征。在自定义异常类时,通常继承 `Exception` 或者它的子类实现,也可以从 `Throwable` 继承。

自定义异常类的定义格式如下:

```
public class 类名称 extends Exception{  
}
```

在实现自定义异常类时,为了方便使用,会给该异常类设计两个构造方法,一个无参数,另一个带 `String` 类型参数。实例代码如下:

```
public class UserNotFoundException extends Exception{  
    //定义无参构造方法  
    public UserNotFoundException(){  
        super();  
    }  
    //定义带参数的构造方法  
    public UserNotFoundException(String msg){  
        super(msg);  
    }  
}
```

(3) 如何使用自定义异常

自定义异常在使用时主要有以下两种格式。

❑ 违反业务逻辑

```
public void f1() throws UserNotFoundException{  
    if(条件不满足){  
        throw new UserNotFoundException("用户不存在");  
    }  
}
```

❑ try-catch 捕获抛出

```
public void f1() throws UserNotFoundException{  
    try{  
        //业务代码  
    }catch(Exception e){  
        throw new UserNotFoundException("用户不存在");  
    }  
}
```

疑难点评

虽然 Java 在底层类库中已经定义了很多异常类型,但是有些情况下不能满足实际需要,此时就需要使用自定义异常类型。

第4章

Java 流和文件操作

本章重点介绍 Java I/O 流和文件操作相关的疑难问题。Java 提供了类型丰富的 I/O 流，用于实现 Java 程序与外界的输入和输出操作。I/O 的内容很庞大，涉及的领域也很广泛，例如标准输入输出、文件操作、网络上的数据流、字符串流、对象流和 ZIP 文件流等。本章内容涉及与 I/O 操作相关的各个领域，主要内容有各种类型流的使用、NIO 技术、各种类型文件的处理、文件的压缩和解压缩、文件的加密和解密、如何处理读写中文字符时的乱码问题以及流操作时其他常见异常的解决等。通过本章的学习，读者可以熟练地掌握 I/O 流相关功能的实现方法。

FAQ4.01 如何获取文件的属性信息？

📖 难度系数：★★

📖 问题频率：70%

核心解答

在 Java 中，提供了对文件及目录进行操作的 File 类，使用该类的方法可以很方便地获取文件相关的信息，具体如下。

❑ boolean exists()

测试此抽象路径名表示的文件或目录是否存在。

❑ String getName()

返回由此抽象路径名表示的文件或目录的名称。

❑ String getParent()

返回此抽象路径名的父路径名的路径名字符串，如果此路径名没有指定父目录，则返回 null。

❑ String getAbsolutePath()

返回抽象路径名的绝对路径名字符串。

❑ String getPath()

将此抽象路径名转换为一个路径名字符串。

❑ boolean isAbsolute()

测试此抽象路径名是否为绝对路径名。

❑ boolean isDirectory()

测试此抽象路径名表示的文件是否是一个目录。

❑ boolean isFile()

测试此抽象路径名表示的文件是否是一个标准文件。

❑ boolean isHidden()

测试此抽象路径名指定的文件是否是一个隐藏文件。

❑ long lastModified()

返回此抽象路径名表示的文件最后一次被修改的时间。

❑ long length()

返回由此抽象路径名表示的文件的长度，以字节为单位。

❑ boolean canRead()

测试应用程序是否可以读取此抽象路径名表示的文件。

❑ boolean canWrite()

测试应用程序是否可以修改此抽象路径名表示的文件。

上述方法的使用示例如下：

```
public static void main(String[] args) {  
    File file = new File("F:\\wy.html");  
    System.out.println("*****文件属性测试*****");  
    System.out.println("是否存在: " + file.exists());  
    System.out.println("文件名: " + file.getName());  
    System.out.println("上级目录: " + file.getParent());  
    System.out.println("是否可读: " + file.canRead());  
    System.out.println("是否可写: " + file.canWrite());  
    System.out.println("绝对路径: " + file.getAbsolutePath());  
    System.out.println("相对路径: " + file.getPath());  
    System.out.println("是否为绝对路径: " + file.isAbsolute());  
    System.out.println("是否为目录: " + file.isDirectory());  
    System.out.println("是否为文件: " + file.isFile());  
    System.out.println("是否为隐藏文件: " + file.isHidden());  
    System.out.println("最后修改时间: " + new Date(file.lastModified()));  
    System.out.println("文件长度: " + file.length());  
}
```

上述代码的执行结果如下：

```
*****文件属性测试*****  
是否存在: true  
文件名: wy.html  
上级目录: F:\  
是否可读: true  
是否可写: true  
绝对路径: F:\wy.html  
相对路径: F:\wy.html  
是否为绝对路径: true  
是否为目录: false
```

是否为文件: true
是否为隐藏文件: false
最后修改时间: Wed Nov 12 10:20:26 CST 2008
文件长度: 19003

疑难点评

通过 File 类的方法可以轻松获取很多文件属性, 例如文件大小、修改时间、是否可读等。但也有些属性无法获取, 例如创建时间。

知识链接

FAQ4.03 如何实现文件的创建、删除和移动?

FAQ4.02 如何判断文件是否为空?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

在 File 类中, 包含一个 length() 方法, 主要功能是返回文件的大小。方法定义如下:

```
public long length()
```

如果文件不存在或文件为空时, length() 方法返回 0, 因此可以借助 length() 方法来判断文件是否为空。

示例代码如下:

```
File file = new File("F:\\b.txt");  
if(file.exists() && file.length() == 0){  
    System.out.println("文件为空!");  
}
```

疑难点评

在 File 类中, 没有提供判断文件是否为空的方法, 但可以借助 length() 方法的返回值进行判断。

知识链接

FAQ4.03 如何实现文件的创建、删除和移动?

FAQ4.03 如何实现文件的创建、删除和移动?

📖 难度系数: ★★★★★

📖 问题频率: 88%

核心解答

□ 文件创建

使用 `File` 类中的 `createNewFile()` 和 `createTempFile()` 方法, 可以实现创建文件的功能。

`createNewFile()` 方法的定义如下:

```
public boolean createNewFile() throws IOException
```

使用 `createNewFile()` 方法时, 如果指定的文件不存在并成功地创建文件, 则返回 `true`; 如果指定的文件已经存在, 则返回 `false`。

`createTempFile()` 方法主要用于创建临时文件, 该方法重载了两次, 具体定义如下:

```
public static File createTempFile(String prefix,String suffix) throws IOException
```

```
public static File createTempFile(String prefix,String suffix,File directory) throws IOException
```

参数 `prefix` 用于指定生成文件名的前缀字符串, 必须至少是 3 个字符长。

参数 `suffix` 用于指定生成文件名的后缀字符串, 可能是 `null`, 在这种情况下, 将使用后缀 `".tmp"`。

参数 `directory` 用于指定将创建的文件所在的目录, 如果使用默认临时文件目录, 则该参数为 `null`。

`createTempFile(prefix, suffix)` 方法用于在默认临时文件目录中创建一个空文件, 使用给定前缀、系统生成的随机数和后缀生成其名称。调用此方法等同于调用 `createTempFile(prefix, suffix, null)`。

示例代码如下:

```
File file = new File("F:\\b.txt");
//如果 F:\\b.txt 不存在, 新建一个文件
if(!file.exists()){
    file.createNewFile();
}
//在 F 盘下创建一个文件, 文件名为 tmp 随机数.log
try {
    File.createTempFile("tmp",".log",new File("F:\\"));
} catch (IOException e) {
    e.printStackTrace();
}
```

注意: 在创建文件时, 必须保证父目录正确存在, 否则文件创建异常。

□ 文件删除

使用 `File` 类中的 `delete()` 方法, 可以实现删除文件的功能。`delete()` 方法的定义如下:

```
public boolean delete()
```

`delete()` 方法用于删除此抽象路径名表示的文件或目录。如果此路径名表示一个目录, 则此目录必须为空才能删除。当且仅当成功删除文件或目录时, 返回 `true`; 否则返回 `false`。

示例代码如下:

```
File file = new File("F:\\b.txt");
//如果 F:\\b.txt 存在, 删除该文件
```



```
if(file.exists()){  
    file.delete();  
}
```

在 File 类中还提供了一个 `deleteOnExit()` 方法, 该方法可以实现在 Java 虚拟机退出的时候请求删除对象所指定的文件的功能。

□ 文件移动

使用 File 类中的 `renameTo()` 方法, 可以实现文件的重命名和移动功能。`renameTo()` 方法的定义如下:

```
public boolean renameTo(File dest)
```

`renameTo()` 方法在对文件重命名时可以修改路径和文件名, 当且仅当重命名成功时, 返回 `true`; 否则返回 `false`。

示例代码如下:

```
//源文件  
File file = new File("F:\\b.txt");  
//要移动的目的地  
File dest = new File("D:\\a.txt");  
//将 F 盘下的 b.txt 移动到 D 盘并重命名为 a.txt  
file.renameTo(dest);
```

疑难点评

文件的创建、删除和移动属于对文件的最基本操作, 在实现时应该先使用 File 类的 `exists()` 方法检测文件或路径是否存在。例如在创建文件操作中, 应首先检测文件是否存在, 如果不存在再创建一个新文件。需要注意的是, 如果文件所在路径不正确, 创建文件时也会发生异常。

知识链接

FAQ4.04 如何创建和删除文件夹?

FAQ4.04 如何创建和删除文件夹?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

□ 创建文件夹

使用 File 类中的 `mkdir()` 和 `mkdirs()` 方法, 可以实现创建文件夹的功能。`mkdir()` 方法用于创建当前目录, 必须保证父目录存在, 否则创建失败。`mkdirs()` 方法可用于创建多层目录, 如果父目录不存在, 可以连同父目录一起创建。`mkdir()` 和 `mkdirs()` 方法的定义如下:

```
public boolean mkdir()  
public boolean mkdirs()
```

上述方法中，当目录创建成功时返回 `true`；否则返回 `false`。

示例代码如下：

```
File file = new File("F:\\test\\soft");  
// mkdir()方法在 F:\\test 目录不存在时将发生异常  
file.mkdir();  
// mkdirs()方法即使 F:\\test 目录不存在也能正常创建  
file.mkdirs();
```

□ 删除文件夹

使用 `File` 类中的 `delete()` 方法，既可以删除文件又可以删除文件夹。在删除文件夹时，要求所删除的文件夹必须为空，否则删除失败。

删除文件夹的示例代码如下：

```
private static void deleteFile(File file) {  
    //判断目录或文件是否存在  
    if (file.exists()) {  
        //如果是文件  
        if (file.isFile()) {  
            //删除文件  
            file.delete();  
        } //如果是目录  
        else if (file.isDirectory()) {  
            //获取目录中的子目录和文件  
            File files[] = file.listFiles();  
            //递归调用 deleteFile()方法  
            for (int i = 0; i < files.length; i++) {  
                deleteFile(files[i]);  
            }  
        }  
        //尝试删除目录  
        file.delete();  
    } else {  
        System.out.println("所删除的文件不存在！" + "\n");  
    }  
}
```

注意：`delete()`方法除了在发生安全侵犯时抛出 `SecurityException` 异常之外，其他异常错误都已在其内部进行了处理。通常情况下，在使用 `delete()`方法时即使删除失败也不会抛出异常，不会影响程序运行。

疑难点评

在删除文件夹时，如果文件夹不为空将不能删除，但不会抛出异常。因此，为了保障文件夹的顺利删除，在删除文件夹之前应该先判断当前文件夹是否为空，不为空应采取递归遍历的方式删除当前文件夹中的内容，然后再删除当前文件夹。

知识链接

FAQ4.05 如何遍历目录中所有的文件?

FAQ4.06 如何获取文件夹大小?

FAQ4.05 如何遍历目录中所有的文件?

📖 难度系数: ★★★★★

📖 问题频率: 75%

核心解答

使用 File 类中的 listFiles()方法可以获取文件夹中的文件和子文件夹信息。listFiles()方法的定义如下:

```
public File[] listFiles()
```

listFiles()方法返回一个包含文件和文件夹对象的 File[]数组,如果目录为空,则数组也将为空。如果抽象路径名不表示一个目录,或者发生 I/O 错误,则返回 null。

利用 listFiles()方法遍历一个目录的示例代码如下:

```
public void fileList(File file) {  
    if (file.isFile()) {  
        //如果是文件,输出文件名字  
        System.out.println("文件----->" + file.getName());  
    } else if (file.isDirectory()) {  
        //如果是文件夹,输出它的目录  
        System.out.println("文件夹,目录是----->" + file.getName());  
        //获取目录中的文件及子目录信息  
        File[] fl = file.listFiles();  
        for (int i = 0; i < fl.length; i++) {  
            //调用递归遍历 fl 数组中的每一个对象  
            fileList(fl[i]);  
        }  
    }  
}
```

在上述代码中, fileList()方法接收一个代表目录的 File 对象参数,首先判断传入的 File 对象的类型,如果是文件则显示文件信息,如果是子目录则调用 listFiles()方法获取目录中的所有内容,再进行递归遍历。

疑难点评

由于一个目录中允许包含文件和子目录,因此,遍历目录中所有文件需要使用递归调用。递归调用就是在当前的函数中调用当前的函数,在调用时传入不同的参数,这一过程是层层进行的,直到满足某一情况的时候才停止递归调用,开始从最后一个递归调用返回。

知识链接

FAQ4.04 如何创建和删除文件夹?

FAQ4.06 如何获取文件夹大小?

FAQ4.06 如何获取文件夹大小?

📖 难度系数: ★★★★★

📖 问题频率: 86%

核心解答

当 File 对象代表一个文件时, 通过 `length()` 方法可以获取文件的大小, 是以字节为单位的。如果需要获取某个目录的大小, 需要遍历目录中所有的文件, 然后将每个文件的大小相加, 相加之和才是文件夹的大小。

实现代码如下:

```
//定义成员变量, 用于累加文件大小
long size = 0;
//统计目录大小的方法
public void getDirSize(File file) {
    if (file.isFile()) {
        //如果是文件, 获取文件大小累加
        size += file.length();
    } else if (file.isDirectory()) {
        //获取目录中的文件及子目录信息
        File[] fl = file.listFiles();
        for (int i = 0; i < fl.length; i++) {
            //调用递归遍历 fl 数组中的每一个对象
            getDirSize(fl[i]);
        }
    }
}
```

在上述方法中, 定义了一个成员变量 `size` 用于统计文件大小之和。统计的结果以字节为单位, 如果需要以 K 为单位可使用 “`size%1024==0? (size/1024):(size/1024+1)`” 公式计算, 因为两个整数做除法, 最后结果取整, 所以在不能整除的情况下需要加 1。

疑难点评

获取文件夹大小, 需要将文件夹中所有文件大小累加, 因此也需要使用文件夹遍历功能。

知识链接

FAQ4.04 如何创建和删除文件夹?

FAQ4.05 如何遍历目录中所有的文件？

FAQ4.07 什么是流？如何分类？具体包含哪些类？

📖 难度系数：★★★

📖 问题频率：90%

核心解答

1. 流的定义

流是一个很形象的概念，大部分的应用程序都接收某种形式的数据输入，并产生某种形式的数据输出。当程序需要读取数据的时候，就会开启一个通向数据源的流，这个数据源可以是文件，内存或者是网络连接。与读取过程相似，当程序需要写入数据的时候，就会开启一个通向目的地的流。在程序读取和写入数据的过程中数据就像在“流”动一样，如图 4-1 所示。

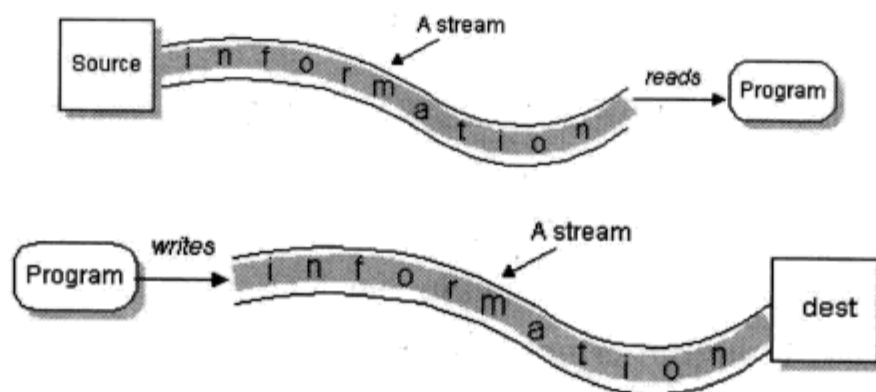


图 4-1 数据的读取和写入过程

通过图 4-1 可知，流代表经过管道流通的数据。为了进行数据的输入和输出操作，Java 把不同的输入和输出源（例如键盘、文件和网络连接等）抽象表述为流。

2. 流的分类

Java 把程序输入和输出的操作进行了封装，提供了很多类，利用这些类可以方便的实现数据的输入和输出操作。按照不同的分类方式，可以将流分为不同的类型。按照流向不同，可以分为输入流和输出流；按照处理单位不同，可以分为字节流和字符流。

(1) 字节流

字节流在数据读取和写入时以字节为单位，包含 `InputStream` 和 `OutputStream` 两个基础类。

`InputStream` 用于按字节读取数据，其子类主要有以下几种。

- ❑ `ByteArrayInputStream`: 把内存中的一个缓冲区作为 `InputStream` 使用。
- ❑ `StringBufferInputStream`: 把一个 `String` 对象作为 `InputStream`。
- ❑ `FileInputStream`: 把一个文件作为 `InputStream`，实现对文件的读取操作。

- ❑ `PipedInputStream`: 实现了 `pipe` 的概念, 主要在线程中使用。
 - ❑ `SequenceInputStream`: 把多个 `InputStream` 合并为一个 `InputStream`。
 - ❑ `DataInputStream`: 从 `stream` 中读取基本类型 (`int`、`char` 等) 数据。
 - ❑ `BufferedInputStream`: 使用缓冲区读取字节数据。
 - ❑ `LineNumberInputStream`: 记录 `InputStream` 内的行数, 然后可以调用 `getLineNumber()` 和 `setLineNumber(int)` 方法。
 - ❑ `PushbackInputStream`: 很少用到, 一般用于编译器开发。
- `OutputStream` 用于按字节输出数据, 其子类主要有以下几种。
- ❑ `ByteArrayOutputStream`: 把信息存入内存中的一个缓冲区中。
 - ❑ `FileOutputStream`: 把字节信息存入文件中。
 - ❑ `PipedOutputStream`: 实现了 `pipe` 的概念, 主要在线程中使用。
 - ❑ `SequenceOutputStream`: 把多个 `OutputStream` 合并为一个 `OutputStream`。
 - ❑ `DataOutputStream`: 往 `stream` 中输出基本类型 (`int`、`char` 等) 数据。
 - ❑ `BufferedOutputStream`: 使用缓冲区将字节数据输出。
 - ❑ `PrintStream`: 产生格式化输出。

(2) 字符流

字符流在数据读取和写入时以字符为单位, 在 JDK 1.1 之前输入输出流只支持 8 位的字节流, 从 JDK 1.1 开始, 能够支持 16 位的 Unicode 字符流。字符流包含 `Reader` 和 `Writer` 两个基础类。

`Reader` 用于按字符读取数据, 其子类主要有以下几种。

- ❑ `CharArrayReader`: 与 `ByteArrayInputStream` 对应。
- ❑ `StringReader`: 与 `StringBufferInputStream` 对应。
- ❑ `FileReader`: 与 `FileInputStream` 对应。
- ❑ `PipedReader`: 与 `PipedInputStream` 对应。
- ❑ `BufferedReader`: 与 `BufferedInputStream` 对应。
- ❑ `LineNumberReader`: 与 `LineNumberInputStream` 对应。
- ❑ `PushBackReader`: 与 `PushbackInputStream` 对应。

`Writer` 用于按字符输出数据, 其子类主要有以下几种。

- ❑ `CharArrayWrite`: 与 `ByteArrayOutputStream` 对应。
- ❑ `FileWrite`: 与 `FileOutputStream` 对应。
- ❑ `PipedWrite`: 与 `PipedOutputStream` 对应。
- ❑ `BufferedWrite`: 与 `BufferedOutputStream` 对应。
- ❑ `PrintWrite`: 与 `PrintStream` 对应。

疑难点评

流是一个很抽象的概念, 而且有很多分类。读者在使用流操作之前, 应首先理解流的概念,

了解流的分类。

知识链接

FAQ4.08 如何实现字节流和字符流之间的转化?

FAQ4.08 如何实现字节流和字符流之间的转化?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

流按处理单位不同可以划分为字节流和字符流, 字节流以字节为单位进行读写操作, 字符流是以字符为单位进行读写操作。一般在处理图片、多媒体等原始字节信息时, 选用字节流, 例如 `FileInputStream`、`FileOutputStream` 等。如果文件包含的是字符信息, 应选用字符流, 例如 `FileReader`、`FileWriter` 等。

注意: Excel、Word 和 PDF 等文件不属于纯文本文件, 因为文件中除了包含字符信息外还有很多格式信息, 因此如果使用字符流读写这类文件会出现乱码问题。对这类文件的读写操作需要使用第三方组件, 对 Word 文件的操作可以使用 `jacob`、`iText`、`POI` 和 `java2word`, 对 Excel 文件的操作可以使用 `POI` 和 `JXL`, 而 PDF 文件的操作可以使用 `iText`、`pdfbox` 和 `xpdf`。

字节流和字符流之间是可以相互转化的, 但是需要注意一个字符等于两个字节的问题。在字符流和字节流相互转化时需注意编码问题, 如果使用错误的编码格式, 流信息将会失真, 出现乱码。

❑ `InputStreamReader`

`InputStreamReader` 是字节流通向字符流的桥梁, 可完成字节流到字符流的转化。它使用指定的字符集读取字节并将其解码为字符, 使用的字符集可以显式给定, 否则可能接受平台默认的字符集。中文操作系统的默认编码为 `GBK`。

每次调用 `InputStreamReader` 中的 `read()` 方法都会导致从基础输入流读取一个或多个字节。要启用从字节到字符的有效转换, 可以提前从基础流读取更多的字节, 使其超过满足当前读取操作所需的字节。为了达到最高效率, 可以考虑使用 `BufferedReader` 包装 `InputStreamReader`。示例代码如下:

```
public static void main(String[] args) {
    FileInputStream fis = null;
    InputStreamReader isr = null;
    BufferedReader br = null;
    try {
        //创建字节输出流
        fis = new FileInputStream("D:\\a.txt");
```

```

//利用 InputStreamReader 包装 FileInputStream 字节流, 并指定字节转换为字符的编码为 GBK
isr = new InputStreamReader(fis,"GBK");
//利用 BufferedReader 包装 InputStreamReader
br = new BufferedReader(isr);
//读取文件中的一行字符
String s = br.readLine();
//打印输出
System.out.println(s);
} catch (IOException e) {
    e.printStackTrace();
}finally{
    //关闭 BufferedReader 对象
    if(br != null){
        try {
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //关闭 InputStreamReader 对象
    if(isr != null){
        try {
            isr.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //关闭 FileInputStream 对象
    if(fis != null){
        try {
            fis.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}
}

```

□ OutputStreamWriter

OutputStreamWriter 是字符流通向字节流的桥梁, 可完成字符流到字节流的转化。它使用指定的字符集将向其写入的字符编码为字节。它使用的字符集可以显式给定, 否则可能接受平台默认的字符集。

每次调用 **write()** 方法都会针对给定的字符 (或字符集) 调用编码转换器。在写入基础输出流之前, 得到的这些字节会在缓冲区累积。可以指定此缓冲区的大小, 不过, 默认的缓冲区对多数操作已足够。注意, 传递到此 **write()** 方法的字符是未缓冲的。为了达到最高效率, 可以考虑使用 **BufferedWriter** 包装 **OutputStreamWriter** 来避免频繁调用转换器。示例代码如下:

```

public static void main(String[] args) {
    FileOutputStream fos = null;

```



```
OutputStreamWriter osr = null;
BufferedWriter bw = null;
try {
    //创建字节输出流
    fos = new FileOutputStream("D:\\a.txt");
    //利用 OutputStreamWriter 包装 FileOutputStream 字节流, 并指定字符转换为字节的编码为 GBK
    osr = new OutputStreamWriter(fos,"GBK");
    //利用 BufferedWriter 包装 OutputStreamWriter
    bw = new BufferedWriter(osr);
    //写一行字符串
    bw.write("测试字符串");
} catch (IOException e) {
    e.printStackTrace();
} finally {
    //关闭 BufferedWriter 对象
    if(bw != null){
        try {
            bw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //关闭 OutputStreamWriter 对象
    if(osr != null){
        try {
            osr.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //关闭 FileOutputStream 对象
    if(fos != null){
        try {
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

疑难点评

流可分为字节流和字符流, 一个字符等于两个字节的大小。在字节流和字符流之间进行相互转化时, 可以使用 `InputStreamReader` 和 `OutputStreamWriter` 类。注意转换时采用的编码, 否则会产生乱码问题。

知识链接

FAQ4.07 什么是流? 如何分类? 具体包含哪些类?

FAQ4.09 如何判断要读的文件是否到达末尾?

📖 难度系数: ★★★

📖 问题频率: 70%

核心解答

当一个文件的内容读到末尾后,再读取时将会发生异常,为了避免异常发生我们在读文件时都会判断文件是否已经读到末尾。在使用输入流读文件时,许多输入流的读取方法都返回一个特殊值表示到达流的末尾。在读文件操作时,主要使用 `InputStream` 字节输入流和 `Reader` 字符输入流,判断文件是否到达末尾的方法如下。

- ❑ `InputStream` 字节输入流的 `read()`方法返回读取的字节信息,如果到达流的末尾,则返回-1。
- ❑ `InputStream` 字节输入流的 `read(byte[] b)`方法返回读取到 `byte[]`数组中的总字节数,如果已到达流末尾而不再有数据,则返回-1。
- ❑ `Reader` 字符输入流的 `read()`方法返回读取的字节信息,如果到达流的末尾,则返回-1。
- ❑ `Reader` 字符输入流的 `read(char[] cbuf)`方法返回读取到 `char[]`数组中的总字符数,如果已到达流末尾而不再有数据,则返回-1。
- ❑ `BufferedReader` 类的 `readLine()`方法读取的一行内容的字符串,不包含换行符和回车符,如果已到达流末尾,则返回 `null`。

利用上述方法返回的特殊值,我们可以判断是否已读到文件末尾,例如利用 `BufferedReader` 类的 `readLine()`方法读文件的示例代码如下:

```
public void testBufferedReader() throws Exception{
    FileReader f = new FileReader("D:\\测试.txt");
    BufferedReader br = new BufferedReader(f);
    String tmp = null;
    //将 readLine()方法每次读到的内容赋值给 tmp 变量,然后再判断 tmp 的值是否为 null
    while((tmp=br.readLine()) != null){
        System.out.println(tmp);
    }
    br.close();
    f.close();
}
```

- ❑ 上述代码,在 `while` 循环控制部分,每次执行先将 `readLine()`方法返回的内容赋值给 `tmp` 变量,然后再判断 `tmp` 变量的值是否等于 `null`,如果等于 `null` 则表示文件内容已读取完毕,结束循环,如果 `tmp` 不等于 `null` 则表示还可以继续读文件内容,执行循环体内的打印输出程序。

疑难点评

在读文件时,如果读到文件末尾后再继续读取,将会发生异常,因此,为了避免该异常发

生有必要进行“文件是否到达末尾”的判断。

知识链接

FAQ4.10 如何读文件、写文件?

FAQ4.10 如何读文件、写文件?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

□ 读文件

读文件时可根据文件类型的不同采用不同的读取方式。

如果是原始字节流文件,例如图片、流媒体等,可使用以下代码读取。

```
public void testBufferedInputStream() throws Exception{
    int tmp = -1;
    FileInputStream fs = new FileInputStream("D:\\测试.txt");
    BufferedInputStream bs = new BufferedInputStream(fs);
    while((tmp=bs.read()) != -1){
        System.out.print((char)tmp);
    }
    bs.close();
    fs.close();
}
```

如果是纯文本字符文件,例如 txt、log、csv 等格式,可使用以下代码读取。

```
public void testBufferedReader() throws Exception{
    FileReader f = new FileReader("D:\\测试.txt");
    BufferedReader br = new BufferedReader(f);
    String tmp = null;
    while((tmp=br.readLine()) != null){
        System.out.println(tmp);
    }
    br.close();
    f.close();
}
```

□ 写文件

写文件时可根据文件类型的不同采用不同的写入方式。在写入时,如果目标文件不存在,首先会尝试自动创建文件,然后再执行写入操作。如果目标文件的父目录不正确,文件自动创建将失败。

如果是原始字节流文件,例如图片、流媒体等,可以使用以下代码进行写入。

```
public void testPrintStream() throws Exception{
    FileOutputStream f = new FileOutputStream("D:/outtest.txt");
}
```



```
BufferedOutputStream b = new BufferedOutputStream(f);
b.write('A');
b.write('B');
ps.close();
f.close();
}
```

如果是纯文本字符文件，例如 txt、log、csv 等格式，可以使用以下代码进行写入。

```
public void testPrintWriter() throws Exception{
    FileWriter fw = new FileWriter("D:/writertest.txt");
    BufferedWriter bw = new BufferedWriter(fw);
    PrintWriter pw = new PrintWriter(bw);
    pw.println("欢迎进入 Java 世界!");
    pw.println("从 HelloWorld 开始");
    pw.close();
    bw.close();
    fw.close();
}
```

注意：除纯文本文件外，例如 doc、xls 等其他文件，由于文件内容除了包含文本信息之外还包含一些样式和格式信息，因此使用字符流和字节流读取时为乱码。这些特殊格式文件，需要使用第三方组件执行读写操作。

疑难点评

读文件和写文件是文件操作中最常用的功能，读者应该重点掌握。

知识链接

FAQ4.11 如何以追加的方式写文件？

FAQ4.13 如何在文件的任意位置进行读写？

FAQ4.17 如何使用 NIO 读写文件？

FAQ4.11 如何以追加的方式写文件？

📖 难度系数：★★★

📖 问题频率：90%

核心解答

FileOutputStream 和 FileWriter 类都提供了带 boolean 类型参数的构造方法，具体定义如下：

//FileOutputStream 的构造方法

```
public FileOutputStream(File file,boolean append) throws FileNotFoundException
```

```
public FileOutputStream(String name,boolean append) throws FileNotFoundException
```

//FileWriter 的构造方法

```
public FileWriter(File file,boolean append) throws IOException
```

```
public FileWriter(String fileName,boolean append) throws IOException
```


在上述构造方法中, append 参数用于指定写入方式, 如果为 true 则将信息写入文件末尾处, 如果为 false 则清空原有信息并将信息写入文件开始处。

示例代码如下:

```
public void testPrintWriter() throws Exception{
    //指定写入方式为追加方式
    FileWriter fw = new FileWriter("D:/writertest.txt",true);
    BufferedWriter bw = new BufferedWriter(fw);
    PrintWriter pw = new PrintWriter(bw);
    pw.println("欢迎进入 Java 世界!");
    pw.println("从 HelloWorld 开始");
    pw.close();
    bw.close();
    fw.close();
}
```

疑难点评

FileOutputStream 和 FileWriter 类的构造方法都提供了一个参数, 该参数可控制是否采取追加写入的方式写文件。一般写文件时, 默认采取覆盖方式, 即文件中只保留最近一次写的内容。如果需要采取追加写入的方式, 只要修改上述两个类的构造参数即可。

知识链接

FAQ4.10 如何读文件、写文件?

FAQ4.13 如何在文件的任意位置进行读写?

FAQ4.12 如何实现文件和文件夹的复制?

📖 难度系数: ★★★★★

📖 问题频率: 86%

核心解答

□ 文件复制

实现文件复制的代码如下:

```
/**
 * 复制单个文件
 * @param oldPath String 原文件路径, 例如: c:/fqf.txt
 * @param newPath String 复制后路径, 例如: f:/fqf.txt
 * @return boolean
 */
public void copyFile(String oldPath, String newPath) {
    try{
        int byteread = 0;
        File oldfile = new File(oldPath);
```

```

        if (oldfile.exists()) { //文件存在时
            InputStream inStream = new FileInputStream(oldPath); //负责文件读取
            FileOutputStream fs = new FileOutputStream(newPath); //负责文件写入
            byte[] buffer = new byte[1024];
            while ((byteread = inStream.read(buffer)) != -1) {
                fs.write(buffer, 0, byteread);
            }
            inStream.close();
        }
    } catch (Exception e) {
        System.out.println("复制单个文件操作出错");
        e.printStackTrace();
    }
}

```

□ 文件夹复制

实现文件夹复制的代码如下：

```

/**
 * 复制整个文件夹内容
 * @param oldPath String 原文件路径，例如：c:/fqf
 * @param newPath String 复制后路径，例如：f:/fqf/ff
 * @return boolean
 */
public void copyFolder(String oldPath, String newPath) {
    try {
        (new File(newPath)).mkdirs(); //如果文件夹不存在则建立新文件夹
        File a=new File(oldPath);
        String[] file=a.list();
        File temp=null;
        for (int i = 0; i < file.length; i++) {
            if(oldPath.endsWith(File.separator)){
                temp=new File(oldPath+file[i]);
            }
            else{
                temp=new File(oldPath+File.separator+file[i]);
            }

            if(temp.isFile()){
                FileInputStream input = new FileInputStream(temp);
                FileOutputStream output = new FileOutputStream(newPath + "/" +
                    (temp.getName()).toString());
                byte[] b = new byte[1024 * 5];
                int len;
                while ( (len = input.read(b)) != -1) {
                    output.write(b, 0, len);
                }
                output.flush();
                output.close();
                input.close();
            }
            if(temp.isDirectory()){ //如果是子文件夹

```

```

        copyFolder(oldPath+"/"+file[i],newPath+"/"+file[i]);
    }
}
} catch (Exception e) {
    System.out.println("复制整个文件夹内容操作出错");
    e.printStackTrace();
}
}

```

疑难点评

在 File 类中提供了文件创建、删除和移动的实现方法,但没有提供复制方法。如果需要使用文件或文件夹的复制功能,需要借助于文件读写功能。

知识链接

FAQ4.10 如何读文件、写文件?

FAQ4.13 如何在文件的任意位置进行读写?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

RandomAccessFile 类是一种特殊的文件流,它同时实现了 DataInput 和 DataOutput 接口,因此可以用它来读/写文件。RandomAccessFile 通过一个文件指针来指定读写的位置,默认该指针处于文件开始处。该文件指针可以通过 getFilePointer()方法读取,并通过 seek()方法设置。可通过 seek()方法控制指针位置,实现在文件的任何位置读取或者写入数据。

RandomAccessFile 类提供了两个构造方法,具体如下:

```

public RandomAccessFile(File file,String mode) throws FileNotFoundException
public RandomAccessFile(String name,String mode) throws FileNotFoundException

```

上述构造方法中,第 1 个参数用于指定目标文件,第 2 个参数 mode 用于指定操作模式,可使用的模式如表 4-1 所示。

表 4-1 RandomAccessFile 可使用的操作模式

参 数 值	具 体 描 述
"r"	以只读方式打开。调用结果对象的任何 write 方法都将导致抛出 IOException
"rw"	打开以便读取和写入。如果该文件尚不存在,则尝试创建该文件
"rws"	打开以便读取和写入,对于 "rw",还要求对文件的内容或元数据的每个更新都同步写入到基础存储设备
"rwd"	打开以便读取和写入,对于 "rw",还要求对文件内容的每个更新都同步写入到基础存储设备

“rwd”模式可用于减少执行的 I/O 操作数量。使用“rwd”仅要求更新要写入存储的文件的内容；使用“rws”要求更新要写入的文件内容及其元数据。

利用 RandomAccessFile 进行文件读写的示例代码如下。

□ 写操作

```
public static void testWrite() throws Exception{
    RandomAccessFile rf = new RandomAccessFile("C:\\a.txt","rw");
    //rf.seek(rf.length()); //可指定位置写入，以字节为单位
    rf.writeByte('a');
    rf.writeByte('b');
    rf.writeByte('c');
    rf.writeUTF("我");
    rf.close();
}
```

□ 读操作

```
public static void testRead() throws Exception{
    RandomAccessFile rf = new RandomAccessFile("C:\\a.txt","r");
    //rf.seek(0); //可指定位置读取，以字节为单位
    System.out.println((char)rf.read());
    System.out.println((char)rf.read());
    System.out.println((char)rf.read());
    System.out.println(rf.readUTF());
    rf.close();
}
```

疑难点评

RandomAccessFile 类是一种特殊的文件流，该类不仅可以实现读文件和写文件的功能，而且通过 seek()方法还可以更加灵活的控制读、写文件内容的位置。

知识链接

FAQ4.10 如何读文件、写文件？

FAQ4.11 如何以追加的方式写文件？

FAQ4.14 使用 Buffered 缓冲流写文件，为什么内容没有写入？

📖 难度系数：★★★

📖 问题频率：85%

核心解答

在 OutputStream 和 Writer 输出流中，都提供了 flush()和 close()两个方法。flush()方法用于刷新输出流并强制写出所有缓冲的输出字节。close()方法用于关闭输出流并释放与此流有关

的所有系统资源。

在使用缓冲流写文件时,是将信息先写入缓冲区,如果不显示调用 `flush()` 和 `close()` 方法,则无法将缓冲区中的信息输出到目标文件中,因此文件为空。

正确的示例代码如下:

```
public static void test1() throws Exception{
    FileWriter fw = new FileWriter("C:\\w.txt");
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write('A');
    bw.write('和');
    bw.write("大家好");
    bw.close(); //此处必须调用 close()或 flush()方法才能正确写入
    fw.close();
}
```

注意: 在使用 `close()` 方法时,会默认执行 `flush()` 方法的功能,因此 `close()` 方法也可以将缓冲区信息输出到目标文件。

疑难点评

在使用缓冲流读写文件时,实现过程中引入了缓冲区的机制,即读写操作先将内容放入缓冲区。因此,写操作完毕后,如果不将缓冲区内容刷新到文件,文件内容将为空。

知识链接

FAQ4.10 如何读文件、写文件?

FAQ4.11 如何以追加的方式写文件?

FAQ4.15 如何实现文件的分割与合并?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

使用 `RandomAccessFile` 类可实现文件的分割和合并功能,该类具有在文件任意位置进行读写的功能。例如多线程下载、断点续传等功能都需要用到文件分割和合并功能。

❑ 文件分割

文件分割功能的示例代码如下:

```
/**
 * 分割文件
 * @param fileName:源文件
 * @param filterFolder: 分割文件所在目录
 * @param size:每一份大小,以 KB 为单位
 * @throws Exception
```

```

    */
    public void cut(String fileName,String filterFolder,int size) throws Exception {
        size = size*1024;
        int maxx = 0;

        //如果输出目录不存在则创建
        File outFolder = new File(filterFolder);
        if(!outFolder.exists()){
            outFolder.mkdirs();
        }
        File inFile = new File(fileName);
        int fileLength = (int) inFile.length(); //取得文件的大小
        int value; // 取得要分割的个数

        RandomAccessFile inn = new RandomAccessFile(inFile, "r");//打开要分割的文件

        value = fileLength / size;

        int i = 0;
        int j = 0;

        //根据要分割的数目输出文件
        for (; j < value; j++) {
            File outFile = new File(filterFolder+File.separator+inFile.getName()+ j + ".tmp");
            RandomAccessFile outt = new RandomAccessFile(outFile, "rw");
            maxx += size;
            for (; i < maxx; i++) {
                outt.write(inn.read());
            }
            outt.close();
        }
        File outFile = new File(filterFolder+File.separator+inFile.getName()+ j + ".tmp");
        RandomAccessFile outt = new RandomAccessFile(outFile, "rw");
        for (; i < fileLength; i++) {

            outt.write(inn.read());
        }
        outt.close();
        inn.close();
    }
}

```

□ 文件合并

文件合并功能的示例代码如下：

```

/**
 * 将分割后的文件合并
 * @param fileName: 合并之后的文件
 * @param filterFolder: 分割文件所在目录
 * @param filterName: 分割后的文件后缀
 * @throws Exception
 */
public void unite(String fileName,String filterFolder,final String filterName) throws Exception {
    File[] tt;
}

```

```
File inFile = new File(filterFolder); //在当前目录下的文件
File outFile = new File(fileName); //取得输出名
RandomAccessFile outt = new RandomAccessFile(outFile, "rw");

//取得符合条件的文件名
tt = inFile.listFiles(new FilenameFilter() {
    public boolean accept(File dir, String name) {
        String rr = new File(name).toString();
        return rr.endsWith(filterName);
    }
});
//打印出取得的文件名
for (int i = 0; i < tt.length; i++) {
    System.out.println(tt[i]);
}

//打开所有的文件再写入一个文件里
for (int i = 0; i < tt.length; i++) {
    RandomAccessFile inn = new RandomAccessFile(tt[i], "r");
    int c;
    while ((c = inn.read()) != -1)
        outt.write(c);
}

outt.close();
}
```

疑难点评

由于 RandomAccessFile 类可以在文件指定位置读写文件内容, 因此 RandomAccessFile 类是实现文件分割和合并功能的最佳选择。

知识链接

FAQ4.10 如何读文件、写文件?

FAQ4.13 如何在文件的任意位置进行读写?

FAQ4.16 什么是 NIO? 与 I/O 有什么区别和联系?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

原有的 I/O 操作都是以字节为单位进行读写的, 虽然应用时使用了很多高级流进行了封装, 不需要直接去处理字节流, 但是底层的实现还是离不开字节处理。原有的 I/O 操作是一次一个字节的处理数据, 速度比较慢; 此外 InputStream 中的 read() 是一种阻塞性的方法, 该方法可用

于从流中读取数据，但是如果数据源没有数据，它将一直等待，其他程序也不能执行。

为了解决原有 I/O 中的一些问题，从 JDK 1.4 开始提供了一系列改进 I/O 处理的新特性，这些改进功能和特性被称为“新 I/O”（New I/O，简称 NIO）。NIO 新增了很多处理 I/O 的类，这些类都放在 `java.nio` 及其子包中，并且对原有 `java.io` 包中的很多类进行了改进，新增了满足 NIO 的功能。

NIO 与原有的 I/O 有同样的作用和目的，是基于原有 I/O 的改进和扩展。NIO 与原有 I/O 不同，它是基于特殊缓冲区块（Buffer）进行的高效的 I/O 操作。NIO 的缓冲区块与普通的缓冲区不同，它是一块连续的空间，它内存的分配不在 Java 的堆栈中，不受 Java 内存回收的影响；它的实现不是纯 Java 的代码，而是本地代码，这样操作系统可以直接与缓冲区进行交互，Java 程序只需要完成对缓冲区的读写，而后续操作由操作系统完成。

异步通道 Channel（又称频道，译法暗示存在多通道可选性）是 NIO 的另外一个重要的新特点。Channel 并不是对原有 I/O 类的扩充和完善，而是完全崭新的实现。通过 Channel，Java 应用程序能够更好地与操作系统的 I/O 服务结合起来，充分地利用 Buffer 缓冲区，完成高性能的 I/O 操作。Channel 的实现也不是纯 Java 的，而是和操作系统结合紧密的本地代码。

NIO 与原有 I/O 相比，NIO 新特性主要体现在以下几个方面。

- ❑ 更加灵活的可伸缩的 I/O 接口（scalable I/O），包括 I/O 抽象 Channels 的出现以及新的多元的（multiplexed），非阻塞（non-blocking）的 I/O 机制。这使得构建产品级的应用服务更加方便灵活，使用户能够轻松应付成千上万个开放的连接，并且可以有效地利用多个处理器。
- ❑ 快速缓存（fast buffered）的二进制和字符 I/O 接口。快速缓存的二进制 I/O API 使得用户可以很容易地编写出操作文件流或者二进制数据流的高性能代码。而快速缓存的字符 I/O API 使得用户可以更加高效地处理字符流和文件，此外它还将正则表达式引入到 Java 平台中来格式化用户的输入与输出。
- ❑ 字符集的编码器和解码器（Character-set encoders and decoders）。这些字符集转换 API 使得用户可以直接访问操作系统内置的字符集转换器，同时还支持那些外来的转换器。
- ❑ 基于 Perl 风格正则表达式的模式匹配机制（A pattern-matching facility based on Perl-style regular expressions）。
- ❑ 改良的文件系统接口，支持锁定和内存映射（locks and memory mapping）。该特性使得用户可以更加容易地处理各种文件系统操作中出现的各种问题，同时使得用户可以更加高效地访问大量的文件属性集。此外如果用户确实需要，还可以访问与平台相关的一些特性。最后，它还提供对非本地文件系统的支持，例如网络文件系统（network filesystems）。
- ❑ 新的 I/O 违例类可以使用户更加有针对性地去处理各种 I/O 错误，让用户能够在各种平台上一致地来对待这些错误。
- ❑ 增加了对并发的支持，NIO 类中的大部分方法都支持多个并发的线程。

注意: NIO 并不是对原有 I/O 的替代, 尽管 NIO 在 I/O 操作时速度快, 但是由于其底层借助了大量的本地代码, 对操作系统和硬件平台有很大依赖性, 这将影响 Java 的可移植性。

疑难点评

NIO 是对原有 IO 的改革, 它引入了很多机制, 使用 NIO 读写文件, 其效率与原有 IO 操作相比得到了很大的提高。

知识链接

FAQ4.17 如何使用 NIO 读写文件?

FAQ4.17 如何使用 NIO 读写文件?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在使用 Java 进行文件读写时有很多方法, 各个方法的效率也大不相同, 例如经过缓冲流包装后的效率会比以前要高。在有些情况下可以使用 NIO 进行文件读写, 因为 NIO 读写效率会更快。

❑ 读操作

NIO 实现读操作的示例代码如下:

```
/**
 *读文件
 *filename:要读取的目标文件
 */
private static String fileReader(File fileName) {
    String fileContent = null;
    FileInputStream fis = null;
    FileChannel fc = null;
    try {
        fis = new FileInputStream(fileName);
        //获取通道 channel
        fc = fis.getChannel();
        //创建 ByteBuffer 缓冲区, 大小可根据实际情况指定
        ByteBuffer bb = ByteBuffer.allocate(10000);
        fc.read(bb);
        bb.flip();
        fileContent = new String(bb.array());
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
```

```
//释放 FileChannel 资源
try {
    fc.close();
} catch (Exception ex) {
}
try {
    fis.close();
} catch (Exception ex) {
}
}
//返回文件内容
return fileContent;
}
```

□ 写操作

NIO 实现写操作的示例代码如下:

```
/**
 *写文件
 *filename:要写入的目标文件
 *s:要写入的信息
 */
private static void fileWriter(File filename,String s) {
    String fileContent = null;
    FileOutputStream fos = null;
    FileChannel fc = null;
    byte[] bts = s.getBytes();
    try {
        fos = new FileOutputStream(fileName);
        //获取通道 channel
        fc = fos.getChannel();
        //创建 ByteBuffer 缓冲区, 大小可根据实际情况指定
        ByteBuffer bb = ByteBuffer.allocate(bts.length);
        bb.put(bts);
        bb.flip();
        fc.write(bb);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        //释放 FileChannel 资源
        try {
            fc.close();
        } catch (Exception ex) {
        }
        try {
            fos.close();
        } catch (Exception ex) {
        }
    }
}
```

□ 文件复制

NIO 实现文件复制的示例代码如下:

```
/**
 *实现文件复制
 *srcFile:源文件
 *destFile:目标文件
 */
public static void copy(String srcFile,String destFile) throws IOException{
    long begin = System.currentTimeMillis();
    ByteBuffer bb = ByteBuffer.allocate(100000);
    //获取 NIO 读取通道
    FileInputStream fis = new FileInputStream(srcFile);
    FileChannel in = fis.getChannel();
    //获取 NIO 写入通道
    FileOutputStream fos = new FileOutputStream(destFile);
    FileChannel out = fos.getChannel();
    int len = -1;
    //当读文件到末尾时结束循环
    while((len = in.read(bb)) != -1){
        //在 write 之前, 将 position 和 limit 标志设置好
        bb.flip();
        //按设定的 position 位置开始读, 到 limit 结束
        out.write(bb);
        //初始化 position\limit\capacity 标志的位置, 为下一次循环读取做准备
        bb.clear();
    }
    long end = System.currentTimeMillis();
    System.out.println(end - begin);
    out.close();
    in.close();
    fos.close();
    fis.close();
}
```

疑难点评

NIO 是在原有 IO 流的基础之上进行的改进, 因此, 使用 NIO 读写文件, 首先还需要使用 IO 流类。注意并不是所有 IO 流类都支持 NIO 操作的, 支持 NIO 操作的类有 `FileInputStream`、`FileOutputStream` 和 `RandomAccessFile`。

知识链接

FAQ4.16 什么是 NIO? 与 I/O 有什么区别和联系?

FAQ4.18 什么是字符编码和解码?

📖 难度系数: ★★★

📖 问题频率: 85%

核心解答

在字符和字节信息相互转化时,需要涉及编码和解码操作。一般情况下,开发者不需要显示指定编码和解码的字符集,默认使用与操作系统一致的字符集。但是在某些特殊情况下,例如在文件保存时与系统字符集不一致,就需要开发者显示指定编码和解码时采用的字符集,否则会出现乱码。

在 NIO 技术中提供了与字符集、编码和解码相关的 API,存放在 `java.nio.charset` 包中,包含了在字节与 Unicode 字符间转换的字符集、解码器与编码器,具体有以下几项。

- ❑ `Charset`: 字符集,用于描述字符和字节之间的命名映射关系。
- ❑ `CharsetDecoder`: 解码器,用于实现将字节解码为字符。
- ❑ `CharsetEncoder`: 编码器,用于实现将字符编码为字节。

疑难点评

字符信息在计算机中存储时,需要涉及编码和解码操作,这是由于计算机语言和人类语言不一致造成的。读者在理解上述的编码和解码操作后,对文件读写操作为什么会产生乱码的问题将会有更深入的理解。

知识链接

FAQ4.19 读写文件时为什么中文字符经常产生乱码?

FAQ4.20 如何解决 `FileReader` 读文件乱码的问题?

FAQ4.19 读写文件时为什么中文字符经常产生乱码?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

由于计算机只能识别二进制信息,因此计算机中的各种文件在存储时都是以二进制形式存储在磁盘中,读取时需要先将磁盘的二进制信息载入,然后通过程序将二进制信息转化并显示。以 `a.txt` 文件为例,其存储和读取过程如图 4-2 所示。

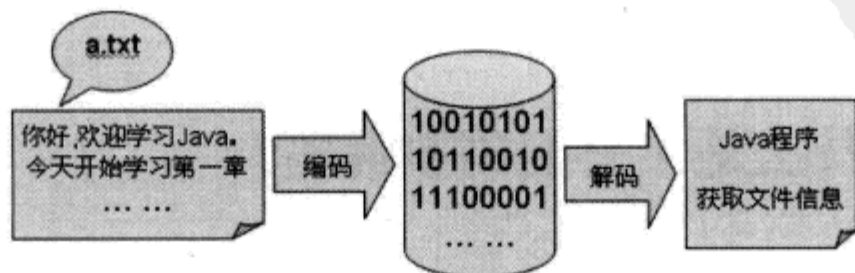


图 4-2 文件存储和读取过程示意图

通过图 4-2 所示的过程可知,当存储 a.txt 文件时,需要将文件中的字符转为二进制字节信息,该过程属于编码;当程序读取时,需要先将字节信息转为字符信息,该过程属于解码,经过解码后才能将原有文件信息正确显示。

字符集用于描述字符与二进制之间的映射关系。在编码和解码过程中,都需要遵循某一种字符集进行转换,如果编码和解码遵循相同的字符集,文件信息能正确显示;如果遵循不同的字符集,文件信息将会产生乱码。

在文件读写时,除了由于编码和解码的字符集不一致引起乱码之外,API 使用不当也会引起中文乱码。例如使用 `InputStream` 读取中文字符的文件,由于一个中文字符需要由两个字节表示,而 a、b、c 等英文字符一个字节就能表示,`InputStream` 是以字节为单位读取的,因此会将中文拆成两个字节分两次读取,这就是读取中文为乱码而英文字符却正常的原因。

疑难点评

读写文件乱码是由于编码和解码时使用的字符集不一致引起的,读者了解乱码产生过程之后有助于解决代码中的此类问题。

知识链接

FAQ4.18 什么是字符编码和解码?

FAQ4.20 如何解决 FileReader 读文件乱码的问题?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

一般情况下,如果文件中包含中文字符,为了避免中文乱码问题,开发者会习惯选用 `FileReader` 类读文件。但是有些情况,即使使用了 `FileReader` 类同样会产生乱码问题。

`FileReader` 属于字符流,是读取字符文件的便捷类。`FileReader` 继承自 `InputStreamReader`,`InputStreamReader` 是将字节流转换为字符流的桥梁,即将字节信息转换为字符信息。实际上 `FileReader` 类在内部实现过程中也是利用 `InputStreamReader` 完成字节流到字符流的转化,只不过转化时采用的字符集为系统平台默认的字符集。

如果文件保存时的编码设定为 UTF-8,那么在中文操作系统使用 `FileReader` 就会发生乱码,因为中文操作系统平台的默认字符集为 GBK。解决该问题的方法是放弃使用 `FileReader`,选用 `InputStreamReader`,在获取 `InputStreamReader` 对象时显示指定合适的字符集。

使用指定字符集读取文件的示例代码如下:

```
public static void testInputStreamReader() throws Exception{  
    //获取读取文件的字节流
```

```
FileInputStream fis = new FileInputStream("C:\\b.txt");  
//将字节流转换为字符流, 编码指定为文件保存的编码  
InputStreamReader isr = new InputStreamReader(fis,"UTF-8");  
//使用缓冲字符流封装, 方便读取操作  
BufferedReader br = new BufferedReader(isr);  
String s = null;  
//以行为单位循环读取文件中的信息  
while((s=br.readLine()) != null){  
    System.out.println(s);  
}  
br.close();  
isr.close();  
fis.close();  
}
```

注意: 在使用 `InputStreamReader` 时, 如果指定的编码与文件的编码不一致时, 读取的内容将会是乱码。

疑难点评

一般情况下, 使用 `FileReader` 读文件是不会发生乱码问题的, 但如果文件内容采取的存储编码与操作系统编码不一致就会发生乱码。因为 `FileReader` 读文件时采取的编码是操作系统编码, 例如中文操作系统为 GBK。

知识链接

FAQ4.18 什么是字符编码和解码?

FAQ4.21 为什么 `DataInputStream` 和 `DataOutputStream` 读写文件时乱码?

📖 难度系数: ★★★

📖 问题频率: 75%

核心解答

使用 `DataInputStream` 和 `DataOutputStream`, 开发者可以以基本类型为单位对文件进行读写, 例如写入一个 `int` 值、读取一个 `boolean` 值等。使用 `DataOutputStream` 写入文件中的信息, 除了包含信息本身之外, 还有类型标识等, 只有使用 `DataInputStream` 才能正确读取原有信息, 其他方式会产生乱码, 这也是使用记事本打开文件时显示乱码的原因。

注意: `DataOutputStream` 和 `DataInputStream` 在读写文件时需要配合使用, 不能单独应用, 否则读写过程将会产生乱码。

❑ `DataOutputStream` 写文件

使用 `DataOutputStream` 写文件的示例代码如下:

```
public static void dataWrite(){
    FileOutputStream fos = null;
    DataOutputStream dos = null;
    try {
        fos = new FileOutputStream("C:\\a.txt",true);
        dos = new DataOutputStream(fos);
        dos.writeBoolean(true);
        dos.writeInt(20);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            dos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

□ DataInputStream 读文件

使用 DataInputStream 读文件的示例代码如下:

```
public static void dataRead(){
    FileInputStream fos = null;
    DataInputStream dos = null;
    try {
        fos = new FileInputStream("C:\\a.txt");
        dos = new DataInputStream(fos);
        boolean b = dos.readBoolean();
        System.out.println(b);
        int i = dos.readInt();
        System.out.println(i);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            dos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```


疑难点评

使用 `DataInputStream` 和 `DataOutputStream` 类可以读写各种类型的信息,例如写入一个整数、读取一个布尔值等。在读写文件时, `DataInputStream` 和 `DataOutputStream` 类必须成对使用,而且读写时应采取相对应的顺序和类型,否则将会发生乱码问题。

知识链接

FAQ4.18 什么是字符编码和解码?

FAQ4.22 如何实现文件锁定功能?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

Java 提供了两种方式,可以实现文件锁定功能。一种是使用 `RandomAccessFile` 实现,另一种是使用 `FileChannel` 实现。

❑ 使用 `RandomAccessFile` 实现锁定

在获取 `RandomAccessFile` 对象时,需要提供一个参数,用于指定文件的访问模式。如果需要以锁定方式对文件进行写入,可以使用“`rws`”参数,示例代码如下:

```
RandomAccessFile rw = new RandomAccessFile(file, "rws")
```

其中的“`rws`”参数中, `rw` 代表读写方式, `s` 代表同步方式,也就是锁。这种方式就是独占方式。

❑ 使用 `FileChannel` 实现锁定

`FileChannel` 是 NIO 提供的一个类,该类提供了锁定的方法。使用示例如下:

```
RandomAccessFile raf = new RandomAccessFile(new File("c:\\test.txt"), "rw");
FileChannel fc = raf.getChannel();
//尝试获取文件锁,如果有其他用户在使用将返回 null
FileLock fl = fc.tryLock();
if (fl.isValid()) {
    System.out.println("允许执行读写操作!");
    //省略写入操作代码
    fl.release(); //释放文件锁
}else{
    System.out.println("其他用户正在执行写入操作");
}
```

在上述方法中,使用 `FileChannel` 的 `tryLock()` 方法获取文件锁,此外也可以使用 `lock()` 方法,该方法与 `tryLock()` 方法的区别在于 `lock()` 是一个阻塞性方法,只有获取文件锁才能进行后续操作,否则一直等待。

注意: `FileOutputStream` 和 `FileInputStream` 也提供了 `getChannel()` 方法, 但是获取的是一个缺省的 `FileChannel` 对象, 该对象不能实现文件的锁定。

疑难点评

文件锁定功能是 NIO 提供的技术, 可以有效地避免出现文件操作冲突的问题。

知识链接

FAQ4.17 如何使用 NIO 读写文件?

FAQ4.23 如何实现对文件和字符串加密、解密?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

为了程序数据的安全性, 有时候需要对文件或其他字符信息进行加密和解密。数据加密和解密的算法有很多, 例如 DES 和 RSA 等, 也可以自己制定加密算法。下面介绍利用 DES 算法实现对文件、字符串的加密和解密操作。

DES (Data Encryption Standard) 算法出自 IBM 的研究工作, 并在 1997 年被美国政府正式采纳。它是一种分组密码, 通过反复使用加密组块替代和换位两种技术, 经过 16 轮的变换后得到密文, 安全性很高。DES 属于传统的对称密码体制, 其加密密钥与解密密钥是相同的, 由于其安全性高、速度快、计算较简单, 因此被应用于许多需要安全加密的场合。例如自动取款机的数据和 UNIX 的密码就是以 DES 算法为基础进行加密的。

注意: 有些算法只能加密不能解密, 例如 MD5 和 SHA-1 就是不可逆算法。

采用 DES 算法对字符串和文件进行加密、解密的示例代码如下:

❑ 获取密钥

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;

public class MySecurity {
    private static String strDefaultKey = "hnztzwm";
    private Cipher encryptCipher = null;
    private Cipher decryptCipher = null;

    private Key getKey(byte[] arrBTmp) throws Exception {
        //创建一个空的 8 位字节数组 (默认值为 0)
        byte[] arrB = new byte[8];
```

```

//将原始字节数组转换为 8 位
for (int i = 0; i < arrBTmp.length && i < arrB.length; i++) {
    arrB[i] = arrBTmp[i];
}

//生成密钥
Key key = new javax.crypto.spec.SecretKeySpec(arrB, "DES");

return key;
}

public MySecurity() throws Exception {
    this(strDefaultKey);
}

public MySecurity(String strKey) throws Exception {
    Security.addProvider(new com.sun.crypto.provider.SunJCE());
    Key key = getKey(strKey.getBytes());

    encryptCipher = Cipher.getInstance("DES");
    encryptCipher.init(Cipher.ENCRYPT_MODE, key);

    decryptCipher = Cipher.getInstance("DES");
    decryptCipher.init(Cipher.DECRYPT_MODE, key);
}

```

□ 字符串加密

```

public static String byteArr2HexStr(byte[] arrB) throws Exception {
    int iLen = arrB.length;
    //每个 byte 用两个字符才能表示，因此字符串的长度是数组长度的两倍
    StringBuffer sb = new StringBuffer(iLen * 2);
    for (int i = 0; i < iLen; i++) {
        int intTmp = arrB[i];
        //把负数转换为正数
        while (intTmp < 0) {
            intTmp = intTmp + 256;
        }
        //小于 0F 的数需要在前面补 0
        if (intTmp < 16) {
            sb.append("0");
        }
        sb.append(Integer.toString(intTmp, 16));
    }
    return sb.toString();
}

public byte[] encrypt(byte[] arrB) throws Exception {
    return encryptCipher.doFinal(arrB);
}

```



```
public String encrypt(String strIn) throws Exception {
    return byteArr2HexStr(encrypt(strIn.getBytes()));
}
```

□ 字符串解密

```
public static byte[] hexStr2ByteArr(String strIn) throws Exception {
    byte[] arrB = strIn.getBytes();
    int iLen = arrB.length;

    //两个字符表示一个字节，因此字节数组长度是字符串长度除以 2
    byte[] arrOut = new byte[iLen / 2];
    for (int i = 0; i < iLen; i = i + 2) {
        String strTmp = new String(arrB, i, 2);
        arrOut[i / 2] = (byte) Integer.parseInt(strTmp, 16);
    }
    return arrOut;
}

public byte[] decrypt(byte[] arrB) throws Exception {
    return decryptCipher.doFinal(arrB);
}

public String decrypt(String strIn) throws Exception {
    return new String(decrypt(hexStr2ByteArr(strIn)));
}
```

□ 文件加密

```
/**
 *以字节数组形式返回文件的内容
 */
public static byte[] getBytesFromFile(File file) throws IOException {
    InputStream is = new FileInputStream(file);

    //获取文件大小
    long length = file.length();

    //限制要读取的文件的大小不能超过 Integer.MAX_VALUE.
    if (length > Integer.MAX_VALUE) {
        return null; //文件太大，退出程序
    }

    //创建字节数组
    byte[] bytes = new byte[(int) length];

    //将文件信息读入数组
    int offset = 0;
    int numRead = 0;
    while (offset < bytes.length
        &&
        (numRead = is.read(bytes, offset, bytes.length - offset)) >= 0) {
        offset += numRead;
    }
}
```

```

    }

    //确认是否将所有信息读入数组
    if (offset < bytes.length) {
        throw new IOException("Could not completely read file " +
                                file.getName());
    }

    //关闭流
    is.close();
    return bytes;
}

/**
 * 将字节数组写入文件
 */
public static File writeBytesToFile(byte[] inByte, String pathAndNameString) throws
    IOException {
    File file = null;
    try {

        file = new File(pathAndNameString);
        file.createNewFile();
        FileOutputStream fos = new FileOutputStream(file);
        fos.write(inByte);
        fos.close();

    } catch (FileNotFoundException ex) {
        System.out.println(ex);
    } catch (IOException e) {
        System.out.println(e);
    }

    return file;
}

/**
 * 文件加密
 * srcFile:要加密的源文件
 * destFile:加密后的文件
 */
public void encryptFile(String srcFile,String destFile){
    File infile = new File(srcFile); //加密前的文件
    byte[] myfileA = getBytesFromFile(infile);
    writeBytesToFile(des.encrypt(myfileA), destFile); //加密后的文件
}

```

□ 文件解密

```

/**
 * 文件加密
 * srcFile:要解密的源文件

```



```
*destFile:解密后的文件
*/
public void decryptFile(String srcFile,String destFile){
    File infile = new File(srcFile); //加密前的文件
    byte[] myfileA = getBytesFromFile(infile);
    writeBytesToFile(des.decrypt(myfileA), destFile); //加密后的文件
}
```

测试代码如下:

```
public static void main(String[] args) {
    try {
        MySecurity des = new MySecurity();
        //加密文件测试
        encryptFile("c:\\test.zip","c:\\enFile.zip");

        //解密文件测试
        decryptFile("c:\\enFile.zip","c:\\deFile.zip");

        //字符串加密测试
        System.out.println("解密前的 String==" + "123456");
        String enString = des.encrypt("123456");
        System.out.println("加密后的 enString==" + enString);
        //字符串解密测试
        String deString = des.decrypt(enString);
        System.out.println("解密后的 deString==" + deString);

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

疑难点评

为了保障信息的安全性,需要对敏感信息进行加密和解密操作。加密和解密的算法有很多,上面介绍了一些常见的加密和解密算法的实现,读者也可以模仿上述过程自己设计加密和解密算法实现。

知识链接

FAQ4.10 如何读文件、写文件?

FAQ4.24 如何实现对文件和目录的压缩、解压缩?

FAQ4.24 如何实现对文件和目录的压缩、解压缩?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

Java 支持的压缩格式有 zip、gzip 和 jar 等。zip 是 Windows 系统常用的压缩格式、gzip 是 Linux 系统常用的压缩格式、而 jar 是 Java 特有的一种压缩格式。

下面重点介绍 zip 格式的压缩和解压缩。在使用 zip 格式压缩、解压缩时，压缩可以使用 ZipEntry 类和 ZipOutputStream 类，解压缩可以使用 ZipEntry 类和 ZipInputStream 类。

□ 文件压缩

可以实现将若干文件进行压缩的代码如下：

```
/**
 *文件压缩
 *files:要解压缩的文件
 *destZip:压缩之后的 zip 文件所放的目录，需要"D:\\\"或"D:\\test\"格式
 */
public static void zip(String[] files, String destZip) throws IOException {

    FileOutputStream fos = new FileOutputStream(destZip);
    BufferedOutputStream bos = new BufferedOutputStream(fos);
    ZipOutputStream zos = new ZipOutputStream(bos);

    //循环读文件，压缩到 zip 中
    for (int i = 0; i < files.length; i++) {
        String file = files[i];
        FileInputStream fis = new FileInputStream(file);
        BufferedInputStream bis = new BufferedInputStream(fis);
        //获取文件名，创建条目并添加到 zip 中
        File f = new File(file);
        ZipEntry z1 = new ZipEntry(f.getName());
        zos.putNextEntry(z1);
        //读取文件中的字节信息，压入条目
        int tmp = -1;
        while ((tmp = bis.read()) != -1) {
            zos.write(tmp);
        }
        zos.closeEntry();
        bis.close();
        fis.close();
    }

    zos.close();
    bos.close();
    fos.close();
}
```

□ 文件解压缩

可以实现将压缩文件解压缩的代码如下：

```

/**
 *文件解压缩
 *zipfile:要解压缩的 zip 文件
 *destpath:解压后文件所放的目录, 需要"D:\\\"或"D:\\test\"格式
 */
public static void unzip(String zipfile,String destpath) throws IOException {

    FileInputStream fis = new FileInputStream("C:\\a.zip");
    ZipInputStream zis = new ZipInputStream(fis);
    ZipEntry z1 = null;

    while ((z1 = zis.getNextEntry()) != null) {
        if (z1.isDirectory()) {
            File f = new File("C:\\\" + z1.getName());
            f.mkdirs();
        } else {
            String file = z1.getName();
            FileOutputStream fos = new FileOutputStream(destpath + file);
            int tmp = -1;
            while ((tmp = zis.read()) != -1) {
                fos.write(tmp);
            }
            zis.closeEntry();
            fos.close();
        }
    }
    zis.close();
}

```

□ 目录压缩

目录压缩需要采用递归方式遍历压缩目录中的文件及其子目录中文件。实现代码如下:

```

/**
 *压缩目录
 * zipFileName: 压缩后 zip 文件的路径
 * inputFile: 需要压缩的源目录
 */
public void zip(String zipFileName, String inputFile) throws Exception {
    File file = new File(inputFile);
    ZipOutputStream out =
        new ZipOutputStream(new FileOutputStream(zipFileName));
    zip(out, file, "");
    System.out.println("zip done");
    out.close();
}

/**
 *递归遍历子目录和文件, 进行压缩
 */
private void zip(ZipOutputStream out, File f, String base)
    throws Exception {

```



```

        System.out.println("Zipping " + f.getName());
        if (f.isDirectory()) {
            File[] fl = f.listFiles();
            out.putNextEntry(new ZipEntry(base + "/"));
            base = base.length() == 0 ? "" : base + "/";
            for (int i = 0; i < fl.length; i++) {
                zip(out, fl[i], base + fl[i].getName());
            }
        } else {
            out.putNextEntry(new ZipEntry(base));
            FileInputStream in = new FileInputStream(f);
            int b;
            while ((b = in.read()) != -1)
                out.write(b);
            in.close();
        }
    }
}

```

□ 目录解压缩

目录解压缩在解压过程中需要根据压缩时的目录结构，在目标位置将目录和文件还原。实现代码如下：

```

/**
 * 解压缩目录
 * zipFileName: 需要解压缩的 zip 文件
 * outputDirectory: 解压缩后文件所放路径
 */
public void unzip(String zipFileName, String outputDirectory)
    throws Exception {
    ZipInputStream in =
        new ZipInputStream(new FileInputStream(zipFileName));
    ZipEntry z;
    while ((z = in.getNextEntry()) != null) {
        System.out.println("unzipping " + z.getName());
        if (z.isDirectory()) {
            String name = z.getName();
            name = name.substring(0, name.length() - 1);
            File f = new File(outputDirectory + File.separator + name);
            f.mkdir();
            System.out.println(
                "mkdir " + outputDirectory + File.separator + name);
        } else {
            File f =
                new File(outputDirectory + File.separator + z.getName());
            f.createNewFile();
            FileOutputStream out = new FileOutputStream(f);
            int b;
            while ((b = in.read()) != -1)
                out.write(b);
            out.close();
        }
    }
}

```



```

    }

    in.close();
}

```

疑难点评

压缩和解压缩是用户经常使用的功能,例如 WINRAR 和 WINZIP 等软件。上面介绍了如何在 Java 中实现压缩和解压缩。读者也可以利用上述代码,自己实现一个类似于 WINRAR 的软件。

知识链接

FAQ4.10 如何读文件、写文件?

FAQ4.23 如何实现对文件和字符串加密、解密?

FAQ4.25 如何读写 properties 文件?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

properties 是 Java 程序特有的一种文件类型,该文件的信息以“键-值”格式存放。通常可以将容易变化的一些系统参数写到 properties 文件中,然后利用 Java 程序读取,这样可以提高程序灵活性。

properties 文件内容以“键-值”格式存储,示例如下:

```

name=tom
pwd=123456

```

在 Java 中可以使用 Properties 类对 properties 文件进行读写,程序中可以通过“pwd”和“name”键值获取对应的内容,也可以对内容修改。示例代码如下:

❑ 读操作

```

/**
 * 根据 key 读取 value
 * @param fileName: 属性文件名
 * @param key: 属性名
 * @return: 属性值
 */
public String read(String fileName,String key) {
    Properties props = new Properties();
    try {

        props.load(this.getClass().getClassLoader().getResourceAsStream(fileName));
    }
}

```

```

        String n = props.getProperty(key);
        System.out.println(n);
        return n;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * 读取 properties 的全部信息
 * @param fileName: 属性文件名
 */
public void readProperties(String fileName) {
    Properties props = new Properties();
    try {
        props.load(this.getClass().getClassLoader().getResourceAsStream(fileName));
        Enumeration en = props.propertyNames();
        while (en.hasMoreElements()) {
            String key = (String) en.nextElement();
            String Property = props.getProperty(key);
            System.out.println(key + ":" + Property);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

注意：程序中读取的 t.properties 文件应放在工程的 src 目录下，如果放到某个包中，例如“day.study”，那么程序中 getResourceAsStream() 读取的路径也要改变，应该改为“day/study/t.properties”

□ 写操作

```

/**
 * 写入 properties 信息
 * @param fileName: 属性文件名
 * @param parameterName: 属性名
 * @param parameterValue: 属性值
 */
public void writeProperties(String fileName, String parameterName,
    String parameterValue) {
    Properties prop = new Properties();
    String path = this.getClass().getClassLoader().getResource("").getPath().substring(1);
    path = path + fileName;
    System.out.println(fileName);
    try {
        InputStream fis = new FileInputStream(path);
        //从输入流中读取属性列表（键和元素对）
        prop.load(fis);
    }
}

```

```

//调用 Hashtable 的方法 put(), 使用 getProperty()方法提供并行性
//强制要求为属性的键和值使用字符串。返回值是 Hashtable 调用 put 的结果
OutputStream fos = new FileOutputStream(path);
prop.setProperty(parameterName, parameterValue);
//以适合使用 load 方法加载到 Properties 表中的格式
//将此 Properties 表中的属性列表 (键和元素对) 写入输出流
prop.store(fos, "Update " + parameterName + " value");
} catch (IOException e) {
    e.printStackTrace();
}
}

```

□ 测试代码

```

public class Test {
    //测试主入口方法
    public static void main(String[] args) {
        Test t = new Test();
        t.read("info.properties", "name");
        t.readProperties("info.properties");
        t.writeProperties("info.properties", "sex", "男");
        t.read("info.properties", "sex");
    }
}

```

Properties 类在读写 properties 文件时, 使用的是 ISO-8859-1 编码, 每个字节是一个 Latin1 字符, 因此对于非 Latin1 的字符和某些特殊字符, 例如中文字符, 在读写时就会发生乱码。为了避免读写中文字符乱码, 可以将中文字符转义, 再进行读写操作, 实现方法如下。

使用 JDK 提供的 native2ascii 工具实现转义, 将文件中的中文字符和要写入文件的中文字符进行编码, 这样 Properties 在读写时就不会发生乱码。

native2ascii 工具在 JDK 安装目录的 bin 目录中, 双击 native2ascii.exe 文件, 输入要转换的中文字符后, 按回车键执行编码转换, 效果如图 4-3 所示。

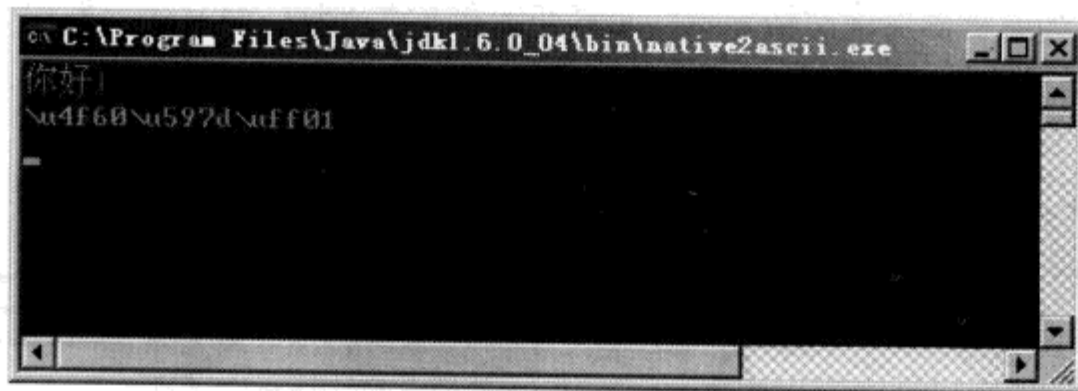


图 4-3 编码转换效果图

也可以在写入文件之前, 将要写入的字符串编码, 然后再进行写入。示例代码如下:

```

public static String convert(String s) {
    String unicode = "";
    char[] charAry = new char[s.length()];
    for(int i=0; i<charAry.length; i++) {

```



```

charAry[i] = (char)s.charAt(i);
if(Character.isLetter(charAry[i])&&(charAry[i]>255))
    unicode+="\\u" + Integer.toString(charAry[i], 16);
else
    unicode+=charAry[i];
}
return unicode;
}

```

疑难点评

properties 文件是 Java 中特有的文件类型,经常会先将一些容易变化的参数信息存储到一个 properties 文件中,然后通过 Java 程序读取。

FAQ4.26 如何读写 XML 文件?

📖 难度系数: ★★★★★

📖 问题频率: 88%

核心解答

解析 XML 文件可以使用 DOM (Document Object Model) 或 SAX (Simple API for XML) 技术。

DOM 文档对象模型是通过树型结构存取 XML 文档的,DOM 解析器在解析时将 XML 文档读入内存并构造成为一个树型结构,XML 文档中的每一个元素都是文档树中的节点,可以通过遍历树节点的形式获取 XML 文档内容。DOM 的解析原理如图 4-4 所示。

SAX 提供了一种快速、串行读取的方式解析 XML 文件。SAX 解析器在解析开始的时候就开始发送事件,当解析器发现文档开始、元素开始和文本等时,会发送一系列事件,调用不同的事件处理方法。SAX 的解析原理如图 4-5 所示。

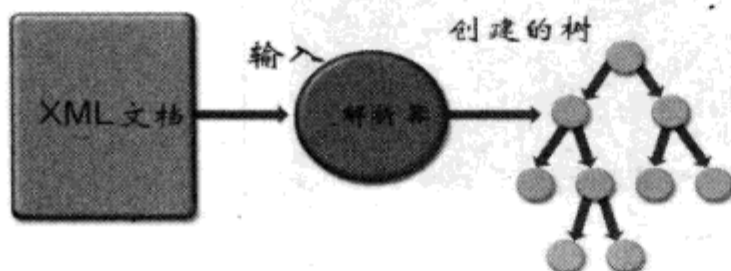


图 4-4 DOM 解析原理图

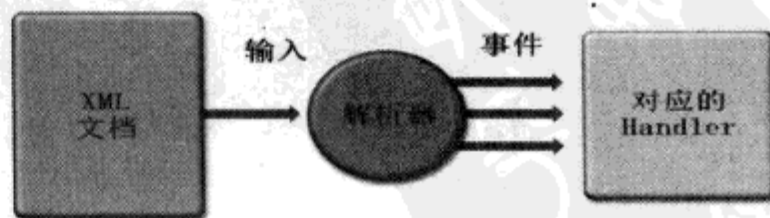


图 4-5 SAX 解析原理图

DOM 可以实现增加、删除、修改和读取功能,但是在文件很大时比较耗内存。而 SAX 对 XML 文件是一边读取一边处理,对内存要求很低,但是仅支持文件读取,不支持对文件的增加、删除和修改操作。

□ DOM 读文件

DOM 读取 XML 文件的示例代码如下:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class TestDOM1 {
    private static final String space = ":";

    private static final String file = "E:\\XML\\home.xml";

    public static void main(String args[]) {
        TestDOM1 test = new TestDOM1();
        try {
            test.parser();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 解析 XML 文件
     *
     * @throws Exception
     */
    public void parser() throws Exception {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(file);
        Element elt = doc.getDocumentElement();
        System.out.println(elt.getNodeName() + space + elt.getNodeValue());
        parserNode(elt);
    }

    /**
     * 递归遍历节点信息
     *
     * @param node
     */
    private void parserNode(Node node) {
        if (node != null) {
            if (node.getChildNodes().getLength() == 1) {
                System.out.println(node.getNodeName() + space
                    + node.getFirstChild().getNodeValue());
            }
            return;
        }
    }
}
```

```

        NodeList nodes = node.getChildNodes();
        for (int i = 0; i < nodes.getLength(); i++) {
            Node e = nodes.item(i);
            parserNode(e);
        }
    }
}

```

□ DOM 写文件

DOM 写文件的示例代码如下:

```

//向 XML 文件中添加节点信息
public void addNode() throws Exception {
    //创建解析器
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    //获取整个文档结构
    Document doc = db.parse("E:\\XML\\home.xml");
    //创建节点元素
    Element node = doc.createElement("成员");
    Element node1 = doc.createElement("姓名");
    Element node2 = doc.createElement("性别");
    Element node3 = doc.createElement("角色");

    Attr attr = doc.createAttribute("年龄");
    //创建文本节点
    Text text1 = doc.createTextNode("王致和");
    Text text2 = doc.createTextNode("男");
    Text text3 = doc.createTextNode("公公");
    Text attrText = doc.createTextNode("68");
    //建立节点之间的关系
    node1.appendChild(text1);
    node2.appendChild(text2);
    node3.appendChild(text3);

    //属性设置
    attr.appendChild(attrText);
    node.appendChild(node1);
    node.appendChild(node2);
    node.appendChild(node3);
    //添加属性
    node.setAttributeNode(attr);
    //获取根节点并添加节点目录
    doc.getDocumentElement().appendChild(node);
    //将内存中的结构转化到 XML 文件里
    transfor(doc);
}

//将内存中的结构转化到 XML 文件里

```



```
private void transfor(Document doc) throws Exception{
    TransformerFactory tf = TransformerFactory.newInstance();
    Transformer t = tf.newTransformer();
    DOMSource ds = new DOMSource(doc);
    StreamResult sr = new StreamResult(file);
    t.transform(ds, sr);
}
```

上述代码实现了向 XML 文件添加信息的功能, 如果修改和删除 XML 中的信息, 可以应用 `removeChild()` 和 `replaceChild()` 方法实现。

□ SAX 读文件

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

public class TestSAX {
    private static final String file = "E:\\XML\\home.xml";
    public static void main(String[] args) {
        TestSAX test = new TestSAX();
        try {
            test.parser1();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 使用 J2EE 自带的解析器进行解析
     * @throws Exception
     */
    private void parser1() throws Exception {
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser sp = spf.newSAXParser();
        XMLReader xr = sp.getXMLReader();
        SaxHandler saxHandler = new SaxHandler();
        xr.setContentHandler(saxHandler);
        xr.parse(file);
    }
}
```

在上述代码中, 为 SAX 解析器指定了文档处理对象, 类型为 `SaxHandler`, 该类的实现代码如下:

```
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.Locator;
import org.xml.sax.SAXException;

public class SaxHandler implements ContentHandler {

    public void characters(char[] arg0, int arg1, int arg2) throws SAXException {
        //过滤空格
    }
}
```

```
String s = new String(arg0, arg1, arg2);
if ("".equals(s.trim())) {
    System.out.print("");
} else {
    System.out.print(s.trim());
}
}

public void endDocument() throws SAXException {
    System.out.println("----解析文档结束! -----");
}

public void endElement(String arg0, String name, String qname)
    throws SAXException {
    System.out.println("</" + qname + ">");
}

public void endPrefixMapping(String arg0) throws SAXException {
}

public void ignorableWhitespace(char[] arg0, int arg1, int arg2)
    throws SAXException {
}

public void processingInstruction(String arg0, String arg1)
    throws SAXException {
}

public void setDocumentLocator(Locator arg0) {
}

public void skippedEntity(String arg0) throws SAXException {
}

public void startDocument() throws SAXException {
    System.out.println("-----解析文档开始! -----");
}

public void startElement(String arg0, String name, String qname,
    Attributes attrs) throws SAXException {
    System.out.print("<" + qname + " ");
    for(int i=0;i<attrs.getLength();i++) {
        System.out.print(attrs.getQName(i) + "=" + attrs.getValue(i));
    }
}
```



```
        System.out.println(">");
    }

    public void startPrefixMapping(String arg0, String arg1)
        throws SAXException {
    }
}
```

注意: DOM 和 SAX 除了使用 JDK 提供的解析器之外,还有很多第三方 API,例如 Apache 组织的 xercesImpl.jar 开发包。

疑难点评

XML 文件具有跨平台的特点,使用也比较广泛,例如 XML 配置文件、实现异构系统的整合,即在不同系统之间利用 XML 文件传递数据信息等。Java 在解析 XML 文件时可以使用 DOM 和 SAX 两种技术,两者都有各自的优缺点,在实际应用时,需要根据使用情况进行选取。

知识链接

FAQ4.27 如何读写 XML 文件中元素的属性?

FAQ4.27 如何读写 XML 文件中的元素属性?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

SAX 技术仅支持读操作,DOM 技术可以实现 XML 的读写操作。对 XML 文件的元素属性读写时,需要使用 DOM 技术实现。

DOM 在解析 XML 文档时,文档中每个元素都会转化为内存文档树中的节点,节点对象的类型为 Element。使用 Element 的 `getAttributeNode()` 和 `getAttribute()` 方法可以获取属性信息,通过 `setAttributeNode()` 和 `setAttribute()` 方法可以设置属性信息。

❑ 获取属性信息的示例代码如下:

```
/**
 * 功能: 解析属性
 * @param elt: 元素节点
 * @param name: 属性名
 */
private void parserAttr(Element elt, String name) {
    //String val = elt.getAttribute(name); // 获取属性值
    // System.out.println(val);
    Attr attr = elt.getAttributeNode(name); // 获取属性对象
```

```
        if(attr != null) {  
            System.out.println(attr.getName() + space+ attr.getValue());  
        }  
    }  
}
```

□ 设置属性信息的示例代码如下：

```
/**  
 * 功能：解析属性  
 * @Document doc 文档对象  
 * @param elt 元素节点  
 */  
private void setAttr(Document doc, Element elt) {  
    Attr attr = doc.createAttribute("年龄");  
    Text attrText = doc.createTextNode("68");  
    //属性设置  
    attr.appendChild(attrText);  
    //添加属性  
    elt.setAttributeNode(attr);  
    //elt.setAttribute(name, value); //设置属性值  
}
```

疑难点评

XML 文件有严格的语法规则，例如必须有一个根元素、元素的开始标记可以包含属性等。对于元素属性值的操作，在 Java 中可以通过 Element 类的方法实现。

知识链接

FAQ4.26 如何读写 XML 文件？

FAQ4.28 如何读写 CSV 格式的文件？

📖 难度系数：★★★★

📖 问题频率：86%

核心解答

CSV 是 Comma Separated Value（逗号分隔值）的英文缩写，属于纯文本文件，文件扩展名为 csv。Emp.csv 文件内容如下：

```
姓名,性别,工资  
张三,男,5000  
李四,男,3100  
张红,女,2800  
王五,男,3000
```

通常 CSV 文件开头不留空行或空格，内容以行为单位，每行记录一条数据，每项数据用英文逗号分隔。如果计算机安装了 Office Excel 软件，CSV 文件会默认使用该软件打开，因此可

以使用 CSV 文件替代一些功能简单的 Excel 文档的输出。Emp.csv 文件使用 Excel 软件打开后,效果如图 4-6 所示。

CSV 文件与普通的 TXT 文件不同的是,CSV 文件中的每项数据都是用逗号分隔。因此 Java 读写 CSV 文件与读写 TXT 文件相比,只是多一些逗号和换行符的处理。示例代码如下:

	A	B	C	D	E	F
1	姓名	性别	工资			
2	张三	男	5000			
3	李四	男	3100			
4	张红	女	2800			
5	王五	男	3000			
6						
7						
8						

图 4-6 Emp.csv 文件显示效果图

□ 写 CSV 文件

```
public static void main(String[] args) {
    try {
        FileWriter fw = new FileWriter("C:\\helloCsv.csv");
        fw.write("aaa,bbb,ccc,ddd,eee,fff,ggg,hhh\r\n");
        fw.write("aa1,bb1,cc1,dd1,ee1,ff1,gg1,hh1\r\n");
        fw.write("aaa\r\n");
        fw.write("aa2,bb2,cc2,dd2,ee2,ff2,gg2,hh2\r\n");
        fw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

□ 读 CSV 文件

```
public static void main(String[] args) {
    InputStreamReader fr = null;
    BufferedReader br = null;
    try {
        fr = new InputStreamReader(new FileInputStream(
            "C:\\helloCsv.csv"));
        br = new BufferedReader(fr);
        String rec = null;
        String[] argsArr = null;
        while ((rec = br.readLine()) != null) {
            System.out.println(rec);
            argsArr = rec.split(",");
            for (int i = 0; i < argsArr.length; i++) {
                System.out.println("num " + (i + 1) + ":" + argsArr[i]);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (fr != null)
                fr.close();
            if (br != null)
                br.close();
        } catch (IOException ex) {
            //
        }
    }
}
```



```
ex.printStackTrace();
```

注意：CSV 文件内容中的逗号为半角，不能使用全角。

疑难点评

CSV 格式的文件也属于纯文本文件，只是在文件打开时默认使用 Excel 软件。如果需要将系统的一些数据以 Excel 格式进行处理，可以考虑使用 CVS 文件，因为 Java 处理 CSV 文件比处理 Excel 文件要简单的多。

知识链接

FAQ4.10 如何读文件、写文件？

FAQ4.29 如何为图片文件生成缩略图？

📖 难度系数：★★★★★

📖 问题频率：80%

核心解答

在某些系统中，需要使用和管理很多图片，现在图片占用的资源空间越来越大，因此需要将图片缩小。

在实现缩略图的过程中，主要使用 `BufferedImage` 和 `ImageIO` 两个类。首先将图片信息读取到 `BufferedImage` 对象中，接着构造缩略图的 `BufferedImage` 对象，最后将缩略图输出。`ImageIO` 类提供了 `read()` 和 `write()` 方法，用于读写图片中的信息。

生成缩略图的示例代码如下：

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.awt.image.ColorModel;
import java.awt.image.WritableRaster;
import java.awt.*;
import java.awt.geom.AffineTransform;
import java.io.InputStream;
import java.io.File;
import java.io.FileOutputStream;

public class Test {
    /**
     * 生成缩略图
     * fromFileStr: 源图片路径
     * saveToFileStr: 缩略图路径
     */
}
```



```

* width: 缩略图的宽
* hight: 缩略图的高
*/
public static void saveImageAsJpg (String fromFileStr,String saveToFileStr,int width,int hight)
    throws Exception {
    BufferedImage srcImage;
    String imgType = "JPEG";
    if (fromFileStr.toLowerCase().endsWith(".png")) {
        imgType = "PNG";
    }
    File saveFile=new File(saveToFileStr);
    File fromFile=new File(fromFileStr);
    srcImage = ImageIO.read(fromFile);
    if(width > 0 || hight > 0)
    {
        srcImage = resize(srcImage, width, hight);
    }
    ImageIO.write(srcImage, imgType, saveFile);
}
/**
*将源图片的 BufferedImage 对象生成缩略图
* source: 源图片的 BufferedImage 对象
* targetW: 缩略图的宽
* targetH: 缩略图的高
*/
public static BufferedImage resize(BufferedImage source, int targetW, int targetH) {
    int type = source.getType();
    BufferedImage target = null;
    double sx = (double) targetW / source.getWidth();
    double sy = (double) targetH / source.getHeight();
    //这里想实现在 targetW, targetH 范围内实现等比缩放。如果不需要等比缩放
    //则将下面的 if else 语句注释掉即可
    if(sx>sy)
    {
        sx = sy;
        targetW = (int)(sx * source.getWidth());
    }else{
        sy = sx;
        targetH = (int)(sy * source.getHeight());
    }
    if (type == BufferedImage.TYPE_CUSTOM) {
        ColorModel cm = source.getColorModel();
        WritableRaster raster = cm.createCompatibleWritableRaster(targetW, targetH);
        boolean alphaPremultiplied = cm.isAlphaPremultiplied();
        target = new BufferedImage(cm, raster, alphaPremultiplied, null);
    } else
        target = new BufferedImage(targetW, targetH, type);
    Graphics2D g = target.createGraphics();
    g.setRenderingHint(RenderingHints.KEY_RENDERING, RenderingHints.VALUE_RENDER_QUALITY);
    g.drawRenderedImage(source, AffineTransform.getScaleInstance(sx, sy));
}

```

```
        g.dispose();
        return target;
    }
    //测试程序主入口方法
    public static void main (String argv[]) {
        try{
            //参数 1(源图片路径), 参数 2(缩略图路径), 参数 3(缩略图宽), 参数 4(缩略图高)
            Test.saveImageAsJpg("E:/Document/My Pictures/3.gif","c:/6.gif",50,50);
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

疑难点评

生成图片缩略图功能在图片处理时比较常见,不同的实现方法生成缩略图的清晰度也有一定的区别,读者在使用时也可以相互比较,择优选用。

知识链接

FAQ4.10 如何读文件、写文件?

FAQ4.30 如何操作 Excel 文件?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

由于 Java 流无法实现对 Excel 文件的读写操作,因此在项目中经常利用第三方开源的组件来实现。支持对 Excel 文件操作的第三方开源组件主要有 Apache 的 POI 和开源社区的 JXL。

(1) POI 组件的应用

POI 组件的下载地址为 <http://poi.apache.org/>, 该组件对 Excel 文档操作的示例代码如下:

❑ 写入 Excel 文件

```
/**
 *创建 excel 文件并写入信息
 * filename: 文件存放位置
 * @throws IOException
 */
public void writeExcel(String filename) throws IOException{
    //创建新的 Excel 工作簿
    HSSFWorkbook workbook = new HSSFWorkbook();

    //在 Excel 中新建一个工作表,起名字为 JSP
```



```

HSSFSheet sheet = workbook.createSheet("JSP");

//创建第 1 行
HSSFRow row = sheet.createRow(0);
//创建第 1 列
HSSFCell cell = row.createCell((short)0);
//定义单元格为字符串类型
cell.setCellType(HSSFCell.CELL_TYPE_STRING);
//在单元格中输入一些内容
cell.setCellValue("作者");
//创建第 2 列
cell = row.createCell((short)1);
//定义单元格为字符串类型
cell.setCellType(HSSFCell.CELL_TYPE_STRING);
//在单元格中输入一些内容
cell.setCellValue("编辑");
//新建一输出流
FileOutputStream fout = new FileOutputStream(filename);
//存盘
workbook.write(fout);

fout.flush();
//结束关闭
fout.close();
}

```

□ 读取 Excel 文件

```

/**
 * 创建 excel 文件并写入信息
 * filename: 文件存放位置
 * @throws IOException
 * @throws FileNotFoundException
 */
public void read(String filename) throws FileNotFoundException, IOException{
    HSSFWorkbook workbook = new HSSFWorkbook(new FileInputStream(filename));
    //按名引用 Excel 工作表
    HSSFSheet sheet = workbook.getSheet("JSP");
    //也可以用以下方式来获取 Excel 的工作表, 采用工作表的索引值
    //HSSFSheet sheet = workbook.getSheetAt(0);
    HSSFRow row = sheet.getRow(0);
    HSSFCell cell = row.getCell((short)0);
    //打印读取值
    System.out.println(cell.getStringCellValue());
    cell = row.getCell((short)1);
    System.out.println(cell.getStringCellValue());
}

```

(2) JXL 组件的应用

JXL 组件的下载地址为 <http://jexcelapi.sourceforge.net/>, 该组件对 Excel 文档操作的示例代

码如下:

□ 写入 Excel 文件

```
/**
 * 生成一个 Excel 文件
 * @param fileName: 要生成的 Excel 文件名
 */
public static void writeExcel(String fileName){
    WritableWorkbook wwb = null;
    try {
        //首先要使用 Workbook 类的工厂方法创建一个可写入的工作簿 (Workbook) 对象
        wwb = Workbook.createWorkbook(new File(fileName));
    } catch (IOException e) {
        e.printStackTrace();
    }
    if(wwb!=null){
        //创建一个可写入的工作表
        //Workbook 的 createSheet 方法有两个参数, 第 1 个是工作表的名称
        //第 2 个是工作表在工作簿中的位置
        WritableSheet ws = wwb.createSheet("sheet1", 0);

        //下面开始添加单元格
        for(int i=0;i<10;i++){
            for(int j=0;j<5;j++){
                //这里需要注意的是, 在 Excel 中, 第 1 个参数表示列, 第 2 个表示行
                Label labelC = new Label(j, i, "这是第" + (i+1) + "行, 第" + (j+1) + "列");
                try {
                    //将生成的单元格添加到工作表中
                    ws.addCell(labelC);
                } catch (RowsExceededException e) {
                    e.printStackTrace();
                } catch (WriteException e) {
                    e.printStackTrace();
                }
            }
        }

        try {
            //从内存中写入文件中
            wwb.write();
            //关闭资源, 释放内存
            wwb.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (WriteException e) {
            e.printStackTrace();
        }
    }
}
```


□ 读取文件

```

/**读取 Excel 文件的内容
 * @param file 待读取的文件
 * @return
 */
public static String readExcel(File file){
    StringBuffer sb = new StringBuffer();
    Workbook wb = null;
    try {
        //构造 Workbook (工作簿) 对象
        wb=Workbook.getWorkbook(file);
    } catch (BiffException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    if(wb==null)
        return null;

    //获得了 Workbook 对象之后, 就可以通过它得到 Sheet (工作表) 对象了
    Sheet[] sheet = wb.getSheets();

    if(sheet!=null&&sheet.length>0){
        //对每个工作表进行循环
        for(int i = 0; i < sheet.length; i++){
            //得到当前工作表的行数
            int rowNum = sheet[i].getRows();
            for(int j = 0; j < rowNum; j++){
                //得到当前行的所有单元格
                Cell[] cells = sheet[i].getRow(j);
                if(cells!=null&&cells.length>0){
                    //对每个单元格进行循环
                    for(int k = 0; k < cells.length; k++){
                        //读取当前单元格的值
                        String cellValue = cells[k].getContents();
                        sb.append(cellValue+"\t");
                    }
                }
                sb.append("\r\n");
            }
            sb.append("\r\n");
        }
        //最后关闭资源, 释放内存
        wb.close();
        return sb.toString();
    }
}

```

□ 搜索 Excel 是否包含指定的字符串

```
/**搜索某一个文件中是否包含某个关键字
 * @param file: 待搜索的文件
 * @param keyWord: 要搜索的关键字
 * @return
 */
public static boolean searchKeyWord(File file,String keyWord){
    boolean res = false;
    Workbook wb = null;
    try {
        //构造 Workbook (工作簿) 对象
        wb=Workbook.getWorkbook(file);
    } catch (BiffException e) {
        return res;
    } catch (IOException e) {
        return res;
    }
    }

    if(wb==null)
        return res;

    //获得了 Workbook 对象之后, 就可以通过它得到 Sheet (工作表) 对象了
    Sheet[] sheet = wb.getSheets();

    boolean breakSheet = false;

    if(sheet!=null&&sheet.length>0){
        //对每个工作表进行循环
        for(int i = 0; i < sheet.length; i++){
            if(breakSheet)
                break;

            //得到当前工作表的行数
            int rowNum = sheet[i].getRows();

            boolean breakRow = false;

            for(int j = 0;j < rowNum; j++){
                if(breakRow)
                    break;
                //得到当前行的所有单元格
                Cell[] cells = sheet[i].getRow(j);
                if(cells!=null&&cells.length>0){
                    boolean breakCell = false;
                    //对每个单元格进行循环
                    for(int k = 0; k < cells.length; k++){
                        if(breakCell)
                            break;
                        //读取当前单元格的值
                        String cellValue = cells[k].getContents();
                        if(cellValue==null)
```



```

        continue;
    if(cellValue.contains(keyWord)){
        res = true;
        breakCell = true;
        breakRow = true;
        breakSheet = true;
    }
}
}
}
}
//最后关闭资源, 释放内存
wb.close();
return res;
}

```

❑ 向 Excel 中插入图片

```

/**向 Excel 中插入图片
 * @param dataSheet: 待插入的工作表
 * @param col: 图片从该列开始
 * @param row: 图片从该行开始
 * @param width: 图片所占的列数
 * @param height: 图片所占的行数
 * @param imgFile: 要插入的图片文件
 */
public static void insertImg(WritableSheet dataSheet, int col, int row, int width,
    int height, File imgFile){
    WritableImage img = new WritableImage(col, row, width, height, imgFile);
    dataSheet.addImage(img);
}

```

注意: JXL 只支持 png 格式的图片, jpg 格式和 gif 格式都不支持。

疑难点评

在对 Excel 文件进行操作时, 通常需要使用一些第三方组件, 例如 POI 和 JXL 等。具体使用细节可以到组件网站阅读相关的使用文档。

知识链接

FAQ4.31 如何操作 Word 文件?

FAQ4.31 如何操作 Word 文件?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

现在有很多第三方组件都支持对 Word 文件的操作, 例如 jacob、iText、POI 和 java2word。

jacob 组件的功能最强大, 可以操作 Word、Excel 等格式的文件。该组件调用的是操作系统底层的 dll 文件, 需要在 system32 目录下放置 jacob.dll, 然后为工程添加 jacob.jar 文件的引用。

在使用 Java 操作 Word 文件时, jacob 组件是最常用的一个, 该组件的下载地址为 <http://sourceforge.net/projects/jacob-project/>, 推荐使用较新版本, 如果版本太低的话可能会报错。使用 jacob 操作 Word 文件的示例代码如下:

```
public class WordBean {
    private ActiveXComponent MsWordApp = null;
    private Dispatch document = null;

    //打开 Word 文档
    public void openWord(boolean makeVisible) {
        if (MsWordApp == null) {
            MsWordApp = new ActiveXComponent("Word.Application");
        }
        //设置 visible 属性
        Dispatch.put(MsWordApp, "Visible", new Variant(makeVisible));
    }

    //新建 Word 文档
    public void createNewDocument() {
        // 获取文档集合
        Dispatch documents = Dispatch.get(MsWordApp, "Documents").toDispatch();
        // 调用 Add 方法向文档集合中添加一个新的 Word 文件
        document = Dispatch.call(documents, "Add").toDispatch();
    }

    //向 Word 文件中写入字符串信息
    public void insertText(String textToInsert) {
        //获取当前执行写入的位置, 如果是新 Word 文件操作位置为文档的开始
        Dispatch selection = Dispatch.get(MsWordApp, "Selection").toDispatch();
        //将字符串写入
        Dispatch.put(selection, "Text", textToInsert);
    }

    //实现文件“另存为”功能
    public void saveFileAs(String filename) {
        Dispatch.call(document, "SaveAs", filename);
    }

    //实现文件打印功能
    public void printFile() {
        //采用默认打印机打印
    }
}
```



```
        Dispatch.call(document, "PrintOut");
    }

    //关闭文档
    public void closeDocument() {
        //0 关闭文档时不保存改变的信息
        //-1 关闭文档时保存改变的信息
        //-2 关闭文档时提示是否保存改变的信息, 请求确认
        Dispatch.call(document, "Close", new Variant(0));
        document = null;
    }

    //退出
    public void closeWord() {
        Dispatch.call(MsWordApp, "Quit");
        MsWordApp = null;
        document = null;
    }
}
```

在上述代码中, WordBean 类封装了对 Word 文件的相关操作, 例如打开、新建、写入、另存为和打印等。WordBean 类的使用示例如下:

```
public class Test {
    public static void main(String[] args) {
        WordBean word = new WordBean();
        word.openWord(true);
        word.createNewDocument();
        word.insertText("Hello word");
    }
}
```

疑难点评

Java 对 Word 文档的操作需要通过第三方组件实现, 例如 jacob、iText、POI 和 java2word 等。具体使用细节可以到组件网站阅读相关的使用文档。

知识链接

FAQ4.30 如何操作 Excel 文件?



第5章

Java GUI 编程

Java GUI 是指 Java 图形用户界面编程, 主要包含 AWT 和 Swing 两个方面。AWT 是 Java GUI 编程的基础, 本章首先介绍了一些 AWT 相关知识的疑难问题, 内容主要涉及 AWT 容器、布局管理器、AWT 事件处理机制等; 然后又介绍了另一种 GUI 技术 Swing, 内容主要涉及常用 Swing 组件介绍、Swing 特殊组件及特性介绍等。通过对本章的学习, 读者可以对 AWT 和 Swing 两种 GUI 技术深入了解, 例如常用组件有哪些、如何为组件添加事件处理以及记事本和贪吃蛇等游戏的实现等。

FAQ5.01 什么是 Java GUI? Swing 与 AWT 有什么关系?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

Java GUI (Graphics User Interface), 即 Java 图形用户接口。它是 Java 为用户提供的界面编程接口, 而 AWT 则是抽象窗口工具 (Abstract Window Tools) 的缩写, 它是一个用于 GUI 编程的基本类库, 为 Java 应用程序或 Applet 提供基本的图形组件。

AWT 作为一个窗口工具, 为图形用户界面编程提供相关的 API 和方法, 同时它也是一个窗口框架, 它从不同的特定平台的窗口系统 (例如 Windows GUI 和 Unix Motif) 中抽象出共同组件, 在运行的时候, 将这些组件的创建和动作委托给程序所在的运行平台 (例如 Windows、Unix 或 Solaris 等) 的图形工具进行处理。这样在编写 AWT 图形应用时, 只需要指定界面组件的位置和行为, Java 会自动创建真正使用的、与平台一致的对等体, 并与操作系统的图形界面风格保持一致。例如在 Windows 操作系统中表现出 Windows 的风格, 而在 Unix 操作系统中表现出 Unix 的风格, 这就在图形界面编程上也实现了 Java 所宣扬的“一次编写, 到处运行”的思想。

在实际应用中, 由于需要去迎合所有主流操作系统的界面设计, 即 AWT 组件只能是采用这些操作系统上图形组件的交集, 这就大大限制了 AWT 的使用。因此, 出现了一个新的图形界面库 “Swing”, 它在很大程度上弥补了 AWT 组件的不足。

1996年, Netscape 公司开发了一个工作方式完全不同的 GUI 库, 称为 IFC(Internet Foundation Classes), 它的所有图形组件, 例如文本框、按钮等, 可以根据操作系统的显示风格进行自我调整, 从而真正实现了图形界面显示风格的跨平台。

由于 AWT 在图形组件方面表现出局限性, 因此 Sun 公司和 Netscape 公司进行合作, 创建了一套新的界面库 Swing。Swing 提供了一套功能强大且界面风格丰富的组件, 可实现 AWT 组件所能实现的各种功能。Swing 除了具有与 AWT 功能等价的组件之外, 还提供了一些功能更加强大的组件, 例如表格组件 JTable 等。

虽然 Swing 对 AWT 原有组件都提供了替代组件, 但是 Swing 并不是 AWT 的替代品, 而是对 AWT 的补充和加强, Swing 中的很多组件类都是从 AWT 继承过来的, 而且依然采取 AWT 方式添加事件处理。

在 Swing 中, 几乎所有的可视组件都可以作为容器使用, 为了和 AWT 中组件类加以区分, Swing 组件类的前面都是以字母 “J” 开始, 例如 JButton、JFrame 等。

疑难点评

AWT 和 Swing 都是 Java GUI 的编程类库, Swing 是以 AWT 组件库为基础的, 对原有 AWT 组件进行了扩展。虽然 Swing 为 AWT 的每一个组件都提供了替代品, 但为组件添加事件处理的操作是相同的。

知识链接

FAQ5.02 什么是布局管理器？常用的布局管理器有哪些？

FAQ5.02 什么是布局管理器？常用的布局管理器有哪些？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

为了在不同的平台环境中都能使组件获得理想的尺寸, 在 Java 语言中提供了布局管理器 LayoutManager 对容器进行管理。AWT 中常用的布局管理器有 FlowLayout、BorderLayout、GridLayout、CardLayout 和 GridBagLayout 等。

□ FlowLayout

FlowLayout 布局管理器, 又称流布局管理器, 即组件从左到右按顺序配置在 Container 中, 如果到达右边界, 则会折回到下一行中, 这就像水流向一个方向 (从左到右) 流动, 遇到边界折回换行一样。Panel 和 Applet 容器都是默认采用 FlowLayout 布局管理器。

示例代码如下:


```
import java.awt.Button;
import java.awt.FlowLayout;
import java.awt.Frame;

public class TestFlowLayout {

    public static void main(String[] args) {
        TestFlowLayout tf = new TestFlowLayout();
        tf.init();
    }

    public void init() {
        //定义一个 Frame 容器
        Frame f = new Frame();
        //定义按钮
        Button left_button = new Button("left");
        Button center_button = new Button("center");
        Button right_button = new Button("right");
        //为 Frame 设置流布局管理器
        f.setLayout(new FlowLayout(FlowLayout.LEFT));
        //添加按钮组件
        f.add(left_button);
        f.add(center_button);
        f.add(right_button);
        //设置初始位置和大小
        f.setBounds(300, 400, 200, 200);
        //显示
        f.setVisible(true);
    }
}
```

在上述代码中,通过 `setLayout()` 方法将 `Frame` 的布局管理器设置为 `FlowLayout` 方式,然后将 3 个 `Button` 按钮添加到 `Frame` 中。程序运行效果如图 5-1 所示。

❑ BorderLayout

`BorderLayout` 将容器空间分为 `EAST`、`SOUTH`、`WEST`、`NORTH` 和 `CENTER` 等 5 个区域,每个区域只能放一个组件和容器组件。`BorderLayout` 是 `Frame` 和 `Dialog` 容器的默认管理器。如果在同一个区域中放入多个 `Component`,后放入的组件会将原来的覆盖,放置在各个区域中的组件的大小根据所处区域的大小而变化。

示例代码如下:

```
import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Frame;

public class TestBorderLayout {
```

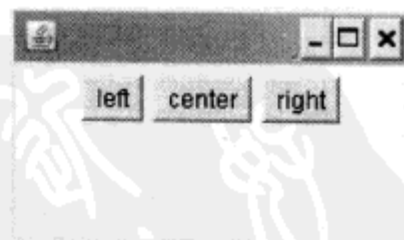


图 5-1 FlowLayout 布局效果

```
public static void main(String[] args) {
    TestBorderLayout tbl = new TestBorderLayout();
    tbl.init();
}

public void init() {
    //定义一个 Frame 容器
    Frame f = new Frame();
    //定义按钮
    Button east_button = new Button("EAST");
    Button south_button = new Button("SOUTH");
    Button west_button = new Button("WEST");
    Button northcenter_button = new Button("NORTH");
    Button center_button = new Button("CENTER");
    //为 Frame 设置流布局管理器
    f.setLayout(new BorderLayout());
    //添加按钮组件
    f.add(east_button, BorderLayout.EAST);
    f.add(south_button, BorderLayout.SOUTH);
    f.add(west_button, BorderLayout.WEST);
    f.add(northcenter_button, BorderLayout.NORTH);
    f.add(center_button, BorderLayout.CENTER);

    //设置初始位置和大小
    f.setBounds(500, 300, 300, 200);
    //显示
    f.setVisible(true);
}
}
```

在上述代码中，通过 `setLayout()` 方法将 `Frame` 的布局管理器设置为 `BorderLayout` 方式，然后将 5 个 `Button` 按钮分别添加到 `Frame` 的 5 个区域中。程序运行效果如图 5-2 所示。

❑ GridLayout

`GridLayout` 又称为网状布局，它将组件配置在纵横交错分割开的网状格子当中，从左到右，从上到下，分割开的格子大小相同。放置在 `GridLayout` 中的组件大小根据组件所处区域的大小决定。

示例代码如下：

```
import java.awt.Button;
import java.awt.Frame;
import java.awt.GridLayout;

public class TestGridLayout {
```

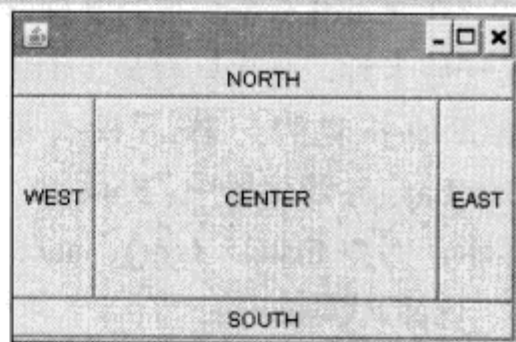


图 5-2 BorderLayout 布局效果


```

public static void main(String[] args) {
    TestGridLayout tg = new TestGridLayout();
    tg.init();
}

public void init() {
    String name[] = { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
        "+", "-", "*", "/", "=", "." };
    Button button[] = new Button[name.length];

    //定义一个 Frame 容器
    Frame f = new Frame();
    //添加按钮
    for (int i = 0; i < name.length; i++) {
        button[i] = new Button(name[i]);
        f.add(button[i]);
    }
    //为 Frame 设置为 4 行 4 列网格布局管理器, 水平垂直间隙均为 4
    f.setLayout(new GridLayout(4, 4, 4, 4));
    //设置初始位置和大小
    f.setBounds(500, 300, 200, 120);
    //显示
    f.setVisible(true);
}
}

```

本示例中将 Frame 容器设置为 4 行 4 列的 GridLayout 布局管理器, 并添加按钮分别放置于对应区域中。程序运行效果如图 5-3 所示。

❑ CardLayout

在 CardLayout 布局管理器中, 它将放置于容器中的组件看成是一叠卡片, 每次只有最上面的那个组件才可见。这就像一叠放置整齐的扑克牌一样, 只有最上面的那一张才能被看到。在 CardLayout 管理器中, 提供了一些方法用于控制和调整这些组件的显示, 例如 `fitst()`、`last()`、`next()` 和 `show()` 等方法。

示例代码如下:

```

import java.awt.Button;
import java.awt.CardLayout;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.Panel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TestCardLayout implements ActionListener {

```

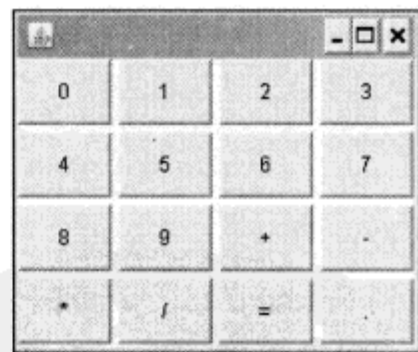


图 5-3 GridLayout 布局效果

```
private CardLayout cl = null; // 定义一个 CardLayout 布局管理器
private Panel p1 = null;
private Panel p2 = null;

public static void main(String[] args) {
    TestCardLayout tcl = new TestCardLayout();
    tcl.init();
}

public void init() {

    String name[] = { "A", "B", "C", "D", "E", "F", "G" };
    Label card[] = new Label[name.length];
    Frame f = new Frame();
    f.setLayout(new FlowLayout(FlowLayout.CENTER)); // 设置 Frame 为流布局管理器模式
    p1 = new Panel();
    cl = new CardLayout(); // new 一个 CardLayout 布局管理器
    p1.setLayout(cl); // 将 p1 设置为 CardLayout 布局管理器模式
    for (int i = 0; i < name.length; i++) {
        card[i] = new Label(name[i]);
        p1.add(name[i], card[i]);
    }

    p2 = new Panel();

    Button previous = new Button("previous");
    Button next = new Button("next");
    // 为按钮添加事件
    previous.addActionListener(this);
    next.addActionListener(this);
    p2.add(previous);
    p2.add(next);

    f.add(p1);
    f.add(p2);
    f.setBounds(300, 400, 300, 300);
    f.setVisible(true);

}

public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    // 事件处理
    if (command.equals("previous")) {
        cl.previous(p1);
    } else if (command.equals("next")) {
        cl.next(p1);
    }
}
}
```


在上述程序中,定义了一个 Panel 容器 p1,并将其布局设置为 CardLayout 方式,然后在容器 p1 中定义了 7 个 Label 对象以 Card 方式放置。在另一个 Panel 容器中定义了 2 个按钮,并添加事件处理,根据不同命令实现向前后翻页的功能。程序运行效果如图 5-4 所示。

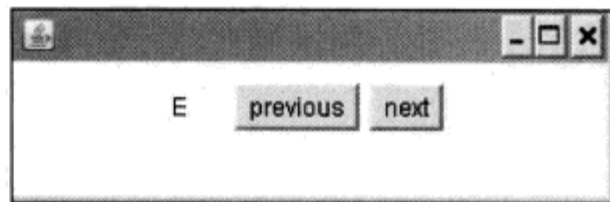


图 5-4 CardLayout 布局效果

疑难点评

布局管理器用于管理容器中组件的布局,常用的布局管理器有 FlowLayout、BorderLayout、GridLayout、CardLayout 和 GridBagLayout 等。在容器中如果使用布局管理器,那么组件的大小将由布局管理器控制,不能通过 setSize()和 setBounds()方法改变大小和位置。如果使用者不想应用布局管理器,可以将容器的 layout 属性设置为 null,即 setLayout(null),但此时需要使用 setSize()和 setBounds()方法自己定义组件的大小和位置。使用这种方式读者可以将组件在容器中任意摆放和设置大小,比使用布局管理器灵活,但缺点就是编码量较大,费时费力。

FAQ5.03 如何在窗体中显示一张图片?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

AWT 组件对图片显示功能的支持比较差,使用 Swing 组件库中的 JLabel 组件,可以非常方便地显示图片,具体实现代码如下:

```
public static void main(String[] args) {  
    JLabel jl = new JLabel();  
    jl.setIcon(new ImageIcon("307.jpg"));  
    JFrame jf = new JFrame();  
    jf.setSize(500, 500);  
    jf.add(jl);  
    jf.setVisible(true);  
}
```

程序运行的效果如图 5-5 所示。

上述代码虽然实现了图片的显示,但是显示大小为原图片的大小,可以通过下列代码自定义 JLabel 中图片的大小。

```
public static void main(String[] args) {  
    JLabel jl = new JLabel();  
    ImageIcon image = new ImageIcon("307.jpg");  
    image.setImage(image.getImage().getScaledInstance(100,100,Image.SCALE_DEFAULT));  
    jl.setIcon(image);  
}
```

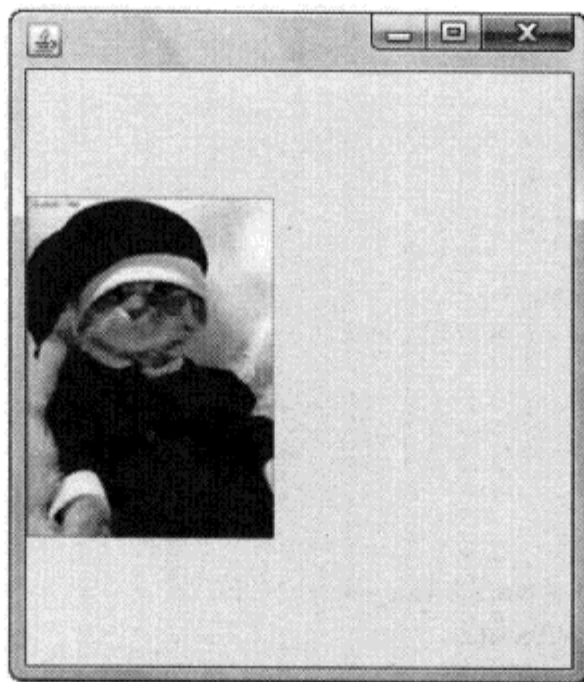


图 5-5 显示图片示例的效果

```
JFrame jf = new JFrame();  
jf.setSize(500, 500);  
jf.add(jl);  
jf.setVisible(true);  
}
```

疑难点评

利用 JLabel 组件可以在窗体上显示图片, 可以增强图形界面的友好性。在实现一些图片管理系统时, 也经常需要用到此功能。JLabel 组件属于 Swing 组件库。

知识链接

FAQ5.09 如何为窗体添加关闭事件?

FAQ5.10 如何实现窗体菜单功能?

FAQ5.04 如何为容器添加滚动条功能?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

默认情况下, Frame、Panel 等容器没有滚动条, 如果容器内放置的组件过多会出现组件显示不完整的情况。AWT 提供了 ScrollPane 类来实现滚动条功能, 具体实现方法很简单, 首先需要将组件放置于 ScrollPane 容器中, 然后将 ScrollPane 放置于 Frame 或者 Panel 等容

器中即可。

示例代码如下：

```
import java.awt.*;

public class TextScrollPane {

    public static void main(String[] args) {
        TextScrollPane tsp = new TextScrollPane();
        tsp.init();
    }

    public void init() {
        Frame f = new Frame();
        // f.setLayout(new FlowLayout(FlowLayout.CENTER));
        ScrollPane sp = new ScrollPane();
        Panel p = new Panel();
        p.setLayout(new GridLayout(3, 3, 5, 5));
        // sp.setLayout(new FlowLayout(FlowLayout.CENTER));
        String[] name = { "A", "B", "C", "D", "E", "F", "G", "H", "I", "J",
                        "K", "L", "M", "O", "P", "Q", "R", "S", "T", "U", "V" };
        Button[] button = new Button[name.length];
        //创建多个按钮并放置于 ScrollPane 容器中
        for (int i = 0; i < name.length; i++) {
            button[i] = new Button(name[i]);
            p.add(button[i]);
        }
        //将 ScrollPane 放置于 Frame 容器中
        sp.add(p);
        f.add(sp);
        //设置初始位置和大小
        f.setBounds(500, 300, 200, 200);
        //显示
        f.setVisible(true);
    }
}
```

在示例中,没有直接将装有 Button 组件的 Panel 容器放置在 Frame 中,而是放在了 ScrollPane 容器中,然后再将容器放在 Frame 中。这样,就可以产生滚动条的功能了。程序运行效果如图 5-6 所示。

疑难点评

为 Frame、Panel 等容器增添滚动条功能,可以方便用户浏览容器中全部内容。在使用 JTable 表格组件时,如果显示的数据记录非常多,也经常为其添加滚动条功能方便用户拖动滚动条浏览。



图 5-6 ScrollPane 容器效果

知识链接

FAQ5.14 如何使用表格组件?

FAQ5.05 如何实现一个打开文件或者是存储文件的对话框?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

在 java.awt 包中, 提供了 FileDialog 类用于实现打开或存储文件的功能, 在创建一个 FileDialog 对话框时, 需要给它指明一个父窗口, 即属主。构造方法如下:

```
FileDialog(Dialog parent, String title, int mode)
sFileDialog(Frame parent, String title, int mode)
```

其中第 1 个参数表示该 FileDialog 的属主, 第 2 个参数表示其标题, 第 3 个参数的取值可以是 LOAD 或者 SAVE, 分别表示打开或者存储文件。

示例代码如下:

```
package question_14;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TestFileDialog {

    public static void main(String[] args) {
        TestFileDialog tfd = new TestFileDialog();
        tfd.init();
    }

    Frame f = new Frame("My Frame");
    //创建 FileDialog 对话框, 属主为 Frame f
    FileDialog fd_l = new FileDialog(f, "My FileDialog", FileDialog.LOAD);
    FileDialog fd_s = new FileDialog(f, "My FileDialog", FileDialog.SAVE);

    public void init() {
        //定义按钮并添加单击事件
        Button b1 = new Button("LOAD");
        b1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                fd_l.setVisible(true);
            }
        });
        Button b2 = new Button("SAVE");
```

```
b2.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        fd_s.setVisible(true);  
    }  
});  
//设置 Frame 为流布局  
f.setLayout(new FlowLayout());  
f.add(b1);  
f.add(b2);  
//设置初始位置和大小  
f.setBounds(500, 300, 200, 120);  
//显示  
f.setVisible(true);  
}
```

在上述示例中，创建了两个 FileDialog 对话框，分别通过两个 Button 事件触发，其运行效果如图 5-7 和图 5-8 所示。

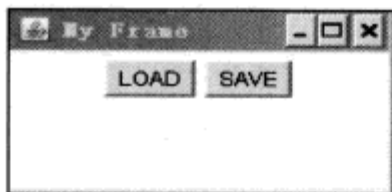


图 5-7 程序运行效果

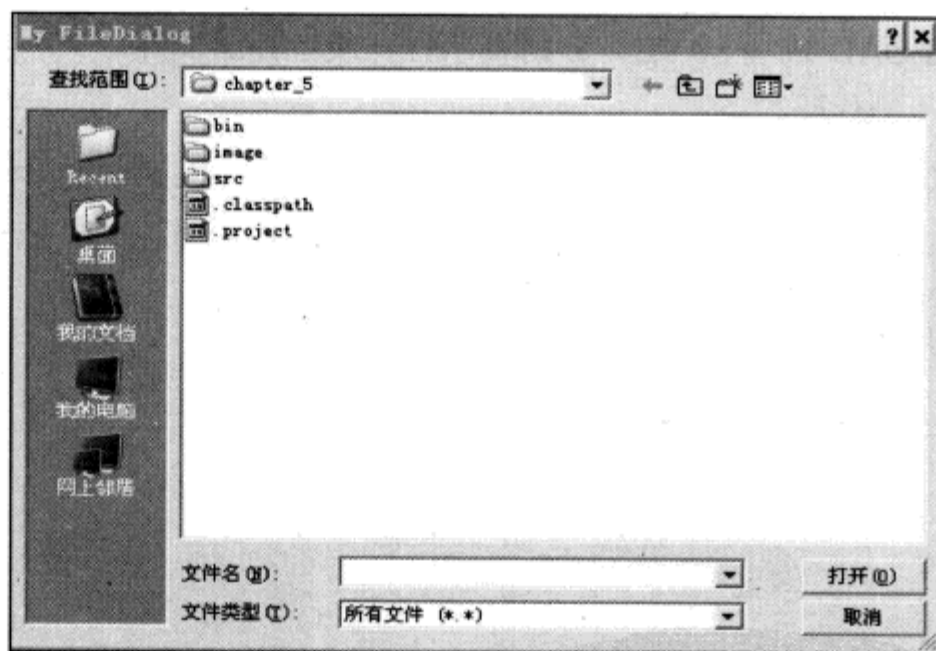


图 5-8 弹出对话框效果

疑难点评

在保存文件和打开文件时，一般都会弹出一个对话框，用户通过对话框选择文件。该功能可以通过 AWT 中的 FileDialog 组件实现，也可以通过 Swing 中的 JFileChooser 组件实现。如果想实现颜色选择器，可以使用 Jcolor Chooser 组件，其使用方式与 JFileChooser 相似。

知识链接

FAQ5.06 如何实现弹出消息框的功能？

FAQ5.07 如何使用 Dialog 对话框？

FAQ5.06 如何实现弹出消息框的功能?

📖 难度系数: ★★★

📖 问题频率: 95%

核心解答

在实现一些功能时, 需要以消息框的形式弹出, 给用户一些提示性信息, 例如操作成功、用户名不对等。

JOptionPane 类可实现弹出对话框的功能, 通过该类可方便地弹出要求用户提供值或向其发出通知的标准对话框。

(1) JOptionPane 简介

JOptionPane 提供了很多方法, 可用于弹出各种风格的对话框, 主要方法及其功能描述如表 5-1 所示。

表 5-1 JOptionPane 主要方法描述

方 法 名	描 述
showConfirmDialog	询问一个确认问题, 例如 yes/no/cancel
showInputDialog	提示要求某些输入
showMessageDialog	告知用户某事已发生
showOptionDialog	上述 3 项的统一

上述 showXxxDialog() 方法弹出的对话框都是有模式的, 对话框在用户交互完成之前, 都一直阻塞当前线程, 只有关闭对话框后才能进行后续的操作。

上述方法在使用时, 需要指定若干参数, 具体如下。

□ parentComponent

定义此对话框的父窗口组件。该参数有两种使用方式, 一种是将包含对话框的 Frame 作为参数值, 对话框的位置使用其屏幕坐标, 一般情况下, 将对话框紧靠组件置于其之下; 另一种是使用 null, 在这种情况下, 将使用默认的 Frame 作为父窗口, 并且对话框将居中位于屏幕上。

□ message

定义对话框中的描述消息, 该消息经常采用 String 类型设置。

□ messageType

定义 message 的样式。外观管理器布置的对话框可能因此值而异, 并且通常提供默认图标。该参数值可以为 ERROR_MESSAGE、INFORMATION_MESSAGE、WARNING_MESSAGE、QUESTION_MESSAGE 和 PLAIN_MESSAGE。

❑ optionType

定义在对话框底部显示选项按钮的集合。该参数值可以为 DEFAULT_OPTION、YES_NO_OPTION、YES_NO_CANCEL_OPTION 和 OK_CANCEL_OPTION。

❑ title

定义对话框的标题。

❑ icon

定义要在对话框中显示的图标。

(2) JOptionPane 使用示例

使用 JOptionPane 弹出各种类型对话框的示例代码如下。

❑ 显示一个错误对话框，该对话框显示的 message 为“alert”。

```
JOptionPane.showMessageDialog(null, "alert", "alert", JOptionPane.ERROR_MESSAGE);
```

❑ 显示一个内部信息对话框，该对话框显示的 message 为“information”。

```
JOptionPane.showInternalMessageDialog(frame, "information", "information", JOptionPane.INFORMATION_MESSAGE);
```

❑ 显示一个信息面板，其 options 为“yes/no”；message 为“choose one”。

```
JOptionPane.showConfirmDialog(null, "choose one", "choose one", JOptionPane.YES_NO_OPTION);
```

❑ 显示一个内部信息对话框，其 options 为“yes/no/cancel”；message 为“please choose one”，并具有 title 信息。

```
JOptionPane.showInternalConfirmDialog(frame, "please choose one", "information", JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE);
```

❑ 显示一个警告对话框，其 options 为 OK、CANCEL；title 为“Warning”；message 为“Click OK to continue”。

```
Object[] options = { "OK", "CANCEL" };  
JOptionPane.showOptionDialog(null, "Click OK to continue", "Warning",  
JOptionPane.DEFAULT_OPTION, JOptionPane.WARNING_MESSAGE,  
null, options, options[0]);
```

❑ 显示一个要求用户键入字符串的对话框。

```
String inputValue = JOptionPane.showInputDialog("Please input a value");
```

❑ 显示一个要求用户选择的对话框。

```
Object[] possibleValues = { "First", "Second", "Third" };  
Object selectedValue = JOptionPane.showInputDialog(null,  
"Choose one", "Input",  
JOptionPane.INFORMATION_MESSAGE, null,  
possibleValues, possibleValues[0]);
```

当其中一个 showXxxDialog() 方法返回整数时，返回值可能为 YES_OPTION、NO_OPTION、CANCEL_OPTION、OK_OPTION 和 CLOSED_OPTION。

疑难点评

JOptionPane 组件提供了多种对话框的显示样式，有只包含确认按钮的提示框，也有包含确定和取消按钮的确认框等。

知识链接

FAQ5.05 如何实现一个打开文件或者是存储文件的对话框?

FAQ5.7 如何使用 Dialog 对话框?

FAQ5.07 如何使用 Dialog 对话框?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

Java.awt.Dialog 类可用于实现对话框的功能,它也有很多子类,可用于显示不同风格的对话框,例如 FileDialog。在创建 Dialog 对象时,需要为它指明父窗口(即属主),构造方法如下:

```
Dialog(Dialog owner, String title, boolean modal, GraphicsConfiguration gc)
```

```
Dialog(Frame owner, String title, boolean modal, GraphicsConfiguration gc)
```

使用上述构造方法创建 Dialog 对象时,需要指定所有者、标题、模式和 GraphicsConfiguration 参数。第 1 个参数 owner 表示的该对话框的属主,即该对话框的所有者;第 2 个参数 title 为该对话框的标题;第 3 个参数 modal 为该对话框所选取的模式,如果为真,则 dialog 将阻断输入到其他应用程序窗口显示的内容;第 4 个参数 gc 表示的是目标屏幕设备的 GraphicsConfiguration,如果为 null,则使用与所拥有的 Dialog 相同的 GraphicsConfiguration。

Dialog 对象默认状态是不可见的,需要使用 setVisible()方法进行显示设置。使用 Dialog 的示例代码如下:

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TestDialog {

    public static void main(String[] args) {
        TestDialog td = new TestDialog();
        td.init();
    }

    Frame f = new Frame("My Frame");
    Dialog d = new Dialog(f, "My Dialog", true);

    public void init() {
        //定义一个按钮,并添加单击事件弹出对话框
        Button b = new Button("Click Me");
        b.addActionListener(new ActionListener() {
```

```
        public void actionPerformed(ActionEvent e) {
            d.setVisible(true);
        }

    });
    //新建一个标签
    Label l = new Label("My First Dialog", Label.CENTER);
    //将标签放置于对话框 Dialog d 中
    d.add(l);
    //设置对话框起始位置和大小
    d.setBounds(550, 350, 200, 120);
    //设置 Frame 为流布局
    f.setLayout(new FlowLayout());
    f.add(b);
    //设置初始位置和大小
    f.setBounds(500, 300, 200, 120);
    //显示
    f.setVisible(true);
}
}
```

上述代码运行结果如图 5-9 所示。

疑难点评

Dialog 表示一个对话框，它必须从属于某个窗口，因此在创建对话框时，务必指明其属主，它的属主可以是 Frame 或者其他的 Dialog。

知识链接

FAQ5.05 如何实现一个打开文件或者是存储文件的对话框？

FAQ5.06 如何实现弹出消息框的功能？

FAQ5.08 如何为按钮添加单击事件？

📖 难度系数：★★★★

📖 问题频率：95%

核心解答

给一个 Button 按钮添加鼠标单击事件有以下几种方法可以实现。

- ❑ 通过当前类实现 ActionListener 接口并实现 actionPerformed() 方法。
- ❑ 通过内部类处理，同样需要实现 ActionListener 接口并实现 actionPerformed() 方法。

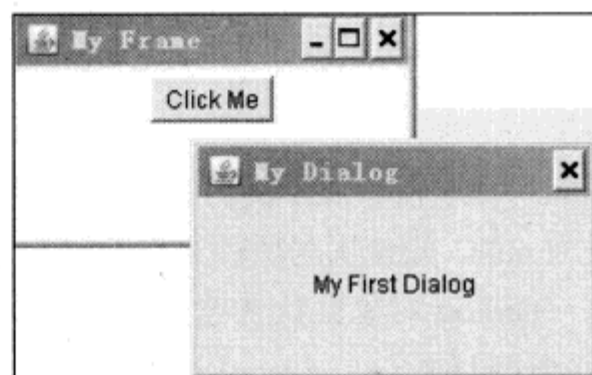


图 5-9 弹出消息对话框

- 通过匿名类来实现事件处理。
- 使用一个实现 ActionListener 接口的外部类来处理。

上述 4 种途径的实现机制都是一致的, 都是通过 ActionListener 接口来实现委派式事件监听和处理, 至于这 4 种方法的选取, 则根据具体的情况而定, 没有硬性的要求。

示例代码如下:

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

public class TestButtonEvent implements ActionListener {

    public static void main(String[] args) {
        TestButtonEvent tbe = new TestButtonEvent();
        tbe.init();
    }

    Frame f;
    Button button1;
    Button button2;
    Button button3;

    public void init() {
        f = new Frame();
        button1 = new Button("当前类事件处理");
        button2 = new Button("匿名类事件处理");
        button3 = new Button("内部类事件处理");
        //当前类实现按钮单击事件处理
        button1.addActionListener(this);
        //匿名类实现按钮单击事件处理
        button2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                //获得 3 个随机数
                Random ran = new Random();
                int r = ran.nextInt(225);
                int g = ran.nextInt(225);
                int b = ran.nextInt(225);
                //创建 2 个 Color 类实例
                Color c1 = new Color(r, g, b);
                Color c2 = new Color(b, g, r);
                //分别为 button 和 f 设置背景色
                button2.setBackground(c1);
                f.setBackground(c2);
            }
        });
        //内部类实现按钮单击事件处理
        button3.addActionListener(new ButtonEvent());
    }
}
```

```
//设置 Frame 为流布局
f.setLayout(new FlowLayout(FlowLayout.CENTER));
//添加按钮
f.add(button1);
f.add(button2);
f.add(button3);
//设置初始位置和大小
f.setBounds(500, 300, 200, 120);
//显示
f.setVisible(true);
}

//当前类实现 ActionListener 接口并实现 actionPerformed()方法处理按钮事件
public void actionPerformed(ActionEvent e) {
    //获得 3 个随机数
    Random ran = new Random();
    int r = ran.nextInt(225);
    int g = ran.nextInt(225);
    int b = ran.nextInt(225);
    //创建 2 个 Color 类实例
    Color c1 = new Color(r, g, b);
    Color c2 = new Color(b, g, r);
    //分别为 button 和 f 设置背景色
    button1.setBackground(c1);
    f.setBackground(c2);
}

//内部类实现按钮单击事件处理
class ButtonEvent implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        //获得 3 个随机数
        Random ran = new Random();
        int r = ran.nextInt(225);
        int g = ran.nextInt(225);
        int b = ran.nextInt(225);
        //创建 2 个 Color 类实例
        Color c1 = new Color(r, g, b);
        Color c2 = new Color(b, g, r);
        //分别为 button 和 f 设置背景色
        button3.setBackground(c1);
        f.setBackground(c2);
    }
}
}
```

在示例中分别对前 3 种方法举例说明了按钮事件的处理, 单击按钮将改变相应按钮和 Frame 的背景颜色, 程序运行效果如图 5-10 所示。

疑难点评

按钮单击事件的处理类在编写时需要实现 ActionListener 接口, 并将事件处理对象注册给按钮组件。这样当按钮发生单击事件时将调用事件处理对象中的特定方法进行处理。

知识链接

FAQ5.09 如何为窗体添加关闭事件?

FAQ5.10 如何实现窗体菜单功能?

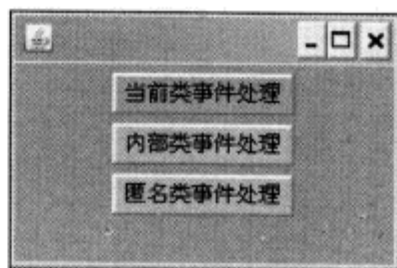


图 5-10 为按钮添加事件示例的效果

FAQ5.09 如何为窗体添加关闭事件?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

窗口关闭事件的处理需要使用 WindowListener 事件监听器, WindowListener 负责监听与窗体相关的各种事件, 例如关闭、最大化和最小化等。

示例代码如下:

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class TestExit {

    public static void main(String[] args) {
        TestExit te = new TestExit();
        te.init();
    }

    Frame f;
    Button b;
    Dialog d;

    public void init() {
```



```
f = new Frame("My Frame");
b = new Button("Click Me");
b.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        d.setVisible(true);
    }
});
d = new Dialog(f, "My Dialog", true);
//匿名类添加窗口关闭事件
d.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0); //安全退出
        d.dispose(); //隐藏窗口
    }
});
//设置初始位置和大小
d.setBounds(580, 340, 200, 120);
//显示
f.add(b);
//设置初始位置和大小
f.setBounds(500, 300, 200, 120);
//显示
f.setVisible(true);
}
```

在上述代码中，给窗口添加了 Window Listener 事件监听器，用于监听关闭按钮事件，并在 windowClosing() 方法中调用 System.exit() 方法安全退出。程序运行效果如图 5-11 所示。

在关闭 Frame 时也可以使用 dispose() 方法，但该方法的功能是释放由此 Window、其子组件及其拥有的所有子组件所使用的所有本机屏幕资源。即这些组件的资源将被破坏，它们使用的所有内存都将返回到操作系统，并将它们标记为不可显示。

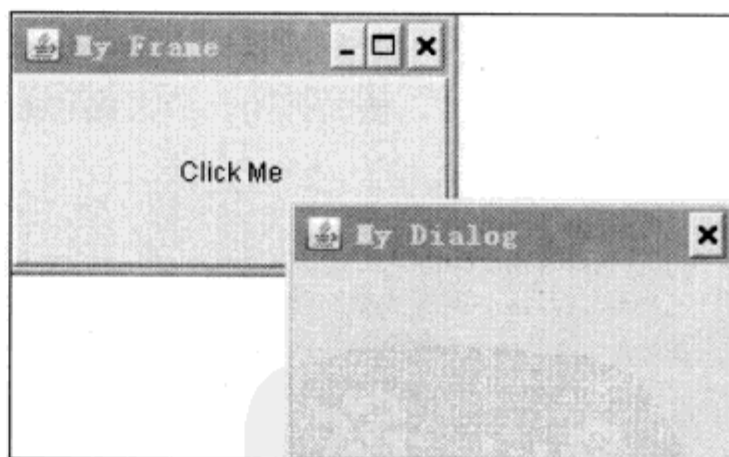


图 5-11 窗口关闭事件

疑难点评

在 AWT 中，Frame 窗体组件默认情况下没有处理关闭操作，要想实现窗体关闭功能，需要为窗体组件指定特定的处理对象。在 Swing 中，JFrame 窗体组件可以通过 setter() 方法设置关闭操作，使用起来比 AWT 的 Frame 方便的多。

知识链接

FAQ5.10 如何实现窗体菜单功能？

FAQ5.10 如何实现窗体菜单功能?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

实现窗体菜单功能涉及 MenuBar (菜单栏)、Menu (菜单) 及 MenuItem (菜单项) 组件, 一个菜单栏可以包含多个菜单, 一个菜单可以包含多个菜单项。使用这些组件构建出菜单效果后, 还需要为其添加单击处理事件, 因为组件视图和事件处理是分开的。

示例代码如下:

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

public class TestMenuEvent implements ActionListener {

    public static void main(String[] args) {
        TestMenuEvent tme = new TestMenuEvent();
        tme.init();
    }

    Frame f;

    public void init() {
        f = new Frame("Test Menu");
        //创建 1 个 MenuBar 对象
        MenuBar mb = new MenuBar();
        //创建 3 个 Menu 对象
        Menu m1 = new Menu("File");
        Menu m2 = new Menu("Tool");
        Menu m3 = new Menu("Help");
        //将 Menu 对象放置到 MenuBar 对象中
        mb.add(m1);
        mb.add(m2);
        mb.setHelpMenu(m3);
        //将 MenuBar 置于 Frame 容器中
        f.setMenuBar(mb);
        //创建 4 个 MenuItem 子菜单对象
        MenuItem mi1 = new MenuItem("New");
        MenuItem mi2 = new MenuItem("Save");
        MenuItem mi3 = new MenuItem("Load");
        MenuItem mi4 = new MenuItem("Quit");
        MenuItem mi5 = new MenuItem("Change BackGround");
```

```
mi1.addActionListener(this);
mi2.addActionListener(this);
mi3.addActionListener(this);
mi4.addActionListener(this);
mi5.addActionListener(this);

m1.add(mi1);
m1.add(mi2);
m1.add(mi3);
m1.add(mi4);
m2.add(mi5);

f.setBounds(300, 300, 200, 200);
f.setVisible(true);

}

@Override
public void actionPerformed(ActionEvent arg0) {
    String command = arg0.getActionCommand();
    if (command.equals("Quit")) {
        System.exit(0); //安全退出
    } else if (command.equals("Change BackGround")) {
        Random ran = new Random();
        int r = ran.nextInt(225);
        int g = ran.nextInt(225);
        int b = ran.nextInt(225);
        //创建 2 个 Color 类实例
        Color c = new Color(r, g, b);
        //分别为 button 和 f 设置背景色
        f.setBackground(c);
    }
}
}
```

在上述代码中，为窗体菜单添加了事件处理。菜单 Quit 实现窗口的正常退出，Tool 中的 Change Background 实现对背景颜色的改变。程序运行效果如图 5-12 所示。

疑难点评

为窗体菜单添加处理事件的过程与按钮事件的添加过程相同，都是实现 ActionListener 接口，并实现 ActionPerformaned() 方法。

知识链接

FAQ5.13 如何实现鼠标右键弹出菜单的功能？

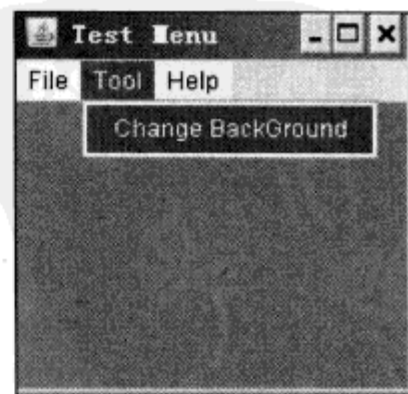


图 5-12 菜单事件处理效果

FAQ5.11 如何处理键盘输入事件?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

监听键盘输入事件, 给组件添加一个键盘事件监听器 `KeyListener` 即可, 同样实现途径可以通过内部类、匿名类、当前类和外部类等 4 种方式。下面通过匿名类方式为键盘添加事件处理, 示例代码如下:

```
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.Random;

public class TestKeyEvent {

    public static void main(String[] args) {
        TestKeyEvent tke = new TestKeyEvent();
        tke.init();
    }

    Frame f;
    Label label;

    public void init() {
        f = new Frame();
        //设置背景颜色
        f.setBackground(Color.cyan);
        //设置布局管理器
        f.setLayout(new FlowLayout());
        label = new Label("My Label");
        f.add(label);
        //为 Frame 添加键盘事件处理
        f.addKeyListener(new KeyListener() {
            //实现 keyPressed 方法处理相关事件
            @Override
            public void keyPressed(KeyEvent arg0) {
                switch (arg0.getKeyCode()) {
                    case KeyEvent.VK_UP:
                        label.setText("您按下了向上键");
                        break;
                    case KeyEvent.VK_DOWN:
                        label.setText("您按下了向下键");
                        break;
                    case KeyEvent.VK_LEFT:
                        label.setText("您按下了向左键");
                        break;
                }
            }
        });
    }
}
```

```
        case KeyEvent.VK_RIGHT:
            label.setText("您按下了向右键");
            break;

    }

}

public void keyReleased(KeyEvent arg0) {

}

public void keyTyped(KeyEvent arg0) {

}

});
//设置起始位置大小和显示
f.setBounds(300, 300, 400, 200);
f.setVisible(true);

}

}
```

在上述代码中,通过 Frame 窗口添加键盘输入事件。具体实现方法为首先创建事件处理对象,然后调用 addKeyListener()方法为 Frame 添加事件监听,当键盘输入触发相关事件时,则调用处理对象的对应方法做出响应。

在示例中,监听键盘输入上、下、左、右方向键,事件发生后,在窗口中通过 Label 将键盘操作显示出来。程序的运行效果如图 5-13 所示。

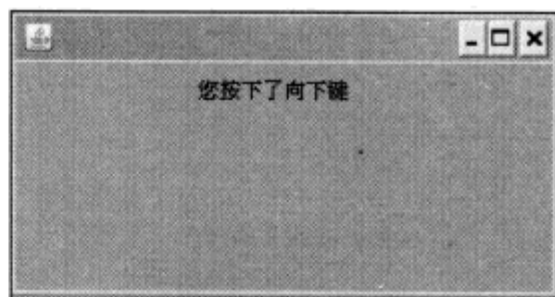


图 5-13 键盘事件示例的运行效果

疑难点评

KeyListener 接口定义了很多处理键盘事件的方法,在使用时可以根据需要在不同的事件方法中添加处理代码。

知识链接

FAQ5.12 如何处理鼠标单击事件?如何区分是左键还是右键?

FAQ5.12 如何处理鼠标单击事件?如何区分是左键还是右键?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

监听鼠标单击事件是通过 `java.awt.event` 包中的 `MouseMotionListener` 接口来实现。当成功创建侦听器对象之后，可以通过 `addMouseListener()` 方法将侦听器对象注册给相应组件，当发生鼠标事件时，将调用该侦听器对象中的相应方法，并将 `MouseEvent` 传递给该方法。

`MouseEvent` 接口提供与鼠标事件相关联的特定上下文信息，其中的 `getButton()` 方法在由按下或释放鼠标的按键引起的鼠标事件期间触发，`button` 用于指示哪一个鼠标按键改变了状态。`button` 值的范围为 1（指示鼠标的左键）、2（指示中间键）和 3（指示右键）。对于为左手使用而配置的鼠标，鼠标按键操作正好相反，值改为从右向左读取。

示例代码如下：

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class TestMouseEvent {

    public static void main(String[] args) {
        TestMouseEvent tme = new TestMouseEvent();
        tme.init();
    }

    Button b1;
    Button b2;
    Button b3;
    Label l;

    public void init() {

        Frame f = new Frame();
        f.setLayout(new FlowLayout());
        b1 = new Button("LEFT");
        b2 = new Button("MIDDLE");
        b3 = new Button("RIGHT");
        l = new Label("监听鼠标单击事件");
        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(l);
        //匿名类为 Frame 添加鼠标事件
        f.addMouseListener(new MouseListener() {

            @Override
            public void mouseClicked(MouseEvent e) {
                int s = e.getButton();
                if (s == 1) {
                    b1.setBackground(new Color(100, 150, 200));
                    l.setText("您单击了鼠标左键" + s);
                } else if (s == 2) {
```



```
        b2.setBackground(new Color(100, 150, 200));
        l.setText("您单击了鼠标中键" + s);
    } else {
        b3.setBackground(new Color(100, 150, 200));
        l.setText("您单击了鼠标右键" + s);
    }
}

public void mouseEntered(MouseEvent arg0) {
    // TODO Auto-generated method stub
}

public void mouseExited(MouseEvent arg0) {
    // TODO Auto-generated method stub
}

public void mousePressed(MouseEvent arg0) {
    // TODO Auto-generated method stub
}

public void mouseReleased(MouseEvent arg0) {
    // TODO Auto-generated method stub
}

});
f.setBounds(300, 300, 200, 200);
f.setVisible(true);
}
}
```

在上述程序中，为 Frame 添加鼠标事件，并通过 `getButton()` 方法判断鼠标单击事件是左键、右键还是中间键，并相应改变对应按钮的标签。程序运行效果如图 5-14 和图 5-15 所示。

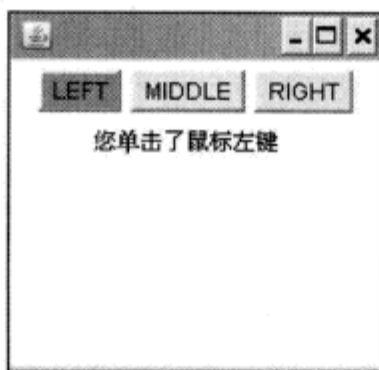


图 5-14 单击鼠标左键效果



图 5-15 单击鼠标右键效果

疑难点评

`MouseMotionListener` 接口定义了很多处理鼠标事件的方法，在使用时可以根据需要在不同的事件方法中添加处理代码。

知识链接

FAQ5.13 如何实现鼠标右键弹出菜单的功能?

FAQ5.13 如何实现鼠标右键弹出菜单的功能?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

Menu 类用于实现菜单功能, 菜单项由 MenuItem 类实现。PopupMenu 类是 Menu 的直接子类, 实现能够在组件的指定位置动态弹出菜单的功能, 其子菜单项同样是用 MenuItem 类实现。

示例代码如下:

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;

public class TestPopupMenu extends Frame {

    private static final long serialVersionUID = 1L;

    public static void main(String[] args) {
        TestPopupMenu tpm = new TestPopupMenu();
        tpm.init();
    }

    TextField tf;
    PopupMenu pm;

    public void init() {

        setLayout(new FlowLayout());
        tf = new TextField(30);
        pm = new PopupMenu();

        MenuItem mi1 = new MenuItem("Refresh");
        MenuItem mi2 = new MenuItem("Copy");
        MenuItem mi3 = new MenuItem("Paste");
        MenuItem mi4 = new MenuItem("Change Background");
        pm.add(mi1);
        pm.add(mi2);
        pm.add(mi3);
        pm.add(mi4);
        //为菜单添加事件
        mi1.addActionListener(new PopupListener());
        mi2.addActionListener(new PopupListener());
    }
}
```

```

mi3.addActionListener(new PopupListener());
mi4.addActionListener(new PopupListener());
//启用鼠标事件监听
enableEvents(AWTEvent.MOUSE_EVENT_MASK);

add(tf);
add(pm);
//设置起始位置大小和显示
setBounds(300, 300, 400, 200);
setVisible(true);
}

//处理鼠标右键单击事件
public void processMouseEvent(MouseEvent e) {
    if (e.isPopupTrigger()) {
        pm.show(e.getComponent(), e.getX(), e.getY());
    }
}

//为菜单项添加事件处理
class PopupListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        tf.setText(e.getActionCommand());
        if (e.getActionCommand().equals("Change Background")) {
            setBackground(Color.CYAN);
        } else if (e.getActionCommand().equals("Refresh")) {
            setBackground(Color.WHITE);
        }
    }
}
}

```

在上述代码中，为 `Frame` 容器添加了 1 个单行文本框和 1 个弹出菜单组件，启用容器的鼠标事件监听，用于监听鼠标右键单击事件，并实现弹出菜单的功能。用户在弹出菜单中选中某个子菜单选项时，将在文本框中显示对应的子菜单标签。程序运行效果如图 5-16 和图 5-17 所示。

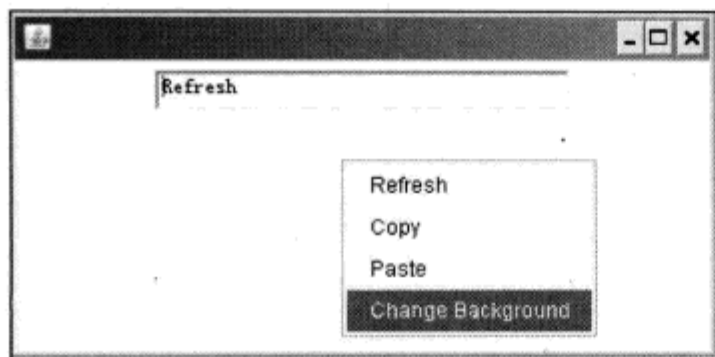


图 5-16 改变窗口背景色

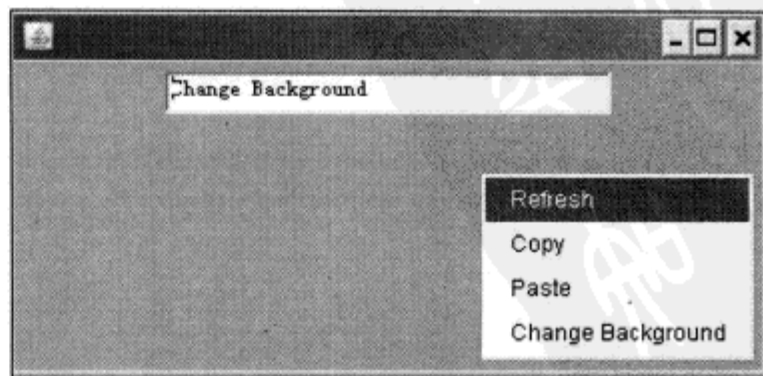


图 5-17 恢复背景色为白色

疑难点评

在很多软件中都具有鼠标右键菜单功能,在实现时,没有使用原有方式注册事件处理对象,而是使用了 `enableEvents()` 方法启用鼠标事件处理。

知识链接

FAQ5.10 如何实现窗体菜单功能?

FAQ5.14 如何使用表格组件?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

当需要把已知数据以表格的形式呈现出来时,便会用到 Swing 中提供的 JTable 组件。JTable 组件可用于显示、浏览和编辑数据。

JTable 类代表了 Swing 中的表格组件,为表格的行为和外观的管理提供了一套成熟的 API。

JTable 类直接继承了 JComponent 类,并实现了 TableModelListener、TableColumnListener、ListSelectionListener、CellEditorListener 和 Scrollable 等接口。每个 JTable 都有 3 个模型,分别为 TableModel、TableColumnModel 和 ListSelectionModel。所有的表格数据通常都是以二维格式(二维数组或者 Vector 向量)存放在 TableModel 中。TableModel 中提供的方法定义了怎样来存储、增加、操作和获取这些数据的操作;TableColumnModel 设计用来维护 TableColumn 对象,即 TableModel 中的每一列对象。为了支持不同的模式,TableColumn 中又提供了一个 ListSelectionModel 来维护列的单选和多选操作。

示例代码如下:

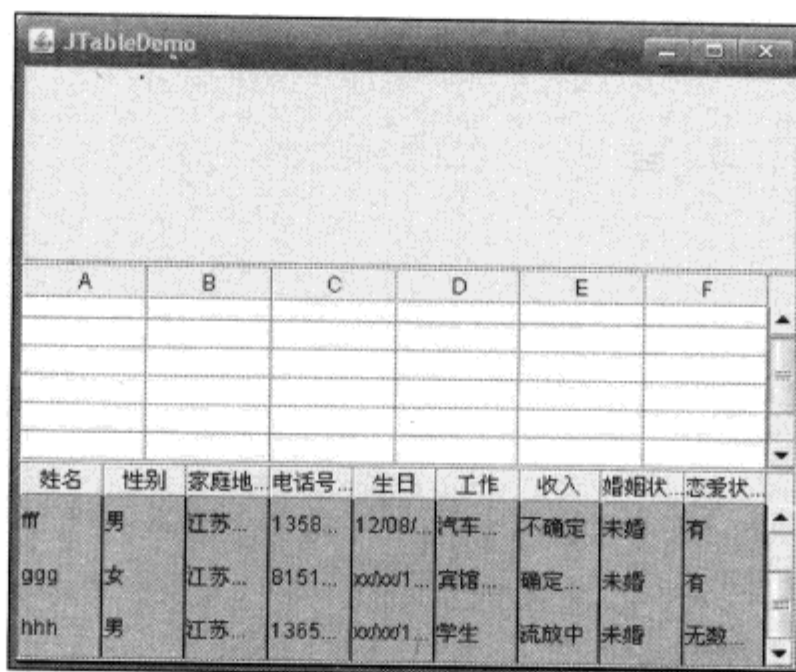
```
import java.awt.Dimension;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JPanel;
import javax.swing.JTable;
import java.awt.Color;
import java.awt.GridLayout;

public class JTableDemo
{
    public static void main(String[] args) {

        JTable example1 = new JTable();//看不到但存在
```

```
JTable example2 = new JTable(8, 6);
final Object[] columnNames = { "姓名", "性别", "家庭地址", //列名最好用 final 修饰
    "电话号码", "生日", "工作", "收入", "婚姻状况", "恋爱状况" };
Object[][] rowData = {
    { "ddd", "男", "江苏南京", "1378313210", "03/24/1985", "学生", "寄生中",
        "未婚", "没" },
    { "eee", "女", "江苏南京", "13645181705", "xx/xx/1985", "家教", "未知",
        "未婚", "好象没" },
    { "fff", "男", "江苏南京", "13585331486", "12/08/1985", "汽车推销员",
        "不确定", "未婚", "有" },
    { "ggg", "女", "江苏南京", "81513779", "xx/xx/1986", "宾馆服务员",
        "确定但未知", "未婚", "有" },
    { "hhh", "男", "江苏南京", "13651545936", "xx/xx/1985", "学生", "流放中",
        "未婚", "无数次分手后没有" }
};
JTable friends = new JTable(rowData, columnNames);
friends.setPreferredScrollableViewportSize(new Dimension(500, 100)); //设置表格的大小
friends.setRowHeight(30); //设置每行的高度为 20
friends.setRowHeight(0, 20); //设置第 1 行的高度为 15
friends.setRowMargin(5); //设置相邻两行单元格的距离
friends.setRowSelectionAllowed(true); //设置可否被选择, 默认为 false
friends.setSelectionBackground(Color.white); //设置所选择行的背景色
friends.setSelectionForeground(Color.red); //设置所选择行的前景色
friends.setGridColor(Color.black); //设置网格线的颜色
friends.selectAll(); //选择所有行
friends.setRowSelectionInterval(0, 2); //设置初始的选择行, 这里是 1 到 3 行都处于选择状态
friends.clearSelection(); //取消选择
friends.setShowGrid(false); //是否显示网格线
friends.setShowHorizontalLines(false); //是否显示水平的网格线
friends.setShowVerticalLines(true); //是否显示垂直的网格线
friends.setValueAt("tt", 0, 0); //设置某个单元格的值, 这个值是一个对象
friends.doLayout();
friends.setBackground(Color.lightGray);
JScrollPane pane1 = new JScrollPane(example1); // JTable 最好加在 JScrollPane 上
JScrollPane pane2 = new JScrollPane(example2);
JScrollPane pane3 = new JScrollPane(friends);
JPanel panel = new JPanel(new GridLayout(0, 1));
panel.setPreferredSize(new Dimension(600, 400));
panel.setBackground(Color.black);
panel.add(pane1);
panel.add(pane2);
panel.add(pane3);
JFrame frame = new JFrame("JTableDemo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setContentPane(panel);
frame.pack();
frame.setVisible(true);
}
```

在上述代码中,构造了一个二维数组 friends 作为 TableModel,并将结果显示在 JTable 组件中,程序运行效果如图 5-18 所示。



A	B	C	D	E	F			
姓名	性别	家庭地...	电话号...	生日	工作	收入	婚姻状...	恋爱状...
fff	男	江苏...	1358...	12/08/...	汽车...	不确定	未婚	有
ggg	女	江苏...	8151...	xx/xx/1...	宾馆...	确定...	未婚	有
hhh	男	江苏...	1365...	xx/xx/1...	学生	流放中	未婚	无数...

图 5-18 JTable 组件使用效果

疑难点评

JTable 是 Swing 类库中新增的一款功能强大的表格组件,通过该组件可以将数据以列表形式显示,并可以在表格中进行编辑。

知识链接

FAQ5.04 如何为容器添加滚动条功能?

FAQ5.15 如何实现记事本功能?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

利用 Swing 类库实现记事本,主要涉及文件打开和存储、事件处理等知识点的综合应用。下面实现了一个包含基本功能的记事本,示例代码如下:

```
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Toolkit;
```



```
public class MyNotePad extends WindowAdapter implements ActionListener {
    JFrame jf;
    JButton jb, jb1;
    JTextArea ta;
    String filename, copy, paste, cut;
    JPanel jp;
    JMenu jmb, jmb2;
    JMenuItem _fm, _fm1, _fm2, _fm3, _fm4, _fe1, _fe2, _fe3, _fe4;
    JMenuBar JMENU;
    JScrollPane jsp;

    public static void main(String[] argv) {
        new MyNotePad();
    }

    public MyNotePad() {
        jp = new JPanel();
        JMENU = new JMenuBar();
        ta = new JTextArea();
        jf = new JFrame();
        jsp = new JScrollPane(ta);
        jf.addWindowListener(this);
        jmb = new JMenu("文件");
        jmb2 = new JMenu("编辑");
        _fm1 = new JMenuItem("打开");
        _fm1.addActionListener(this);
        _fm2 = new JMenuItem("储存");
        _fm2.addActionListener(this);
        _fm4 = new JMenuItem("另存为");
        _fm4.addActionListener(this);
        _fm3 = new JMenuItem("关闭");
        _fm3.addActionListener(this);
        _fm = new JMenuItem("新建");
        _fm.addActionListener(this);

        _fe1 = new JMenuItem("复制");
        _fe1.addActionListener(this);
        _fe2 = new JMenuItem("粘贴");
        _fe2.addActionListener(this);
        _fe3 = new JMenuItem("剪切");
        _fe3.addActionListener(this);
        _fe4 = new JMenuItem("版本");
        _fe4.addActionListener(this);
        jf.setJMenuBar(JMENU);
        jf.setTitle("记事本");

        jmb.add(_fm);
        jmb.addSeparator();
        jmb.add(_fm1);
        jmb.addSeparator();
        jmb.add(_fm2);
        jmb.addSeparator();
        jmb.add(_fm4);
        jmb.addSeparator();
    }
}
```

```

jmb.add(_fm3);
jmb2.add(_fe1);
jmb2.addSeparator();
jmb2.add(_fe2);
jmb2.addSeparator();
jmb2.add(_fe3);
jmb2.addSeparator();
jmb2.add(_fe4);

JMENU.add(jmb);
JMENU.add(jmb2);

jb = new JButton("保存");
jb.addActionListener(this);
jb1 = new JButton("关闭");
jb1.addActionListener(this);
jp.add(jb);
jp.add(jb1);
jf.add(jp, "South");
ta.setWrapStyleWord(true);
jf.add(jsp);
jf.setSize(600, 400);
jf.setVisible(true);
int W = (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
int H = (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
jf.setLocation((W - jf.getWidth()) / 2, (H - jf.getHeight()) / 2);
}

public void actionPerformed(ActionEvent e) {
    if (e.getSource() == jb || e.getSource() == _fm2) {
        try {
            if (filename == null) {
                filename = JOptionPane.showInputDialog("请输入文件名", "java");
                FileOutputStream fout = new FileOutputStream(filename
                    + ".txt");
                byte bb[] = ta.getText().getBytes();
                fout.write(bb);
                fout.close();
                JOptionPane.showMessageDialog(null, "已保存");
            } else {
                FileOutputStream fout = new FileOutputStream(filename
                    + ".txt");
                byte bb[] = ta.getText().getBytes();
                fout.write(bb);
                fout.close();
                JOptionPane.showMessageDialog(null, "已保存");
            }
        } catch (IOException ioe) {
            System.err.println(e);
        }
    }
    if (e.getSource() == _fm) {
        if (!(ta.getText().equals(""))) {
            Object[] options = { "确定", "取消" };

```



```

        int response = JOptionPane
            .showOptionDialog(null, "你是否保存", "提 示",
                JOptionPane.YES_OPTION,
                JOptionPane.QUESTION_MESSAGE, null, options,
                options[0]);

        if (response == 0) {
            try {
                FileDialog d = new FileDialog(jf, "保存文件",
                    FileDialog.SAVE);
                d.setVisible(true);
                filename = d.getDirectory() + d.getFile();
                FileOutputStream fout = new FileOutputStream(filename
                    + ".txt");
                byte bb[] = ta.getText().getBytes();
                fout.write(bb);
                fout.close();
                JOptionPane.showMessageDialog(null, "已保存");
                ta.setText("");
            } catch (IOException ioe) {
                System.err.println(e);
            }
        }
        if (response == 1) {
            JOptionPane.showMessageDialog(null, "你选择了取消");
            ta.setText("");
        }
    }
}

if (e.getSource() == _fm1) {
    FileDialog d = new FileDialog(jf, "打开文件", FileDialog.LOAD);
    d.setVisible(true);
    File f = null;
    f = new File(d.getDirectory() + d.getFile());
    for (int i = 0; i <= f.length(); i++) {
        char[] ch = new char[100];
        try {
            FileReader fr = new FileReader(f);
            fr.read(ch);
            String str = new String(ch);
            ta.setText(str);

        } catch (FileNotFoundException fe) {

        }
        catch (IOException ie) {

        }
    }
}

if (e.getSource() == _fm4) {
    FileDialog d = new FileDialog(jf, "另存为", FileDialog.SAVE);
    d.setVisible(true);
    try {
        filename = d.getDirectory() + d.getFile();
        FileOutputStream fout = new FileOutputStream(filename + ".txt");
    }
}

```



```
        byte bb[] = ta.getText().getBytes();
        fout.write(bb);
        fout.close();
    } catch (IOException ioe) {
        System.err.println(e);
    }
}
if (e.getSource() == _fm3 || e.getSource() == jb1) {
    System.exit(0);
}
if (e.getSource() == _fe1) {
    copy = ta.getSelectedText();
}
if (e.getSource() == _fe2) {
    ta.setText(copy);
}
if (e.getSource() == _fe3) {
    copy = ta.getSelectedText();
    ta.setText("");
}
if (e.getSource() == _fe4) {
    JOptionPane.showMessageDialog(jf,
        "My Note Pad 1.0");
}
}

public void windowClosing(WindowEvent e) {
    System.exit(0);
}
}
```

上述程序运行效果如图 5-19 所示。

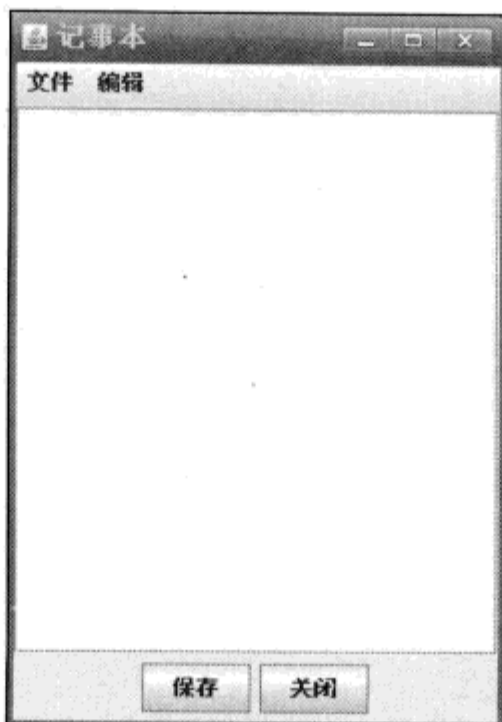


图 5-19 记事本示例的效果

疑难点评

该示例是一个综合示例，应用知识点主要涉及流操作、Swing 和 AWT 组件、组件事件处理等方面。读者可以借此示例提升自己在 Java 编程方面的综合能力。

知识链接

FAQ5.16 如何实现贪吃蛇游戏？

FAQ5.16 如何实现贪吃蛇游戏？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

利用 Swing 类库实现贪吃蛇游戏，主要涉及键盘事件处理、窗体菜单、组件绘图和线程等知识点的综合应用。下面实现了一个贪吃蛇游戏，示例代码如下：

```
package question_32;

import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;

import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;

public class SnakeGame {
    public static void main(String[] args) {
        SnakeFrame frame = new SnakeFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

//记录状态的线程

```
class StatusRunnable implements Runnable {
    public StatusRunnable(Snake snake, JLabel statusLabel, JLabel scoreLabel) {
        this.statusLabel = statusLabel;
        this.scoreLabel = scoreLabel;
        this.snake = snake;
    }

    public void run() {
        String sta = "";
        String spe = "";
        while (true) {

            switch (snake.status) {
                case Snake.RUNNING:
                    sta = "Running";
                    break;
                case Snake.PAUSED:
                    sta = "Paused";
                    break;
                case Snake.GAMEOVER:
                    sta = "GameOver";
                    break;
            }
            statusLabel.setText(sta);
            scoreLabel.setText("" + snake.score);
            try {
                Thread.sleep(100);
            } catch (Exception e) {
            }
        }
    }

    private JLabel scoreLabel;
    private JLabel statusLabel;
    private Snake snake;
}
```

//蛇运动以及记录分数的线程

```
class SnakeRunnable implements Runnable {
    public SnakeRunnable(Snake snake, Component component) {
        this.snake = snake;
        this.component = component;
    }

    public void run() {
        while (true) {
            try {
                snake.move();
                component.repaint();
            }
        }
    }
}
```



```
        Thread.sleep(snake.speed);
    } catch (Exception e) {
    }
}

private Snake snake;
private Component component;
}

class Snake {
    boolean isRun;//是否运动中
    ArrayList<Node> body;//蛇体
    Node food;//食物
    int direction;//方向
    int score;
    int status;
    int speed;
    public static final int SLOW = 500;
    public static final int MID = 300;
    public static final int FAST = 100;
    public static final int RUNNING = 1;
    public static final int PAUSED = 2;
    public static final int GAMEOVER = 3;
    public static final int LEFT = 1;
    public static final int UP = 2;
    public static final int RIGHT = 3;
    public static final int DOWN = 4;

    public Snake() {
        speed = Snake.SLOW;
        score = 0;
        isRun = false;
        status = Snake.PAUSED;
        direction = Snake.RIGHT;
        body = new ArrayList<Node>();
        body.add(new Node(60, 20));
        body.add(new Node(40, 20));
        body.add(new Node(20, 20));
        makeFood();
    }

    //判断食物是否被蛇吃掉
    //如果食物在蛇运行方向的正前方, 并且与蛇头接触, 则被吃掉
    private boolean isEaten() {
        Node head = body.get(0);
        if (direction == Snake.RIGHT && (head.x + Node.W) == food.x
            && head.y == food.y)
            return true;
        if (direction == Snake.LEFT && (head.x - Node.W) == food.x
```

```

        && head.y == food.y)
            return true;
        if (derection == Snake.UP && head.x == food.x
            && (head.y - Node.H) == food.y)
            return true;
        if (derection == Snake.DOWN && head.x == food.x
            && (head.y + Node.H) == food.y)
            return true;
        else
            return false;
    }

```

//是否碰撞

```

private boolean isCollsion() {
    Node node = body.get(0);
    //碰壁
    if (derection == Snake.RIGHT && node.x == 280)
        return true;
    if (derection == Snake.UP && node.y == 0)
        return true;
    if (derection == Snake.LEFT && node.x == 0)
        return true;
    if (derection == Snake.DOWN && node.y == 380)
        return true;
    //蛇头碰到蛇身
    Node temp = null;
    int i = 0;
    for (i = 3; i < body.size(); i++) {
        temp = body.get(i);
        if (temp.x == node.x && temp.y == node.y)
            break;
    }
    if (i < body.size())
        return true;
    else
        return false;
}

```

//在随机的地方产生食物

```

public void makeFood() {
    Node node = new Node(0, 0);
    boolean isInBody = true;
    int x = 0, y = 0;
    int X = 0, Y = 0;
    int i = 0;
    while (isInBody) {
        x = (int) (Math.random() * 15);
        y = (int) (Math.random() * 20);
        X = x * Node.W;
        Y = y * Node.H;
    }
}

```



```
        for(i = 0; i < body.size(); i++) {
            if (X == body.get(i).x && Y == body.get(i).y)
                break;
        }
        if (i < body.size())
            isInBody = true;
        else
            isInBody = false;
    }
    food = new Node(X, Y);
}

//改变运行方向
public void changeDerection(int newDer) {
    if (derection % 2 != newDer % 2) //如果与原来方向相同或相反, 则无法改变
        derection = newDer;
}

public void move() {
    if (isEaten()) { //如果食物被吃掉
        body.add(0, food); //把食物当成蛇头成为新的蛇体
        score += 10;
        makeFood(); //产生食物
    } else if (isCollsion()) //如果碰壁或碰到自身
    {
        isRun = false;
        status = Snake.GAMEOVER; //结束
    } else if (isRun) { //正常运行 (不吃食物, 不碰壁, 不碰自身)
        Node node = body.get(0);
        int X = node.x;
        int Y = node.y;
        //蛇头按运行方向前进一个单位
        switch (derection) {
            case 1:
                X -= Node.W;
                break;
            case 2:
                Y -= Node.H;
                break;
            case 3:
                X += Node.W;
                break;
            case 4:
                Y += Node.H;
                break;
        }
        body.add(0, new Node(X, Y));
        //去掉蛇尾
        body.remove(body.size() - 1);
    }
}
```



```
    }  
}  
  
//组成蛇身的单位, 食物  
class Node {  
    public static final int W = 20;  
    public static final int H = 20;  
    int x;  
    int y;  
  
    public Node(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
//画板  
class SnakePanel extends JPanel {  
    Snake snake;  
  
    public SnakePanel(Snake snake) {  
        this.snake = snake;  
    }  
  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        Node node = null;  
        for (int i = 0; i < snake.body.size(); i++) {  
            g.setColor(Color.BLACK);  
            node = snake.body.get(i);  
            g.fillRect(node.x, node.y, node.H, node.W); // ***试用***  
        }  
        node = snake.food;  
        g.setColor(Color.red);  
        g.fillRect(node.x, node.y, node.H, node.W);  
    }  
}  
  
class SnakeFrame extends JFrame {  
    private JLabel statusLabel;  
    private JLabel speedLabel;  
    private JLabel scoreLabel;  
    private JPanel snakePanel;  
    private Snake snake;  
    private JMenuBar bar;  
    JMenu gameMenu;  
    JMenu helpMenu;  
    JMenu speedMenu;  
    JMenuItem newItem;  
    JMenuItem pauseItem;
```

```
JMenuItem beginItem;
JMenuItem helpItem;
JMenuItem aboutItem;
JMenuItem slowItem;
JMenuItem midItem;
JMenuItem fastItem;

public SnakeFrame() {
    init();
    ActionListener l = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == pauseItem)
                snake.isRun = false;
            if (e.getSource() == beginItem)
                snake.isRun = true;
            if (e.getSource() == newItem) {
                newGame();
            }
            //菜单控制运行速度
            if (e.getSource() == slowItem) {
                snake.speed = Snake.SLOW;
                speedLabel.setText("Slow");
            }
            if (e.getSource() == midItem) {
                snake.speed = Snake.MID;
                speedLabel.setText("Mid");
            }
            if (e.getSource() == fastItem) {
                snake.speed = Snake.FAST;
                speedLabel.setText("Fast");
            }
        }
    };
    pauseItem.addActionListener(l);
    beginItem.addActionListener(l);
    newItem.addActionListener(l);
    aboutItem.addActionListener(l);
    slowItem.addActionListener(l);
    midItem.addActionListener(l);
    fastItem.addActionListener(l);
    addKeyListener(new KeyListener() {
        public void keyPressed(KeyEvent e) {
            switch (e.getKeyCode()) {
                //方向键改变蛇运行方向
                case KeyEvent.VK_DOWN://
                    snake.changeDerection(Snake.DOWN);
                    break;
                case KeyEvent.VK_UP://
                    snake.changeDerection(Snake.UP);
                    break;
            }
        }
    });
}
```



```

        case KeyEvent.VK_LEFT://
            snake.changeDerection(Snake.LEFT);
            break;
        case KeyEvent.VK_RIGHT://
            snake.changeDerection(Snake.RIGHT);
            break;
        //空格键, 游戏暂停或继续
        case KeyEvent.VK_SPACE://
            if (snake.isRun == true) {
                snake.isRun = false;
                snake.status = Snake.PAUSED;
                break;
            }
            if (snake.isRun == false) {
                snake.isRun = true;
                snake.status = Snake.RUNNING;
                break;
            }
        }
    }

    public void keyReleased(KeyEvent k) {
    }

    public void keyTyped(KeyEvent k) {
    }
});
}

private void init() {
    speedLabel = new JLabel();
    snake = new Snake();
    setSize(380, 460);
    setLayout(null);
    this.setResizable(false);
    bar = new JMenuBar();
    gameMenu = new JMenu("Game");
    newItem = new JMenuItem("New Game");
    gameMenu.add(newItem);
    pauseItem = new JMenuItem("Pause");
    gameMenu.add(pauseItem);
    beginItem = new JMenuItem("Continue");
    gameMenu.add(beginItem);
    helpMenu = new JMenu("Help");
    aboutItem = new JMenuItem("About");
    helpMenu.add(aboutItem);
    speedMenu = new JMenu("Speed");
    slowItem = new JMenuItem("Slow");
    fastItem = new JMenuItem("Fast");
    midItem = new JMenuItem("Middle");
}

```



```
speedMenu.add(slowItem);
speedMenu.add(midItem);
speedMenu.add(fastItem);
bar.add(gameMenu);
bar.add(helpMenu);
bar.add(speedMenu);
setJMenuBar(bar);
statusLabel = new JLabel();
scoreLabel = new JLabel();
snakePanel = new JPanel();
snakePanel.setBounds(0, 0, 300, 400);
snakePanel.setBorder(BorderFactory.createLineBorder(Color.darkGray));
add(snakePanel);
statusLabel.setBounds(300, 25, 60, 20);
add(statusLabel);
scoreLabel.setBounds(300, 20, 60, 20);
add(scoreLabel);
JLabel temp = new JLabel("状态");
temp.setBounds(310, 5, 60, 20);
add(temp);
temp = new JLabel("分数");
temp.setBounds(310, 105, 60, 20);
add(temp);
temp = new JLabel("速度");
temp.setBounds(310, 55, 60, 20);
add(temp);
speedLabel.setBounds(310, 75, 60, 20);
add(speedLabel);
}

private void newGame() {
    this.remove(snakePanel);
    this.remove(statusLabel);
    this.remove(scoreLabel);
    speedLabel.setText("Slow");
    statusLabel = new JLabel();
    scoreLabel = new JLabel();
    snakePanel = new JPanel();
    snake = new Snake();
    snakePanel = new SnakePanel(snake);
    snakePanel.setBounds(0, 0, 300, 400);
    snakePanel.setBorder(BorderFactory.createLineBorder(Color.darkGray));
    Runnable r1 = new SnakeRunnable(snake, snakePanel);
    Runnable r2 = new StatusRunnable(snake, statusLabel, scoreLabel);
    Thread t1 = new Thread(r1);
    Thread t2 = new Thread(r2);
    t1.start();
    t2.start();
    add(snakePanel);
    statusLabel.setBounds(310, 25, 60, 20);
```

```
add(statusLabel);  
scoreLabel.setBounds(310, 125, 60, 20);  
add(scoreLabel);  
}  
}
```

上述程序通过空格键可控制暂停,还可以根据需要调整速度,程序运行效果如图 5-20 所示。

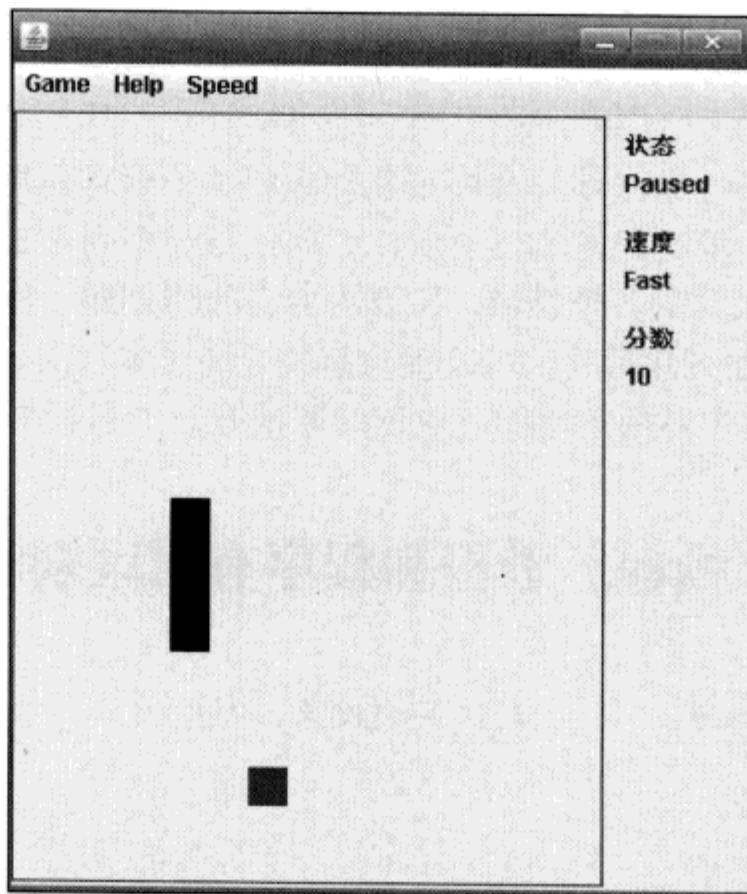


图 5-20 贪吃蛇游戏的效果

疑难点评

该示例是一个综合示例,应用知识点主要涉及线程操作、Swing 和 AWT 组件、组件事件处理等方面。读者可以借此示例提升自己在 Java 编程方面的综合能力。

知识链接

FAQ5.15 如何实现记事本功能?

第6章

Java 线程和序列化

本章重点介绍一些与 Java 线程和序列化知识相关的疑难问题。Java 网络编程部分的内容主要涉及线程创建、多线程通信与同步、序列化与反序列化等几个方面。本章内容涵盖与 Java 线程和序列化相关的各个领域，主要内容有线程创建、线程控制（例如停止和启动等）、线程同步和对象序列化等方面的功能实现等。通过本章的学习，读者可以灵活的应用线程和序列化完成项目中常用的功能，快速的解决开发中遇到的疑难问题。

FAQ6.01 线程、进程和程序有何区别和联系？

📖 难度系数：★★★

📖 问题频率：95%

核心解答

1. 什么是线程

线程是进程的一个实体，是 CPU 调度和分配的基本单位，其本身不拥有系统资源，只含有程序计数器、寄存器和栈等一些运行时必不可少的基本资源。它的存在是为进程服务的，同属一个进程的线程共享进程所拥有的全部资源。

2. 什么是进程

进程是具有一定独立功能的程序块关于某个数据集合上的一次运行活动，它是系统进行资源调度分配的一个独立单位。

3. 什么是程序

程序是一组指令的集合，由多个进程共同完成，它是一个静态的实体，没有执行的含义。

4. 线程和进程之间的区别

- (1) 线程是进程的一部分，因此线程有的时候被称为轻权进程或者轻量级进程。
- (2) 一个没有线程的进程是可以被看做单线程的，如果一个进程内拥有多个进程，那么进程的执行过程就不是由一条线（线程），而是由多条线（线程）共同完成的。
- (3) 系统在运行的时候会为每个进程分配不同的内存区域，但是不会为线程分配内存（线程所

使用的资源是它所属的进程的资源), 线程组只能共享资源。即除了 CPU 之外 (线程在运行的时候要占用 CPU 资源), 计算机内部的软硬件资源的分配与线程无关, 线程只能共享它所属进程的资源。

(4) 与进程的控制表 PCB 相似, 线程也有自己的控制表 TCB, 但是 TCB 中所保存的线程状态比 PCB 表中少得多。

(5) 进程是系统所有资源分配时候的一个基本单位, 拥有一个完整的虚拟空间地址, 并不依赖线程而独立存在。

5. 进程与程序的区别

程序是一组指令的集合, 是静态的实体, 没有执行的含义。而进程是一个动态的实体, 有自己的生命周期。一般说来, 一个进程一定与一个程序相对应, 并且只有一个, 但是一个程序可以有多个进程, 或者一个进程都没有。除此之外, 进程还有并发性和交往性。简单地说, 进程是程序的一部分, 程序运行的时候会产生进程。

疑难点评

线程的概念比较难于理解, 可以与进程、程序概念相结合, 加以区分理解。有许多初学者在学习多线程编程时, 就是因为不能正确理解线程的概念, 因此无法完全掌握。由此可见, 在掌握一项技术之前对相关概念的理解非常重要。

知识链接

FAQ6.02 如何创建和启动一个线程?

FAQ6.02 如何创建和启动一个线程?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

与其他语言不同, 在 Java 语言中专门为线程的实现提供了 `java.lang.Thread` 类和一整套的 API, 具备一个完整的自建线程系统, 因此在使用 Java 语言创建线程时非常简单, 具体的实现途径有以下两种。

(1) 方法一

通过继承 `Thread` 类来创建一个线程。编写一个实现类, 继承 `Thread` 类并覆盖其 `run()` 方法, 在 `run()` 方法中加入线程所要执行的处理逻辑。

示例代码如下:

```
public class MyThread_exte extends Thread {  
  
    public static void main(String[] args) {  
        MyThread_exte thread = new MyThread_exte();  
        thread.start();  
    }  
}
```

```
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + 1);
        }
    }
}
```

这种方法实现比较简洁，MyThread_exte 类本身就是一个线程，操纵起来也比较方便。但是 MyThread_exte 类却不能再继承其他类，因为在 Java 中只允许继承一个父类。

启动线程的示例代码如下：

```
public class Test {
    public static void main(String[] args){
        MyThread_exte t = new MyThread_exte();
        t.start();
    }
}
```

(2) 方法二

通过实现 Runnable 接口来创建一个线程。编写一个实现类，实现 Runnable 接口并实现其中惟一的方法 run()，在 run() 方法中加入线程所要执行的处理逻辑。

示例代码如下：

```
public class MyThread_impl implements Runnable {

    public static void main(String[] args) {
        MyThread_impl mythread = new MyThread_impl();
        Thread thread = new Thread (mythread);
        thread.start();
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + 1);
        }
    }
}
```

由于 Runnable 接口对线程没有任何支持，因此在获得线程的实例时，必须通过 Thread 类的构造函数 public Thread(Runnable target) 来实现。启动线程的示例代码如下：

```
public class Test {
    public static void main(String[] args){
        MyThread_impl myrun = new MyThread_impl();
        Thread t = new Thread(myrun);
        t.start();
    }
}
```

在 Java 中是可以实现多个接口，因此在方法一中遇到的问题得到了很好的解决，而且可以

很好地将 CPU、代码和数据分开,形成清晰的模式,保证了程序风格的一致性。但是如果想创建多个线程并使各个线程执行不同的代码,则必须额外创建类。

在启动一个线程时,调用 `start()` 方法,线程进入 `Runnable` (可运行) 状态,并向线程调度器注册该线程。但是,调用 `start()` 方法并不是马上会执行该线程,只是进入 `Runnable` 状态,而并非 `Running` 状态。只有当线程调度器给该线程分配 CPU 资源并进入执行状态时,JVM 才会调用该线程对象的 `run()` 方法,执行线程体中的相关操作。

如果显式调用 `run()` 方法来启动一个线程,该线程将不受线程调度器的调度和管理,就属于单纯的方法调用,因此这种做法是不提倡的。

疑难点评

线程的创建有两种方法,一种是实现 `Runnable` 接口;另一种是继承 `Thread` 类。读者需要熟练掌握创建线程的方法。

知识链接

FAQ6.03 线程的基本状态有哪些?它们之间有何关系?

FAQ6.03 线程的基本状态有哪些?它们之间有何关系?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

线程的基本状态大体包括 5 种,分别为新建状态、可运行状态、运行状态、阻塞状态和死亡状态。如图 6-1 所示。

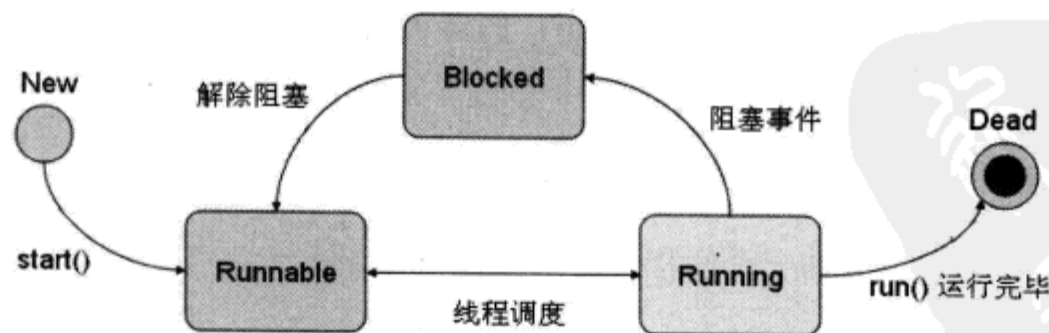


图 6-1 Java 线程基本状态转换

(1) 新建状态 (New)

当一个线程对象成功创建,且没有调用 `start()` 方法。

(2) 可运行状态 (Runnable)

当线程有资格运行,但调度程序还没有把它选定为运行线程时线程所处的状态。当 `start()`

方法调用时, 线程首先进入可运行状态, 在线程调度运行之后如果被阻塞(例如调用 `sleep()` 方法后); 则返回可运行状态, 继续等待进程的调度分配。

(3) 运行状态 (Running)

线程调度程序从可运行池中选择一个线程作为当前线程时该线程则处于运行状态。这也是线程进入运行状态的惟一一种方式。

(4) 阻塞状态 (Blocked)

这是当线程遭遇异常或者人为的 `sleep()` 方法、`wait()` 方法等操作时所处的一种临时状态。在此状态下的线程仍然是“活的”, 但是当前条件不允许其运行, 如果某种时间出现, 它便有可能返回可运行状态。

(5) 死亡状态 (Dead)

当线程 `run()` 方法中的处理过程全部完成之后便认为该线程处于死亡状态。虽然该线程对象也许是活的, 但是其已经不是一个可执行的线程, 程序员可以显式将其置为 `null` 等待 Java 垃圾回收器的回收。

注意: 调用 `stop()` 或 `destroy()` 方法也会使当前线程处于死亡状态, 但是这种用法不被推荐, 前者会产生异常, 后者是强制终止, 不会释放线程锁。线程一旦死亡, 就不能复生。如果在一个死去的线程上调用 `start()` 方法, 会抛出 `java.lang.IllegalThreadStateException` 异常。

基于线程概念的描述可知, 多个线程的“同步”其实并非真正意义上的同步。因为对于单个 CPU 来说不可能在同一时刻同时处理两个线程中的执行逻辑, 在此处所说的“同步”是相对意义上的解释, 只指当计算机的运行速率非常快时, 在 CPU 的指挥下依次运行对应的线程, 在用户看来就好像是“同步”完成的一样。当一个线程被阻塞之后, 是不能够直接返回运行状态的, CPU 在同一时刻只能处理一个线程操作, 如果直接进入将会和当前运行的线程发生冲突, 因此只能恢复可执行状态返回队列中重新等待系统资源调度。

疑难点评

线程在创建和销毁之间, 有多种存在状态, 例如可运行、运行和阻塞。通过线程的 API 可以将线程在这些状态之间进行转换。

知识链接

FAQ6.02 如何创建和启动一个线程?

FAQ6.04 什么是线程优先级? 线程依据什么原则调度执行?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

线程调用的意义在于 JVM 应对运行的多个线程进行系统级的协调，以避免多个线程争用有限资源而导致应用系统死机或者崩溃。为了使线程对于操作系统和用户的重要性区分开，Java 定义了线程的优先级策略。将线程的优先级分为 10 个等级，分别用数字 1~10 表示。数字越大表明线程的级别越高。

在 Thread 类中定义了表示线程最低、最高和普通优先级的成员变量 MIN_PRIORITY、MAX_PRIORITY 和 NORMAL_PRIORITY，代表的优先级等级分别为 1、10 和 5。当一个线程对象被创建时，其默认的线程优先级为 5。

为了控制线程的运行策略，Java 定义了线程调度器来监控系统中处于就绪状态的所有线程。线程调度器按照线程的优先级决定哪个线程投入处理器运行。在多个线程处于就绪状态的情况下，具有高优先级的线程会在低优先级线程之前得到执行。

但是在 Java 线程技术中，通常是以“抢占式”调度模式进行的，而不需要时间片分配进程，因此在使用时间片的 Windows 平台运行环境中，出现低优先级线程抢占高优先级线程的 CPU 时间的情况不可避免。从 JVM 的实现来看，并没有绝对的抢占式或是时间片分配，但是大多数的 JVM 的实现结果在行为上表现出了严格的抢占，因此在线程调度过程中需要程序员灵活地使用 wait()、notify() 和 sleep() 等方法来协助 JVM 完成线程的调度和控制。

示例代码如下：

```
public class TestPrio extends Thread {
    public TestPrio() {
        super();
    }
    public TestPrio(String name) {
        super(name);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(this.getName() + "----" + i);
        }
    }
    public static void main(String[] args) {
        TestPrio thread_1 = new TestPrio("Thread one ");
        TestPrio thread_2 = new TestPrio("Thread two ");
        thread_1.setPriority(6);
        thread_2.setPriority(8);
        thread_1.start();
        thread_2.start();
    }
}
```

输出结果如下：

```
Thread one----0
Thread two----0
```

```
Thread two---1
Thread two---2
Thread two---3
Thread two---4
Thread two---5
Thread two---6
Thread two---7
Thread two---8
Thread two---9
Thread one---1
Thread one---2
Thread one---3
Thread one---4
Thread one---5
Thread one---6
Thread one---7
Thread one---8
Thread one---9
```

注意：线程 `thread_2` 的优先级高于线程 `thread_1`，但是由于线程的执行采取抢占式同步模式工作，而 Windows 平台运行则采取的是时间片机制，因此尽管 JVM 用队列来实现线程池，但是却没有保证行为。

疑难点评

线程优先级是线程调度执行的一个重要原则，每次需要执行一个线程时，优先级高的将优先被选择执行。

知识链接

FAQ6.02 如何创建和启动一个线程？

FAQ6.05 什么是后台线程？如何创建一个后台线程？

📖 难度系数：★★★

📖 问题频率：80%

核心解答

在 Java 中有一种比较特殊的线程，被称为守护（Daemon）线程，这种线程具有最低的优先级，用于为系统中的其他对象和线程提供服务。将一个用户线程设置为守护线程的方法是在调用线程对象 `start()` 方法之前调用 `setDaemon()` 方法。JVM 中的系统资源自动回收线程就是守护线程的典型示例，它始终在低级别的状态中运行，用于实时监控和管理系统中的可回收资源。

守护线程是一类特殊的线程，和普通线程的区别在于其并不是应用程序的核心部分，当一个应用程序的所有非守护线程终止运行时，即使仍然有守护线程在运行，应用程序也将终止；

反之，只要有一个非守护线程在运行，守护线程就不会终止。守护线程一般被用于在后台为其他线程提供服务，因此也被称为“后台线程”。

示例代码如下：

```
public class TestDaemon extends Thread {

    public static void main(String[] args) {
        TestDaemon dt = new TestDaemon();
        dt.setDaemon(true);
        dt.start();
        for(int i=1;i<=10;i++){
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("---DaemonThread ---"+i);
        }
    }

    public void run() {
        for (int i = 1; i <= 1000; i++) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("---Current Thread ---"+i);
        }
    }
}
```

在上述示例中，由于线程对象 dt 被设置为守护线程，因此在 main() 方法后续打印功能执行结束后，即使线程 dt 的功能没有执行结束也要自动终止。示例的输出结果如下：

```
-----1
---DaemonThread ---1
---DaemonThread ---2
-----2
---DaemonThread ---3
-----3
---DaemonThread ---4
-----4
-----5
---DaemonThread ---5
---DaemonThread ---6
-----6
---DaemonThread ---7
-----7
---DaemonThread ---8
-----8
```

```
---DaemonThread ---9  
-----9  
-----10  
---DaemonThread ---10
```

疑难点评

守护线程也称为后台线程，可以用来执行一些非主要的、辅助性的处理功能。

知识链接

FAQ6.02 如何创建和启动一个线程？

FAQ6.06 如何使正在运行的线程在指定时间内休眠？

📖 难度系数：★★★★

📖 问题频率：85%

核心解答

一个正在运行的线程的状态为 **Running**，即正在执行，可以通过调用线程对象的 **sleep()** 方法实现在指定时间内休眠，待休眠时间过去，该线程又重新回到 **Runnable** 状态，等待线程调度器的调用。

Thread 类中定义的静态方法 **sleep()** 能够使当前运行中的线程暂停执行（即休眠）一段指定的时间。**sleep()** 方法可以指定一个 **Long** 类型的参数值，用于指定休眠的时间，单位是毫秒；也可以使用毫秒数和 **int** 值表示的纳秒数的组合。毫秒的分辨率在大多数情况下是足够的，因此通常使用 **sleep()** 的最简形式。例如以下代码可以使当前线程暂停 2 秒。

```
Thread.sleep(2000);
```

使用以下代码可以使线程休眠 100 纳秒。

```
Thread.sleep(0, 100);
```

当线程被休眠中断时，**sleep()** 方法都会抛出 **InterruptedException** 异常，在使用时需要使用 **try-catch** 语句捕获处理。此外 **sleep()** 方法只能影响当前执行中的线程，无法强制休眠其他线程。

注意：**sleep()** 方法是静态的，因此正确使用方式为 **Thread.sleep()**，不建议通过“对象.sleep()”的形式使用。

示例代码如下：

```
public class TestSleep extends Thread {  
  
    public TestSleep() {  
        super();  
    }  
  
    public TestSleep(String name) {
```

```
        super(name);
    }

    public static void main(String[] args) {
        TestSleep st = new TestSleep("Thread one");
        st.start();
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                if (i == 5) {
                    System.out.println("Sleeping Thread one 3s");
                    Thread.sleep(3000);
                    System.out.println("Waking Thread one");
                }
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(this.getName() + "---" + i);
        }
    }
}
```

输出结果如下:

```
Thread one---0
Thread one---1
Thread one---2
Thread one---3
Thread one---4
Sleeping Thread one 3s
Waking Thread one
Thread one---5
Thread one---6
Thread one---7
Thread one---8
Thread one---9
```

疑难点评

在线程正在运行时, 可以通过 `sleep()` 方法使线程休眠一段时间, 此时可以将系统资源让给其他线程程序。

知识链接

FAQ6.02 如何创建和启动一个线程?

FAQ6.07 如何终止一个正在运行的线程?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

终止一个正在运行的线程, Java 线程系统中提供的方法有很多, 但是考虑到线程安全和其他的一些不确定因素, Thread 类中提供的 stop() 和 suspend() 方法是不推荐使用的, 具体原因请参见 FAQ6.08 “为何 stop() 和 suspend() 方法不推荐使用?”。

除了 stop() 和 suspend() 方法可以终止线程之外, Thread 类中还提供了一个 interrupt() 方法, 使用 Thread.isInterrupted() 和 Thread.interrupted() 便可以实现线程的安全终止。

示例代码如下:

```
public class TestStop extends Thread {  
  
    public static void main(String[] args) {  
        TestStop thread = new TestStop();  
        thread.start();  
    }  
  
    public void run() {  
        int count = 0;  
        while (!this.isInterrupted()) {  
            System.out.println("---" + count + "---" + this.isAlive());  
            count++;  
            if (count == 5) {  
                this.interrupt();  
                System.out.println("---" + count + "---" + this.isAlive());  
            }  
        }  
    }  
}
```

输出结果如下:

```
---0---true  
---1---true  
---2---true  
---3---true  
---4---true  
---5---true
```

注意: 通常情况下, 要安全终止一个线程, 都是采用一个标志位来控制 run() 方法正常结束, 而不使用 Thread 类中提供的方法, 这是因为如果涉及多线程通信问题时, 很容易造成不安全性和出现异常。

疑难点评

在线程 API 初期, 提供了一些结束线程的方法, 例如 stop()和 suspend()等。在使用线程时, 不推荐使用 stop()和 suspend()等方法结束线程, 如果需要结束线程, 可以使用线程的 run()方法正常退出。

知识链接

FAQ6.08 为何 stop()和 suspend()方法不推荐使用?

FAQ6.09 如何控制线程的暂停和启动?

FAQ6.08 为何 stop()和 suspend()方法不推荐使用?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

(1) stop()方法

stop()方法作为一种粗暴的线程终止行为, 在线程终止之前没有对其做任何的清除操作, 因此具有固有的不安全性。

用 Thread.stop()方法来终止线程将会释放该线程对象已经锁定的所有监视器。如果以前受这些监视器保护的任意对象都处于一种不连贯状态, 那么损坏的对象对其他线程可见, 这有可能导致不安全的操作。

由于上述原因, 因此不应该使用 stop()方法, 而应该在自己的 Thread 类中置入一个标志, 用于控制目标线程是活动还是停止。如果该标志指示它要停止运行, 可使其结束 run()方法。如果目标线程等待很长时间, 则应使用 interrupt()方法来中断该等待。

(2) suspend()方法

该方法已经遭到反对, 因为它具有固有的死锁倾向。调用 suspend()方法的时候, 目标线程会停下来。如果目标线程挂起时在保护关键系统资源的监视器上保持有锁, 则在目标线程重新开始以前, 其他任何线程都不能访问该资源。除非被“挂起”的线程恢复运行。对任何其他线程来说, 如果想恢复目标线程, 同时又试图使用任何一个锁定的资源, 就会造成死锁。

由于上述原因, 因此不应该使用 suspend()方法, 而应在自己的 Thread 类中置入一个标志, 用于控制线程是活动还是挂起。如果标志指出线程应该挂起, 那么用 wait()方法命令其进入等待状态。如果标志指出线程应当恢复, 那么用 notify()方法重新启动线程。

疑难点评

stop()和 suspend()方法虽然在早期 API 中给出, 但由于其不安全, 因此不推荐使用。推荐使

用正常结束 `run()` 方法的形式来结束线程。

知识链接

FAQ6.07 如何终止一个正在运行的线程?

FAQ6.09 如何控制线程的暂停和启动?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

(1) 使用 `sleep()` 方法

`sleep()` 方法允许指定以毫秒为单位的一段时间作为参数, 它使得线程在指定的时间内进入阻塞状态, 不能得到 CPU 时间, 指定的休眠时间过去之后, 线程重新进入可执行状态。

`sleep()` 方法在具体的应用时, 可能出现一定的偏差。例如 `sleep(3600000)`, 线程休眠一小时之后返回可执行状态等待, 而不是立即执行。因此在实际操作中, 如果实现类似功能, 只使用 `sleep()` 方法显然是不行的, 因为线程苏醒之后等待 CPU 再次调度的时间是不可预知的。如果对一个正在 `sleeping` 的线程使用 `Interrupt()` 方法中断时将会抛出一个 `InterruptedException` 异常。

(2) 使用 `wait()` 和 `notify()` 方法

`wait()` 和 `notify()` 这两种方法通常配套使用, `wait()` 使得线程进入阻塞状态, 它有两种形式, 一种允许指定以毫秒为单位的一段时间作为参数, 另一种没有参数。前者当对应的 `notify()` 方法被调用或者超出指定时间时线程重新进入可执行状态, 后者则必须当对应的 `notify()` 方法被调用时生效。

示例代码如下:

```
public class TestWait {
    public static void main(String[] args) {
        Mythread thread1 = new Mythread("Thread one");
        Mythread thread2 = new Mythread("Thread two");
        try {
            Thread.sleep(2000);
            thread1.mywait();
            System.out.println("Suspend Thread one");
            Thread.sleep(2000);
            thread1.mynotify();
            System.out.println("Resume Thread two");
            thread2.mywait();
            System.out.println("Suspend Thread one");
            Thread.sleep(2000);
            thread2.mynotify();
            System.out.println("Resume Thread two");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```



```
    } catch (InterruptedException ex) {
        System.out.println("Main thread Interrupted.");
    }
    try {
        System.out.println("Waiting for threads to finish.");
        thread1.t.join();
        thread2.t.join();
    } catch (InterruptedException ex) {
        System.out.println("Main thread Interrupted.");
    }
    System.out.println("Main Thread exit");
}
}
```

```
class Mythread implements Runnable {
    String name;
    Thread t;
    boolean waitFlag;

    Mythread(String threadname) {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("New Thread: " + t);
        waitFlag = false;
        t.start();
    }

    public void run() {
        try {
            for (int i = 10; i > 0; i--) {
                System.out.println(name + ": " + i);
                Thread.sleep(1000);
                synchronized (this) {
                    while (waitFlag) {
                        wait();
                    }
                }
            }
        } catch (InterruptedException ex) {
            System.out.println(name + " interrupted.");
        }
        System.out.println(name + "exit");
    }

    void mywait() {
        waitFlag = true;
    }

    synchronized void mynotify() {
```

```
        waitFlag = false;
        notify();
    }
}
```

输出结果如下:

```
New Thread: Thread[Thread one,5,main]
New Thread: Thread[Thread two,5,main]
Thread two: 10
Thread one: 10
Thread two: 9
Thread one: 9
Suspend Thread one
Thread two: 8
Thread two: 7
Resume Thread one
Suspend Thread two
Thread one: 8
Thread one: 7
Resume Thread two
Thread two: 6
Thread one: 6
Waiting for threads to finish.
Thread two: 5
Thread one: 5
Thread two: 4
Thread one: 4
Thread two: 3
Thread one: 3
Thread two: 2
Thread one: 2
Thread one: 1
Thread two: 1
Thread one exit
Main Thread exit
Thread two exit
```

(3) 使用 yield()方法

yield()方法可以使正在运行的线程放弃当前分得的 CPU 时间,但是不会使线程阻塞,即线程仍处于可执行状态,随时可能再次分得 CPU 时间。调用 yield()方法的效果相当于调度程序认为该线程已执行了足够的时间从而转到另一个线程。

示例代码如下:

```
public class TestYield extends Thread {

    TestYield() {
        super();
    }

    TestYield(String name) {
```



```
        super(name);
    }

    public static void main(String[] args) {
        TestYield t1 = new TestYield("Thread one");
        TestYield t2 = new TestYield("Thread two");
        t1.setPriority(2);
        t2.setPriority(6);
        t2.yield();
        t1.start();
        t2.start();
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(this.getName() + i);
        }
    }
}
```

输出结果如下:

```
Thread one 0
Thread two 0
Thread one 1
Thread two 1
Thread one 2
Thread two 2
Thread one 3
Thread two 3
Thread one 4
Thread two 4
Thread two 5
Thread one 5
Thread two 6
Thread one 6
Thread two 7
Thread one 7
Thread one 8
Thread two 8
Thread one 9
Thread two 9
```

(4) 使用 join() 方法

join() 方法允许指定一个以毫秒为单位的参数, 表示等待该线程终止的最长时间, 如果没有

参数则表示等待该线程执行完毕，再调度其他线程。但是如果另一个线程中断了当前线程，则会抛出 `InterruptedException` 异常。

示例代码如下：

```
public class TestJoin extends Thread {

    TestJoin() {
        super();
    }

    TestJoin(String name) {
        super(name);
    }

    public static void main(String[] args) {
        TestJoin t1 = new TestJoin("Thread one");
        TestJoin t2 = new TestJoin("Thread two");
        t1.setPriority(2);
        t2.setPriority(6);
        t1.start();
        t2.start();
        try {
            t1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(this.getName() + i);
        }
    }
}
```

输出结果如下：

```
Thread one 0
Thread two 0
Thread one 1
Thread two 1
Thread one 2
Thread two 2
Thread one 3
```

```
Thread two 3
Thread one 4
Thread two 4
Thread one 5
Thread two 5
Thread one 6
Thread two 6
Thread one 7
Thread two 7
Thread one 8
Thread two 8
Thread one 9
Thread two 9
```

疑难点评

线程的暂停和启动功能在很多软件中都有应用, 例如下载软件和计时器等。

知识链接

FAQ6.07 如何终止一个正在运行的线程?

FAQ6.10 如何实现多个线程同步?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

在编写一个类时, 如果该类中的代码可能运行于多线程环境下, 那么就要考虑同步的问题。Java 实现线程同步的方法很多, 具体如下。

(1) synchronized 关键字

在 Java 中内置了语言级的同步原语 `synchronized` 关键字, 其在多线程条件下实现了对共享资源的同步访问。根据 `synchronized` 关键字修饰的对象不同可分为以下几种情况。

❑ `synchronized` 关键字同步方法

```
public synchronized void method(){
    //do something
}
```

如果使用 `synchronized` 关键字同步方法, 很容易误认为同步关键字锁住了它所包围的代码。但是实际情况却不是这样, 同步加锁的是对象而并非代码。因此, 如果在一个类中有一个同步方法, 该方法是可以被两个不同的线程同时执行的, 只要每个线程自己创建一个该类的实例即可。

示例代码如下:

```
public class TestSync{
    public static void main(String args[]) {
```

```

        MyThread my1 = new MyThread(1);
        my1.start();
        MyThread my2 = new MyThread(3);
        my2.start();
    }
}

class MyThread extends Thread {
    private int val;

    public MyThread(int v) {
        val = v;
    }

    public synchronized void printVal(int v) {
        for (int i = 0; i < 100; i++) {
            System.out.print(v);
        }
    }

    public void run() {
        printVal(val);
    }
}

```

输出结果如下：

```

131111113333333333333333331313313131
31333333333333333333333333333333
33333111111111111111111111111111
11111111113333333333333333333311
11111111111111111111111111333333

```

从结果可以看出，1 和 3 是交叉输出的，即 1 和 3 两个线程在并发执行 `printVal` 方法，并没有实现同步功能。原因在于 `synchronized` 关键字锁定的是对象而并非代码块，如果真正实现真正的同步，必须同步一个全局对象或者对类进行同步。`synchronized` 关键字同步类的格式如下：

```
synchronized (MyThread.class){}
```

下面对上文示例做了相关改进，具体代码如下：

```

public class TestSync_2 {
    public static void main(String args[]) {
        MyThread my1 = new MyThread(1);
        my1.start();
        MyThread my2 = new MyThread(3);
        my2.start();
    }
}

class MyThread_2 extends Thread {
    private int val;

    public MyThread_2(int v) {

```



```

        val = v;
    }

    public void printVal(int v) {
        synchronized (MyThread_2.class) {
            for (int i = 0; i < 100; i++) {
                System.out.print(v);
            }
        }
    }

    public void run() {
        printVal(val);
    }
}

```

输出结果如下:

```

33333333333333333333333333333333
33333333333333333333333333333333
33333333333333333333333333333333
33333331111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111

```

或者如下:

```

11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111133333333333333333333333333
33333333333333333333333333333333
33333333333333333333333333333333
33333333333333333333333333333333
33333333333333333333333333333333

```

在上面的示例中, `printVal()` 方法的功能不再对个别的类实行同步, 而是对当前类进行同步。对于 `MyThread` 而言, 它只有惟一的类定义, 两个线程在相同的锁上同步, 因此在同一时刻只有一个线程可以执行 `printVal()` 方法。至于输出结果的两种可能, 则是由于 Java 线程调度的抢占式实现模式所决定的。

□ synchronized 关键字同步公共的静态成员变量

在上面的示例中, 通过对当前类进行加锁, 达到了线程同步的效果, 但是基于线程同步的一般原理是应该尽量减小同步的粒度以达到更高的性能。其实针对上文中的示例, 也可以通过公共对象加锁, 即添加一个静态成员变量来实现, 两种方法都通过同步该对象而达到线程安全。示例代码如下:

```

public class TestSync_3 {
    public static void main(String args[]) {
        MyThread_3 my1 = new MyThread_3(1);
        my1.start();
    }
}

```

```

        MyThread_3 my2 = new MyThread_3(3);
        my2.start();
    }
}

class MyThread_3 extends Thread {
    private int val;
    private static Object lock = new Object();

    // private static byte[] lock = new byte[0];
    // private static String str = "";
    public MyThread_3(int v) {
        val = v;
    }

    public void printVal(int v) {
        synchronized (lock) {
            for (int i = 0; i < 100; i++) {
                System.out.print(v);
            }
        }
    }

    public void run() {
        printVal(val);
    }
}

```

输出结果如下：

```

33333333333333333333333333333333
33333333333333333333333333333333
33333333333333333333333333333333
33333331111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111

```

或者如下：

```

11111111111111111111111111111111
11111111111111111111111111111111
11111111111111111111111111111111
11111113333333333333333333333333
33333333333333333333333333333333
33333333333333333333333333333333
33333333333333333333333333333333

```

注意：为了能够提高程序的性能，针对示例代码中对于对象的选取做一个简单介绍：基于 JVM 的优化机制，由于 String 类型的对象是不可变的，因此当用户使用""的形式引用字符串时，如果 JVM 发现内存已经有一个这样的对象，那么它就使用该对象而再生成一个新的 String 对象，这样是为了减小内存的使用；而零长度的 byte 数组对象创建起来将比任何对象都要实用，

生成零长度的 byte[] 对象只需要 3 条操作码, 而 Object lock = new Object() 则需要 7 行操作码。

□ synchronized 关键字同步游离块

```
synchronized{  
    //do something  
}
```

synchronized 关键字同步代码块和上文中所提到的 synchronized 关键字同步公共的静态成员变量类似, 都是为了降低同步粒度避免对整个类进行同步而极大降低了程序的性能, 具体使用请参见该问题前面的示例, 在此不再赘述。

□ synchronized 关键字同步静态方法

```
public synchronized static void methodAAA(){  
    //do something  
}  
  
public void methodBBB(){  
    synchronized(Test.class){  
        //dosomething  
    }  
}
```

synchronized 关键字同步静态方法与 synchronized 关键字同步类实现的效果相同, 惟一不同的是获得的锁对象不同, 当同一个 object 对象访问 methodAAA() 和 methodBBB() 时, 同步静态方法获得的锁就是该 object 对象, 而同步类方法获得的对象锁则是 object 对象所属的那个类。

(2) Metux 互斥体的设计和使用

Metux 称为互斥体, 同样广泛应用于多线程编程中。其中以 Doug Lea 教授编写的 concurrent 工具包中实现的 Mutex 最为典型, 本文将以此为例, 对多线程编程中互斥体的设计和使用做简单介绍。在 Doug Lea 的 concurrent 工具包中, Mutex 实现了 Sync 接口, 该接口是 concurrent 工具包中所有锁 (lock)、门 (gate) 和条件变量 (condition) 的公共接口, Sync 的实现类主要有 Mutex、Semaphore 及其子类、Latch、CountDown 和 ReentrantLock 等。这也体现了面向对象抽象编程的思想, 使开发者可以在不改变代码或者改变少量代码的情况下, 选择使用 Sync 的不同实现。Sync 接口的定义如下:

```
public interface Sync {  
    public void acquire() throws InterruptedException; //获取许可  
    public boolean attempt(long msecs) throws InterruptedException; //尝试获取许可  
    public void release(); //释放许可  
}
```

通过使用 Sync 接口可以替代 synchronized 关键字, 并提供更加灵活的同步控制。但是并不是说 concurrent 工具包是独立于 synchronized 的技术, 其实 concurrent 工具包也是在 synchronized 的基础上搭建的。区别在于 synchronized 关键字仅在方法内或者代码块内有效, 而使用 Sync 却可以跨越方法甚至通过在对象之间传递, 跨越对象进行同步。这是 Sync 及 concurrent 工具包比直接使用 synchronized 更加强大的地方。

需要注意的是 Sync 中的 acquire() 和 attempt() 方法都会抛出 InterruptedException 异常, 因此

在使用 Sync 及其子类时, 调用这些方法一定要捕获 InterruptedException。而 release()方法并不会抛出 InterruptedException 异常, 这是因为在 acquire()和 attempt()方法中都有可能会调用 wait()方法等待其他线程释放锁。而 release()方法在实现上进行了简化, 不管是否真的持有锁, 都直接释放。因此, 如果对一个并没有 acquire()方法的线程调用 release()方法不会存在什么问题。而由于 release()方法不会抛出 InterruptedException, 因此必须在 catch 或 finally 子句中调用 release()方法以保证获得的锁能够被正确释放。示例代码如下:

```
import com.sun.corba.se.impl.orbutil.concurrent.Sync;
```

```
public class TestSync_4 {  
  
    Sync gate;  
  
    public void test() {  
        try {  
            gate.acquire();  
  
            try {  
                // do some thing  
            } finally {  
                gate.release();  
            }  
        } catch (InterruptedException ex) {  
  
        }  
    }  
}
```

Mutex 是一个非重入的互斥锁。广泛应用于需要跨越方法的 before or after 类型的同步环境中。下面是一个 Doug Lea 的 concurrent 工具包中的 Mutex 的实现, 代码如下:

```
import com.sun.corba.se.impl.orbutil.concurrent.Sync;
```

```
public class TestMutex implements Sync {  
  
    protected boolean flg = false;  
    public void acquire() throws InterruptedException {  
        if (Thread.interrupted())  
            throw new InterruptedException(); //(1)  
        synchronized (this) {  
            try {  
                while (flg)  
                    wait();  
                flg = true;  
            } catch (InterruptedException ex) { //(2)  
                notify();  
                throw ex;  
            }  
        }  
    }  
}
```

```

    }

    public synchronized void release() {
        flg = false;
        notify();
    }

    public boolean attempt(long msec) throws InterruptedException {
        if (Thread.interrupted())
            throw new InterruptedException();
        synchronized (this) {
            if (!flg) {
                flg = true;
                return true;
            } else if (msec <= 0)
                return false;
            else {
                long waitTime = msec;
                long start = System.currentTimeMillis();
                try {
                    for (;;) {
                        wait(waitTime);
                        if (!flg) {
                            flg = true;
                            return true;
                        } else {
                            waitTime = msec
                                - (System.currentTimeMillis() - start);
                            if (waitTime <= 0) //(3)
                                return false;
                        }
                    }
                } catch (InterruptedException ex) {
                    notify();
                    throw ex;
                }
            }
        }
    }
}

```

在上述示例中，在 `acquire()` 和 `attempt()` 方法的开始都要检查当前线程的中断标志是为了在当前线程已经被打断时可以立即返回，而不会在锁标志上等待。调用一个线程的 `interrupt()` 方法根据当前线程所处的状态，可能产生两种不同的结果，即当线程在运行过程中被打断，则设置当前线程的中断标志为 `true`；当前线程阻塞于 `wait()`、`sleep()` 或 `join()` 方法，则当前线程的中断标志被清空，同时抛出 `InterruptedException`。因此在上面代码的位置 (2) 也捕获了 `InterruptedException`，然后再次抛出 `InterruptedException`。`release()` 方法简单地重置 `flg` 标志，并通知其他线程。`attempt()` 方法是利用 Java 的 `Object.wait(long)` 进行计时的，由于 `Object.wait(long)`

不是一个精确的时钟，因此 `attempt(long)` 方法也是一个粗略的计时。注意代码中位置 (3) 是用来在超时时返回。

示例中给出的 `Mutex` 类是 `Sync` 的一个基本实现，除了实现了 `Sync` 接口中的方法外，并没有添加新的方法。因此，`Mutex` 的使用和 `Sync` 的完全相同。其实在 `concurrent` 包的 API 中 Doug 给出了很多更加精细锁定的实现，由于篇幅有限，在此不再赘述，读者可参考 JDK 1.5 中 `Java.util.concurrent` 工具包相关介绍。

(3) ThreadLocal 的设计与使用

早在 Java 1.2 推出之时，Java 平台中就引入了一个新的支持，即 `java.lang.ThreadLocal` 类，它为编写多线程程序提供了一种新的选择。使用此工具类可以很简洁地编写出多线程程序。

`ThreadLocal` 指的是一个线程的本地实现，但是它并不是一个线程，确切地说它只是一个线程局部变量 (`thread local variable`)，它为每一个使用该变量的线程都提供一个变量值的副本，每一个线程都可以独立的修改自己的副本，而不会和其他的线程副本产生任何冲突。从线程的角度看，就好像每一个线程都完全拥有改变量。

与其他所有的同步机制相同，`ThreadLocal` 同样是为了解决多线程中的对同一变量的访问冲突，不同的是在普通的同步机制中是通过对象加锁来实现多个线程对同一变量的安全访问的。这时该变量是多个线程共享的，使用这种同步机制需要很细致地分析在什么时候对变量进行读写，什么时候需要锁定某个对象，什么时候释放该对象的锁等问题。这些都是因为多个线程共享了资源造成的，而 `ThreadLocal` 却从另一个角度来解决多线程的并发访问，它为每一个线程维护一个和该线程绑定的变量的副本，从而隔离了多个线程的数据，每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。

如果读者对 Spring、Hibernate 框架有所了解，相信对 `ThreadLocal` 将会并不陌生，Spring 框架的事务管理、任务调度、Aop 编程以及 Hibernate 中的 session 会话和连接池的实现都是 `ThreadLocal` 的经典应用。下面是 Hibernate 官方文档手册中通过 `ThreadLocal` 维护 session 的示例。

```
public class HibernateUtil {
    public static final SessionFactory sessionFactory;
    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static final ThreadLocal session = new ThreadLocal();
    public static Session currentSession() throws HibernateException {
        Session s = session.get();
        if(s == null) {
            s = sessionFactory.openSession();
            session.set(s);
        }
        return s;
    }
}
```



```
}  
public static void closeSession() throws HibernateException {  
    Session s = session.get();  
    if(s != null) {  
        s.close();  
    }  
    session.set(null);  
}  
}
```

ThreadLocal 提供了线程安全的共享对象,在编写多线程代码时,只需要把不安全的整个变量封装进 ThreadLocal,或者把该对象的特定于线程的状态封装进 ThreadLocal,即可完全解决多线程同步访问数据的问题。但是由于在 Java 中并没有在线程局部变量上提供语言层次的直接支持,使用起来比较麻烦,因此不能像 synchronized 关键字一样得到编程者的青睐。

注意: ThreadLocal 并不能替代同步机制,因为二者面向的问题领域并不相同。同步机制是为了同步多个线程对相同资源的并发访问,是多个线程之间进行通信的有效方式;而 ThreadLocal 是隔离多个线程的数据共享,从根本上就不在多个线程之间共享资源(变量),这样就不需要对多个线程进行同步。因此如果需要进行多个线程之间进行通信,则使用同步机制;如果需要隔离多个线程之间的共享冲突,可以使用 ThreadLocal,这将极大地简化用户的程序,使程序更加易读、简洁。

疑难点评

在多个线程执行时,由于资源争夺很容易发生死锁和数据混乱等问题,使用上述方法可以有效地避免多线程引发的问题。

知识链接

FAQ6.09 如何控制线程的暂停和启动?

FAQ6.11 什么是对象序列化和对象反序列化?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

序列化是一种用来处理对象流的机制,所谓对象流也就是将对象的内容进行流化。对象序列化和反序列化的过程就是将对象写入字节流和从字节流中读取对象的过程。将对象状态转换成字节流之后,可以用 java.io 包中的各种 I/O 流类将其保存到文件中,或者应用 NIO 技术将其传输到另一线程中或通过网络连接将对象数据发送到另一主机。对象序列化功能非常简单强大,在 RMI、Socket、JMS 和 EJB 中都有应用。

对象序列化机制是为了解决对象在磁盘上读写操作和在网络中传递出现的问题而提出的,通过流化后的对象可以通过调用该类的 `writeObject()` 方法方便地向特定的文件或者网络输出对象,另一方也可以通过 `readObject()` 方法方便地接收该对象。

序列化的实现场景有以下几种。

- ☐ 永久性保存对象,保存对象的字节序列到本地文件中。
- ☐ 对象序列化可以实现分布式对象。例如 RMI 要利用对象序列化运行远程主机上的服务,与在本地机上运行对象时相同。
- ☐ 通过序列化对象在网络中传递对象。
- ☐ 通过序列化在进程间传递对象。

疑难点评

对象序列化是 Java 中一个非常重要的概念,在很多情况下,都需要使用对象序列化,读者需要了解序列化的概念、作用及其使用环境。

知识链接

FAQ6.12 实现对象序列化的方法有哪些?

FAQ6.12 实现对象序列化的方法有哪些?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

Java 序列化有两种实现方法,一种在类定义的时候实现 `Serializable` 接口,另一种是实现 `Externalizable` 接口。实现了这两种接口的类的对象便可以转换成字节流或从字节流恢复,不需要在类中增加任何代码。只有极少数情况下才需要定制代码保存或恢复对象状态,这涉及显式定制序列化过程,在后文中将做相关介绍,在此不再赘述。

注意:一般选择使用 `Serializable` 接口实现序列化,因为该接口不需要实现任何方法;而 `Externalizable` 接口定义了 `writeExternal()` 和 `readExternal()` 方法,实现该接口的类必须要实现这两种方法。

(1) 利用 `Serializable` 接口实现序列化

示例代码如下:

```
public class Person implements Serializable {  
    private String name;  
  
    private int age;  
  
    private Date birthday;
```

```
public int getAge() {  
    return age;  
}  
  
public void setAge(int age) {  
    this.age = age;  
}  
  
public Date getBirthday() {  
    return birthday;  
}  
  
public void setBirthday(Date birthday) {  
    this.birthday = birthday;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
}
```

(2) 利用 Externalizable 接口实现序列化
示例代码如下:

```
public class Person implements Externalizable {  
    private String name;  
  
    private int age;  
  
    private Date birthday;  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
  
    public Date getBirthday() {  
        return birthday;  
    }  
  
    public void setBirthday(Date birthday) {  
        this.birthday = birthday;  
    }  
}
```



```
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        // TODO 添加反序列化代码
    }

    public void writeExternal(ObjectOutput out) throws IOException {
        // TODO 添加序列化代码
    }
}
```

利用 `Externalizable` 接口实现序列化, 需要使用者自己定义序列化和反序列化过程, 即实现 `Externalizable` 接口中的 `readExternal()` 和 `writeExternal()` 方法。具体实现示例可参考 FAQ6.17 “如何使用 `Externalizable` 接口定制序列化过程?”。

疑难点评

上述两种实现对象序列化的方法, 程序员一般习惯使用 `Serializable` 接口, 因为这种方式实现起来非常方便。为了保障对象能够实现序列化, 对象类不仅需要实现 `Serializable` 或 `Externalizable` 接口, 而且它的每个属性的类型也都要具有序列化特性, 如果某个属性不具有序列化特性, 并且有没有使用 `static` 和 `transient` 关键字修饰, 在使用时将出现错误。`static` 和 `transient` 关键字在序列化中的作用可以参考 FAQ6.15 “对象中的成员哪些参与序列化? 哪些不参与序列化?”。

知识链接

FAQ6.17 如何使用 `Externalizable` 接口定制序列化过程?

FAQ6.11 什么是对象序列化和对象反序列化?

FAQ6.15 对象中的成员哪些参与序列化? 哪些不参与序列化?

FAQ6.13 如何实现对象在磁盘中的存取操作?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

对象序列化的过程分为序列化和反序列化两大部分。序列化是该过程的第 1 部分, 将数据分解成字节流, 以便存储在文件中或在网络上传输。反序列化就是打开字节流并重构对象。具体可以通过 `Java.io.ObjectOutputStream` 类中的 `writeObject(Object obj)` 方法和 `Java.io.ObjectInputStream` 类中的 `readObject(Object obj)` 方法来实现。

`String` 和 `Date` 类都已经实现 `Serializable` 接口, 以 `String` 类型对象和 `Date` 类型对象为例, 实现磁盘存取的操作代码如下:

❑ 向文件中写入序列化对象

```
public class TestWrite {

    public static void main(String[] args) {
        TestWrite e = new TestWrite();
        try {
            e.write();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }

    public void write() throws IOException {
        FileOutputStream fos = new FileOutputStream("C://tmp.txt");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject("Today");
        oos.writeObject(new Date());
        oos.flush();
    }
}
```

注意: 以对象为单位进行文件写入时, 该对象类型必须是可序列化的, 否则会抛出 `NotSerializableException` 异常。

❑ 从文件中读取序列化对象

```
public class TestRead {

    public static void main(String[] args) {
        TestRead e = new TestRead();
        try {
            e.read();
        } catch (IOException e1) {
            e1.printStackTrace();
        } catch (ClassNotFoundException e1) {
            e1.printStackTrace();
        }
    }

    public void read() throws IOException, ClassNotFoundException {
```

```
FileInputStream fis = new FileInputStream("C://tmp.txt");
ObjectInputStream ois = new ObjectInputStream(fis);
String today = (String) ois.readObject();
System.out.println("-----" + today);
Date date = (Date) ois.readObject();
System.out.println("-----" + date.toString());

}
```

输出结果如下:

```
-----Today
-----Sun Nov 30 23:57:24 CST 2008
```

疑难点评

以对象为单位在网络中传递和在磁盘上存储时, 需要使用序列化功能。上述示例介绍了对象序列化的一种具体应用。

知识链接

FAQ6.12 实现对象序列化的方法有哪些?

FAQ6.14 使用 ObjectInputStream 读取对象时为什么会发生 StreamCorruptedException 异常?

FAQ6.14 使用 ObjectInputStream 读取对象时为什么会发生 StreamCorruptedException 异常?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

StreamCorruptedException 是流中的控制信息不一致异常, 当从对象流中读取的控制信息与内部一致性检查相冲突时, 就会抛出此异常。

在使用 ObjectInputStream 读取文件时, 比较容易发生 StreamCorruptedException 异常。例如通过 ObjectOutputStream 向一个文件中多次以追加方式写入对象, 使用 ObjectInputStream 读取这些对象时会产生 StreamCorruptedException。示例代码如下:

```
public static void main(String[] args) {
    try {
        //用两个 ObjectOutputStream 对象向 p.txt 中写入两个 Person 对象
        for (int i = 0; i < 2; i++) {
            FileOutputStream fos = new FileOutputStream("F:\\p.txt", true);
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(new Person("zhang", 20, "男"));
        }
    }
}
```



```
        oos.close();
    }
    //用一个 ObjectInputStream 对象读文件中的两个 Person 对象
    FileInputStream fis = new FileInputStream("F:\\p.txt");
    ObjectInputStream ois = new ObjectInputStream(fis);
    Person p = (Person) ois.readObject();
    System.out.println(p.getName());
    Person p1 = (Person) ois.readObject();
    System.out.println(p1.getName());

} catch (Exception e) {
    e.printStackTrace();
}
}
```

Person 类的实现代码如下:

```
public class Person implements java.io.Serializable {
    private String name;

    private int age;

    private String sex;

    public Person() {
    }

    public Person(String name, int age, String sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSex() {
        return sex;
    }
}
```

```
    }  
  
    public void setSex(String sex) {  
        this.sex = sex;  
    }  
  
}
```

上述代码的执行结果如下:

```
zhang  
java.io.StreamCorruptedException: invalid type code: AC  
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1356)  
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:351)  
    at Test.main(Test4.java:26)
```

上述异常发生的原因是由于使用缺省的序列化机制,该机制在实现时要求 `ObjectOutputStream` 对象和 `ObjectInputStream` 对象必须一一对应,即一个 `ObjectOutputStream` 对象输出的对象要使用一个 `ObjectInputStream` 对象读入。

当创建 `ObjectOutputStream` 对象时,其构造函数会向输出流中写入一个标识头,而 `ObjectInputStream` 会首先读入这个标识头。示例代码用两个不同的 `ObjectOutputStream` 对象向文件写入,该文件将会包含两个不同的标识头。当使用一个 `ObjectInputStream` 对象读文件时,只能与第一个 `ObjectOutputStream` 对象的标示头对应,在读第二个 `Person` 对象时,又发现另一个标示头无法对应,因此第一个 `Person` 对象可以顺利读入,在读第二个 `Person` 时产生了 `StreamCorruptedException` 异常。

上述示例的正确写法如下:

```
public static void main(String[] args) {  
    try {  
        // 用一个 ObjectOutputStream 对象向 p.txt 中写入两个 Person 对象  
        FileOutputStream fos = new FileOutputStream("F:\\p.txt", true);  
        ObjectOutputStream oos = new ObjectOutputStream(fos);  
        for (int i = 0; i < 2; i++) {  
            oos.writeObject(new Person("zhang", 20, "男"));  
        }  
        oos.close();  
  
        // 用一个 ObjectInputStream 对象读文件中的两个 Person 对象  
        FileInputStream fis = new FileInputStream("F:\\p.txt");  
        ObjectInputStream ois = new ObjectInputStream(fis);  
        Person p = (Person) ois.readObject();  
        System.out.println(p.getName());  
        Person p1 = (Person) ois.readObject();  
        System.out.println(p1.getName());  
  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

注意：ObjectOutputStream 和 ObjectInputStream 在读写对象时，要求该对象必须可以被序列化和反序列化。

疑难点评

在实现对象的文件存取和网络传递时，都需要使用到 ObjectOutputStream 和 ObjectInputStream 类。在使用时一定要注意一一对应的原则，否则就会发生 StreamCorruptedException 异常。

知识链接

FAQ7.08 如何利用 Socket 传递对象信息？

FAQ6.15 对象中的成员哪些参与序列化？哪些不参与序列化？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

Java 对象序列化时参与序列化的内容包含以下几个方面。

- ☐ 属性，包括基本数据类型、数组以及其他对象的引用。
- ☐ 类名。

不能被序列化的内容有以下几个方面。

- ☐ 方法。
- ☐ 有 static 修饰的属性。
- ☐ 有 transient 修饰的属性。

在序列化过程中不仅保留当前类对象的数据，而且递归保存对象引用的每个对象的数据。将整个对象层次写入字节流中，这也就是序列化对象的“深复制”，即复制对象本身及引用的对象本身。序列化一个对象将可能得到整个对象序列。

在序列化的过程中，由于有些属性值比较敏感（例如密码），或者有些属性值的信息量比较大，它们不需要在网络中传递或在磁盘中存储，即不需要参与序列化。对于此类属性只需要在定义时为其添加 transient 关键字即可，对于 transient 属性序列化机制会跳过而不会将其写入文件，但在读取时也不可被恢复，该属性值保持默认初始化值。

示例代码如下。

- ☐ 定义序列化类型 Person

```
public class Person implements java.io.Serializable{
```

```
    public String name;
```



```
public String sex;
public transient int age;
public static String address;

public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getSex() {
    return sex;
}
public void setSex(String sex) {
    this.sex = sex;
}
public int getAge() {
    return age;
}
public void setAge(int age) {
    this.age = age;
}
public int getAge() {
    return age;
}
public void setAge(int age) {
    this.age = age;
}
public String toString(){
    return "姓名: "+name+"性别: "+sex+"年龄: "+age+"地址: "+address;
}
}
```

□ 对 Person 对象执行读写操作

```
public class TestSeria {

    public static void main(String[] args) {
        TestSeria e = new TestSeria();
        try {
            // e.write();
            e.read();
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }

    public void write() throws IOException {
        FileOutputStream fos = new FileOutputStream("C://person.txt");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
    }
}
```

```
Person p = new Person();
p.setName("tom");
p.setSex("male");
p.setAge(22);
p.setAddress("beijing");
oos.writeObject(p);
oos.flush();
}

public void read() throws IOException, ClassNotFoundException {
    FileInputStream fis = new FileInputStream("C://person.txt");
    ObjectInputStream ois = new ObjectInputStream(fis);
    Person person = (Person) ois.readObject();
    System.out.println("-----" + person.toString());
}
}
```

输出结果如下:

```
-----姓名: tom 性别: male 年龄: 0 地址: null //age 属性有 transient 关键字修饰时
-----姓名: tom 性别: male 年龄: 22 地址: null
```

通过上述结果可知, 当 age 属性 transient 关键字修饰时, 该属性值没有参与文件的读写操作。age 属性值为 0 是由于 Person 对象在初始化时 int 类型属性默认的初始化值为 0, 其他类型默认初始化规则可参见 FAQ2.13。

疑难点评

有些时候, 对象中的某些信息不需要参与网络传递和磁盘存储, 可以使用 transient 关键字将这些信息隐藏, 不参与序列化过程。此外, static 关键字所修饰的属性也不会参与序列化过程。

FAQ6.16 如何自定义序列化和反序列化过程?

📖 难度系数: ★★★★★

📖 问题频率: 88%

核心解答

在对象序列化的过程中, 如果涉及一些敏感信息, 例如用户密码、信用卡账号等, 通常需考虑到保密和如何防止在传递过程中泄露的问题。使用 transient 关键字可以保护这些属性, 但是这些属性将不会参与序列化过程。此时可以考虑显式定制序列化和反序列化的过程, 在序列化时加入加密、在反序列化时加入解密等操作, 而不是使用默认的序列化方式。

对象在序列化和反序列化时, 是调用内部的 writeObject() 和 readObject() 方法实现的。writeObject() 和 readObject() 方法的定义如下:


```
private void writeObject(ObjectOutputStream out)throws IOException  
private void readObject(ObjectInputStream in)throws IOException,ClassNotFoundException
```

如果显示定制序列化和反序列化过程, 重写上述两个方法即可。如果需要在 `writeObject()` 方法中使用默认的序列化机制, 只需要在 `writeObject()` 方法中调用 `out.defaultWriteObject()` 方法即可; 同样如果需要在 `readObject()` 方法中使用默认的序列化机制, 只需要在 `readObject()` 方法中调用 `out.defaultReadObject()` 方法。

注意: 为了防止在对象序列化过程中调用默认的 `writeObject()` 和 `readObject()` 方法, 必须将这两个方法设置为 `private`。

在自定义序列化和反序列化过程中, 对于简单类型的属性, 必须使用默认的 `writeInt (int i)`、`readInt (int i)`、`writeDouble (double d)` 和 `readDouble (double d)` 等方法对简单类型进行读写操作; 而对于引用类型的属性, 可以在 `writeObject(ObjectOutputStream out)` 和 `readObject(ObjectInputStream in)` 中使用 `readObject()` 和 `writeObject()` 方法进行读写。另外, 在文件或者网络中读取属性时, 读取的顺序必须与写入的顺序保持一致。

示例代码如下:

□ 定义 `Person` 类并重写 `writeObject()` 和 `readObject()` 方法。

```
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
  
public class Person implements java.io.Serializable {  
  
    private String name;  
  
    private int age;  
  
    private String sex;  
  
    private String password;  
  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getAge() {  
        return age;  
    }  
    public void setAge(int age) {  
        this.age = age;  
    }  
    public String getSex() {  
        return sex;  
    }  
}
```



```

    public void setSex(String sex) {
        this.sex = sex;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    //实现 writeObject(ObjectOutputStream out)方法
    public void writeObject(ObjectOutputStream out) throws IOException {
        String psw = reverse(password);
        out.writeObject(name);
        out.writeInt(age);
        out.writeObject(sex);
        out.writeObject(psw);
    }
    //实现 readObject(ObjectInputStream in)方法
    public void readObject(ObjectInputStream in) throws IOException,
        ClassNotFoundException {
        this.name = (String) in.readObject();
        this.age = in.readInt();
        this.sex = (String) in.readObject();
        this.password = (String) in.readObject();
    }
    //通过简单的反转字符串自定义密码的序列化方法
    public String reverse(String in) {
        StringBuffer sb = new StringBuffer(in);
        sb.reverse();
        return sb.toString();
    }
    //重写 toString()方法
    public String toString() {
        return "Name:" + name + "\nAge:" + age + "\nSex:" + sex + "\nPassword:"
            + password;
    }
}

```

对 Person 类的对象执行读写操作

```

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class TestMySeria {

```

```
public static void main(String[] args) {
    TestMySeria tms = new TestMySeria();
    tms.testWrite();
    tms.testRead();
}

public void testWrite() {
    Person p = new Person();
    p.setName("Tom");
    p.setAge(22);
    p.setSex("male");
    p.setPassword("serializable");
    FileOutputStream fos = null;
    ObjectOutputStream oos = null;
    try {
        fos = new FileOutputStream("D:\\person.tmp");
        oos = new ObjectOutputStream(fos);
        oos.writeObject(p);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            fos.close(); //关闭 I/O 流
            oos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public void testRead() {
    Person p = null;
    FileInputStream fis = null;
    ObjectInputStream ois = null;
    try {
        fis = new FileInputStream("D:\\person.tmp");
        ois = new ObjectInputStream(fis);
        p = (Person) ois.readObject();
        System.out.println(p);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            fis.close(); //关闭 I/O 流
            ois.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



```
    }  
    }  
}  
}
```

输出结果如下:

```
Name:Tom  
Age:22  
Sex:male  
Password:serializable
```

疑难点评

一般情况下, 程序员都使用默认的对象序列化和反序列化过程, 如果在序列化和反序列化过程中需要作一些特殊处理, 例如加密等, 可以通过上述方法显示定义序列化和反序列化过程。

知识链接

FAQ6.12 实现对象序列化的方法有哪些?

FAQ6.17 如何使用 Externalizable 接口定制序列化过程?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

Externalizable 接口定义如下:

```
public interface Externalizable extends java.io.Serializable{  
    void writeExternal(ObjectOutput out)throws IOException;  
    void readExternal(ObjectInput in)throws IOException,ClassNotFoundException;  
}
```

Externalizable 接口与 Serializable 接口不同的是声明了 writeExternal(ObjectOutput out)和 readExternal(ObjectInput in)方法用于读写对象流信息。但从保存了序列化对象的介质中读取使用接口实现序列化的对象时, 该接口会调用该类的不带有任何参数的默认构造器来构造对象, 因此使用接口实现序列化时如果自定义了构造器, 则必须同时提供一个不带参数的构造器, 否则在序列化过程中将会出现异常。

示例代码如下:

❑ 定义 Person 类并实现 readExternal()和 writeExternal()方法

```
import java.io.Externalizable;  
import java.io.IOException;  
import java.io.ObjectInput;
```



```
import java.io.ObjectOutput;

public class MyPerson implements Externalizable {

    private String name;
    private int age;
    private double salary;
    //
    public MyPerson() {
        System.out.println("---default constructor---");
    }

    public MyPerson(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException,
        ClassNotFoundException {
        System.out.println("start to read object");
        name = (String) in.readObject();
        age = in.readInt();
        salary = in.readDouble();
    }

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        System.out.println("start to write object");
        out.writeObject(name);
        out.writeInt(age);
        out.writeDouble(salary);
    }

    public String toString() {
        return "Name:" + name + "\nAge:" + age + "\nSalary:" + salary;
    }
}
```

□ 对 Person 类的对象执行读写操作

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class TestExtern {
```

```
public static void main(String[] args) {
    TestExtern te = new TestExtern();
    te.writeObject();
    te.readObject();
}

public void readObject() {
    MyPerson mp = null;
    FileInputStream fis = null;
    ObjectInputStream ois = null;
    try {
        fis = new FileInputStream("D:\\myPerson.tmp");
        ois = new ObjectInputStream(fis);
        mp = (MyPerson) ois.readObject();
        System.out.println(mp);
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
    } finally {
        try {
            fis.close();
            ois.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public void writeObject() {
    MyPerson mp = new MyPerson("Jack", 23, 5000);
    FileOutputStream fos = null;
    ObjectOutputStream oos = null;
    try {
        fos = new FileOutputStream("D:\\myPerson.tmp");
        oos = new ObjectOutputStream(fos);
        oos.writeObject(mp);
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
    } finally {
        try {
            fos.close();
            oos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



```
    }  
    }  
}
```

输出结果如下:

```
start to write object  
---default constructor---  
start to read object  
Name:Jack  
Age:23  
Salary:5000.0
```

疑难点评

使用 `Externalizable` 接口也是对象序列化的一种实现方法,但是要求对象的实现类必须实现 `Externalizable` 接口中定义的 `writeExternal()` 和 `readExternal()` 方法。

知识链接

FAQ6.12 实现对象序列化的方法有哪些?

FAQ6.18 在序列化类中添加 `serialVersionUID` 属性有什么作用?

📖 难度系数: ★★★

📖 问题频率: 85%

核心解答

`serialVersionUID` 是一个私有的静态 `final` 属性,用于表明类之间不同版本的兼容性,该属性不是必须的。当一个类实现了 `java.io.Serializable` 接口,但是没有显式定义一个 `Long` 类型的 `serialVersionUID` 属性,Java 序列化机制会根据编译的 `class` 自动生成一个 `serialVersionUID` 作为该类的序列化版本 ID 号,只有同一次编译生成的 `class` 才会生成相同的 `serialVersionUID`。在反序列化过程中, JVM 会把传来的字节流中的 `serialVersionUID` 与本地类的 `serialVersionUID` 进行比较,如果相同就认为版本一致,可以进行反序列化,否则就会出现序列化版本不一致的 `InvalidClassException` 异常。如果不希望通过编译来强制划分软件版本,即实现序列化接口的类能够兼容以前版本,只需要显式地定义一个名为 `serialVersionUID`、类型为 `Long` 的 `final` 属性,即可保证相同的版本号,且在进行序列化和反序列化时不会出现 `InvalidClassException` 异常。

疑难点评

在定义序列化类时,显式声明属性 `serialVersionUID` 是为了在对象序列化过程中保障程序的

兼容性，但该属性不是必要的。

知识链接

FAQ6.11 什么是对象序列化和对象反序列化？

FAQ6.19 当序列化遭遇继承时，如何正确处理对象序列化过程？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

在序列化过程中，如果父类实现了 `Serializable` 或 `Externalizable` 接口，则其所有子类都将自动实现序列化，用这种方式为子类实现序列化，显然非常简单。

示例代码如下：

❑ 定义父类，实现了序列化

```
public class SuperC implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
    int supervalue;  
  
    public SuperC(int supervalue) {  
        this.supervalue = supervalue;  
    }  
  
    public String toString() {  
        return "supervalue: " + supervalue;  
    }  
}
```

❑ 定义子类

```
public class SubC extends SuperC {  
  
    private static final long serialVersionUID = 1L;  
    int subvalue;  
  
    public SubC(int supervalue, int subvalue) {  
        super(supervalue);  
        this.subvalue = subvalue;  
    }  
  
    public String toString() {  
        return super.toString() + " sub: " + subvalue;  
    }  
}
```

□ 对子类对象进行读写操作

```
public class TestSub {  
  
    public static void main(String[] args) {  
        SubC subc = new SubC(100, 200);  
        FileInputStream fis = null;  
        FileOutputStream fos = null;  
        ObjectInputStream ois = null;  
        ObjectOutputStream oos = null;  
        try {  
            fos = new FileOutputStream("D:\\TestSub.txt");//子类序列化  
            oos = new ObjectOutputStream(fos);  
            oos.writeObject(subc);  
  
            fis = new FileInputStream("Test1.txt");  
            ois = new ObjectInputStream(fis);  
            SubC subc2 = (SubC) ois.readObject();//子类反序列化  
            System.out.println(subc2);  
        } catch (Exception ex) {  
            ex.printStackTrace();  
        } finally {  
            try {  
                fis.close();  
                fis = null;  
                fos.close();  
                fos = null;  
                ois.close();  
                ois = null;  
                oos.close();  
                oos = null;  
  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

输出结果如下:

```
supervalue: 100 sub: 200
```

注意: 在实际操作中, 如果需要对子类进行序列化, 直接将其父类实现序列化的方式是不可取的, 因为父类不能强制其所有子类都具有序列化的能力。

如果父类没有序列化, 而子类却实现了 `Serializable` 接口, 那么子类对象也不能序列化。示例代码如下:

□ 父类未实现序列化

```
public class SuperC {  
  
    private static final long serialVersionUID = 1L;
```

```
int supervalue;

public SuperC(int supervalue) {
    this.supervalue = supervalue;
}

public String toString() {
    return "supervalue: " + supervalue;
}
}
```

定义子类实现了序列化

```
public class SubC extends SuperC implements Serializable {

    private static final long serialVersionUID = 1L;
    int subvalue;

    public SubC(int supervalue, int subvalue) {
        super(supervalue);
        this.subvalue = subvalue;
    }

    public String toString() {
        return super.toString() + " sub: " + subvalue;
    }
}
```

在上述实例中，父类被定义为不可序列化的，而其子类却实现序列化，此时在序列化过程中，会抛出 `java.io.InvalidClassException` 异常。这是因为子类继承了父类的属性，而父类没有序列化。此时正确的做法应该是编写一个能够实现序列化的子类，首先将自身序列化，同时为其父类提供一个无参的构造方法并序列化父类的属性。示例代码如下：

❑ 定义父类并增加无参的构造方法

```
public abstract class SuperAbsC { //定义为抽象类
    int supervalue;

    public SuperAbsC(int supervalue) {
        this.supervalue = supervalue;
    }
    public SuperAbsC() {
        System.out.println("--default constructor--"); //增加一个无参的 constructor
    }
    public String toString() {
        return "supervalue: " + supervalue;
    }
    public int getSupervalue() {
        return supervalue;
    }
    public void setSupervalue(int supervalue) {
        this.supervalue = supervalue;
    }
}
```



```

    }
}

```

□ 定义子类并显示定义序列化和反序列化过程

```

import java.io.IOException;
import java.io.Serializable;

public class Sub_SeriaC extends SuperAbsC implements Serializable {
    private static final long serialVersionUID = 1L;
    private int subvalue;
    public Sub_SeriaC(int supervalue, int subvalue) {
        super(supervalue);
        this.subvalue = subvalue;
    }
    public String toString() {
        return super.toString() + " sub: " + subvalue;
    }
    private void writeObject(java.io.ObjectOutputStream out) throws IOException {
        out.defaultWriteObject(); //先序列化对象
        out.writeInt(supervalue); //再序列化父类的域
    }
    private void readObject(java.io.ObjectInputStream in) throws IOException,
        ClassNotFoundException {
        in.defaultReadObject(); //先反序列化对象
        supervalue = in.readInt(); //再反序列化父类的域
    }
}

```

□ 对子类对象执行读写操作

```

public class TestSub_SeriaC {

    public static void main(String[] args) {
        Sub_SeriaC subc = new Sub_SeriaC(100, 200);
        FileInputStream fis = null;
        FileOutputStream fos = null;
        ObjectInputStream ois = null;
        ObjectOutputStream oos = null;
        try {
            fos = new FileOutputStream("D:\\TestSub.txt");//子类序列化
            oos = new ObjectOutputStream(fos);
            oos.writeObject(subc);

            fis = new FileInputStream("D:\\TestSub.txt ");
            ois = new ObjectInputStream(fis);
            Sub_SeriaC subc2 = (Sub_SeriaC) ois.readObject();//子类反序列化
            System.out.println(subc2);
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            try {
                fis.close();
            }

```

```
        fis = null;
        fos.close();
        fos = null;
        ois.close();
        ois = null;
        oos.close();
        oos = null;

    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

输出结果如下：

```
supervalue: 100 sub: 200
```

疑难点评

在实现类使用继承时，要求父类也必须可序列化，否则在子类序列化时就会出现错误。

知识链接

FAQ6.15 对象中的成员哪些参与序列化？哪些不参与序列化？

新华书店
PDG

第7章

Java 网络编程

本章重点介绍一些与 Java 网络编程相关的疑难问题, Java 的网络功能很强大, 开发和使用也非常简单, 可以方便的访问 Internet 资源和与局域网其他计算机通信。Java 网络编程部分的内容主要涉及网络协议 (例如 TCP、IP、UDP 和 SMTP 等)、URL 技术和 Socket 技术等方面。本章涵盖了与 Java 网络编程相关的各个领域, 主要内容包括常用协议介绍、利用 Socket 实现聊天系统、利用 URL 技术获取 Internet 资源和网络编程常见功能的具体实现等。通过本章的学习, 读者可以了解常用网络协议的基本原理, 熟悉 Java 网络编程时各种常用功能的实现方法。

FAQ7.01 什么是 TCP/IP? 什么是 IP?

📖 难度系数: ★★★

📖 问题频率: 75%

核心解答

1. 什么是 TCP/IP

TCP/IP 是 Transmission Control Protocol/Internet Protocol 的缩写, 即传输控制协议/网际互联协议, 它是 Internet 国际互联网络的基础。TCP/IP 是 1978 年~1979 年由美国国防部高级研究计划局开发的用于异构网络通信的一组协议, 适用于 Internet 中不同计算机系统的互联, 该组协议主要包括 TCP、IP、UDP、ICMP、RIP、TELNET、FTP、SMTP、ARP 和 TFTP 等, 具体含义如下。

- ☐ TCP (Transmission Control Protocol) 传输控制协议。
- ☐ IP (Internet Protocol) 网际协议。
- ☐ UDP (User Datagram Protocol) 用户数据报协议。
- ☐ ICMP (Internet Control Message Protocol) 互联网控制信息协议。
- ☐ SMTP (Simple Mail Transfer Protocol) 简单邮件传输协议。
- ☐ SNMP (Simple Network manage Protocol) 简单网络管理协议。
- ☐ FTP (File Transfer Protocol) 文件传输协议。

□ ARP (Address Resolution Protocol) 地址解析协议。

TCP/IP 通常被看成一个 4 层模型, 分为应用层、传输层、网络层以及网络接口层, 具体介绍如下。

(1) 应用层

应用层向用户提供一组常用的应用程序, 例如电子邮件、文件传输访问、远程登录等。TCP/IP 协议族在这一层面有着很多协议来支持不同的应用, 例如万维网访问用到了 HTTP 协议、文件传输用 FTP、电子邮件发送用 SMTP、域名解析用 DNS 协议、远程登录用 Telnet 协议等, 都属于 TCP/IP 应用层。就用户而言, 看到的是由一个个软件所构筑的大多为图形化的操作界面, 而实际后台运行的便是上述协议。

(2) 传输层

传输层的功能主要是提供应用程序间的通信, 包括格式化信息流和提供可靠传输等。TCP/IP 协议族在这一层的协议有 TCP 和 UDP。

(3) 网络层

网络层是 TCP/IP 协议族中非常关键的一层, 负责相邻计算机之间的通信, 主要定义了 IP 地址格式, 从而能够使得不同应用类型的数据在 Internet 上通畅地传输, IP 协议就是一个网络层协议。

(4) 网络接口层

网络接口层是 TCP/IP 软件的最底层, 负责接收 IP 数据包并通过网络进行发送, 或者从网络上接收物理帧, 抽出 IP 数据报, 交给 IP 层。

2. 什么是 IP

IP 协议又称互联网协议, 是支持网间互连的数据报协议, 它与 TCP 协议一起构成了 TCP/IP 协议族的核心。它提供网间连接的完善功能, 包括 IP 数据报规定互连网络范围内的 IP 地址格式。Internet 为了实现连接到互联网上的结点之间的通信, 必须为每个结点 (入网的计算机) 分配一个地址, 并且应当保证这个地址是全网惟一的, 即 IP 地址。

目前的 IP 地址 (IPv4: IP 第 4 版本) 由 32 个二进制位表示, 每 8 位二进制数为一个整数, 中间由小数点间隔, 例如 159.226.41.98, 整个 IP 地址空间有 4 组 8 位二进制数, 由表示主机所在的网络地址以及主机在该网络中的标识共同组成。为了便于寻址和层次化的构造网络, IP 地址被分为 A、B、C、D、E 五类, 商业应用中只用到 A、B、C 三类。

(1) A 类地址

A 类地址的网络标识由第 1 组 8 位二进制数表示, 网络中的主机标识占 3 组 8 位二进制数, A 类地址的特点是网络标识的第 1 位二进制数取值必须为“0”。不难算出, A 类地址允许有 126 个网段, 每个网络大约允许有 1670 万台主机, 通常分配给拥有大量主机的网络, 例如主干网。

(2) B 类地址

B 类地址的网络标识由前两组 8 位二进制数表示, 网络中的主机标识占两组 8 位二进制数, B 类地址的特点是网络标识的前两位二进制数取值必须为“10”。B 类地址允许有 16 384 个网段, 每个网络允许有 65 533 台主机, 适用于结点比较多的网络, 例如区域网。

(3) C 类地址

C 类地址的网络标识由前 3 组 8 位二进制数表示, 网络中主机标识占 1 组 8 位二进制数, C 类地址的特点是网络标识的前 3 位二进制数取值必须为“110”。具有 C 类地址的网络允许有 254 台主机, 适用于结点比较少的网络, 例如校园网。

为了便于记忆, 通常习惯采用 4 个十进制数来表示一个 IP 地址, 十进制数之间采用句点“.”予以分隔。这种 IP 地址的表示方法也被称为点分十进制法。如以这种方式表示, A 类网络的 IP 地址范围为 1.0.0.1~127.255.255.254; B 类网络的 IP 地址范围为 128.1.0.1~191.255.255.254; C 类网络的 IP 地址范围为 192.0.1.1~223.255.255.254。

注意: 由于网络地址紧张, 主机地址相对过剩, 通常采取子网掩码的方式来指定网段号。

疑难点评

TCP/IP 和 IP 是网络传输的基本协议, 在进行网络编程之前, 读者应需要对网络协议有一定的了解。

知识链接

FAQ7.02 TCP 和 UDP 有什么区别?

FAQ7.02 TCP 和 UDP 有什么区别?

📖 难度系数: ★★★

📖 问题频率: 70%

核心解答

TCP (Transmission Control Protocol) 是传输控制协议, UDP (User Datagram Protocol) 是用户数据报协议。

在 TCP/IP 分层模型中, TCP 和 UDP 均属于传输层协议。其中 TCP 提供 IP 环境下的数据可靠传输, 它提供的服务包括数据流传送、可靠性、有效流控、全双工操作和多路复用, 可用于实现面向连接、端到端和可靠的数据包发送。通俗说, 它是事先为所发送的数据开辟出连接好的通道, 然后再进行数据发送; 而 UDP 则不为 IP 提供可靠性、流控或差错恢复功能。一般来说, TCP 对应的是可靠性要求高的应用, 而 UDP 对应的则是可靠性要求低、传输经济的应用。TCP 支持的应用协议主要有 Telnet、FTP、SMTP 等; UDP 支持的应用层协议主要有 NFS (网络文件系统)、SNMP (简单网络管理协议)、DNS (主域名称系统) 和 TFTP (通用文件传输协议) 等。

疑难点评

TCP 和 UDP 均属于传输层协议, Java 使用 Socket 可以实现两种不同协议类型的通信, 在

此主要介绍下 TCP 和 UDP 两种协议的区别。

知识链接

FAQ7.01 什么是 TCP/IP? 什么是 IP 协议?

FAQ7.03 什么是 HTTP? HTTP 的工作原理如何?

FAQ7.03 什么是 HTTP? HTTP 的工作原理如何?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

1. HTTP 概念

HTTP (Hypertext Transfer Protocol) 被称为超文本传输协议, 用于控制万维网 (WWW) 服务器和本地浏览器之间的超文本信息传送。该协议可以使浏览器访问速率更加高效, 减少网络传输。它不仅保证计算机正确快速地传输超文本文档, 还确定传输文档中的哪一部分, 以及哪部分内容首先显示等。

自万维网诞生后, 为了方便浏览器和服务端之间信息的传输和控制, 决定使用超文本作为万维网文档的标准格式, 于是在 1990 年, 科学家们制定了能够快速查找这些超文本文档的协议, 即 HTTP 协议。

当用户想浏览一个网站的时候, 只要在浏览器的地址栏中输入网站的地址就可以了, 例如 `http://www.baidu.com`, 该地址就是 URL (Uniform Resource Locator), 即统一资源定位符。以 `http://www.baidu.com/china/index.htm` 为例, 该 URL 的具体含义如下。

- ☐ “http://” 代表访问服务器的协议。
- ☐ “www” 代表一个 Web 服务器。
- ☐ “baidu.com/” 代表 Web 服务器的域名, 或者服务器站点的名称。
- ☐ “china/” 代表服务器上的一个子目录。
- ☐ “index.htm” 代表 china 目录中的一个 HTML 文件。

2. HTTP 工作原理

从用户在浏览器中输入 URL 到网页显示, HTTP 的工作主要经历以下几个过程。

(1) 发送请求

用户输入 URL 地址后, 客户机与服务器建立连接, 客户机通过 HTTP 向 Web 服务器发送请求, 请求格式为统一资源标识符 (URL)、协议版本号, 后边是 MIME 信息, 包括请求修饰符、客户端信息和其他可能的内容。

(2) 响应请求

Web 服务器在接收到用户请求后, 将请求资源通过 HTTP 发送给客户浏览器, 响应信息的

格式为信息的协议版本号、一个成功或错误的代码，后边是 MIME 信息，包括服务器信息、实体信息和其他可能的内容。

(3) 处理响应

客户端接收服务器返回的响应信息，通过浏览器解析并显示，然后客户机与服务器断开连接。

如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端，在浏览器中显示输出。对于用户来讲，这些过程都是由 HTTP 自动完成，用户只需要用鼠标单击，然后等待信息显示。

3. HTTP 请求和响应格式

客户机向服务器的请求消息和服务器向客户机的响应消息都是通过 HTTP 完成的，消息的格式由一个起始行，一个或者多个头域，一个指示头域结束的空行和可选的消息体组成。HTTP 的头域包括通用头、请求头、响应头和实体头 4 个部分。每个头域由一个域名，冒号和域值 3 部分组成。域名是大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

请求信息的格式如下所示：

```
GET http://download.microtool.de:80/somedata.exe
Host: download.microtool.de
Accept: */*
Pragma: no-cache
Cache-Control: no-cache
Referer: http://download.microtool.de/
User-Agent: Mozilla/4.04[en](Win95;I;Nav)
Range: bytes=554554-
```

在上述示例中，第 1 行表示 HTTP 客户端（可能是浏览器或下载程序）通过 GET 方法获得指定 URL 的文件。

- ☐ Host 头域指定请求资源所在的服务器或网关的位置。HTTP/1.1 请求必须包含 Host 头域，否则系统会以 400 状态码返回。
- ☐ Referer 头域允许客户端指定请求 URI 的源资源地址。
- ☐ Range 头域可以请求实体的一个或者多个子范围。例如“bytes=0-499”表示头 500 个字节；“bytes=500-999”表示第 2 个 500 字节。
- ☐ User-Agent 头域的内容包含发出请求的用户信息。

响应消息的格式如下所示：

```
HTTP/1.0200OK
Date: Mon, 31 Dec 2001 04:25:57 GMT
Server: Apache/1.3.14 (Unix)
Content-type: text/html
Last-modified: Tue, 17 Apr 2001 06:46:28 GMT
Etag: "a030f020ac7c01:1e9f"
Content-length: 39725426
Content-range: bytes 554554-40279979/40279980
```

在上述示例中，第 1 行表示 HTTP 服务端响应一个 GET 方法，包含响应的信息版本号和状态码。常用的响应状态码有 200（正常）、404（无法找到文件）和 500（内部服务器错误）等。

- ☐ Date 代表响应时间。
- ☐ Server 代表服务器信息。
- ☐ Last-modified 代表上次修改时间。
- ☐ Content-type 代表响应信息的格式及类型。
- ☐ Content-length 表示响应信息大小，以字节为单位。
- ☐ Content-range 表示响应信息的范围。

疑难点评

HTTP 是超文本传输协议，用于控制 Java 与互联网之间信息的传输。URL 访问互联网资源、JSP 和 Servlet 程序都是基于 HTTP 协议传输请求和响应信息。

知识链接

FAQ7.01 什么是 TCP/IP? 什么是 IP?

FAQ7.04 在 Socket 通信时如何获取主机和客户机的 IP 地址?

📖 难度系数: ★★★

📖 问题频率: 85%

核心解答

Java 提供了一个 `java.net.InetAddress` 类，该类用于表示互联网协议 (IP) 地址。使用 `InetAddress` 类的方法可以实现由主机名获取 IP 地址和由 IP 地址获取主机名的功能，具体如下所示。

- ☐ `String getHostAddress()`

获取 IP 地址字符串。

- ☐ `String getHostName()`

获取此 IP 地址的主机名。

- ☐ `InetAddress getLocalHost()`

获取本地主机的 `InetAddress` 对象。

- ☐ `InetAddress[] getAllByName (String host)`

在给定主机名的情况下，获取给定主机名对应的所有 IP 地址。

- ☐ `InetAddress getByName (String host)`

在给定主机名的情况下，获取给定主机名对应的 IP 地址。

- ☐ `InetAddress getByAddress (byte[] addr)`

在给定原始 IP 地址的情况下，返回 `InetAddress` 对象。如果给定的 IP 地址为 IPv4 格式，则

字节数组的长度必须为 4 个字节；如果为 IPv6 格式，则字节数组的长度必须为 16 个字节，高位位于字节数组的 0 索引处。

获取本机 IP 地址和主机名的示例代码如下：

```
public class Test {

    public static void main(String[] args) {
        System.out.println("主机 IP: " + getLocalHostIP());
        System.out.println("主机名: " + getLocalHostName());

        String[] localIP = getAllLocalHostIP();
        for (int i = 0; i < localIP.length; i++) {
            System.out.println(localIP[i]);
        }
    }

    //获取本机 IP 地址
    public static String getLocalHostIP() {
        String ip;
        try {
            InetAddress addr = InetAddress.getLocalHost();
            ip = addr.getHostAddress();
        } catch (Exception ex) {
            ip = "";
        }
        return ip;
    }

    //获取主机名
    public static String getLocalHostName() {
        String hostName;
        try {
            InetAddress addr = InetAddress.getLocalHost();
            hostName = addr.getHostName();
        } catch (Exception ex) {
            hostName = "";
        }
        return hostName;
    }

    //获取本机所有的 IP 地址
    public static String[] getAllLocalHostIP() {
        String[] ret = null;
        try {
            String hostName = getLocalHostName();
            if (hostName.length() > 0) {
                InetAddress[] addrs = InetAddress.getAllByName(hostName);
                if (addrs.length > 0) {
                    ret = new String[addrs.length];
                    for (int i = 0; i < addrs.length; i++) {
                        ret[i] = addrs[i].getHostAddress();
                    }
                }
            }
        }
    }
}
```



```

    }

    } catch (Exception ex) {
        ret = null;
    }
    return ret;
}

//根据主机名获取其对应的所有 IP 地址
public static String[] getAllHostIPByName(String hostName) {
    String[] ret = null;
    try {
        if (hostName.length() > 0) {
            InetAddress[] addrs = InetAddress.getAllByName(hostName);
            if (addrs.length > 0) {
                ret = new String[addrs.length];
                for (int i = 0; i < addrs.length; i++) {
                    ret[i] = addrs[i].getHostAddress();
                }
            }
        }
    } catch (Exception ex) {
        ret = null;
    }
    return ret;
}
}

```

在 Socket 编程时, 利用 Socket 类的 `getInetAddress()` 方法可以获取此套接字连接到的远程 IP 地址。示例代码如下:

```

public static void main(String[] args) {
    try {
        //创建服务器 ServerSocket
        ServerSocket ss = new ServerSocket(8080);
        //获取客户端的 Socket
        Socket s = ss.accept();
        InetAddress address = s.getInetAddress();
        //获取主机名
        System.out.println(address.getHostName());
        //获取主机 IP
        System.out.println(address.getHostAddress());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

疑难点评

在使用 Socket 实现两台计算机之间通信时, 有时候需要获取对方 IP 地址, 该功能在一些聊天软件中经常会使用到。

知识链接

FAQ7.01 什么是 TCP/IP? 什么是 IP 协议?

FAQ7.05 如何利用 Socket 实现基于 TCP 的通信?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

基于 TCP 协议实现两台计算机之间的通信, 主要使用 Socket 和 ServerSocket 类。

Socket 类用于实现客户端套接字, 套接字是两台计算机之间的通信端点。Socket 类的重要方法如下所示:

```
//创建一个流套接字并将其连接到指定主机上的指定端口号
public Socket(String host,int port) throws UnknownHostException,IOException
//返回套接字连接到的远程 IP 地址
public InetAddress getInetAddress()
//返回将字节写入此套接字的输出流
public OutputStream getOutputStream() throws IOException
//返回从此套接字读取字节的输入流
public InputStream getInputStream() throws IOException
//关闭此套接字,与此套接字有关联的通道也将关闭
public void close() throws IOException
```

ServerSocket 类用于实现服务器套接字, 提供 TCP 服务。服务器套接字可以接收客户端的接入请求, 然后向客户端套接字发送信息。ServerSocket 类的重要方法如下所示:

```
//创建绑定到特定端口的服务器套接字
public ServerSocket(int port) throws IOException
//侦听并接收到此套接字的连接, 此方法在进行连接之前一直阻塞
public Socket accept() throws IOException
//关闭此套接字, 与此套接字有关联的通道也将关闭
public void close() throws IOException
```

使用 Socket 和 ServerSocket 类实现两台计算机之间通信的代码如下所示。

❑ 服务器端示例代码

```
public class Server {
    public static void main(String args[]) {
        ServerSocket server;
        try {
            server = new ServerSocket(1234);
            Socket sk = server.accept();

            BufferedReader br = new BufferedReader(new InputStreamReader(sk.getInputStream()));
            System.out.println(br.readLine());
            br.close();
        }
    }
}
```

```
        server.close();
    } catch (IOException e) {
        System.out.println(e);
    }
}
```

□ 客户端示例代码

```
public class Client {
    public static void main(String args[]) {
        Socket client;
        PrintStream ps;
        try {
            client = new Socket("localhost", 1234);
            System.out.println("连接成功");
            ps = new PrintStream(client.getOutputStream());
            ps.println("你好");
            client.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

在上述示例中, 首先启动服务器端示例, 服务器启动并阻塞 `accept()` 方法, 等待接收客户端连接。当客户端示例启动后, 服务器会接收客户端的连接, 并接收客户端发送的消息。

疑难点评

基于 TCP 实现计算机之间的通信, 需要编写客户端和服务端程序, 在客户端主要使用 `Socket` 类, 在服务器端主要使用 `ServerSocket` 类。

知识链接

FAQ7.06 如何利用 Socket 传输中文字符?

FAQ7.08 如何利用 Socket 传递对象信息?

FAQ7.09 如何利用 Socket 实现文件传输?

FAQ7.06 如何利用 Socket 传输中文字符?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

在网络中传递中文字符出现乱码主要是因为发送方编码和接收方解码不一致造成的。在网

络中传递的信息为字节,因此发送方发送的字符需要编码为字节。如果开发者不指定编码类型,将采用默认的 ISO-8859-1。接收方在接收时,与发送编码一致才能正确获取中文。

在使用 Socket 传递中文字符时,通过以下两种方式可实现正确接收。

□ 方法一

使用 `DataOutputStream` 的 `writeUTF()`方法发送信息,代码如下:

```
/**
 * 发送信息
 * @param ip: 服务器 ip
 * @param port: 服务器通信端口
 * @throws Exception
 */
public static void clientStart(String ip, int port) throws Exception {
    Socket s = new Socket(ip, port);
    DataOutputStream dos = new DataOutputStream(new BufferedOutputStream(s
        .getOutputStream()));
    dos.writeUTF("Hello 你好!");
    dos.flush();
}
```

使用 `DataInputStream` 的 `readUTF()`方法接收信息,代码如下:

```
/**
 * 接收信息
 * @param port: 服务器通信端口
 * @throws Exception
 */
public static void startServer(int port) throws Exception {
    ServerSocket ss = new ServerSocket(port);
    System.out.println("服务器端已启动,等待客户端连接");
    Socket s = ss.accept();
    System.out.println("客户端连接成功!" + s);
    DataInputStream dis = new DataInputStream(new BufferedInputStream(s
        .getInputStream()));
    String msg = dis.readUTF();
    System.out.println("服务器端接收到: " + msg);
}
```

□ 方法二

使用 `Writer` 字符流发送信息,代码如下:

```
/**
 * 发送信息
 * @param ip: 服务器 ip
 * @param port: 服务器通信端口
 * @throws Exception
 */
public static void clientStart(String ip, int port) throws Exception {
    Socket s = new Socket(ip, port);
    OutputStreamWriter osw = new OutputStreamWriter(s.getOutputStream(), "UTF-8");
    PrintWriter pw = new PrintWriter(osw, true);
    pw.println("Hello 你好!");
}
```

使用 Reader 字符流接收信息, 代码如下:

```
/**
 * 接收信息
 * @param port: 服务器通信端口
 * @throws Exception
 */
public static void startServer(int port) throws Exception{
    ServerSocket ss = new ServerSocket(port);
    System.out.println("服务器端已启动,等待客户端连接");
    Socket s = ss.accept();
    System.out.println("客户端连接成功!" + s);

    InputStreamReader isr = new InputStreamReader(s.getInputStream(), "UTF-8");
    BufferedReader br = new BufferedReader(isr);
    String msg = br.readLine();
    System.out.println("服务器端接收到: " + msg);
}
```

在使用字符流发送和接收信息时, 需要使用统一的编码, 例如上述代码使用的是“UTF-8”, 也可以改用“GBK”。

疑难点评

使用 Socket 实现计算机之间的通信时, 如果传递的是中文信息, 由于编码问题经常会产生乱码。上面给出了两种解决方法, 可实现中文的正确传输。

知识链接

FAQ7.05 如何利用 Socket 实现基于 TCP 协议的通信?

FAQ7.07 如何在 Socket 读取数据时使用超时设置?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

在早于 1.4 的 Java 版本中, 当从 Socket 中读取数据时, 读数据的方法会将程序阻塞, 阻塞时间长短取决于数据发送方。但是设置了读取数据的超时时间后, 可以改变这种阻塞状态。

Java 1.4 在 Java API 中加入了 NIO 技术, NIO 支持非阻塞 I/O (non-blocking I/O) 操作, Java 1.3 和更早的版本都不支持这样的特性。然而, 可以用 `java.net.Socket` 类的超时属性来实现一些类似于非阻塞 I/O 的功能。

要使用超时属性, 首先需要创建一个 Socket 对象, 然后通过 `setSoTimeout()` 方法设置所期望的超时时间, 单位是毫秒。当设置超时后, 输入流如果超过了指定的时间还读不到任何数据,

程序将抛出一个 `java.io.InterruptedIOException` 异常，可以捕获该异常并决定是否尝试进行另一次读取操作。

示例代码如下：

```
public static void main(String[] args) {
    int bytesRead = 0;
    byte[] buffer = new byte[1024];

    try {
        Socket s = new Socket("localhost", 1234);
        //获取默认的超时时间
        System.out.println(s.getSoTimeout());
        //设置超时时间
        s.setSoTimeout(5000);
        //获取输入流读取数据
        InputStream in = s.getInputStream();
        while (true) {
            try {
                bytesRead = in.read(buffer);
                if (bytesRead == -1) {
                    break;
                }
                System.out.println(new String(buffer, 0, bytesRead));
            } catch (InterruptedException e) {
                System.out.print("timeout on read");
                //决定是否继续读取
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

疑难点评

在使用 `Socket` 读取数据时，设置超时可以避免程序读操作阻塞的问题，但需要注意使用 `try-catch` 语句捕获处理超时异常。

知识链接

FAQ7.05 如何利用 `Socket` 实现基于 TCP 协议的通信？

FAQ7.08 如何利用 `Socket` 传递对象信息？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

在 Socket 通信时, 可以用对象将传输的信息封装, 然后以对象为单位在网络中进行传输。具体实现过程如下。

(1) 定义消息类, 用于封装消息信息。

示例代码如下:

```
public class Message implements java.io.Serializable{
    //消息发送者
    private String name;

    //发送者 IP
    private String ip;

    //发送的内容
    private String msg;

    //发送时间
    private Date date;

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }

    public String getName() {
        return name;
    }
}
```

```

    public void setName(String name) {
        this.name = name;
    }
}

```

注意：由于消息对象要在网络中传递，因此消息类必须实现序列化。

(2) 客户端示例程序

客户端程序在创建 Socket 对象后，需要将 Socket 的输出流用 ObjectOutputStream 封装，ObjectOutputStream 提供了发送对象的 writeObject() 方法。示例代码如下：

```

/**
 * 发送信息
 * @param ip: 服务器 ip
 * @param port: 服务器通信端口
 * @throws Exception
 */
public static void clientStart(String ip,int port) throws Exception{
    Socket s = new Socket(ip,port);
    ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
    Message msg = new Message();
    //获取客户端主机名并设置发送者
    msg.setName(InetAddress.getLocalHost().getHostName());
    //获取客户端 IP 地址并设置发送者 IP
    msg.setIp(InetAddress.getLocalHost().getHostAddress());
    msg.setMsg("你好，好久没见了");
    msg.setDate(new Date());
    oos.writeObject(msg);
    oos.flush();
    s.close();
}

```

(3) 服务器端示例程序

服务器的程序在获取客户端的 Socket 连接后，需要将 Socket 的输入流用 ObjectInputStream 封装，ObjectInputStream 提供了接收对象的 readObject() 方法。示例代码如下：

```

/**
 * 接收信息
 * @param port: 服务器通信端口
 * @throws Exception
 */
public static void startServer(int port) throws Exception{
    ServerSocket ss = new ServerSocket(port);
    System.out.println("服务器端已启动,等待客户端连接");
    Socket s = ss.accept();
    System.out.println("客户端连接成功!" + s);

    ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
    Message msg = (Message)ois.readObject();
    System.out.println(msg.getName() + " " + msg.getIp() + "说: ");
    System.out.println(msg.getMsg());
    System.out.println("消息发送时间" + msg.getDate());
}

```

```
ois.close();  
ss.close();  
}
```

疑难点评

以对象为单位在网络中传递信息,可以简化传递代码。读者需要注意,并不是所有对象都可以在网络中传输的,传输对象必须具备可序列化的特性。

知识链接

FAQ7.06 如何利用 Socket 传输中文字符?

FAQ7.09 如何利用 Socket 实现文件传输?

FAQ7.09 如何利用 Socket 实现文件传输?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

在读取其他计算机中的文件时,需要用到读写文件的流,还有 Socket 技术。该功能的实现代码需要由客户端和服务端组成,具体实现过程如下所示。

❑ 客户端的实现

客户端程序主要负责连接服务器,并读取服务器发送的文件信息。实现代码如下:

```
public class RemoteFileClient {  
    protected String hostIp;  
  
    protected int hostPort;  
  
    protected BufferedReader socketReader;  
  
    protected PrintWriter socketWriter;  
  
    public static void main(String[] args) {  
        RemoteFileClient remoteFileClient = new  
RemoteFileClient("127.0.0.1",3000);  
        remoteFileClient.setUpConnection();  
        String fileContents = remoteFileClient.getFile("F:\\a.txt");  
        remoteFileClient.tearDownConnection();  
        System.out.println(fileContents);  
    }  
  
    public RemoteFileClient(String aHostIp, int aHostPort) {  
        hostIp = aHostIp;  
        hostPort = aHostPort;  
    }  
}
```



```

    }

    //与服务器建立连接
    public void setUpConnection() {
        try {
            Socket client = new Socket(hostIp, hostPort);
            socketReader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
            socketWriter = new PrintWriter(client.getOutputStream());
        } catch (UnknownHostException e) {
            System.out
                .println("Error setting up socket connection: unknown host at "
                    + hostIp + ":" + hostPort);
        } catch (IOException e) {
            System.out.println("Error setting up socket connection: " + e);
        }
    }

    //读取文件信息以字符串的形式返回
    public String getFile(String fileNameToGet) {
        StringBuffer fileLines = new StringBuffer();
        try {
            socketWriter.println(fileNameToGet);
            socketWriter.flush();
            String line = null;
            while ((line = socketReader.readLine()) != null)
                fileLines.append(line + " ");
        } catch (IOException e) {
            System.out.println("Error reading from file: " + fileNameToGet);
        }
        return fileLines.toString();
    }

    //关闭流和 Socket 连接
    public void tearDownConnection() {
        try {
            socketWriter.close();
            socketReader.close();
        } catch (IOException e) {
            System.out.println("Error tearing down socket connection: " + e);
        }
    }
}

```

□ 服务器端的实现

服务器端程序负责接收客户端的连接请求，在接收到客户端发送的文件信息后，将信息返回给客户端。实现代码如下：

```

public class RemoteFileServer {
    protected int listenPort = 3000;

```

```

public static void main(String[] args) {
    RemoteFileServer server = new RemoteFileServer();
    server.acceptConnections();
}

//创建服务器 Socket, 接收客户端连接
public void acceptConnections() {
    try {
        ServerSocket server = new ServerSocket(listenPort);
        Socket incomingConnection = null;
        while (true) {
            incomingConnection = server.accept();
            handleConnection(incomingConnection);
        }
    } catch (BindException e) {
        System.out.println("Unable to bind to port " + listenPort);
    } catch (IOException e) {
        System.out.println("Unable to instantiate a ServerSocket on port: "
            + listenPort);
    }
}

//获取客户端传递过来的文件内容, 并将文件内容返回
public void handleConnection(Socket incomingConnection) {
    try {
        OutputStream outputToSocket = incomingConnection.getOutputStream();
        InputStream inputFromSocket = incomingConnection.getInputStream();
        BufferedReader streamReader = new BufferedReader(
            new InputStreamReader(inputFromSocket));
        FileReader fileReader = new FileReader(new File(streamReader.readLine()));
        BufferedReader bufferedFileReader = new BufferedReader(fileReader);
        PrintWriter streamWriter = new PrintWriter(outputToSocket);
        String line = null;
        while ((line = bufferedFileReader.readLine()) != null) {
            streamWriter.println(line);
        }
        fileReader.close();
        streamWriter.close();
        streamReader.close();
    } catch (Exception e) {
        System.out.println("Error handling a client: " + e);
    }
}
}

```

注意: 为了降低文件传送的数据量, 可以在传递时使用 `ZipOutputStream` 和 `ZipInputStream` 等压缩流封装, 实现文件的压缩传输。

疑难点评

利用 Socket 传输文件与传输字符信息的过程非常相似, 只不过在传输文件过程中添加了读

写文件的操作。

知识链接

FAQ7.06 如何利用 Socket 传输中文字符?

FAQ7.08 如何利用 Socket 传递对象信息?

FAQ7.10 如何基于 Socket 实现聊天系统?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在“如何利用 Socket 实现基于 TCP 的通信”问题中已经实现了客户端和服务端之间传递消息的功能,聊天系统也是在该问题基础之上实现的。聊天系统主要由客户端和服务端构成,允许客户端之间一对一和一对多进行聊天。客户端之间相互聊天时,消息借助服务器行转发。聊天系统的实现结构如图 7-1 所示。

如图 7-1 所示,如果客户端 1 与客户端 2 聊天,客户端 1 发送的消息首先到达服务器,然后服务器再转发给客户端 2。一对多聊天的原理与该过程相同。

(1) 客户端的实现

在聊天系统中,客户端需要具有以下功能。

- ☐ 可以给其他人发消息。
- ☐ 可以接收其他人的消息。
- ☐ 消息发送和接收互不干涉,属于并发操作。

在实现接收消息功能时,读信息的方法具有阻塞性质,因此需要将消息发送和接收用两个不同的线程分开,否则消息接收会阻塞消息发送。客户端的示例代码如下:

```
public class MyClient {  
    JFrame jf = new JFrame("聊天室客户端");  
    JTextArea jt = new JTextArea();  
    DataOutputStream dos = null;  
    JTextField jtf = new JTextField(15);  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        //TODO Auto-generated method stub
```

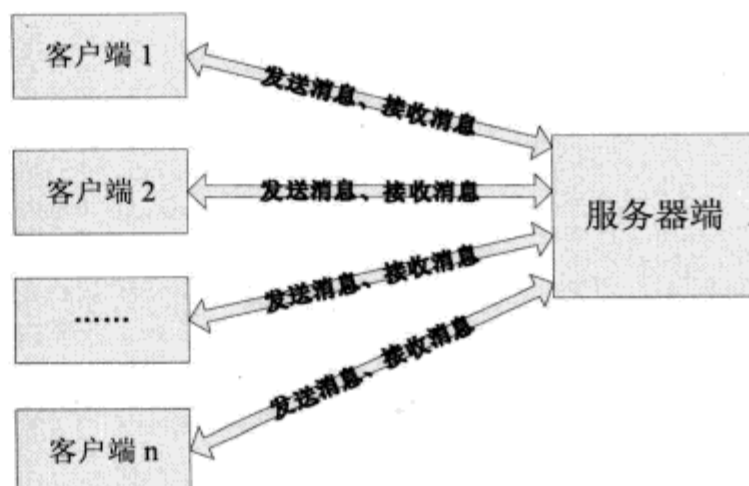


图 7-1 聊天系统结构图


```

MyClient c = new MyClient();
try {
    c.createForm();
    c.clientStart("192.168.2.2", 8888);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
//创建客户端用户的图形窗体
public void createForm(){
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    jf.getContentPane().add(jt, BorderLayout.CENTER);

    JButton jb = new JButton("发送");
    SendButtonAction sendAction = new SendButtonAction();
    jb.addActionListener(sendAction);
    JPanel jp = new JPanel();
    jp.add(jtf);
    jp.add(jb);

    jf.getContentPane().add(jp, BorderLayout.SOUTH);

    jf.setSize(300, 200);
    jf.setVisible(true);
}

//创建客户端 Socket, 连接服务器
public void clientStart(String ip, int port) throws Exception {
    Socket s = new Socket(ip, port);
    //将发消息的输出流实例化, 为后续发送消息做准备
    dos = new DataOutputStream(new
BufferedOutputStream(s.getOutputStream()));
    //打开一个接收消息的线程
    new MySocketReadServer(s).start();
}

//为图形窗体界面的发送按钮添加处理事件
class SendButtonAction implements ActionListener{
    //当用户单击发送按钮后, 将输入的消息发送给服务器
    @Override
    public void actionPerformed(ActionEvent arg0) {
        String msg = jtf.getText();
        jtf.setText("");
        if(msg.equals("")){
            JOptionPane.showMessageDialog(jf, "发送的内容不能为空");
            return;
        }
    }
    try {

```

```

        dos.writeUTF(msg);
        dos.flush();
    } catch (IOException e) {
        JOptionPane.showMessageDialog(jf, "发送失败,请重新尝试");
        e.printStackTrace();
    }
}
}
//定义一个线程类,封装接收消息功能
class MySocketReadServer extends Thread {
    private Socket s;

    public MySocketReadServer(Socket s) {
        this.s = s;
    }
    //时刻准备接收服务器发送过来的消息,并添加到显示窗口
    public void run() {
        try {
            DataInputStream dis = new DataInputStream(
                new BufferedInputStream(s.getInputStream()));
            while (true) {
                String msg = jt.getText()+"\n\r"+dis.readUTF();
                jt.setText(msg);

                if ("88".equals(msg)) {
                    break;
                }
            }
        } catch (Exception e) {
            System.out.println(s+"已退出聊天室");
        }
    }
}
}

```

(2) 服务器端的实现

在聊天系统中,服务器端需要具有以下功能。

- ☐ 可以接收客户端发送的消息。
- ☐ 将客户端发送的消息转发。
- ☐ 需要随时接收和转发每一个客户端的消息。

在实现服务器的接收和转发功能时,为了时刻为每一个客户端提供服务,服务器需要将接收和转发功能用线程进行封装,然后为每一个客户端都开启一个线程为其提供服务。服务器端的示例代码如下:

```

public class MyServer {
    private Vector<Socket> vect = new Vector<Socket>();
    /**
     * @param args

```



```

*/
public static void main(String[] args) {
    //TODO Auto-generated method stub
    MyServer server = new MyServer();
    try {
        server.startServer(8888);
    } catch (Exception e) {
        System.out.println("服务器端启动失败!");
        e.printStackTrace();
    }
}

//启动服务器, 并循环监听接收客户端的连接
public void startServer(int port) throws Exception {
    ServerSocket ss = new ServerSocket(port);
    System.out.println("服务器端已启动,等待客户端连接");
    while (true) {
        //监听并接收客户端的连接, 注意该方法有阻塞性
        Socket s = ss.accept();
        //将客户端的 Socket 连接添加到一个 Vector 集合
        vect.add(s);
        System.out.println("客户端连接成功!" + s);
        //打开一个线程, 为该客户端的 Socket 连接提供接收和转发服务
        new MySocketOpt(s).start();
    }
}

//定义一个线程, 提供接收和转发服务
class MySocketOpt extends Thread {
    private Socket s;

    public MySocketOpt(Socket s) {
        this.s = s;
    }

    public void run() {
        try {
            DataInputStream dis = new DataInputStream(
                new BufferedInputStream(s.getInputStream()));
            while (true) {
                //等待接收客户端发送的消息
                String msg = dis.readUTF();
                System.out.println("服务器端接收到: " + msg);

                //遍历集合中的 Socket 连接, 向外转发消息
                for(Socket tmp_s:vect){
                    if(tmp_s != s){
                        DataOutputStream dos = new
DtaOutputStream(new BufferedOutputStream (tmp_s.getOutputStream()));
                        dos.writeUTF(msg);
                        dos.flush();
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```



```
        }  
        //如果接收到的消息为“88”，退出该线程  
        if ("88".equals(msg)) {  
            break;  
        }  
    }  
} catch (Exception e) {  
    System.out.println(s+"已退出聊天室");  
    vect.remove(s);  
}  
}  
}
```

疑难点评

上述聊天室是基于 TCP 协议进行多用户聊天的，读者应首先理解聊天室的实现原理，然后再进行代码实现。

知识链接

FAQ7.06 如何利用 Socket 传输中文字符？

FAQ7.11 如何利用 Socket 实现基于 UDP 的通信？

FAQ7.11 如何利用 Socket 实现基于 UDP 的通信？

📖 难度系数：★★★★★

📖 问题频率：80%

核心解答

基于 UDP 协议实现两台计算机之间的通信，主要使用 `DatagramSocket` 和 `DatagramPacket` 类。

`DatagramSocket` 类表示用来发送和接收数据报包的套接字。数据报套接字是包投递服务的发送或接收点。每个在数据报套接字上发送或接收的包都是单独编址和路由的。从一台计算机发送到另一台计算机上的多个包可能选择不同的路由，也可能按不同的顺序到达。

`DatagramSocket` 总是启用 UDP 广播发送。为了接收广播包，应该将 `DatagramSocket` 绑定到通配符地址。在某些实现中，将 `DatagramSocket` 绑定到一个具体的地址时广播包也可以被接收。

`DatagramSocket` 类的重要方法如下：

```
//创建数据报套接字并将其绑定到本地主机上的指定端口  
public DatagramSocket(int port) throws SocketException  
//将套接字连接到此套接字的远程地址  
//当套接字连接到远程地址时，包就只能从该地址发送或接收，默认情况下不连接数据报套接字
```

```
//如果套接字要连接的远程目标不存在或不可到达,那么可能会抛出 PortUnreachableException
public void connect(InetAddress address, int port)
//从此套接字接收数据报包。此方法在接收到数据报前一直阻塞
public void receive(DatagramPacket p) throws IOException
//从此套接字发送数据报包
public void send(DatagramPacket p) throws IOException
//断开套接字的连接
public void disconnect()
```

DatagramPacket 类表示数据报包。数据报包用来实现无连接包投递服务,每条报文仅根据该包中包含的信息从一台计算机路由到另一台计算机。从一台计算机发送到另一台计算机的多个包可能选择不同的路由,也可能按不同的顺序到达。DatagramPacket 类的重要方法如下:

```
//创建数据报包,用来将长度为 length 的包发送到指定主机上的指定端口号
public DatagramPacket(byte[] buf,int length,InetAddress address,int port)
//返回将要发送或接收到的数据报中数据块的字节大小
public int getLength()
//得到接收数据报中的数据
public byte[] getData();
//为发送者提供一个 InetAddress 对象
public InetAddress getAddress();
//得到 UDP 端口
public int getPort();
```

使用 DatagramSocket 和 DatagramPacket 类实现两台计算机之间通信的代码如下所示。

❑ 消息接收示例

```
public class UDPServer {

    public static void main(String args[]) {
        try {
            DatagramSocket ser = new DatagramSocket(10005);
            byte[] rb = new byte[1024];
            DatagramPacket pac = new DatagramPacket(rb, rb.length);
            String rev = "";
            int i = 0;
            while (i == 0)
                //无数据,则循环
            {
                ser.receive(pac);
                i = pac.getLength();
                //接收数据
                if (i > 0) {
                    //指定接收到数据的长度,可以使接收数据正常显示,开始时很容易忽略这一点
                    rev = new String(rb, 0, pac.getLength());
                    System.out.println(rev);
                    i = 0; //循环接收
                }
            }
        } catch (Exception e) {
```

```
        e.printStackTrace();
    }
}
```

□ 消息发送示例

```
public class UDPClient {

    public static void main(String args[]) {
        try {
            //指定端口号，避免与其他应用程序发生冲突
            DatagramSocket cli = new DatagramSocket(10002);
            byte[] sb = new byte[1024];
            String sen = "UDP 方式发送数据";
            sb = sen.getBytes();
            DatagramPacket pac = new DatagramPacket(sb, sb.length,
            InetAddress .getByName("localhost"), 10005);
            cli.send(pac);
        } catch (SocketException se) {
            se.printStackTrace();
        } catch (IOException ie) {
            ie.printStackTrace();
        }
    }
}
```

在上述示例中，首先启动消息接收示例，启动后程序会阻塞在 `receive()` 方法中，等待接收发送的数据包。当消息发送示例启动后，接收发送的数据包并显示。

疑难点评

UDP 与 TCP 是两种不同的协议，虽然程序功能相似，但使用的却是不同的 Java 类。UDP 传输主要使用 `DatagramSocket` 和 `DatagramPacket` 类。

知识链接

FAQ7.12 如何利用 UDP Socket 技术实现 IP 多点传送？

FAQ7.12 如何利用 UDP Socket 技术实现 IP 多点传送？

📖 难度系数：★★★★★

📖 问题频率：75%

核心解答

IP 多点传送针对点到点的传送和广播传送两种方式而言，它是指在一定的组内对其成员进行的广播，是一种有限的广播。组中的某个成员发出的信息，组中的其他所有成员都能收到。

IP 多点传送特别适合于高带宽的应用,例如在网络上发送视频和音频、网上聊天及网上会议、分布式数据存储、联机事务处理和交互式游戏等方面。

(1) UDP 基础

使用用户数据报协议 (User Datagram Protocol, 简称 UDP) 进行会话必须将信息装配成一定尺寸的小报文。当发送一条信息以后,接收方能否收到并返回信息永远是不确定的,如果无法收到返回信息,就无法确定发送的信息是否被接收,信息可能在途中丢失,接收者返回的响应信息也可能丢失,另外,接收者也可能忽略发送的信息,因此,UDP 被描述为是不可靠的、无连接的和面向消息的协议。

(2) IP 多点传送原理

为了支持 IP 多点传送,某些范围的 IP 地址被单独留出专门用于该目的,这些 IP 地址是 D 类地址,其地址的最高 4 比特的位模式为“1110”,即 IP 地址的范围在 224.0.0.0 和 239.255.255.255 之间。它们中的每一个 IP 地址都可以被引用作为一个多点传送组,任何以该 IP 地址编址的 IP 报文将被该组中的其他所有计算机接收,即一个 IP 地址就相当于一个邮箱。另外,组中的成员是动态的并随时间而改变的。

对于 IP 多点传送,网间网组管理协议 (Internet Group Management Protocol, 简称 IGMP),用于管理多点传送组中的成员。支持多点传送的路由可以使用 IGMP 决定本地的计算机是否能够加入某个组,一个多点传送路由可以决定是否转发一个多点传送报文。

影响多点传送报文的一个重要参数是 time-to-live (TTL)。TTL 用于描述发送者希望传送的信息能通过多少不同的网络。当报文被路由器转发,报文中的 TTL 将减一,当 TTL 为零时,报文将不再向前发送。

(3) 技术要点

实现 IP 多点传送,主要涉及 DatagramPacket 和 MulticastSocket 类。DatagramPacket 类用于创建一个发送和接收的数据报包。MulticastSocket 类是实施 IP 多点传送的关键,它允许使用多点传送 IP,发送或接收多个 UDP 数据包。

MulticastSocket 类的重要方法如下所示:

```
//创建一个多点传送 Socket
public MulticastSocket () throws IOException;
//在指定端口创建一个多点传送 Socket
public MulticastSocket(int port)throws IOException;
//加入多点传送组
public void joinGroup(InetAddress mcastaddr)throws IOException
//退出多点传送组
public void leaveGroup(InetAddress mcastaddr)throws IOException
//发送数据报
public synchronized void send(DatagramPacket p) throws IOException
//接收数据报
public synchronized void receive(DatagramPacket p) throws IOException
```

注意: 实现 TCP 通信,无需将传送的数据分块,然而,创建一个基于 UDP 的网络通信应用程序时,必须创建一套方法,在运行时刻分割数据。TCP/IP 允许最大的数据报可以含有 65 507

字节, 然而, 主机最多仅能接收 548 字节, 支持 8 192 字节的大数据报的平台利用 IP 层对数据报进行分割。如果在传送期间, 任何含有 IP 报文的一个数据块丢失, 都会造成整个 UDP 数据报的丢失, 因此, 在确定应用中数据报尺寸时, 对其尺寸的合理性的把握一定要谨慎。

实现 IP 多点传送功能的示例如下所示。

□ 消息接收示例

```
public class MultiCastReceiver {

    private static final int DATAGRAM_BYTES = 1024;

    private int mulcastPort;

    private InetAddress mulcastIP;

    private MulticastSocket mulcastSocket;

    private boolean keepReceiving = true;

    public static void main(String[] args) {

        // This must be the same port and IP address used by the sender.
        MultiCastReceiver cast = new MultiCastReceiver();
        try {
            cast.receiver("224.1.1.1", "12345");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //开始接收消息
    public void receiver(String ip, String port) throws Exception {
        DatagramPacket mulcastPacket; //用于接收的 UDP 数据包
        byte[] mulcastBuffer; //用于接收的字节数组
        InetAddress fromIP; //发送方的 IP 地址
        int fromPort; //发送方的端口号
        String mulcastMsg;
        //设置接收消息的 Socket
        mulcastIP = InetAddress.getByName(ip);
        mulcastPort = Integer.parseInt(port);
        mulcastSocket = new MulticastSocket(mulcastPort);
        //将 IP 加入 Socket 转发组
        mulcastSocket.joinGroup(mulcastIP);
        while (keepReceiving) {
            //创建一个数据包
            mulcastBuffer = new byte[DATAGRAM_BYTES];
            mulcastPacket = new DatagramPacket(mulcastBuffer,
                mulcastBuffer.length);
            //接收数据包
            mulcastSocket.receive(mulcastPacket);
            fromIP = mulcastPacket.getAddress();
        }
    }
}
```



```

        fromPort = multicastPacket.getPort();
        multicastMsg = new String(multicastPacket.getData());
        //打印输出接收到的消息
        System.out.println("Received from " + fromIP + " on port "
            + fromPort + ": " + multicastMsg);
    }
    //退出 Socket 转发组
    multicastSocket.leaveGroup(multicastIP);
    multicastSocket.close(); //关闭 Socket 连接
}
//停止接收消息
public void stop() {
    if (keepReceiving) {
        keepReceiving = false;
    }
}
}
}

```

□ 消息发送示例

```

public class MultiCastSender {

    private static final byte TTL = 1;

    private static final int DATAGRAM_BYTES = 1024;

    private int multicastPort;

    private InetAddress multicastIP;

    private BufferedReader input;

    private MulticastSocket multicastSocket;

    public static void main(String[] args) {

        //接收和发送方的 port 端口和 IP 地址必须一致
        MultiCastSender cast = new MultiCastSender();
        try {
            cast.sender("224.1.1.1", "12345");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 发送消息
     * @param ip: IP 地址
     * @param port: 端口
     * @throws Exception
     */
}

```



```
public void Sender(String ip, String port) throws Exception {
    DatagramPacket mulcastPacket; // 用于发送的 UDP 数据包
    String nextLine; // 读取键盘输入的字符串
    byte[] mulcastBuffer; // 数据包使用的字节数组
    byte[] lineData; // 存放键盘输入的内容
    int sendLength;
    // 读取键盘信息
    input = new BufferedReader(new InputStreamReader(System.in));
    // 创建 MulticastSocket 多点广播对象
    mulcastIP = InetAddress.getByName(ip);
    mulcastPort = Integer.parseInt(port);
    mulcastSocket = new MulticastSocket();
    // 循环读取键盘信息, 向多点广播
    while ((nextLine = input.readLine()) != null) {
        mulcastBuffer = new byte[DATAGRAM_BYTES];
        // 如果读取的键盘信息长度大于缓冲区, 将发送大小设置为数据包的字节大小
        if (nextLine.length() > mulcastBuffer.length) {
            sendLength = mulcastBuffer.length;
            // 否则设置为键盘信息的大小
        } else {
            sendLength = nextLine.length();
        }
        // 将键盘输入的字符串转换成字节数组
        lineData = nextLine.getBytes();
        // 将字节数组复制到数据包中
        for (int i = 0; i < sendLength; i++) {
            mulcastBuffer[i] = lineData[i];
        }
        // 创建数据包
        mulcastPacket = new DatagramPacket(mulcastBuffer,
            mulcastBuffer.length, mulcastIP, mulcastPort);
        // 发送数据包
        mulcastSocket.send(mulcastPacket, TTL);
    }
    mulcastSocket.close(); // Close the socket.
}
```

在上述示例中, 首先启动若干个消息接收示例, 启动后程序都会阻塞在 `receive()` 方法中, 等待接收发送的数据包, 当消息发送示例启动后进行多点传送, 每一个接收方都可以获取到发送的数据包, 实现了在线群发的效果。

疑难点评

使用 IP 多点传送功能可以实现基于 UDP 协议的一对多聊天。

知识链接

FAQ7.05 如何利用 Socket 实现基于 TCP 协议的通讯?

FAQ7.11 如何利用 Socket 实现基于 UDP 协议的通讯?

FAQ7.13 如何获取 Internet 资源的大小?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

获取网络中资源文件的大小, 需要使用 `java.net` 包下的 `URL` 类和 `URLConnection` 类。

`java.net.URL` 类代表一个统一资源定位符, 它是指向互联网“资源”的指针。
`java.net.URLConnection` 类代表应用程序和 `URL` 之间的通信链接。此类的实例可用于读取和写入此 `URL` 所引用的资源。通常, 创建一个到 `URL` 的连接需要几个步骤。

- ❑ 通过在 `URL` 上调用 `openConnection()` 方法创建连接对象。
- ❑ 设置操作参数和一般请求属性。
- ❑ 使用 `connect()` 方法建立到远程对象的实际连接。
- ❑ 远程对象变为可用。远程对象的头字段和内容变为可访问。

实现获取网络资源大小的功能, 示例代码如下:

```
public static void main(String[] args) {  
  
    try {  
        URL url = new  
URL("http://gpxz.com/live/UploadFile/2008-8/20088413571256734.mp3");  
        URLConnection conn=url.openConnection();  
        conn.connect();  
        //显示资源大小, 单位字节  
        System.out.println(conn.getContentLength());  
        //显示资源内容类型  
        System.out.println(conn.getContentType());  
        //显示资源的内容编码  
        System.out.println(conn.getContentEncoding());  
    } catch (Exception e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}
```

疑难点评

获取网络资源大小的功能在下载软件中非常常见, 每一个下载都会显示资源的总大小、已下载量等信息。

知识链接

FAQ7.14 如何实现 Internet 资源的单线程下载?

FAQ7.15 URL 如何通过 proxy 代理访问 Internet 资源?

FAQ7.14 如何实现 Internet 资源的单线程下载?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

URLConnection 类可实现网络资源的读取和写入功能。URLConnection 类提供的主要方法如下所示:

```
//打开与 URL 引用的资源的通信链接
public abstract void connect() throws IOException
//返回 URL 引用的资源的内容长度, 如果内容长度未知, 则返回-1
public int getContentLength()
//返回 URL 引用的资源的内容编码, 或者如果编码为未知, 则返回 null
public String getContentEncoding()
//返回从此打开的连接读取的输入流
public InputStream getInputStream() throws IOException
//返回写入此连接的输出流
public OutputStream getOutputStream() throws IOException
```

实现单线程下载网络资源, 示例代码如下:

```
public static void main(String[] args) {
    try {
        URL url = new
URL("http://gpxz.com/live/UploadFile/2008-8/20088413571256734.mp3");
        URLConnection conn=url.openConnection();
        conn.connect();
        InputStream is = conn.getInputStream();

        String file = url.getFile();
        //获取文件名
        String name = file.substring(file.lastIndexOf('/')+1);
        System.out.println(name);
        //将文件保存在 F 盘下
        FileOutputStream fos = new FileOutputStream("F:\\"+name);
        byte[] buf = new byte[1024];
        int size = -1;
        //循环读取网络资源信息, 写入本地文件
        while ((size = is.read(buf)) != -1) {
            fos.write(buf,0,size);
        }
        fos.close();
        is.close();
        conn.connect();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



```
}  
}
```

疑难点评

单线程下载是实现多线程下载的基础, 主要使用 `URLConnection` 类。

知识链接

FAQ7.13 如何获取 Internet 资源的大小?

FAQ7.15 URL 如何通过 proxy 代理访问 Internet 资源?

FAQ7.17 如何实现 Internet 资源的多线程下载?

FAQ7.15 URL 如何通过 proxy 代理访问 Internet 资源?

📖 难度系数: ★★★

📖 问题频率: 85%

核心解答

对于在局域网内发布的 Java 应用程序, 由于其没有直接连接到 Internet 网络, 因此不能直接用 `java.net.URL` 访问网络上的资源。在这种情况下, 需要为 `java.net.URL` 指定代理服务器。设置代理访问网络资源的代码如下:

```
public static void main(String[] args) {  
    String sUrl = "http://java.sun.com/index.html";  
    //获取系统属性集合  
    Properties prop = System.getProperties();  
    //设置代理服务器的地址  
    prop.put("http.proxyHost", "172.16.0.10");  
    //设置代理服务器端口  
    prop.put("http.proxyPort", "11080");  
    try {  
        URL url = new URL(sUrl);  
        URLConnection conn = url.openConnection();  
        conn.connect();  
        //获取网络资源的大小  
        System.out.println(conn.getContentLength());  
        //获取网络资源的输入流读取信息  
        InputStream is = conn.getInputStream();  
        System.out.println("ic : " + is.read());  
        is.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

疑难点评

如果程序运行的环境是通过代理服务器上网,此时,需要设置代理服务器信息之后才能正确访问网络资源。

知识链接

FAQ7.13 如何获取 Internet 资源的大小?

FAQ7.14 如何实现 Internet 资源的单线程下载?

FAQ7.16 如何实现 Internet 资源下载的断点续传?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

(1) 断点续传原理

在使用 URL 类访问 Internet 资源时,首先需要向 Internet 资源服务器发送一个访问请求,然后服务器根据请求信息将资源信息返回。请求和响应信息的格式都是基于 HTTP 协议的,可以在请求和响应时设置一些参数。

假设服务器域名为 www.qmrz.com, 文件名为 down.zip, 那么访问 down.zip 文件的请求信息,其格式如下:

```
GET /down.zip HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Connection: Keep-Alive
```

在发送请求时,除了上述请求参数外,还可以设置一个“RANGE”参数,该参数用于指定访问资源的字节区间。“RANGE: bytes=10000-20000”的意思是告诉服务器将目标资源中 10 000~20 000 字节之间的内容返回。“RANGE: bytes=10000-”的意思是将目标资源中从 10 000 字节开始到资源结束位置的内容返回。

断点续传的原理就是在每次发送请求时,将上次断点的位置通过“RANGE”参数设定,实现资源的继续访问。

(2) 实现案例

实现断点续传功能,示例代码如下:

```
public class Download {
    private long start = 0;
```

```

private long end = 0;

public DownLoad(long start,long end){
    this.start = start;
    this.end = end;
}

public DownLoad(long start){
    this.start = start;
}

public void down(){
    try {
        URL url = new
URL("http://bz.mtv0.com/d/21234352191/81195231587319.jpg");
        URLConnection conn=url.openConnection();
        //设置请求的 RANGE 和 User-Agent 参数
        conn.setRequestProperty("User-Agent","NetFox");
        String sProperty = "bytes="+start+"-";
        //如果指定结束位置大于 0, 设置结束位置, 否则默认到文件结尾
        if(end > 0){
            sProperty = "bytes="+start+"-"+end;
        }
        conn.setRequestProperty("RANGE",sProperty);

        conn.connect();
        InputStream is = conn.getInputStream();
        //将文件保存在 F 盘下
        String file = url.getFile();
        String name = file.substring(file.lastIndexOf('/')+1);
        System.out.println(name);
        //以追加方式写文件
        FileOutputStream fos = new FileOutputStream("F:\\"+name,true);
        byte[] buf = new byte[1024];
        int size = -1;
        while ((size = is.read(buf)) != -1) {
            fos.write(buf,0,size);
        }
        fos.close();
        is.close();
        conn.connect();
    } catch (Exception e) {
        //TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}

```

测试代码如下:

/**

* 注意: 执行完 d1 的 down 方法后, 再执行 d2 的 down 方法, 同时执行会并发


```
* 下载, 发生冲突
* @param args
*/
public static void main(String[] args) {
    Download d1 = new Download(1000,2000);
    d1.down();

    //    Download d2 = new Download(2000);
    //    d2.down();
}
```

疑难点评

断点续传功能可以实现某一资源分多次下载, 现在主流的下载软件都具有该功能。例如下载一个资源时可以断掉, 下次再继续下载。

知识链接

FAQ7.14 如何实现 Internet 资源的单线程下载?

FAQ7.17 如何实现 Internet 资源的多线程下载?

FAQ7.17 如何实现 Internet 资源的多线程下载?

□ 难度系数: ★★★★★

□ 问题频率: 95%

核心解答

多线程下载的示例由 MultiDownload 和 DownThread 两个类构成。

□ DownThread 类

DownThread 类是一个下载线程, 该线程在启动时需要指定下载文件的 URL、下载起始点和下载结束点等参数。下载过程中利用 RandomAccessFile 类将下载的内容写入目标文件, 通过 RandomAccessFile 类的 seek() 方法可以指定写入目标文件的起始位置。DownThread 类的实现代码如下:

```
public class DownThread implements Runnable{
    String url = "";
    File file;
    long startPosition;
    long endPosition;

    public DownThread(String url,File file,long startPosition,long endPosition){
        this.url = url;
        this.file = file;
        this.startPosition = startPosition;
        this.endPosition = endPosition;
    }
}
```

```

public void run(){
    try{
        URL downUrl = new URL(url);
        HttpURLConnection connection =
(HttpURLConnection)downUrl.openConnection();
        //设置请求参数
        connection.setRequestProperty("User-Agent","NetFox");
        String sProperty = "bytes="+startPosition+"-";
        if(endPosition > 0){
            sProperty+=endPosition;
        }
        connection.setRequestProperty("RANGE",sProperty);
        System.out.println(sProperty);
        //利用 RandomAccessFile 将下载的信息写入本地文件
        RandomAccessFile dst = new RandomAccessFile(file,"rw");
        //从 startPosition 字节的位置开始写入信息
        dst.seek(startPosition);
        //读取 URL 定位的文件信息
        InputStream is = connection.getInputStream();
        BufferedInputStream bis = new BufferedInputStream(is);
        byte buf[] = new byte[1024];
        long size = -1;
        while ((size = bis.read(buf)) > 0) {
            dst.write(buf,0,(int)size);
        }

        dst.close();
        bis.close();
        connection.disconnect();
    }catch(Exception e){
        e.printStackTrace();
    }
    System.out.println("一个线程结束");
}
}

```

❑ MultiDownload 类

MultiDownload 类主要功能是获取下载文件的大小, 并根据指定的线程数量计算出每一个线程应该下载的起始点和结束点, 然后创建并启动线程。MultiDownload 类的实现代码如下:

```

public class MultiDownload {

    public static void main(String[] args){
        MultiDownload down = new MultiDownload();
        try {
            //指定 3 个线程下载 img/baidu_logo.gif, 下载后放到 F 盘下
            down.downProcess("http://www.baidu.com/img/baidu_logo.gif",
"F:\\ logo.gif", 3);
        } catch (Exception e) {

```



```

        e.printStackTrace();
    }
}

public void downProcess(String url,String dest,int threadNum) throws
Exception{
    //获取文件大小
    long fileSize = getFileLength(url);
    //计算每个线程需要下载多少字节
    long byteCount = fileSize/threadNum+1;
    File file = new File(dest);
    int i= 0;
    while(i<threadNum){
        //计算每个线程需要下载文件的起始点和结束点
        final long startPosition = byteCount*i;
        final long endPosition = byteCount*(i+1);
        //启动一个线程
        if(i == threadNum-1){
            DownThread fileThread = new
DownThread(url,file,startPosition,0);
            new Thread(fileThread).start();
        }else{
            DownThread fileThread = new
DownThread(url,file,startPosition,endPosition);
            new Thread(fileThread).start();
        }
        i++;
    }
}

public long getFileLength(String url) throws Exception{
    URL downladURL = new URL(url);
    HttpURLConnection con = (HttpURLConnection)
downladURL.openConnection();
    long size = -1;
    int stateCode = con.getResponseCode();
    if (stateCode == 200) {
        size = con.getContentLength();
        con.disconnect();
    }
    return size;
}
}

```

疑难点评

多线程下载可以提高下载速度，具有较强的实战意义，读者可重点分析学习该功能。

知识链接

FAQ7.16 如何实现 Internet 资源下载的断点续传?

FAQ7.18 如何解析 Internet 网页内容?

 难度系数: ★★★★★

 问题频率: 85%

核心解答

在很多情况下，需要获取 Internet 网页中指定的信息，例如获取所有的超链接等。

解析网页内容的重要思路是：首先使用 URL 类和 URLConnection 类获取指定网页的流信息，然后使用字符串切割或正则表达式技术将需要的信息获取。

下面的示例实现了解析网页功能，将网页中含有的超链接都解析出来。示例代码如下：

```

public class Test
{
    /**
     *遍历输入的页面里的内容，以找出 URL
     *@param urlString
     *@return urls
     */
    public Collection searchURL(String urlString)
    {
        URL url2=null;
        URLConnection conn=null;
        String nextLine=null;
        StringTokenizer tokenizer=null;
        Collection urlCollection=new ArrayList();
        try
        {
            //定位网页资源
            url2=new URL(urlString);
            //打开资源连接
            conn=url2.openConnection();
            conn.connect();
            BufferedReader Reader1=new BufferedReader(new
InputStreamReader(conn.getInputStream()));
            //读取网页信息，解析出 HTTP 超链接，放入一个 Collection 集合
            while((nextLine=Reader1.readLine())!=null)
            {
                tokenizer=new StringTokenizer(nextLine);
                while(tokenizer.hasMoreTokens())
                {
                    String urlToken=tokenizer.nextToken();

```

```

        if (hasMatch(urlToken))
            urlCollection.add(trimURL(urlToken));
    }
}

catch(MalformedURLException ex)
{
    ex.printStackTrace();
}
catch(IOException ex)
{
    ex.printStackTrace();
}
return urlCollection;
}

/**
 *判断字符串中是否含有 http 字符, 即超链接
 * @param token
 * @return true or false
 */
private boolean hasMatch(String token)
{
    return token.indexOf("http")!=-1;
}

/**
 *将 HTTP 链接从参数字符串中截取出来
 * @param string
 * @return string has been trimed
 */
private String trimURL(String url1)
{
    String tempStr=null;
    int beginIndex=url1.indexOf("http");
    int endIndex=url1.length();
    tempStr=url1.substring(beginIndex,endIndex);
    endIndex=tempStr.indexOf("");
    if(endIndex==-1)
        endIndex=tempStr.length();
    return tempStr.substring(0,endIndex);
}

/**
 * 在 main 方法中测试
 */
public static void main(String args[])
{
    Test t = new Test();
    Collection urlCollection=t.searchURL("http://www.baidu.com");
    Iterator iter=urlCollection.iterator();
}

```

```
while (iter.hasNext())  
{  
    System.out.println(iter.next());  
}  
}
```

疑难点评

网页也是网络中的文件资源，因此通过 URL 等类也可以获取网页中的 HTML 信息。

知识链接

FAQ7.15 URL 如何通过 proxy 代理访问 Internet 资源?

数字资源
PDF

第8章

Java 常用功能

本章重点介绍一些 Java 编程中常用 API 的应用和相关疑难问题，内容主要涉及 System、String、Math、Date、Collections 等常用类的具体应用，以及 Runtime、MessageDigest 等类在执行系统 cmd 命令和信息加密等方面的使用。

FAQ8.01 如何使字符串中包含 “” 字符

📖 难度系数：★★★

📖 问题频率：85%

核心解答

定义一个字符串通常有以下两种方式可以实现。

(1) String s1 = "";

(2) String s2 = new String("");

以上两种方式都可以初始化一个字符串，但是当需要初始化的字符串里包含有双引号时如果直接按照习惯的方式编写，将会引起歧义而不能通过编译，例如：

```
String s1 = "";
```

因为在编译时第 2 个双引号将被认为是第 1 个双引号的结束，所以第 3 个双引号由于是单独出现从而不能通过编译。这时需要对字符串中包含的双引号进行转义，在 Java 中的转义符是反斜杠 “\”，可以通过反斜杠来对可能引起歧义的字符进行转义。示例代码如下：

```
public class TestNewString {  
  
    public static void main(String[] args) {  
        String s1 = new String("new a string include \"\"");  
        String s2 = "I am a \"";  
        System.out.println("s1:" + s1);  
        System.out.println("s2:" + s2);  
    }  
}
```

输出结果如下：

```
s1:new a string include ""  
s2:I am a "
```

疑难点评

如果在字符串中包含一些特殊字符,例如“,”、“”等,需要使用转义字符“\”,否则会将字符串截断,打乱代码格式。

知识链接

FAQ8.03 如何替换字符串中的字符或子字符串?

FAQ8.04 如何过滤字符串前后以及中间出现的空格?

FAQ8.02 如何实现字符串和整数之间的转化?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

String 类型和 int 类型、String 类型和 Double 类型等数据类型之间的相互转化,在 Java 编程中被普遍用到,因此本章以 String 类型和 int 类型之间的转化为例对该问题进行解答。

从前面的章节可以知道 int 属于简单类型,对应的封装类为 Integer 类,而 String 本身就属于引用类型,因此如果需要将 String 类型转化为 int 类型,实质上首先需要调用 Integer 类中的 parseInt(String str)方法将其转化为 Integer 类型,而且需要转化的字符串必须只包含数字,如果字符串不是数字,那么在运行时将会出现抛出 NumberFormatException 异常。

而 int 类型如果转化为 String 类型,只需要简单地通过字符串的连接操作即可实现,但是其内在实现机制同样是用到了 JDK 1.5 的自动拆装箱,当在 int 后面使用字符串连接符“+”时,JVM 首先需要将 int 类型转化为 Integer 类型,然后再转化成 String 类型。

示例代码如下:

```
public class TestStringToInt {  
  
    public static void main(String[] args) {  
        TestStringToInt tst = new TestStringToInt();  
        tst.StringToInt();  
        tst.IntToString();  
    }  
  
    //将 String 类型转化为 int 类型  
    public void StringToInt() {  
        String str1 = "100";  
        String str2 = "200";  
        Integer a = Integer.parseInt(str1); //使用 Integer 类的 parseInt(String str)方法  
        Integer b = Integer.parseInt(str2);
```

```
Integer c = a * b;
System.out.println("String parse to int:" + c);
}

//将 int 类型转化为 String 类型
public void IntToString() {
    int a = 100;
    int b = 200;
    String str = a + b + "";
    System.out.println("int parse to String:" + str);
}
}
```

对于 String 和 Double 类型, int 和 Double 类型等之间的转化, 与上文所述方法类似, 在此不再赘述。

```
//int 类型转化为 Double 类型
int i = 12;
Double d = i+0.0;
//String 类型转化为 Double 类型
String s = "123.34";
Double d2 = Double.parseDouble(s);
```

疑难点评

编程时经常涉及数值和字符串之间相互转化, 在将字符串转化为数值型时, 字符串中的每个字符必须符合数值格式, 否则会发生异常。

知识链接

FAQ8.06 如何判断一个字符串是否符合数值格式?

FAQ8.10 如何实现日期格式和字符串之间的转化?

FAQ8.03 如何替换字符串中的字符或子字符串?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

Java.lang.String 类中提供了以下 3 种字符串替换的方法。

```
public String replace(CharSequence target,CharSequence replacement)
public String replaceFirst(String regex,String replacement)ReplaceAll()
public String replaceAll(String regex,String replacement)
```

replace()方法使用指定的字面值替换序列, 替换此字符串匹配字面值目标序列的每个子字符串; replaceFirst()方法使用给定的字符串替换此字符串匹配给定的正则表达式的第 1 个子字符串并返回; replaceAll()方法使用给定的字符串替换此字符串匹配给定的正则表达式的每一个子字符串

并返回。该替换从此字符串的开始一直到结束,例如,用“b”替换字符串“aaaaa”中的“aa”将生成“bba”而不是“abb”或者“bab”。

示例代码如下:

```
public class TestReplace {  
    public static void main(String[] args) {  
        String str = "lang lang way to go";  
        str = str.replace('e', 'a');  
        str = str.replace("lang", "long");  
        System.out.println(str);  
    }  
}
```

输出结果如下:

```
long long way to go
```

疑难点评

替换字符串中的字符或子串,最常见的应用是替换某字符串中的换行符或空格字符。

知识链接

FAQ8.04 如何过滤字符串前后以及中间出现的空格?

FAQ8.04 如何过滤字符串前后以及中间出现的空格?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

对于过滤字符串空格的问题,分为以下两种情况。

- (1) 只需要过滤字符串前后的空格,而中间的空格不需要考虑。
- (2) 字符串的前后和中间可能出现的空格,都需要过滤。

当只需要对字符串前后出现的空格进行过滤时,只需要通过 `String` 类中提供的 `trim()` 方法即可实现。该方法返回字符串对象的一个副本,忽略前导和尾部空格。如果此字符串对象表示一个空的字符串序列或者该字符串的前导和尾部都没有空格出现,则返回该字符串本身。

示例代码如下:

```
public class TestTrim {  
    public static void main(String[] args) {  
        String str = " Merry Christmas!";  
        TestTrim tt = new TestTrim();  
        str = tt.methodA(str);  
        System.out.println(str);  
    }  
}
```

```
public String methodA(String str) {  
    //只能过滤字符串左右两端的空格  
    str = str.trim();  
    return str;  
}
```

输出结果如下:

MerryChristmas!

在上面的示例中只对字符串前后的空格进行了过滤,而中间的空格 trim()方法却无法过滤。对于需要对中间的空格或者其他特殊字符进行过滤时可采用以下方法。

□ 使用字符串替换

```
public String methodB(String str) {  
    //该方法较为简洁,是 JDK 1.4 新增的方法  
    str = str.replaceAll(" ", "");  
    return str;  
}
```

□ 使用 substring()

```
public String methodC(String str) {  
    while (true) {  
        //String.indexOf(int ch)方法当字符串中没有对应字符时返回-1  
        //当字符串中存在空格时  
        if (str.indexOf(" ") != -1) {  
            //算法一  
            str = str.replace(" ", "");  
            //算法二  
            //同样的功能的实现方法和算法非常多  
            //在编程时优先选择效率高的,算法二显然没有算法一效率高  
            str = str.substring(0, i) + str.substring(i + 1, str.length());  
        } else {  
            break;  
        }  
    }  
    return str;  
}
```

□ 使用 StringTokenizer 类

```
public String methodD(String str) throws UnsupportedOperationException {  
    byte[] bt = str.getBytes();  
    //创建一个 StringTokenizer 类  
    StringTokenizer st = new StringTokenizer(str, " ");  
    //StringBuffer 类在处理字符串连接时比 String 效率高  
    StringBuffer sb = new StringBuffer();  
    int i = 1;  
    //如果字符串中存在下一个标记  
    while (st.hasMoreTokens()) {  
        i++;  
        //得到下一个标记并连接  
        System.out.println(sb.append(st.nextToken()));  
    }  
}
```

```
    }  
    return sb.toString();  
}
```

上述代码示例中,给出了多种过滤字符串中间空格的方法,其中比较常用的是 `methodA()` 和 `methodB()` 方法, `methodC()` 方法虽然同样能实现对应功能,但是效率比较低;而 `methodD()` 方法是使用 `StringTokenizer` 类中的方法来实现的,该类通常也会在对字符串进行截取操作时被广泛地使用。

疑难点评

过滤字符串中字符的方法有很多,上面介绍了3种方法,使用 `replaceAll()` 方法最为方便,但使用 `StringTokenizer` 类效率最高。

知识链接

FAQ8.03 如何替换字符串中的字符或子字符串?

FAQ8.05 如何对字符串中的子字符或子字符串进行截取?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

对于字符串的截取操作, `String` 类中提供了以下两个方法供选择。

(1) `public String substring(int beginIndex)`。

(2) `public String substring(int beginIndex, int endIndex)`。

`substring(int beginIndex)` 方法返回从字符串索引 `beginIndex` 开始一直到该字符串结尾的子字符串,如果开始索引 `beginIndex` 为负数或者超过指定大小(字符串长度-1),则会抛出 `IndexOutOfBoundsException` 异常; `substring(int beginIndex, int endIndex)` 方法返回一个从指定的 `beginIndex` 处开始,一直到索引 `endIndex-1` 处的子字符或字符串,其中包含 `endIndex` 而不包含 `beginIndex`。因此,该子字符串的长度为 `endIndex-beginIndex`。

示例代码如下:

```
public class TestSubstring {  
  
    public static void main(String[] args) {  
        TestSubstring ts = new TestSubstring();  
        ts.test1();  
        ts.test2();  
    }  
  
    public void test1() {
```



```
String str = "Happy_new_year";
String str1 = str.substring(0, 5); //从索引为 0 (包括) 处开始截取到 5 (不包括) 处结束
String str2 = str.substring(6, 9); //从索引为 6 (包括) 处开始截取到 9 (不包括) 处结束
String str3 = str.substring(10); //从索引为 10 (包括) 处开始截取到字符串结尾
System.out.println("test1:" + str1 + " " + str2 + " " + str3 + "!");
}

public void test2() {
    String str = "Happy_new_year";
    String str1 = str.substring(0, str.indexOf('_'));
    String str2 = str.substring(str.indexOf('_')+1, str.lastIndexOf('_'));
    String str3 = str.substring(str.lastIndexOf('_')+1);
    System.out.println("test2:" + str1 + " " + str2 + " " + str3 + "!");
}
}
```

输出结果如下:

```
test1:Happy new year!
test2:Happy new year!
```

疑难点评

substring()方法重载了两次,在使用时需要注意参数的含义,确认截取的边界值是否包含参数指定位置的字符。

知识链接

FAQ8.07 如何实现字符串的切割和查找?

FAQ8.06 如何判断一个字符串是否符合数值格式?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

Java 中判断一个字符串是否符合数值类型的方法有很多,下面介绍两种方法。

1. 正则表达式

在判断一个字符串是否满足指定格式时,使用正则表达式会非常方便。首先利用模板字符编写正则表达式,然后使用 Pattern 和 Matcher 比较字符串和正则表达式是否匹配。示例代码如下:

```
public static boolean checkNum1(String s){
    Pattern p = Pattern.compile("\\d*");
    Matcher m = p.matcher(s);
    boolean b = m.matches();
    return b;
}
```

在上述代码中,模板“\\d*”中的“\\d”代表一个数字字符,“*”表示前面字符可出现 0 次到多次。

模板字符指定规则和含义请参考 JDK 帮助文档中关于 Pattern 类的介绍。

2. 字母循环比较

字符串可以看做是字母的数组,利用字符串的 charAt()方法可以获取字符串中的每一个字母,然后循环判断是否在字母‘0’~‘9’的范围。示例代码如下:

```
public static boolean checkNum(String s){
    StringBuffer sb = new StringBuffer(s);
    for(int i=0;i<sb.length();i++){
        char c = sb.charAt(i);
        if(c<'0' || c>'9'){
            return false;
        }
    }
    return true;
}
```

疑难点评

判断一个字符串是否符合数值格式的方法有很多,推荐使用正则表达式的方法。

知识链接

FAQ8.02 如何实现字符串和整数之间的转化?

FAQ8.10 如何实现日期格式和字符串之间的转化?

FAQ8.07 如何实现字符串的切割和查找?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

1. 字符串切割

字符串切割主要指获取字符串的某一部分或按特定标识将字符串分解。其实现方法有很多,主要有以下几种。

(1) 使用 String 类的 subString()方法。

subString()方法具有字符串截取的功能,可实现从字符串中截取出一部分子字符串的功能。

例如字符串“unhappy”,使用该方法可以获取“happy”。示例代码如下:

```
public void test1() {
    String s = "unhappy".substring(2); //返回 "happy"
    System.out.println(s);
}
```

```
s = "Harbison".substring(3);    //返回 "bison"
System.out.println(s);
s = "emptiness".substring(9);   //返回 ""
System.out.println(s);
s = "hamburger".substring(4, 8); //返回 "urge"
System.out.println(s);
s = "smiles".substring(1, 5);   //返回 "mile"
System.out.println(s);
}
```

(2) 使用 String 类的 split()方法。

split()方法具有切割字符串的功能，可按照指定字符对字符串进行切割。例如字符串“javaee.javase.javame”，使用该方法可以按“.”进行切割获取“javaee”、“javase”和“javame”。示例代码如下：

```
public void test2(){
    String s = "javaee.javase.javame";
    String ss[] = s.split(".");
    for(String i:ss){
        System.out.println(i);
    }
}
```

(3) 使用 StringTokenizer 类。

StringTokenizer 类与 split()方法的功能相似，也可以用于切割字符串。使用 StringTokenizer 类实现 split()方法示例的代码如下：

```
public void test3(){
    String s = "javaee.javase.javame";
    StringTokenizer st = new StringTokenizer(s, ".");
    while (st.hasMoreTokens()) {
        System.out.println(st.nextToken());
    }
}
```

(4) 使用正则表达式。

正则表达式的使用非常灵活，主要用于实现字符串格式验证、切割和分析。使用时首先利用模板字符制定表达式模板，然后通过 Pattern 和 Matcher 类进行操作。示例代码如下：

```
public void test4(){
    String s = "javaee.javase.javame";
    Pattern p = Pattern.compile(".");
    String ss[] = p.split(s);
    for(String i:ss){
        System.out.println(i);
    }
}
```

2. 字符串分析

字符串分析主要指判断字符串是否包含指定条件的字符信息，也可以在分析过后，利用切割功能获取需要的内容。其实现方法有很多，主要有以下几种。

(1) 使用 String 类的 indexOf() 方法。

indexOf() 方法可以确定指定字符或字符串在源字符串中第一次出现的位置。示例代码如下:

```
public void test5(){
    //源字符串
    String s = "javaeejavasejavame";
    //寻找的目标字符串
    String p = "javase";
    //确定目标字符串在源字符串中的位置
    int i = s.indexOf(p);
    System.out.println(i);
    //截取出目标字符信息
    String t = s.substring(i,i+p.length());
    System.out.println(t);
}
```

lastIndexOf() 方法与 indexOf() 方法相似, 功能是判断字符或字符串最后一次出现的位置。使用 String 类的方法只能分析指定字符串在源字符串中第一次和最后一次出现的位置, 如果指定字符串在源字符串中出现很多次, 就不适合使用, 可以使用正则表达式方法分析。

(2) 使用正则表达式

使用正则表达式可以在源字符串中寻找任意格式的指定字符或字符串信息。示例代码如下:

```
public void test6(){
    //源字符串
    String s = "javaeejavasejavame";
    //寻找的目标字符串
    String d = "java";
    //从源字符串中寻找“java”出现的位置
    Pattern p = Pattern.compile(d);
    Matcher m = p.matcher(s);
    //寻找是否含有目标字符串
    while(m.find()){
        //将目标字符串出现的位置获取, 从源字符串中截取
        String t = s.substring(m.start(),m.end());
        System.out.println(t);
    }
}
```

疑难点评

字符串切割和分析是比较常用的功能, 在使用时可以根据实际情况选择不同方法。StringTokenizer、split() 方法和正则表达式都可以实现相同功能, 如果字符串信息量比较小, 可使用 split() 方法, 该方法使用起来比较简单; 如果字符串信息量比较大, 可以选用 StringTokenizer 和正则表达式方式, 推荐使用正则表达式, 因为 StringTokenizer 类是出于兼容性的原因才被保留下来, 所以在新代码中不推荐使用。正则表达式使用非常灵活, 可实现很多与字符串相关的处理功能, 建议读者重点学习。

知识链接

FAQ8.03 如何替换字符串中的字符或子字符串?

FAQ8.05 如何对字符串中的子字符或子字符串进行截取?

FAQ8.08 如何实现十进制和二进制之间的相互转化?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

十进制和二进制的转换问题, 可以通过 `Integer` 类中的相关方法实现。

1. 十进制转化为二进制

`Integer.toString(int i)`方法以二进制无符号整数形式返回一个 `int` 类型整数参数的字符串序列。如果需要返回数值类型, 只需要调用 `Integer` 或者 `Double` 类中的 `parseInt()` 或 `parseDouble()` 方法即可。

2. 二进制转化为十进制

`Integer` 类中提供的 `parseInt(String s, int radix)` 方法, 返回字符串序列以指定进制表示的整数表示形式。

示例代码如下:

```
public class TestToBinary {  
    public static void main(String[] args) {  
        int i = 111;  
        String str = Integer.toString(i);  
        Integer in = Integer.parseInt(str);  
        System.out.println("十进制转化为二进制" + in);  
        int no = Integer.parseInt(str, 2);  
        System.out.println("二进制转化为十进制" + no);  
    }  
}
```

输出结果如下:

十进制转化为二进制 111011

二进制转化为十进制 123

疑难点评

利用 `Integer` 的 `toString()` 和 `parseInt()` 方法可以实现数值十进制和二进制的转化。通过 `Integer` 的 `toHexString()` 方法可以将十进制数转化成十六进制的表示。

知识链接

FAQ8.02 如何实现字符串和整数之间的转化?

FAQ8.10 如何实现日期格式和字符串之间的转化?

FAQ8.09 如何将字节流转换为指定编码的字符串?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

在 `Java.lang.String` 类中提供了以下获取字符串字节序列的方法。

```
getBytes()  
getBytes(String charsetName)
```

无参的 `getBytes()` 方法使用平台默认的字符集将此 `String` 解码为字节序列, 并将结果存储到一个新的字节数组中; 而 `getBytes(String charsetName)` 方法则使用指定的字符集将此 `String` 解码为字节序列, 并将结果存储到一个新的字节数组中。

而需要对解码之后的字节序列重新编码成字符串时, 只需要调用 `String` 类带参数的构造方法 `String(byte[] bytes, String charsetName)` 并指定对应的字符集编码方式即可。

示例代码如下:

```
public class TestGetBytes {  
    public static void main(String[] args) {  
        try {  
            String str = "字符串编码转换";  
            byte[] bt = str.getBytes();  
            String str2 = new String(bt, "GBK");  
            System.out.println(str2);  
        } catch (UnsupportedEncodingException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

疑难点评

将字符串进行编码转换时, 需要首先将字符串转化为字节流信息, 然后再将字节流信息转换为指定编码的字符串。

知识链接

FAQ8.02 如何实现字符串和整数之间的转化?

FAQ8.10 如何实现日期格式和字符串之间的转化?

FAQ8.10 如何实现日期格式和字符串之间的转化?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在 Java 语言中,专门提供了 `DateFormat` 类来实现日期格式和字符串格式之间的转化。但是通常情况下,对于日期格式化问题还是运用其子类 `SimpleDateFormat` 中的 `format()`和 `parse()`方法来完成。

1. `format()`方法

`format()`方法,接收一个 `Date` 类型参数,将该 `Date` 格式化为指定格式的字符串,并返回。

示例代码如下:

```
public void test1() {
    Date date = new Date();
    System.out.println("System Date:" + date);
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    String date_format = sdf.format(date);
    System.out.println("Format Date:" + date_format);
}
```

上述代码中,指定格式为“yyyy-MM-dd”。在定义格式时可以使用模式字母,如表 8-1 所示。

表 8-1 模式字母介绍

模式字母	含 义	表 示 示 例
G	Era 标志符	AD
y	年	1996; 96
M	年中的月份	July; Jul; 07
w	年中的周数	27
W	月份中的周数	2
D	年中的天数	189
d	月份中的天数	10
F	月份中的星期	2
E	星期中的天数	Tuesday; Tue
a	Am/pm 标记	PM
H	一天中的小时数 (0~23)	0
k	一天中的小时数 (1~24)	24
K	am/pm 中的小时数 (0~11)	0
h	am/pm 中的小时数 (1~12)	12

续表

模式字母	含 义	表 示 示 例
m	小时中的分钟数	30
s	分钟中的秒数	55
S	毫秒数	978
z	时区 General time zone	Pacific Standard Time; PST; GMT-08:00
Z	时区 RFC 822 time zone	-0800

注意：在格式字符串中，如果需要使用模式字母本身代表的字符，可以使用单引号，例如“yyyy.MM.dd G 'at' HH:mm:ss z”。

2. parse()方法

parse()方法接收一个 String 类型参数，并将其转化为 Date 类型。示例代码如下：

```
public void test2() {
    try {
        String str = "2008.08.08";
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy.MM.dd");
        Date date_parse = sdf.parse(str);
        System.out.println("Parse String to Date:" + date_parse);
    } catch (ParseException e) {
        e.printStackTrace();
    }
}
```

将字符串转化为 Date 类型，还可以使用 java.sql.Date.valueOf(String s)方法。示例代码如下：

```
public void test3() {
    try {
        String str = "2008-08-08";
        Date date_parse = java.sql.Date.valueOf(str);
        System.out.println("Parse String to Date:" + date_parse);
    } catch (ParseException e) {
        e.printStackTrace();
    }
}
```

疑难点评

SimpleDateFormat 和 DateFormat 都可用于实现日期和字符串之间的转换，DateFormat 主要应用于国际化环境，而 SimpleDateFormat 则适用于本地化环境。

知识链接

FAQ8.02 如何实现字符串和整数之间的转化？

FAQ8.17 如何获取系统当前时间？

FAQ8.11 String、StringBuffer 和 StringBuilder 有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

String 类代表定长字符串, 其内容在创建之后是不可更改的。

StringBuffer 类与 String 类相似, 代表的是可变长的字符串缓冲区, 通过特定的方法可以改变字符串序列的长度和内容, 并且对于多线程操作是安全的。在字符串的连接操作上提供了性能和效率都优于 String 类的 “+” 的 append() 方法, 因此如果需要大量频繁地进行字符连接操作时, 优先采用 StringBuffer 类的 append() 方法; 如果从编程习惯上讲, “+” 比 append() 方法更具有可观性, 如果只是简单的字符串连接可以采用 String 类的 “+” 来提高代码的可读性。

StringBuilder 类是 StringBuffer 类的一个等价类, 该类与 StringBuffer 类具有相同的方法, 且同样代表的是可变长的字符串缓冲区, 不同的地方在于 StringBuilder 类是非线程安全的。但是也正是因为少了很多的同步操作, 在效率上会高于 StringBuffer 类。因此如果不涉及多线程操作, 可以优先考虑使用 StringBuilder 类来提高方法的执行效率。

示例代码如下:

```
public class TestStringAppend {
    public static void main(String[] args) {
        String s = "I love you";
        String str = "";
        long start = System.nanoTime();
        for (int i = 0; i < 1000; i++) {
            str += s;
        }
        long end = System.nanoTime();
        System.out.println("String's \"\"+\"\" :\" + (end - start));
        StringBuffer sbuffer = new StringBuffer("");
        long start1 = System.nanoTime();
        for (int i = 0; i < 1000; i++) {
            sbuffer = sbuffer.append(str);
        }
        long end1 = System.nanoTime();
        System.out.println("StringBuffer's append:\" + (end1 - start1));
        StringBuilder sbuilder = new StringBuilder("");
        long start2 = System.nanoTime();
        for (int i = 0; i < 1000; i++) {
            sbuilder = sbuilder.append(str);
        }
        long end2 = System.nanoTime();
    }
}
```



```
        System.out.println("StringBuilder's append:" + (end2 - start2));  
    }  
}
```

输出结果如下:

```
String's "+" :29271877  
StringBuffer's append:140974062  
StringBuilder's append:135180309
```

疑难点评

虽然 String、StringBuffer 和 StringBuilder 都属于字符串类型,但是在使用时不能混用,程序员需要根据实际情况选择合适的字符串类型。

知识链接

FAQ8.01 如何使字符串中包含 “ ” 字符?

FAQ8.09 如何将字节流转换为指定编码的字符串?

FAQ8.12 如何获得一个随机数?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

Math 类中提供的 random()方法能够返回 0~1 (包含 0) 之间的伪随机 double 类型数值,该方法提供多线程同步的机制。例如下面的实例中需要返回 1~100 的随机数:

```
Math.round(Math.random() * 100);
```

另外,在 Java 体系中还为随机数的生成提供了一个专门的 Random 类,用于生成伪随机数,其中对应多种数据类型和需要有多种对应方法可供调用,例如下面的实例中同样返回 1~100 之间的随机数。

```
new Random().nextInt(1000);
```

疑难点评

使用 Math 和 Random 类都可以获取随机数,Math.random()方法可获取 0~1 之间的浮点数;而 Random 的 nextInt()方法可获取 0 至参数之间的整数。这里所谓的随机也并不是完全随机的,而是计算机按某种特定概率算法生成的。

知识链接

FAQ8.02 如何实现字符串和整数之间的转化?

FAQ8.13 List、Set 和 Map 是否继承自 Collection 接口？有什么区别？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

在 Java 体系中，容器类库分为两大类，即 Collection（集合）和 Map（映像）。Collection 中存放的是一组各自独立的对象，而 Map 中存放的是“键—值”对象。

List 和 Set 都是 Collection（集合）的子接口，List 是一个有序可重复列表，Set 是一个无序重复集。

List 示例代码如下：

```
public void testList(){
    List<String> list = new ArrayList<String>();
    list.add("one");
    list.add("two");
    list.add("one");
    list.add("three");
    list.add("four");
    list.add("five");
    list.add("five");
    //for 循环遍历 List 集合
    for (int i = 0; i < list.size(); i++) {
        System.out.println(list.get(i));
    }
}
```

输出结果如下：

```
one
two
one
three
four
five
```

Set 示例代码如下：

```
public void testSet(){
    Set<String> set = new HashSet<String>();
    set.add("one");
    set.add("two");
    set.add("one");
    set.add("three");
    set.add("four");
    set.add("five");
    set.add("five");
    set.add("");
}
```

```
set.add("");  
// Iterator 迭代器遍历 Set  
Iterator<String> it = set.iterator();  
while (it.hasNext()) {  
    String str = it.next();  
    System.out.println(str);  
}  
}
```

输出结果如下:

```
two  
five  
one  
three  
four
```

Map 是映像的接口, 是一个用于存放“键—值”对的映像集合。Map 示例代码如下:

```
public void testMap(){  
    Map map = new HashMap();  
    map.put("name", "tom");  
    map.put("age", 20);  
    System.out.println(map.get("name"));  
    System.out.println(map.get("age"));  
}
```

输出结果如下:

```
tom  
20
```

疑难点评

List、Set 和 Map 都是集合类型的接口, 这 3 种类型的集合具有不同的特性, 读者需要仔细了解, 以便在实际应用时选取合适的类型。

知识链接

FAQ8.13 List、Set 和 Map 是否继承自 Collection 接口? 有什么区别?

FAQ8.14 ArrayList 与 LinkedList、Vector 的区别是什么?

FAQ8.15 HashMap 和 Hashtable 有什么区别?

FAQ8.14 ArrayList 与 LinkedList、Vector 的区别是什么?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

(1) ArrayList 与 LinkedList 的区别

ArrayList 是一种可以自动增加容量并可以存放不同类型对象的集合类, 由于是实现自 List

接口，因此可用来保存有序可重复对象，并增加了索引的定义，放置在其中的对象都被有序排列，可以通过一个整体索引来访问其中的元素（索引值从 0 开始）。由于 ArrayList 采用的是数组的形式来保存对象，这种方式将对象存放在连续的位置中，虽然在对集合中元素进行遍历或查看时非常方便，但是如果需要向集合中插入或者删除元素时将会非常麻烦。例如要删除一个元素，需要将其后的所有元素的索引都向前挪动一个位置，而插入操作时需要将插入位置后面的所有元素都向后挪动一个位置。

为了解决快速插入和删除操作的需要，JDK 中还提供了另一种列表的实现，即 LinkedList（链接列表）。链接列表将每个对象存放在独立的空间中，而且在空间中还存放有下一个链接的索引，如果是双向链表，那么它还应保存上一个链接的索引，这对快速地插入和删除操作提供了很大的便利，但是不支持快速随机访问，如果需要访问 LinkedList 中的第 n 个元素，必须从头开始查找，依次检索 $n-1$ 个元素。

因此如果需要实现快速的随机访问功能时通常选取 ArrayList；如果需要进行快速地插入和删除操作，则应该选择 LinkedList。

（2）ArrayList 与 Vector 的区别

Vector 和 ArrayList 的用法非常类似，同样是可以自动增加容量且可以存放不同类型对象的集合类，存放的元素具有有序可重复的特征。但是不同的地方在于 Vector 是多线程安全的，而 ArrayList 却不是，也正是因为该原因，Vector 在执行效率上会比 ArrayList 低。因此如果不考虑多线程问题，应该优先使用 ArrayList，否则应该选择 Vector。

疑难点评

ArrayList、LinkedList 和 Vector 虽然有很多相似的用法，但是它们都有各自的一些优点，比如 ArrayList 在随机访问方面效率比较高，LinkedList 在插入和删除操作方面效率较高，而 Vector 具有线性安全的特点，可用于多线程环境中。在不同环境下使用不同的类可以增强程序的执行效率。

知识链接

FAQ8.16 如何遍历 Map 和 Vector 集合？

FAQ8.15 HashMap 和 Hashtable 有什么区别？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

Hashtable 是 Dictionary 的子类并实现了 Map 接口；而 HashMap 是 Map 接口的一个实现类。

除了不同步和允许使用 null 之外, HashMap 类与 Hashtable 大致相同。

在 HashMap 中, null 可以作为键, 但是这样的键只允许有一个, 而作为键值允许存在一个或者多个 null 值。当调用 get() 方法返回 null 值时, 即可以表示 HashMap 中没有该键, 也可以表示该键所对应的值为 null。因此在 HashMap 中不能由 get() 方法来判断 HashMap 中是否存在某个键, 而应该使用 containsKey() 方法来判断。

Hashtable 中的方法是同步的, 而 HashMap 中方法在默认情况下是非同步的。

疑难点评

HashMap 和 Hashtable 都实现了 Map 接口, 除了不同步和允许使用 null 之外, HashMap 类与 Hashtable 大致相同。

知识链接

FAQ8.13 List、Set 和 Map 是否继承自 Collection 接口? 有什么区别?

FAQ8.16 如何遍历 Map 和 Vector 集合?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

1. 遍历 Map

对于 Map 集合的循环遍历有以下两种方法可以实现。

(1) Iterator 迭代器遍历。

示例代码如下:

```
public void test1() {
    Map<String, Integer> map = new HashMap<String, Integer>();
    for (int i = 0; i < 10; i++) {
        map.put("key" + i, i);
    }
    //获得迭代器
    Iterator it = map.entrySet().iterator();
    while (it.hasNext()) {
        Map.Entry entry = (Map.Entry) it.next();
        Object key = entry.getKey();
        Object value = entry.getValue();
        System.out.println("key:" + key + "\tvalue:" + value);
    }
}
```

(2) JDK 1.5 新式 for 循环遍历。

示例代码如下：

```
public void test2() {  
    Map<String, Integer> map = new HashMap<String, Integer>();  
    //JDK 1.4 及其之前的 for 循环  
    for (int i = 0; i < 10; i++) {  
        map.put("key" + i, i);  
    }  
    //JDK 1.5 之后出现的新式 for 循环  
    for (String key : map.keySet()) {  
        Integer value = (Integer) map.get(key);  
        System.out.println("key:" + key + "\tvalue:" + value);  
    }  
}
```

2. 遍历 Vector

对于 Vector 集合的遍历可用以下两种实现方法。

(1) Enumeration 枚举器遍历。

```
//Enumeration 枚举器遍历 Vector 集合  
public void test1() {  
    Vector<String> vector = new Vector<String>();  
    vector.add("one");  
    vector.add("two");  
    vector.add("one");  
    vector.add("three");  
    vector.add("four");  
    vector.add("five");  
    vector.add("five");  
    //调用 Vector 的 elements()方法获取一个 Enumeration 枚举器  
    Enumeration<String> en = vector.elements();  
    //如果有下一个元素  
    while (en.hasMoreElements()) {  
        String str = en.nextElement();  
        System.out.println(str);  
    }  
}
```

(2) for 循环遍历。

示例代码如下：

```
public void test2() {  
    Vector<String> vector = new Vector<String>();  
    vector.add("one");  
    vector.add("two");  
    vector.add("one");  
    vector.add("three");  
    vector.add("four");  
    vector.add("five");  
    vector.add("five");  
    for (int i = 0; i < vector.size(); i++) {  
        System.out.println(vector.get(i));  
    }  
}
```



```
}  
}
```

疑难点评

遍历 Map 和 Vector 类型的集合时,可以使用 for 循环,也可以使用迭代器 Iterator 或枚举器 Enumeration,在使用时读者可以灵活选择。虽然 Iterator 和 Enumeration 都是为方便遍历集合中元素而提供的接口,Iterator 迭代器是为了代替 Java Collections Framework 中的 Enumeration,因为 Iterator 接口添加了一个可选的移除操作,并使用较短的方法名,因此新的实现应该优先考虑使用 Iterator 接口。

知识链接

FAQ8.14 ArrayList 与 LinkedList、Vector 的区别是什么?

FAQ8.17 如何获取系统当前时间?

📖 难度系数: ★★★

📖 问题频率: 95%

核心解答

在 Java 中的 System 类提供返回当前系统时间的方法,具体有以下几种。

(1) currentTimeMillis()方法,返回当前时间与协调世界时间 1970 年 1 月 1 日午夜之间的时间差(以毫秒为单位)。

(2) nanoTime()方法,返回最准确的可用系统计时器的当前值,以毫微秒为单位。此方法只能用于测量已过的时间,与系统或钟表的其他任何时间概念无关。返回值表示从某一固定但任意的时间算起的毫微秒数。通常用于较为精确的计算某个代码块执行的时间。

示例代码如下:

```
public class TestSysdate {  
  
    public static void main(String[] args) {  
        //以毫秒为单位返回当前时间与协调世界时间 1970 年 1 月 1 日午夜之间的时间差  
        long start = System.currentTimeMillis();  
        //以毫微秒为单位返回最准确的可用系统计时器的当前值  
        long start2 = System.nanoTime();  
        for (int i = 1; i <= 100000; i++) {  
            i *= i;  
        }  
        long end = System.currentTimeMillis();  
        long end2 = System.nanoTime();  
        System.out.println(end2 - start2);  
        System.out.println(end - start);  
    }  
}
```

```
}  
  
}
```

Java 中的 `Date` 类, 是专门为日期时间设计的类, 如果需要获取当前操作系统的钟表时间刻度, 可以通过以下代码, 获得精确的系统时间。

```
import java.util.Date;  
  
public class TestDate {  
  
    public static void main(String[] args) {  
        Date date = new Date();  
        System.out.println("System date:" + date);  
    }  
}
```

疑难点评

获取系统时间的方法有很多, 可以使用 `System` 类的 `currentTimeMillis()` 方法, 也可以使用 `java.util.Date` 类, 此外还可以使用 `Calendar` 类。`Calendar` 类包含了很多关于日期和时间操作的方法, 替代了原有 `java.util.Date` 类中的很多操作。

知识链接

FAQ8.18 如何获得系统属性?

FAQ8.21 如何获取当前工程目录?

FAQ8.18 如何获得系统属性?

📖 难度系数: ★★★

📖 问题频率: 85%

核心解答

`Properties` 类表示了一个持久的属性集, 用于提供对系统属性的相关操作方法, 其属性列表中每个键及其对应值都是一个字符串。因为 `Properties` 继承于 `Hashtable`, 所以可对 `Properties` 对象应用 `put()` 和 `putAll()` 方法。但不推荐使用这两个方法, 因为它们允许调用方插入其键或值不是 `Strings` 的项, 建议使用 `Properties` 类所提供的 `getProperty()` 和 `setProperty()` 方法。

使用 `Properties` 读取属性文件的示例代码如下:

```
try {  
    Properties prop = new Properties();  
    //从本地读取属性文件
```

```
prop.load(new FileInputStream("D:\\test.properties"));
String name = prop.getProperty("name");
String sex = prop.getProperty("sex");
String age = prop.getProperty("age");
String address = prop.getProperty("address");
System.out.println("name:" + name + "sex:" + sex + "age:" + age
    + "address:" + address);
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

使用 Properties 获取系统属性的示例代码如下：

```
//获取系统属性
Properties prop2 = System.getProperties();
//得到 keySet 迭代器
Iterator it = prop2.keySet().iterator();
//遍历所有系统属性
while (it.hasNext()) {
    String key = (String) it.next();
    System.out.println(key + "=" + prop2.get(key));
}
```

疑难点评

Properties 类可用于获取与 Java 环境相关的一些系统属性，也可用于读取属性文件中的信息。

知识链接

FAQ8.17 如何获取系统当前时间？

FAQ8.21 如何获取当前工程目录？

FAQ8.19 什么是反射机制？有什么作用？

📖 难度系数：★★★★★

📖 问题频率：95%

核心解答

1. 反射机制定义

反射的概念是由 Smith 在 1982 年首次提出的，主要是指程序可以访问、检测和修改其本身状态或行为的一种能力。

在 Java 环境中，反射机制允许程序在执行时获取某个类自身的定义信息，例如属性和方法

等也可以实现动态创建类的对象、变更属性的内容或执行特定的方法的功能。从而使 Java 具有动态语言的特性，增强了程序的灵活性和可移植性。

2. 反射机制的作用

Java 反射机制主要用于实现以下功能。

- (1) 在运行时判断任意一个对象所属的类型。
- (2) 在运行时构造任意一个类的对象。
- (3) 在运行时判断任意一个类所具有的成员变量和方法。
- (4) 在运行时调用任意一个对象的方法，甚至可以调用 `private` 方法。

注意：上述功能都是在运行时环境中，而不是在编译时环境中。

3. Java 反射机制 API

实现 Java 反射机制的 API 在 `java.lang.reflect` 包下，具有以下几点。

- (1) `Class` 类：代表一个类。
- (2) `Filed` 类：代表类的成员变量。
- (3) `Method` 类：代表类的方法。
- (4) `Constructor` 类：代表类的构造方法。
- (5) `Array` 类：提供了动态创建数组以及访问数组的元素的静态方法。该类中的所有方法都是静态的。

4. 应用示例

(1) 根据类名获取类中定义的所有属性和方法，示例代码如下：

```
/**
 * 打印出 String 类的所有的属性和方法
 */
public void test1() {
    Class c = String.class;
    Method[] methods = c.getMethods();
    for (int i = 0; i < methods.length; i++) {
        System.out.println(methods[i].getName());
    }
    Field[] fields = c.getFields();
    for (Field f: fields) {
        System.out.println(f.getType()+"."+f.getName());
    }
}
```

(2) 根据类名动态创建类的对象，示例代码如下：

```
/**
 * 创建 Student 类的对象
 */
public void test2() throws Exception {
    Class e = Class.forName("tutorial.reflection.Student");
    Student student = (Student) e.newInstance();
    student.setName("java");
}
```

```
student.setId("1001");  
student.show();  
}
```

① 上述代码中，Student 类的定义如下：

```
public class Student {  
    private String id;  
  
    private String name;  
  
    public String getId() {  
        return id;  
    }  
  
    public void setId(String id) {  
        this.id = id;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public void show(){  
        System.out.println(id+"."+name);  
    }  
}
```

② 根据类名和方法名，执行对象的方法，示例代码如下：

```
/**  
 * 执行 Student 对象指定名称的方法  
 */  
public void test3(Student student, String method, String value)  
    throws Exception {  
    String s1 = method.substring(0, 1).toUpperCase();  
    String s2 = method.substring(1);  
    String m = "set" + s1 + s2;  
    System.out.println(m);  
    Class c = student.getClass();  
    Method set = c.getMethod(m, new Class[] { String.class });  
    set.invoke(student, new Object[] { value });  
}
```

③ 动态创建数组对象，对数组元素赋值和取值，示例代码如下：

```
public void test4() {  
    try {  
        Class cls = Class.forName("java.lang.String");  
        //创建一个 String 类型的数组，大小为 10  
        Object arr = Array.newInstance(cls, 10);
```

```
//在数组 5 索引的位置赋值
Array.set(arr, 5, "this is a test");
//获取数组 5 索引位置的值
String s = (String) Array.get(arr, 5);
System.out.println(s);
} catch (Throwable e) {
    System.err.println(e);
}
}
```

Java 语言反射提供一种动态链接程序组件的多功能方法。它允许程序创建和控制任何类的对象，无需提前硬编码目标类。这些特性使得反射特别适用于创建以普通方式与对象协作的库。

反射在性能方面会有一定的损耗，用于字段和方法接入时反射要远慢于直接代码。如果它作为程序运行中相对很少涉及的部分将不会造成影响，因为即使测试最坏情况下的计时图显示的反射操作也只耗用几微秒。

疑难点评

反射机制是 Java 中非常重要的一项功能，应用也非常广泛。在现在流行的 Struts、Hibernate 和 Spring 等各种框架都是基于反射机制实现的，首先需要将 XML 配置文件中的配置信息读取，然后利用反射机制创建对象、执行方法等。

FAQ8.20 如何读取键盘输入的信息？

📖 难度系数：★★★★

📖 问题频率：85%

核心解答

在 System 中定义了 3 个静态流类型的属性，分别为 in、out 和 err，通过这些属性可以读取外部设备的输入信息和将信息向外部设备输出。

在 Java I/O 流操作中，为了方便流的读写操作，提供了一些高级流类，这些类有些用于提高读写效率，有些是为了方便读写信息。

在读取键盘信息时，可以使用高级流 BufferedReader 对 System.in 传入的输入流进行封装，然后以行为单位读取字符串信息。示例代码如下：

```
public static void main(String[] args) {
    InputStream is = System.in;
    InputStreamReader isr = new InputStreamReader(is);
    BufferedReader br = new BufferedReader(isr);
    try {
        String readin = br.readLine();
    }
```



```
        if (readin != null) {  
            System.out.println("Read Keyboard in:" + readin);  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

利用 `Scanner` 类也可以完成读取键盘信息的功能。示例代码如下:

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    String s = sc.next();  
    System.out.println(s);  
    int i = sc.nextInt();  
    System.out.println(i);  
}
```

在上述示例中,通过 `nextInt()` 方法可以将键盘输入的内容自动转换为 `int` 类型返回,如果转化失败将抛出异常。`next()` 方法返回由键盘输入的字符串信息。

在使用 `Scanner` 读取键盘信息时,空格字符是 `next()` 方法读取的一个结束标记,如果键盘输入的字符串被空格分隔, `next()` 只能返回空格字符前面的内容,空格后面的内容需要使用下一次 `next()` 方法读取。

疑难点评

使用 `BufferedReader` 和 `Scanner` 类都可以实现读取键盘信息的功能, `BufferedReader` 可以一次读取一行, `Scanner` 则读取空格或换行符前面的内容。当一行信息包含空格字符时,使用 `BufferedReader` 才可以完整读取。

FAQ8.21 如何获取当前工程目录?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

获取 class 路径的代码如下:

```
System.out.println(Thread.currentThread().getContextClassLoader().getResource(""));  
System.out.println(Test.class.getClassLoader().getResource(""));  
System.out.println(ClassLoader.getSystemResource(""));  
System.out.println(Test.class.getResource(""));  
System.out.println(Test.class.getResource("/"));
```

获取工程路径的示例代码如下:

```
System.out.println(new File("").getAbsolutePath());  
System.out.println(System.getProperty("user.dir"));
```

如果需要获取工程 src 目录, 可以通过工程路径+ “/src” 获取。

疑难点评

在实现文件加载和文件保存时, 经常要获取文件所在路径和文件保存路径, 使用上述方法可以获取工程或 class 路径。

知识链接

FAQ8.17 如何获取系统当前时间?

FAQ8.18 如何获得系统属性?

FAQ8.22 如何使用 Java 调用系统的 exe 文件?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

每个 Java 应用程序都有一个 Runtime 类实例, 使应用程序能够与其运行的环境相连接。Runtime 对象可以通过 Runtime 类提供的静态方法 `getRuntime()` 获取。代码如下:

```
Runtime runtime = Runtime.getRuntime();
```

调用系统的 exe 可执行文件, 首先需要获得一个 Runtime 对象, 然后使用该对象的 `exec()` 方法即可执行一个 exe 文件。示例代码如下:

```
public class TestExe {  
    public static void main(String args[]) {  
        //获取一个 Runtime 对象实例  
        Runtime rn = Runtime.getRuntime();  
        try {  
            System.out.println("-----BEGIN-----");  
            //执行 exe 文件  
            //rn.exec("D:\\QQ\\QQ.exe");  
            //调用系统的记事本软件  
            rn.exec("notepad.exe");  
            System.out.println("-----END-----");  
        } catch (Exception e) {  
            System.out.println("Error exec AnyQ");  
        }  
    }  
}
```

疑难点评

通过 Runtime 类可以方便的调用外部的 exe 文件, 执行代码也非常简便。

知识链接

FAQ8.23 如何使用 Java 执行 cmd 命令?

FAQ8.24 如何使用 Java 程序打开一个 Word 文档?

FAQ8.23 如何使用 Java 执行 cmd 命令?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

通过 Java 程序执行 cmd 命令需要使用 Runtime 类和 Process 类。Runtime 能够使应用程序与其运行环境相连接。Process 可用于控制进程并获取相关信息。Process 类提供了进程输入、进程输出、等待进程完成、检查进程的退出状态以及销毁进程的方法的功能。

利用 Java 程序执行“ping www.163.com”命令的示例代码如下:

```
public static void main(String args[]) {  
    try {  
        //执行“ping www.163.com”命令  
        Process pr = Runtime.getRuntime().exec("ping www.163.com");  
        //获取执行命令进程的响应信息  
        BufferedReader br = new BufferedReader(new InputStreamReader(pr.getInputStream()));  
        while (true) {  
            //读取响应信息  
            String s = br.readLine();  
            //读取完毕,退出循环  
            if (s == null) {  
                break;  
            }  
            System.out.println(s);  
        }  
        br.close();  
        //等待 Process 对象表示的进程终止  
        pr.waitFor();  
        //标志为 0 表示正常结束  
        if (pr.exitValue() == 0) {  
            System.out.println("运行成功!! ");  
        }  
    } catch (Exception e) {  
        System.out.println("Unexpected error executing cmd: " + e);  
    }  
}
```


疑难点评

通过 Runtime 类除了可以调用外部的 exe 文件之外,还可以执行系统的 cmd 命令。在实现某些功能时,借助 cmd 命令会显得非常方便,例如使用 ping 命令判断是否与目标计算机连接等。

知识链接

FAQ8.22 如何使用 Java 调用系统的 exe 文件?

FAQ8.24 如何使用 Java 程序打开一个 Word 文档?

FAQ8.24 如何使用 Java 程序打开一个 Word 文档?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

利用程序打开一个文件,可通过 Runtime 类的 exec()方法执行一个 cmd 命令实现。

Java 打开 Word 文档的示例代码如下:

```
public static void open(){
    try {
        Runtime.getRuntime().exec("cmd.exe /c start f:\\a.doc");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

在上述代码中,打开了系统 F 盘下的一个 a.doc 文档。如果要打开的文件名中包含空格,需要使用下列格式的命令替换。示例代码如下:

```
public static void openBlank(){
    try {
        Runtime.getRuntime().exec("cmd /c \"f:\\a 副本.doc\"");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

上述方法不仅适用于 Word 类型的文档,其他格式的文件也一样适用,但前提是系统安装了打开文件的软件。

疑难点评

通过上述方法可以方便地打开一个目标文件,文件类型可以是 doc、jpg 等类型。

知识链接

FAQ8.22 如何使用 Java 调用系统的 exe 文件?

FAQ8.23 如何使用 Java 执行 cmd 命令?

FAQ8.25 如何使用 MD5 和 SHA 算法加密信息?

📖 难度系数: ★★★★★ · 📖 问题频率: 95%

核心解答

在系统中经常要存储一些用户信息,例如登录名和密码等。出于安全性考虑,有些信息需要采用某些算法加密之后再存储。目前广泛使用的算法有 MD5 和 SHA-1 等。

Hash 算法主要用于信息安全领域中加密,它可以把一些不同长度的信息转化成固定长度的 128 位编码。MD5 和 SHA 是目前应用最为广泛的 Hash 算法,主要应用文件校验、数字签名和鉴权协议等领域。MD5 和 SHA 属于非对称性加密算法,一般被认为是不可逆的。

Java 在实现 MD5 和 SHA-1 算法加密时,主要使用了 `java.security.MessageDigest` 类。`MessageDigest` 类为应用程序提供信息摘要算法的功能,例如 MD5 或 SHA 算法。信息摘要安全的单向散列函数,它接收任意大小的数据,输出固定长度的散列值。

实现 MD5 和 SHA 加密的示例代码如下:

```
/**
 * 将信息按指定的算法加密
 * @param str: 要加密的字符串信息
 * @param digestType: 加密类型, 选择 MD5 或 SHA-1
 * @return: 加密之后的内容
 */
public String digestString(String str,String digestType) {
    MessageDigest md;
    String message = "";
    try {
        md = MessageDigest.getInstance(digestType);
        md.update(str.getBytes());
        message = byteToHex(md.digest());
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return message;
}

/**
 * 将字节数组变成十六进制的字符串
```

```
* @param bytes: 字节数组
* @return: 十六进制的字符串
*/
private String byteToHex(byte[] bytes) {
    StringBuffer sb = new StringBuffer();
    for(int i=0;i<bytes.length;i++) {
        int num = 0xFF & bytes[i];
        if(num < 0x10) {
            sb.append("0"+Integer.toHexString(num));
        } else {
            sb.append(Integer.toHexString(num));
        }
    }
    return sb.toString();
}
```

在上述代码中, digestString()方法在使用时可以指定“MD5”或“SHA-1”算法。使用的代码如下:

```
public static void main(String args[]) {
    MessDigest md1 = new MessDigest();
    System.out.println(md1.digestString("liangjq","MD5"));
    System.out.println(md1.digestString("liangjq","SHA-1"));
}
```

疑难点评

加密功能使用的非常普遍,可以提高信息的安全性,加密算法也有很多,比较常用的有MD5和SHA-1。在Java中通过java.security.MessageDigest类可以实现MD5和SHA-1加密算法。



第9章

Java 数据库操作

随着近年来计算机网络和信息技术的快速发展,人们对信息的需求越来越多也越来越具体,这种需求不仅涵盖了要尽可能地实现信息共享,还包括及时、准确、方便地传输、存储和处理数据。

本章主要介绍一些 Java 数据库访问技术,内容主要涉及 JDBC 基本应用、事务处理、图片处理、数据库连接池以及常见疑难问题功能的实现等几个方面。通过本章的学习,读者可以使用 Java 对数据库实现各种常用操作,并快速地处理数据库操作时遇到的各种问题。

FAQ9.01 什么是 JDBC? 有什么作用?

📖 难度系数: ★★★

📖 问题频率: 95%

核心解答

JDBC 全称为 Java DataBase Connectivity standard,它是 Java 访问数据库的应用程序接口(API),通过这套 API 可以访问各种类型的关系数据库。例如 Oracle、MySQL、SQL Server 等。

JDBC 为开发人员提供了一套标准的 API,都是由 Java 语言编写的类和接口,可用于连接数据库和执行 SQL 语句。JDBC 也是 Java 核心类库的一部分,位于 java.sql 包下。

JDBC 包含的核心 API 主要有以下几个。

- ☐ DriverManager: 管理一组 JDBC 驱动程序的基本服务。
- ☐ Connection: Java 程序与特定数据库的连接。
- ☐ Statement: 用于执行静态 SQL 语句并返回它所生成结果的对象。
- ☐ PreparedStatement: 表示预编译的 SQL 语句的对象。
- ☐ CallableStatement: 用于执行 SQL 存储过程的接口。
- ☐ ResultSet: 表示数据库查询的结果集。

疑难点评

JDBC 大部分 API 都采用接口形式设计,接口的实现类都交给数据库提供商完成,这样使

得数据库访问程序不依赖于特定类型的数据库，开发者可以利用统一的方式操作各种数据库。

知识链接

FAQ9.02 Java 与数据库的连接方式有哪些？

FAQ9.02 Java 与数据库的连接方式有哪些？

📖 难度系数：★★★

📖 问题频率：90%

核心解答

要使用 JDBC 连接某一特定的数据库，必须有和数据库相对应的驱动程序，驱动程序往往是由数据库的厂商提供的，是连接 JDBC API 和数据库之间的桥梁。驱动程序是适应某种特定数据库，实现 JDBC 接口的一组实现类。

在 Java 访问数据库时，Java 程序首先使用 DriverManager 类载入指定的驱动程序，然后使用 JDBC API 与 DriverManager 类交互，完成数据库的增加、删除、修改和查询操作。

JDBC 驱动程序一共有 4 种类型，可以实现对数据库执行不同方式的连接。具体有以下几种类型。

❑ 类型 1: JDBC-ODBC bridge plus ODBC driver

JDBC-ODBC 桥驱动，该方式是将 JDBC 调用转换为 ODBC 的调用。此方式适合用于快速的原型系统，没有提供 JDBC 驱动的数据库，例如 Access 数据库、TXT 等文件，但对于大量数据存取的应用场合不适合。在连接时，客户端需要先配置 ODBC 数据源与其他数据库的连接，并且必须已安装 ODBC 驱动。

❑ 类型 2: Native-API partly-Java driver

本地 API 驱动，该方式是将 JDBC 调用转换为对数据库客户端 API 的调用。此方式比 JDBC-ODBC 桥方式效率高，但是客户端需要安装数据库厂商提供的客户端和代码库。

❑ 类型 3: Pure Java Driver for Database Middleware

网络协议驱动，该方式是先将 JDBC 调用转换为 DBMS-independent 网络协议，然后由服务器端的中间件转换为具体数据库服务器可以接收的网络协议。此方式是基于中间件服务器的，因此不需要客户端加载任何驱动和代码库，但是需要配置中间件服务器。由于多了一个中间层传递数据，因此执行效率不是最好的。

❑ 类型 4: Direct-to-Database Pure Java Driver

本地协议驱动，该方式是将 JDBC 调用直接转换为具体数据库服务器可以接收的网络协议。此方式的执行效率非常高，不需要在客户端装载任何的软件或驱动，这种驱动程序可以被动态下载，但是对于不同的数据库需要下载不同的驱动程序。

疑难点评

对于以上 4 种连接类型, 类型 1 由于执行效率不高, 因此更适合作为开发应用时的一种过渡方案, 大量数据操作的应用应考虑其他 3 种。

在 Intranet 方面的应用可以考虑类型 2, 但是类型 3 和 4 比 2 效率有着明显优势, 而且目前开发的趋势是使用纯 Java 实现, 因此类型 3 和 4 也可以作为考虑对象。

在 Internet 方面的应用只能考虑类型 3 和 4, 类型 3 适合于那些需要同时连接多个不同种类的数据库, 并且对并连接要求高的应用。类型 4 则适合那些单一数据库的工作组应用。

知识链接

FAQ9.03 如何连接各种类型的数据库?

FAQ9.03 如何连接各种类型的数据库?

📖 难度系数: ★★★

📖 问题频率: 95%

核心解答

目前, Java 访问数据库使用最多的连接方式是类型 4 本地协议驱动方式。在使用该方式连接数据库之前, 需要先下载驱动包, 然后将驱动包添加到工程的 classpath 中, 后续可以使用 JDBC API 实现数据库的访问操作。

(1) JDBC API 使用介绍

JDBC API 的使用方法如下所示。

❑ 注册 JDBC 驱动程序

驱动程序注册的方法, 有以下 3 种方式。

Class.forName(String drivename);	//方式 1, 最常用方式
DriverManager.registerDriver(Driver driver);	//方式 2
new com.mysql.jdbc.Driver();	//方式 3, 创建驱动类的对象

❑ 建立与 SQL 数据库的连接

利用 DriverManager 的 getConnection() 方法获取 Connection 连接, getConnection() 方法的定义如下:

Connection getConnection(String url)	//url 表示数据库地址字符串
Connection getConnection(String url,String user,String pwd)	
Connection getConnection(String url,Properties info)	

❑ 执行 SQL 语句

利用 Connection 的 createStatement() 方法获取 Statement 对象, Statement 可以执行 SQL 语句, 得到 SQL 查询结果。createStatement() 方法的定义如下:

Statement createStatement();

□ 获得结果集

Statement 执行 SQL 语句的方法如下:

ResultSet executeQuery(String sql)

//执行 select 语句

int executeUpdate(String sql)

//执行更新语句, 如 insert, delete, update

□ 取出查询结果

利用 ResultSet 可以获取查询结果的内容, 主要方法如下:

boolean next()

//没有行时返回 false

String getString(String columnName)

//返回列名对应的值

(2) 数据库访问示例

以 Oracle 数据库为例, 连接数据库执行 SQL 语句的代码如下:

```
import java.sql.*;
public class JdbcExample{
    public static void main(String args[]){
        String serverName = "localhost";
        try{
            //注册驱动
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String url="jdbc:oracle:thin:@"+serverName+":1521:ora9i";
            //获取连接
            Connection
            conn=DriverManager.getConnection(url,"scott","tiger");
            //获取 SQL 执行器
            Statement stmt=conn.createStatement();
            //执行查询的 SQL 语句
            ResultSet rs=stmt.executeQuery("select * from dept");
            //遍历结果集
            while(rs.next()){
                //获取结果中第 1 个字段的值
                System.out.print("DeptNo: "+rs.getInt(1));
                //获取结果中第 2 个字段的值
                System.out.print("\tDeptName: "+rs.getString(2));
                //获取结果中第 3 个字段的值
                System.out.println("\tLOC: "+rs.getString(3));
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

(3) 各种类型数据库的连接

各种不同数据库的连接方式如下所示。

□ Oracle 数据库 (thin 模式)

Oracle 驱动包 classes12.jar, 下载地址: <http://www.oracle.com/index.html>, 也可以从 Oracle 安装目录下获取。

```
Class.forName("oracle.jdbc.driver.OracleDriver");
String url="jdbc:oracle:thin:@localhost:1521:orcl"; //orcl 为数据库的 SID
String user="test";           //用户名
String password="test";       //密码
Connection conn= DriverManager.getConnection(url,user,password);
```

❑ DB2 数据库

DB2 驱动包 db2java.zip, 可以从 DB2 安装目录下获取, 并更改扩展名为 db2java.jar。

```
Class.forName("com.ibm.db2.jdbc.app.DB2Driver ");
String url="jdbc:db2://localhost:5000/sample"; //sample 为数据库名
String user="admin";
String password="";
Connection conn= DriverManager.getConnection(url,user,password);
```

❑ SQLServer 7.0/2000 数据库

SQLServer 驱动包有 3 个, 分别为 msbase.jar、mssqlserver.jar 和 msutil.jar, 下载地址: <http://www.microsoft.com>。

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
String url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=mydb"; //mydb 为数据库
String user="sa";
String password="";
Connection conn= DriverManager.getConnection(url,user,password);
```

或者使用 <http://jtds.sourceforge.net/> 地址下载驱动, 连接代码如下:

```
Class.forName( "net.sourceforge.jtds.jdbc.Driver" );
Connection cn = DriverManager.getConnection( "jdbc:jtds:sqlserver:// localhost:1433/master", sUsr, sPwd );
```

❑ Sybase 数据库

Sybase 驱动包 jconn2.jar, 下载地址: <http://jtds.sourceforge.net>。

```
Class.forName("com.sybase.jdbc.SybDriver");
String url = " jdbc:sybase:Tds:localhost:5007/myDB"; //myDB 为数据库名
Properties sysProps = System.getProperties();
SysProps.put("user","userid");
SysProps.put("password","user_password");
Connection conn= DriverManager.getConnection(url, SysProps);
```

❑ Informix 数据库

MySQL 驱动包 ifxjdbc.jar, 从 <http://www.inforbus.com/service/download/> 地址下载 driver2.21.jc5.rar, 驱动包在该压缩包中。

```
Class.forName("com.informix.jdbc.IfxDriver");
String url =
"jdbc:informix-sqli://123.45.67.89:1533/myDB:INFORMIXSERVER=myserver;
user=testuser;password=testpassword";
//端口号 1533
//数据库名为 myDB
//INFORMIXSERVER 为 myserver
Connection conn= DriverManager.getConnection(url);
```

❑ MySQL 数据库

MySQL 驱动包 mysql-connector-java-2.0.14-bin.jar, 下载地址: <http://www.mysql.com>。

```
Class.forName("com.mysql.jdbc.Driver");
String url = "jdbc:mysql://localhost:3306/myDB"; //myDB 为数据库名
String user = "myuser";
String password = "mypassword";
Connection conn = DriverManager.getConnection(url, user, password);
```

❑ PostgreSQL 数据库

PostgreSQL 驱动包 pgjdbc2.jar, 下载地址: <http://www.de.postgresql.org>。

```
Class.forName("org.postgresql.Driver");
String url = "jdbc:postgresql://localhost/myDB" //myDB 为数据库名
String user = "myuser";
String password = "mypassword";
Connection conn = DriverManager.getConnection(url, user, password);
```

❑ Access 数据库

Access 数据库利用 JDBC-ODBC 桥方式连接, 该驱动类 JDK 自带, 不需要下载。

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:testText", "", ""); // testText 为 ODBC 数据源名
```

❑ Postgresql 数据库

驱动包的下载地址为: <http://jdbc.postgresql.org/download.html>。

```
Class.forName("org.postgresql.Driver");
Connection conn =
DriverManager.getConnection("jdbc:postgresql://host:port/database", "user", "password");
```

❑ IBM AS400 数据库

从安装目录下寻找驱动包 jt400.zip, 并更改扩展名成为 jt400.jar。

```
Class.forName("com.ibm.as400.access.AS400JDBCConnection");
Connection conn =
DriverManager.getConnection("jdbc:as400://IP", "user", "password");
```

❑ SAP DB 数据库

```
Class.forName("com.sap.dbtech.jdbc.DriverSapDB");
Connection connection = DriverManager.getConnection("jdbc:sapdb://" + host + "/" + database_name, user_name, password)
```

❑ InterBase 数据库

```
Class.forName("interbase.interclient.Driver");
String url = "jdbc:interbase://localhost/e:/testbed/database/employee.gdb";
Connection conn = DriverManager.getConnection(url, "sysdba", "masterkey");
```

疑难点评

JDBC 在对各种不同数据库进行连接时, 只需要使用不同的驱动包, 传递不同的参数。这样使得开发者在对各种类型数据库操作时显得非常方便。

知识链接

FAQ9.04 如何实现对数据库数据的查询?

FAQ9.05 如何实现对数据库数据的增加、删除和修改?

FAQ9.04 如何实现对数据库数据的查询?

📖 难度系数: ★★★

📖 问题频率: 95%

核心解答

Statement 可用于执行 SQL 语句,不同类型的 SQL 语句需要使用不同的方法,具体如下:

```
ResultSet executeQuery(String sql)    //执行 select 语句
int executeUpdate(String sql)         //执行 insert、delete 和 update 语句
boolean execute(String sql)           //执行 create 和 drop 等语句
```

上述的 executeQuery()方法返回一个 ResultSet 类型对象,利用该对象可获取查询结果。在遍历 ResultSet 对象结果时,可使用 next()方法判断是否还有下一行记录,可使用 getString()等方法获取当前记录的字段值。

以 Oracle 数据库为例,查询数据库,遍历查询结果的示例如下:

```
import java.sql.*;
public class Test{
    public static void main(String args[]){
        String serverName = "localhost";
        try{
            //注册驱动
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String
url="jdbc:oracle:thin:@"+serverName+":1521:ora9i";//ora9i 为数据库的 SID 名
            //获取连接
            Connection
conn=DriverManager.getConnection(url,"scott","tiger");
            //获取 SQL 执行器
            Statement stmt=conn.createStatement();
            //执行查询的 SQL 语句
            ResultSet rs=stmt.executeQuery("select * from dept");
            //遍历结果集中的每一行记录
            while(rs.next()){
                //获取结果中 deptno 字段的值
                System.out.print("DeptNo: "+rs.getInt("deptno"));
                //获取结果中 dname 字段的值
                System.out.print("\tDeptName: "+rs.getString("dname"));
                //获取结果中 loc 字段的值
                System.out.println("\tLOC: "+rs.getString("loc"));
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

注意：在执行上述代码前，需要获取 Oracle 的 class12.jar 或 ojdbc14.jar 驱动包，将其引入到工程的 classpath 中，即将驱动包添加到工程的 WEB-INF、lib 目录下。

疑难点评

不同类型的数据库在获取 Connection 连接过程中参数会有些差异，但后续执行 SQL 的过程都是相同的。

知识链接

FAQ9.05 如何实现对数据库数据的增加、删除和修改？

FAQ9.05 如何实现对数据库数据的增加、删除和修改？

📖 难度系数：★★★

📖 问题频率：95%

核心解答

数据库的添加、删除和修改功能需要使用 insert、delete 和 update 3 种 SQL 语句。Statement 执行 insert、delete 和 update SQL 语句的方法为 executeUpdate()。

□ 添加操作

insert 语句可实现向数据库添加记录的功能，insert 语句的语法格式如下：

```
insert into 表名 [(字段 1, 字段 2, 字段 3...)] values (字段 1 值, 字段 2 值, 字段 3 值...)
```

在上述格式中，[] 部分表示可以选择。不加 [] 部分时，values 部分指定的字段值必须和数据表的个数、类型和顺序保持一致；加 [] 部分时，可以在数据库写入时指定部分字段值，values 部分指定的字段值必须和 [] 部分的个数、类型和顺序保持一致。示例如下：

//添加记录时，为部分字段指定值

```
insert into emp (empno, ename, sal) values (7801, 'tom', 3000)
```

//添加记录时，为所有字段指定值

```
insert into emp values (7801, 'tom', 3000, '男', '大本', '软件工程师')
```

实现记录添加功能的代码如下：

```
public static void add(){
    Connection conn=null;
    Statement stmt=null;
    try{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url="jdbc:oracle:thin:@localhost:1521:ora9i";
        conn=DriverManager.getConnection(url,"scott","tiger");
        stmt=conn.createStatement();
        String sql="insert into emp values (7801, ' tom' ,3000, ' 男' , ' 大本' , ' 软件工程师' )";
        stmt.executeUpdate(sql);
        conn.close();
    }
}
```

```
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        try {  
            stmt.close();  
        } catch (SQLException e) {  
        }  
        try {  
            conn.close();  
        } catch (SQLException e) {  
        }  
    }  
}
```

□ 删除操作

delete 语句可实现数据库记录删除的功能, delete 语句的语法格式如下:

```
delete from 表名 [where 条件]
```

在上述格式中, 如果不加 where 条件部分, 默认删除表中所有记录; 如果使用 where 条件, 可删除指定条件的记录, 如果没有符合条件的记录也不会发生异常。示例如下:

```
//删除 emp 表中所有记录  
delete from emp  
//删除 emp 表中 empno 字段为 7801 的记录  
delete from emp where empno=7801
```

实现记录删除功能的代码如下:

```
public static void delete() {  
    Connection conn=null;  
    Statement stmt=null;  
    try {  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
        String url="jdbc:oracle:thin:@localhost:1521:ora9i";  
        conn=DriverManager.getConnection(url,"scott","tiger");  
        stmt=conn.createStatement();  
        String sql="delete from emp where empno=7801";  
        stmt.executeUpdate(sql);  
        conn.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        try {  
            stmt.close();  
        } catch (SQLException e) {  
        }  
        try {  
            conn.close();  
        } catch (SQLException e) {  
        }  
    }  
}
```


□ 更新操作

update 语句可实现数据库记录更新的功能, update 语句的语法格式如下:

```
update 表名 set 字段 1=值 1,字段 2=值 2,..... [where 条件]
```

在上述格式中,如果不加 where 条件部分,默认更新表中所有记录;如果使用 where 条件,可更新指定条件的记录,如果没有符合条件的记录也不会发生异常。示例如下:

```
//将 emp 表中所有记录的 sal 字段值更新成 10000
```

```
update emp set sal=10000
```

```
//将 emp 表中 empno 为 7801 记录的 sal 字段值更新成 10 000
```

```
update emp set sal=10000 where empno=7801
```

实现记录更新功能的代码如下:

```
public static void update(){
    Connection conn=null;
    Statement stmt=null;
    try{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url="jdbc:oracle:thin:@localhost:1521:ora9i";
        conn=DriverManager.getConnection(url,"scott","tiger");
        stmt=conn.createStatement();
        String sql = "update emp set ename='rose',sal=5000 where empno=7801";
        stmt.executeUpdate(sql);
        conn.close();
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        try {
            stmt.close();
        } catch (SQLException e) {
        }
        try {
            conn.close();
        } catch (SQLException e) {
        }
    }
}
```

疑难点评

使用 JDBC 实现数据库添加、修改和删除操作时,事务提交功能默认启用,每一次操作都是一个独立事务。此外 SQL 语句是不区分大小写的,因此 SQL 语句中的关键字、表名和字段名,大小写都无所谓,但建议用大小写显示区分关键字和表名、字段名。

知识链接

FAQ9.04 如何实现对数据库数据的查询?

FAQ9.06 如何使用 PreparedStatement 对数据库操作?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

PreparedStatement 是 Statement 接口的子接口, 它也可以完成 Statement 所能实现的功能。PreparedStatement 表示预编译的 SQL 语句的对象, SQL 语句被预编译并且存储在 PreparedStatement 对象中, 然后可以使用此对象高效地多次执行该语句。

通过 Connection 的 prepareStatement() 方法可以获取一个 PreparedStatement 对象, prepareStatement() 方法在使用时必须传入一个待封装的 SQL 语句。使用 PreparedStatement 实现添加功能的代码如下:

```
public class Test{
    public static void main(String[] args){
        try{
            //注册 Oracle 驱动
            Class.forName("oracle.jdbc.driver.OracleDriver");
            //定义连接字符串
            String url = "jdbc:oracle:thin:@localhost:1521:oracle9i";
            //获取数据库连接
            Connection
conn=DriverManager.getConnection(url,"scott","tiger");
            //获取 PreparedStatement 对象, 封装一个添加操作
            PreparedStatement pstmt=conn.prepareStatement("insert into dept values(?,?,?)");
            //为第 1 个? 设置一个 int 类型的参数值
            pstmt.setInt(1,17);
            //为第 2 个? 设置一个 String 类型的参数值
            pstmt.setString(2,"Dirk");
            //为第 3 个? 设置一个 String 类型的参数值
            pstmt.setString(3,"10");
            //执行添加操作
            pstmt.executeUpdate();
            pstmt.close();
            conn.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

在上述代码中, PreparedStatement 对象封装了一个 insert 操作, insert 语句中不确定的字段值可以使用“?”代替, 后续可以为其指定不同参数值, 并多次执行记录添加的功能。

在使用 setter() 方法为“?”设置参数值时, 参数值必须与其所代表的字段类型兼容。例如,

如果字段类型为 INTEGER, 那么应该使用 `setInt()` 方法; 如果字段类型为 VARCHAR 或 CHAR, 那么应该使用 `setString()` 方法等。

疑难点评

在实现更新和删除操作时, 需要在获取 `PreparedStatement` 对象时为其指定 `update` 和 `delete` 语句, 代码过程与添加示例相似。

知识链接

FAQ9.07 Statement 和 PreparedStatement 有什么区别?

FAQ9.07 Statement 和 PreparedStatement 有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

`Statement` 和 `PreparedStatement` 都可以执行 SQL 语句实现对数据表的操作。

`Statement` 用于执行静态 SQL 语句并返回它所生成结果的对象。`Statement` 在执行 SQL 语句时, 必须指定一个事先准备好的 SQL 语句。`Statement` 的使用方法如下:

```
String sql = "insert into tb_name (col1,col2,col2,col4) values  
('"+var1+"','"+var2+"','"+var3+"','"+var4+"')";  
Statement stmt = conn.createStatement();  
stmt.executeUpdate(sql);
```

`PreparedStatement` 表示预编译的 SQL 语句的对象, SQL 语句被预编译并存储在对象中。被封装的 SQL 语句代表某一类操作, SQL 语句中允许包含动态参数 “?”, 在执行时可以为 “?” 动态设置参数值。`PreparedStatement` 的使用方法如下:

```
String sql = "insert into tb_name (col1,col2,col2,col4) values (?, ?, ?, ?)";  
PreparedStatement perstmt = conn.prepareStatement(sql);  
perstmt.setString(1,var1);  
perstmt.setString(2,var2);  
perstmt.setString(3,var3);  
perstmt.setString(4,var4);  
perstmt.executeUpdate();
```

在使用 `PreparedStatement` 对象执行 SQL 命令时, SQL 命令被数据库进行解析和编译, 然后被放到命令缓冲区。然后, 每当执行同一个 `PreparedStatement` 对象时, 它就会被再解析一次, 但不会被再次编译。在缓冲区中可以发现预编译的命令, 并且可以重新使用。在企业级应用软件中, 经常会重复执行相同的 SQL 操作, 使用 `PreparedStatement` 对象带来的编译次数的减少能够提高数据库的总体性能。

在多次执行相同 SQL 操作的情况下,可以选用 PreparedStatement;在仅执行一次 SQL 操作的情况下,两者都适合。

疑难点评

通常,在使用 Statement 之前,需要利用字符串拼写 SQL 语句,如果遇到参数值含有 “'”、“” 和 “,” 等特殊字符时,拼写出的是一个非法的 SQL 语句。但由于 PreparedStatement 是将参数值和 SQL 语句分开的,因此不会出现问题,即使遇到特殊字符也能正确执行。这也是有些开发者无论是在 SQL 执行一次还是多次的情况,都使用 PreparedStatement 的原因。

知识链接

FAQ9.06 如何使用 PreparedStatement 对数据库操作?

FAQ9.08 如何调用数据库中的存储过程?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

存储过程是存储在数据库服务器中的一组 SQL 操作的单元。

CallableStatement 接口继承自 PreparedStatement,可用于执行 SQL 存储过程。JDBC API 提供了一个存储过程 SQL 转义语法,该语法允许对所有数据库使用标准方式调用存储过程。此转义语法有一个包含结果参数的形式和一个不包含结果参数的形式。语法格式如下:

```
{?= call <procedure-name>[<arg1>,<arg2>,...]}  
{call <procedure-name>[<arg1>,<arg2>,...]}
```

如果使用结果参数,则必须将其注册为 OUT 型参数,其他参数为 IN 型,用于输入。IN 参数值使用从 PreparedStatement 中继承的 setter() 方法设置。在执行存储过程之前,必须注册所有 OUT 参数的类型,在执行后通过此类型提供的 getter() 方法获取结果。

以 Oracle 为例,调用各种类型的存储过程的示例如下所示。

❑ 调用无参的存储过程

存储过程的代码如下:

```
create or replace procedure MyProcedure1  
as  
begin  
    insert into dept1 values(1,'computer');  
end;
```

调用代码如下:

```

try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection conn =
    DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:oracle9i","scott","tiger");
    CallableStatement stmt = conn.prepareCall("{call MyProcedure1()}");

    stmt.execute();
    stmt.close();
    conn.close();
} catch (Exception e) {
    e.printStackTrace();
}

```

在上述代码中,通过“{call MyProcedure1()}”语句调用了名为 MyProcedure1 的无参的存储过程。需要注意的是即使存储过程无参,在过程名后面也要加()。

□ 调用具有 IN 参数的存储过程

存储过程的代码如下:

```

create or replace procedure MyProcedure1(sn in number,name in char)
as
begin
    insert into dept1 values(sn,name);
end;

```

调用代码如下:

```

try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection conn=

    DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ora9i","scott","tiger");
    CallableStatement stmt=conn.prepareCall("{call MyProcedure1(?,?)}");
    stmt.setInt(1,2);
    stmt.setString(2,"math");
    stmt.execute();
    stmt.close();
    conn.close();
} catch (Exception e){
    e.printStackTrace();
}

```

在上述代码中,注意 setter()方法传入的值,类型需要和存储过程的定义一致。

□ 调用具有 IN 和 OUT 参数的存储过程

存储过程的代码如下:

```

create or replace procedure MyProcedure1 (vlength in number,vwidth in number,alength out number)
as
begin
    alength := 2 * (vlength + vwidth);
end;

```

调用代码如下:

```

try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection conn=
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ora9i","scott","tiger");
    CallableStatement stmt=conn.prepareCall("{call MyProcedure1(?,?,?)}");
    int i = 0;
    stmt.setInt(1,77);
    stmt.setInt(2,30);
    stmt.registerOutParameter(3,Types.INTEGER);
    stmt.execute();
    i = stmt.getInt(3);
    System.out.println(i);
    stmt.close();
    conn.close();
}catch(Exception e){
    e.printStackTrace();
}

```

□ 调用返回结果集的存储过程

存储过程的代码如下:

```

create or replace package PKG_HOTLINE
is
    type HotlineCursorType is REF CURSOR;
    procedure getAllHotline (rs out HotlineCursorType);
end;

create or replace package body PKG_HOTLINE
is
    procedure getAllHotline (rs out HotlineCursorType)
    is
    begin
        open rs for select * from hotline;
    end;
end;

```

调用代码如下:

```

try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection conn =
        DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:xd",
            "scott",
            "tiger");
    String sql = "{call PKG_HOTLINE.getAllHotline(?)}";
    CallableStatement stmt = conn.prepareCall(sql);
    stmt.registerOutParameter(1,OracleTypes.CURSOR);
    stmt.execute();
    ResultSet rs = ((OracleCallableStatement)stmt).getCursor(1);
    while(rs.next()){
        String country = rs.getString(1);
    }
}

```



```
String pno = rs.getString(2);
System.out.println("country:"+country+"|pno:"+pno);
}
stmt.close();
conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

疑难点评

Oracle、MySQL 和 SQL Server 等数据库都支持存储过程，可以将一组 SQL 操作写成一个存储过程，然后通过 Java 程序发送参数值并调用存储过程。利用存储过程可以减少 SQL 的传递，减少与数据库的交互次数，从而提高程序效率。

知识链接

FAQ9.06 如何使用 PreparedStatement 对数据库操作？

FAQ9.09 如何通过 JDBC-ODBC 桥访问 Access 数据库？

📖 难度系数：★★★

📖 问题频率：70%

核心解答

在开发一些小型系统的时候，限于系统需求或服务器环境的原因需要采用小型数据库 Access。

ODBC（Open Database Connectivity，开放数据库连接）是微软公司开放服务结构中有关数据库的一个组成部分，它建立了一组规范，并提供了一组对数据库访问的标准 API。通过 ODBC 可以访问各种关系数据库。

Java 在访问 Access 数据库时，一般都采用 JDBC-ODBC 桥方式，即通过 JDBC-ODBC 驱动访问 ODBC 数据源，然后由 ODBC 数据源访问数据库。连接 Access 数据库的操作步骤如下。

（1）创建 ODBC 数据源

首先选择【开始】→【设置】→【控制面板】菜单，进入“控制面板”界面。然后选择【管理工具】→【数据源（ODBC）】选项，打开“ODBC 数据源管理器”窗口，如图 9-1 所示。

在图 9-1 所示的“ODBC 数据源管理器”窗口中，单击“添加”按钮，打开“创建数据源”窗口，如图 9-2 所示。

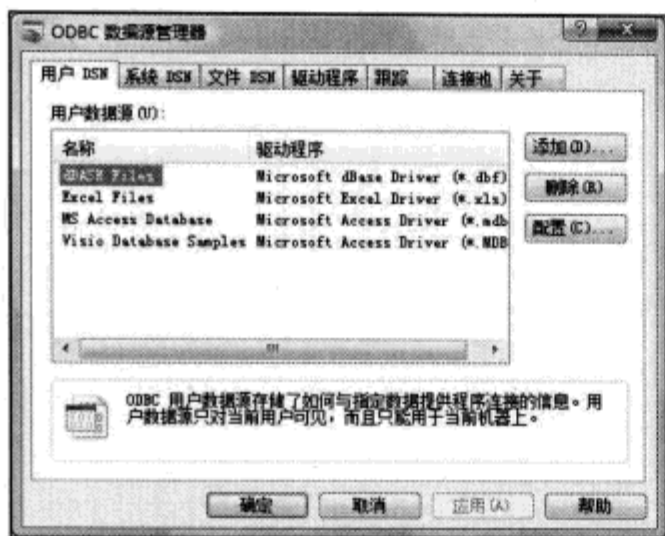


图 9-1 “ODBC 数据源管理器”窗口

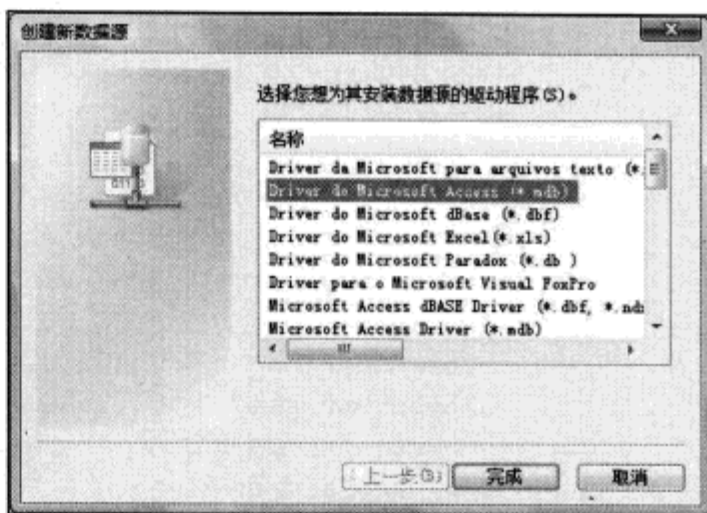


图 9-2 “创建数据源”窗口

在图 9-2 所示的“创建数据源”窗口中,选择 Access 数据库的驱动,单击“完成”按钮,会弹出“ODBC Microsoft Access 安装”窗口,如图 9-3 所示。

在图 9-3 所示的“ODBC Microsoft Access 安装”窗口中的“数据源名”输入框输入“myaccess”,该名称可自定义,在使用 Java 程序连接时需要与其一致。单击“选择”按钮,打开“选择数据库”窗口,如图 9-4 所示。



图 9-3 “ODBC Microsoft Access 安装”窗口

在图 9-4 所示的“选择数据库”窗口中,选择 Access 的数据库文件,然后单击“确定”退出,返回“ODBC 数据源管理器”窗口,完成配置,如图 9-5 所示。

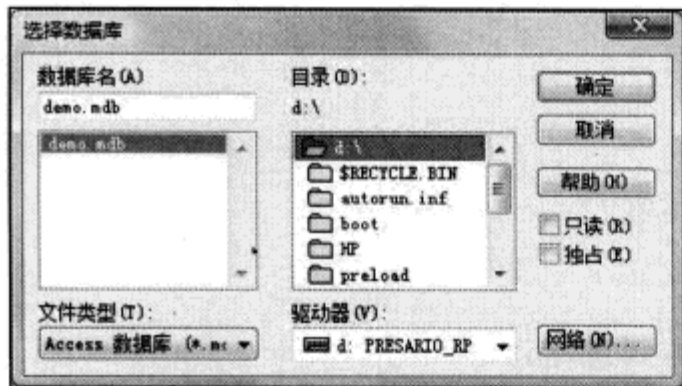


图 9-4 “选择数据库”窗口

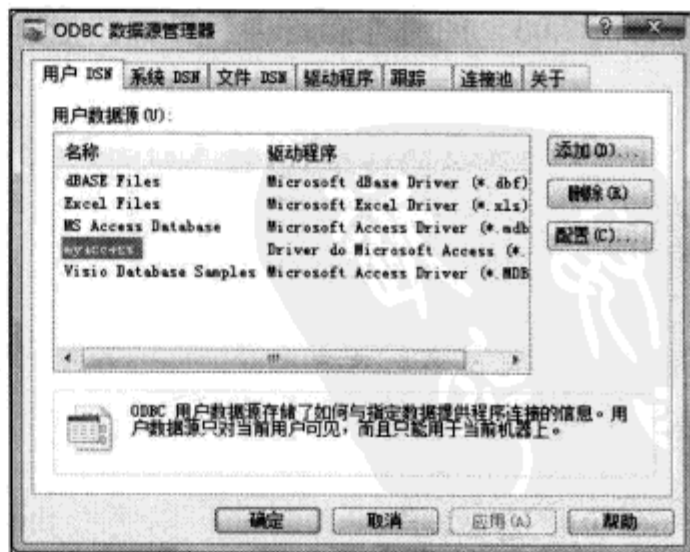


图 9-5 “ODBC 数据源管理器”窗口

(2) 编写访问程序

利用 JDBC-ODBC 桥方式连接 Access 的示例代码如下:

```
public static void main(String[] args) {  
    try {  
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
        //myaccess 为数据源名  
        Connection con =  
        DriverManager.getConnection("jdbc:odbc:myaccess", "", "");  
        Statement stmt=con.createStatement();  
        ResultSet rs=stmt.executeQuery("select * from person");  
        while(rs.next()){  
            System.out.print("\t 编号: " + rs.getString(1));  
            System.out.print("\t 姓名: " + rs.getString(2));  
            System.out.print("\t 性别: " + rs.getString(3));  
            System.out.println();  
        }  
        con.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

疑难点评

Access 数据源都没有提供驱动包，因此在访问时只能通过 JDBC-ODBC 桥方法连接。TXT、CSV 和 Excel 文件也可以充当数据源，其访问过程与 Access 相同，只需要在“创建数据源”窗口选择合适的驱动即可。如果数据源提供了驱动包，建议采用类型 4 本地协议驱动方式访问。

知识链接

FAQ9.02 Java 与数据库的连接方式有哪些？

FAQ9.03 如何连接各种类型的数据库？

FAQ9.10 连接 Oracle 数据库时 thin 和 oci 方式有什么区别？

📖 难度系数：★★★

📖 问题频率：70%

核心解答

在访问 Oracle 数据库时，可以使用 thin 和 oci 方式连接。

thin 方式的连接字符串格式如下：

`jdbc:oracle:thin:@<主机名或 IP>:1521:<数据库 SID 名>`

示例如下：

`jdbc:oracle:thin:@127.0.0.1:1521:test`

oci 方式的连接字符串格式如下:

```
java:oracle:oci@<本地服务名>
```

示例如下:

```
java:oracle:oci@test
```

thin 和 oci 属于两种不同的连接类型, thin 属于 Direct-to-Database Pure Java Driver 类型, 只要有数据库驱动包就可以直接通过网络端口访问数据库; 而 oci 是 Oracle Call Interface 的缩写, 属于 Native-API partly-Java driver 类型, 需要在客户端安装 Oracle 的客户端软件, 并注册一个本地服务名。理论上 oci 性能要好于 thin。

疑难点评

Oracle 提供了两套 Java 访问 Oracle 数据库的方法。thin 就是纯粹用 Java 完成访问数据库的所有方法, 优点是不用安装客户端; oci 就是使用 Java 来调用本机的 Oracle 客户端, 然后再访问数据库, 优点是速度快, 但是需要安装和配置数据库。

知识链接

FAQ9.03 如何连接各种类型的数据库?

FAQ9.11 如何判断 ResultSet 结果集为空?

📖 难度系数: ★★★

📖 问题频率: 85%

核心解答

ResultSet 表示 select 语句的查询结果集。ResultSet 对象具有指向其当前数据行的指针, 最初, 指针被置于第 1 行记录之前, 通过 next() 方法可以将指针移动到下一行记录。next() 方法在 ResultSet 对象没有下一行记录时返回 false, 因此可以在 while 循环中使用它来遍历结果集, 也可以利用该方法判断结果集是否为空。

示例代码如下:

```
try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    String url="jdbc:oracle:thin:@localhost:1521:ora9i";
    Connection conn=DriverManager.getConnection(url,"scott","tiger");
    Statement stmt=conn.createStatement();
    ResultSet rs=stmt.executeQuery("select * from dept");
    if(rs.next()){
        System.out.print("结果集有记录返回");
    }else{
        System.out.print("结果集为空");
    }
}
```

```
}catch(Exception e){  
    e.printStackTrace();  
}
```

疑难点评

默认的 `ResultSet` 对象不可更新，仅有一个向前移动的指针。因此，只能遍历它一次，并且只能按从第 1 行到最后一行的顺序进行。如果需要 `ResultSet` 指针具有移动和可更新的特性，可以在使用 `createStatement()` 方法获取 `Statement` 对象时指定一些参数。

知识链接

FAQ9.12 如何获取 `ResultSet` 中含有的记录数量？

FAQ9.13 如何获取 `ResultSet` 中 $n \sim m$ 位置区间的记录？

FAQ9.12 如何获取 `ResultSet` 中含有的记录数量？

📖 难度系数：★★★★

📖 问题频率：85%

核心解答

在已获取 `ResultSet` 结果集的情况下，可以使用该对象的 `last()` 和 `getRow()` 方法取得记录数量。`last()` 方法用于将 `ResultSet` 指针指向到最后一行记录，`getRow()` 方法用于返回当前指针所在的位置。

`ResultSet` 默认情况下，只能使用 `next()` 方法向前逐行移动指针，不支持 `last()`、`first()` 以及 `absolute()` 等。如果要使用 `last()` 和 `absolute()` 方法，由 `Connection` 生成 `Statement` 时需要指定参数，格式如下：

```
Statement stmt=conn. createStatement(游标类型, 记录更新权限);
```

游标类型参数有以下几种。

- ☐ `ResultSet.TYPE_FORWARD_ONLY`：指针只可以向前移动。
- ☐ `ResultSet.TYPE_SCROLL_INSENSITIVE`：指针可滚动，但是不受其他用户对数据库更改的影响。
- ☐ `ResultSet.TYPE_SCROLL_SENSITIVE`：指针可滚动，当其他用户更改数据库时这个记录也会改变。

记录更新权限参数有以下几种。

- ☐ `ResultSet.CONCUR_READ_ONLY`：只读。
- ☐ `ResultSet.CONCUR_UPDATABLE`：可更新。

`createStatement()` 缺省参数等价于 `createStatement (ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY)`。

获取 `ResultSet` 结果集记录数量的代码如下：

```
public static void main(String[] args) {
    int count = 0;
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url="jdbc:oracle:thin:@localhost:1521:ora9i";
        Connection
conn=DriverManager.getConnection(url,"scott","tiger");
        Statement stmt=conn.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.TYPE_FORWARD_ONLY);
        ResultSet rs=stmt.executeQuery("select * from dept");
        if(rs.last()){
            count = rs.getRow();
        }
        System.out.print("记录数量: "+count);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

使用上述方法,在数据量很大时会出现内存溢出异常,因此不推荐使用。可以使用 SQL 统计函数获取符合查询条件的记录数量。示例代码如下:

```
public static void main(String[] args) {
    int count = 0;
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url="jdbc:oracle:thin:@localhost:1521:ora9i";
        Connection
conn=DriverManager.getConnection(url,"scott","tiger");
        Statement stmt=conn.createStatement();
        ResultSet rs=stmt.executeQuery("select count(*) from dept");
        if(rs.next()){
            count = rs.getInt(1);
        }
        System.out.print("记录数量: "+count);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

疑难点评

在遇到大量记录的时候,推荐使用统计函数获取,不推荐使用 ResultSet 的方法。

知识链接

FAQ9.11 如何判断 ResultSet 结果集为空?

FAQ9.13 如何获取 ResultSet 中 n~m 位置区间的记录?

FAQ9.13 如何获取 ResultSet 中 n ~ m 位置区间的记录?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在分页操作时,经常需要将某一页显示的记录获取,然后在界面上显示。

从 ResultSet 中获取指定区间的记录,主要是使用 absolute()方法将指针定位到参数指定的位置,然后通过 getter 方法获取指针指定记录的字段值。实现代码如下:

```
public void show(int begin, int end) {  
    try {  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
        String url = "jdbc:oracle:thin:@localhost:1521:ora9i";  
        Connection conn = DriverManager.getConnection(url, "scott", "tiger");  
        Statement stmt = conn.createStatement(  
            ResultSet.TYPE_SCROLL_INSENSITIVE,  
            ResultSet.TYPE_FORWARD_ONLY);  
        ResultSet rs = stmt.executeQuery("select * from dept");  
        for (int i = begin; i < end; i++) {  
            //定位指针位置如果超出结果集范围,失败退出  
            if (!rs.absolute(i)) {  
                break;  
            }  
            System.out.print("DeptNo: " + rs.getInt(1));  
            System.out.print("\tDeptName: " + rs.getString(2));  
            System.out.println("\tLOC: " + rs.getString(3));  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

疑难点评

在开发一些数据量较小的系统时,使用上述方法实现分页功能,是非常普遍的做法。在大型系统中,通常需要书写特殊的 SQL 查询语句,仅将当前页需要显示的记录载入 ResultSet 对象。

知识链接

FAQ9.11 如何判断 ResultSet 结果集为空?

FAQ9.12 如何获取 ResultSet 中含有的记录数量?

FAQ9.14 如何利用 ResultSet 更新数据库数据?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

更新数据库数据既可以使用 Update 等 SQL 语句, 也可以使用 ResultSet。

默认情况下, Statement 对象获取的 ResultSet 集合是只读, 并且指针只能逐步向前。如果需要 ResultSet 具有可更新、指针可滚动功能, 在使用 Connection 的 createStatement() 方法获取 Statement 时, 为其指定一些参数。详细介绍请参照 FAQ9.12 “如何获取 ResultSet 中含有的记录数量” 的介绍。

基于 ResultSet 更新数据库时, 可以使用以下两种更新方法。

❑ 更新当前行中的列值

在可滚动的 ResultSet 对象中, 可以向前和向后移动指针, 将其置于绝对位置或相对于当前行的位置。以下代码更新 ResultSet 对象的第 5 行中的 NAME 列, 然后使用 updateRow() 方法更新到数据库表中。

```
public void update1() {
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url = "jdbc:oracle:thin:@localhost:1521:ora9i";
        Connection conn = DriverManager.getConnection(url, "scott", "tiger");
        //设置 Statement 获取的 ResutSet 具有游标可滚动, 结果可更新
        Statement stmt = conn.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = stmt.executeQuery("select * from dept");
        rs.absolute(5); //将指针定位到第 5 行记录
        rs.updateString("NAME", "AINSWORTH"); //更新 "name" 字段的值
        rs.updateRow(); //将 ResultSet 对象中的数据更新到数据库表
        rs.close();
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

❑ 将列值插入到插入行中

可更新的 ResultSet 对象具有一个与其关联的特殊行, 该行用作构建要插入的行的暂存区域。以下代码将指针移动到插入行, 构建一个 3 列的行, 并使用方法 insertRow() 将其插入到 ResultSet 对象和数据库表中。

```
public void update2() {
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url = "jdbc:oracle:thin:@localhost:1521:ora9i";
        Connection conn = DriverManager.getConnection(url, "scott", "tiger");
        //设置 Statement 获取的 ResutSet 具有游标可滚动, 结果可更新
        Statement stmt = conn.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = stmt.executeQuery("select * from dept");
        rs.moveToInsertRow();           //将指针移到插入行
        rs.updateString(1, "AINSWORTH"); //设置插入行第 1 个字段的值
        rs.updateInt(2, 35);           //设置插入行第 2 个字段的值
        rs.updateBoolean(3, true);     //设置插入行第 3 个字段的值
        rs.insertRow();                //将数据插入数据库和 ResultSet 对象中
        rs.moveToCurrentRow();         //将指针指向原来的位置
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

疑难点评

虽然从 JDK 1.2 版本开始已提供了 `ResultSet` 可更新数据库的功能, 但现实中使用的不多, 一般都是使用 SQL 语句。SQL 语句比 `ResultSet` 方式要灵活, 可以根据需要随意拼写, 并且对计算机资源耗费较少。

知识链接

FAQ9.11 如何判断 `ResultSet` 结果集为空?

FAQ9.12 如何获取 `ResultSet` 中含有的记录数量?

FAQ9.13 如何获取 `ResultSet` 中 $n \sim m$ 位置区间的记录?

FAQ9.15 如何使用 LIKE 关键字实现模糊查询?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在进行数据查询时, 可以通过用 `LIKE` 关键字搭配通配符的方式来搜索结果。

SQL 语言提供了 4 种形式的通配符来帮助设置模糊查询的匹配模式, 而 `LIKE` 关键字则在数据文件中搜索与指定模式匹配的字符串、日期或时间值。4 种通配符的功能描述如表 9-1 所示。

表 9-1 通配符的功能

通 配 符	功 能 描 述
%	代表 0 个或多个字符
_	代表任意单一字符
[]	在指定区域或集合中的任意单一字符
[^]	不在指定区域或集合中的任意单一字符

如表 9-1 所示, 模板 “王%” 表示与第 1 个字符为 “王” 的字符串匹配, 如图 9-6 所示。

如果使用模板 “王_”, 则表示与含有两个字符并且第 1 个字符为 “王” 的字符串匹配, 如图 9-7 所示。

学生姓名	学生性别	学生年龄
王男	女	28
王力勤	男	25
王耗	男	22

图 9-6 使用 “王%” 通配符进行模糊查询

学生姓名	学生性别	学生年龄
王男	女	28
王耗	男	22

图 9-7 使用 “王_” 通配符进行模糊查询

使用 LIKE 关键字查询数据库的示例代码如下:

```
public static void main(String[] args) {
    String sql = "select * from emp where ename like '王%'";
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url = "jdbc:oracle:thin:@localhost:1521:ora9i";
        Connection conn = DriverManager.getConnection(url, "scott", "tiger");
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql);
        while(rs.next()){
            System.out.print("Empno: "+rs.getInt(1));
            System.out.print("\tEname: "+rs.getString(2));
            System.out.println("\tSal: "+rs.getString(3));
        }
        rs.close();
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

疑难点评

使用模糊查询对数据文件进行预处理可以过滤掉大量的无用数据, 缩小待操作对象的范围, 从而提高后续处理的效率, 避免对无用数据进行操作造成资源的浪费。

注意: 模糊查询不适用于数值类型的字段。

知识链接

- FAQ9.04 如何实现对数据库数据的查询？
- FAQ9.17 如何实现多表联合查询？

FAQ9.16 如何实现查询的分组统计和排序？

 难度系数：★★★★

 问题频率：95%

核心解答

在进行数据处理时，为了方便操作，往往希望先将在某一字段内具有相同值的数据归为一组，然后再针对每组做出统一的处理。

SQL 语言中提供了 GROUP BY 子句和分组函数来执行分组操作。GROUP BY 子句的使用格式如下：

```
SELECT      字段名,分组函数
FROM        table
[WHERE      查询条件]           //指定过滤条件
[GROUP BY  分组字段]           //指定分组字段
[HAVING     分组函数]           //按分组函数的结果排序
[ORDER BY  排序字段];           //按普通的字段排序
```

在上述格式中，[]部分的内容表示在书写 SQL 查询语句时是可选部分，以下内容含义相同。如果需要按照分组函数的结果排序，必须使用 HAVING 子句，不能使用 ORDER BY。

SQL 语言中提供了 5 个分组函数，用于实现分组统计的功能，具体如表 9-2 所示。

表 9-2 分组函数的功能

分 组 函 数	功 能 描 述
AVG ([DISTINCT]字段名)	统计指定字段非空部分的平均值
COUNT ({ * [DISTINCT]字段名})	统计指定字段非空部分的值的数量，“*”表示统计查询结果中记录的数量
MAX ([DISTINCT]字段名)	统计指定字段非空部分的最大值
MIN ([DISTINCT]字段名)	统计指定字段非空部分的最小值
SUM ([DISTINCT]字段名)	统计指定字段非空部分的合计值

在 9-2 表中，字段名参数前可以加 DISTINCT 关键字，表示对非重复的字段值进行统计。在使用分组函数时，如果遇到字段值为空，将不参与统计操作。

实现分组查询的示例代码如下：

```
public static void main(String[] args) {
    StringBuffer sql = new StringBuffer();
    sql.append("select deptno as no,");
```

```
sql.append(" avg(sal) as a,");
sql.append(" sum(sal) as s");
sql.append(" from emp");
sql.append(" group by deptno");
sql.append(" having avg(sal)");
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    String url = "jdbc:oracle:thin:@localhost:1521:ora9i";
    Connection conn = DriverManager.getConnection(url, "scott", "tiger");
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(sql.toString());
    while(rs.next()){
        System.out.print("部门编号: "+rs.getInt("no"));
        System.out.println("\t 平均工资: "+rs.getDouble("a"));
        System.out.println("\t 工资合计: "+rs.getLong("s"));
    }
    rs.close();
    conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

在使用分组查询的 SQL 语句时, 需要注意一个重要的规则, 即 SELECT 关键字后出现的字段, 除分组函数使用的参数外, 其他的都要在 GROUP BY 子句中出现。

注意: 在使用代码拼写 SQL 语句时, 不要忽略关键字和其他元素之间的空格, 否则会造成语法格式错误。

疑难点评

分组查询使得以组为单位的统一处理成为可能, 这样做不仅能够方便数据的管理, 还可以降低数据操作的复杂度。

知识链接

FAQ9.04 如何实现对数据库数据的查询?

FAQ9.17 如何实现多表联合查询?

FAQ9.17 如何实现多表联合查询?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

在数据处理中, 如果需要使用一个表的全部数据以及另一个表中全部或满足某些条件的部

分数据时,可以使用各种连接格式的 SQL 查询语句来实现。

在 SQL 语法中,表之间的连接类型很多,使用最多的是等值连接、外连接和自连接。

(1) 等值连接

等值连接的作用是:查询结果是由两个表的记录共同决定的,只有两个表的连接字段值相等的记录,才会作为结果返回。语法格式如下:

```
SELECT    table1.column, table2.column
FROM      table1, table2
WHERE     table1.column = table2.column;
```

或者

```
SELECT    table1.column, table2.column
FROM      table1
JOIN table2 ON (table1.column = table2.column)
```

等值连接的示例如下:

```
SELECT    e.empno, e.ename, e.sal, d.name
FROM      emp e
JOIN dept d ON (e.deptno = d.deptno);
```

(2) 外连接

外连接的作用是:查询结果是由某一主表的记录决定的,即使另一方没有对应的记录,主表记录也要作为结果返回,另一方记录的字段值为 NULL。外连接分为左外连接和右外连接两种。

左外连接语法格式如下, table1 为主表。

```
SELECT    table1.column, table2.column
FROM      table1
LEFT OUTER JOIN table2 ON (table1.column = table2.column)
```

右外连接语法格式如下, table2 为主表。

```
SELECT    table1.column, table2.column
FROM      table1
RIGHT OUTER JOIN table2 ON (table1.column = table2.column)
```

左外连接和右外连接的示例如下:

```
SELECT    e.empno, e.ename, e.sal, d.name
FROM      emp e
LEFT OUTER JOIN dept d ON (e.deptno = d.deptno);
//等价于
SELECT    e.empno, e.ename, e.sal, d.name
FROM      dept d
RIGHT OUTER JOIN emp e ON (e.deptno = d.deptno);
```

(3) 自连接

自连接的作用与等值连接类似,只不过连接双方都是同一个表,一般是同一个表的两个不同字段做等值。语法格式如下:

```
SELECT    t1.column, t2.column
FROM      table1 t1
JOIN table1 t2 ON (t1.column1 = t2.column2)
```

自连接的示例如下:

```
SELECT    e1.empno,e.ename,e.sal,e2.empno,e2.name
FROM      emp e1
JOIN emp e2 ON (e1.mgr = e2.empno);
```

使用 Java 代码执行上述 SQL 语句的过程, 与其他查询功能相同, 因此不再重复介绍。

疑难点评

多表连接查询在现实中使用非常频繁, 需要熟练掌握其书写格式和语法作用。通过上述的左外连接和右外连接示例可以看出, 这两种连接是可以相互转化的, 可以实现相同的功能。

知识链接

FAQ9.04 如何实现对数据库数据的查询?

FAQ9.15 如何使用 LIKE 关键字实现模糊查询?

FAQ9.16 如何实现查询的分组统计和排序?

FAQ9.18 如何使用 JDBC 的批处理操作?

📖 难度系数: ★★★★★

📖 问题频率: 89%

核心解答

JDBC 提供了批处理功能, 可以将一些相关的 SQL 操作封装在一起, 然后发送给数据库批量执行, 从而提高了程序的效率。

Statement 和 PreparedStatement 等 SQL 执行器都支持批处理功能, 可以使用 addBatch() 方法向批处理中追加 SQL 操作语句, 然后使用 executeBatch() 执行批处理中的 SQL 语句。

Statement 批处理操作的示例代码如下:

```
public static void main(String[] args) {
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url = "jdbc:oracle:thin:@localhost:1521:ora9i";
        Connection conn = DriverManager.getConnection(url, "scott", "tiger");
        Statement stmt = conn.createStatement();
        stmt.addBatch("insert into emp (empno,ename,sal,deptno) vlaues (7908,'tom',3000,10)");
        stmt.addBatch("update emp set ename='rose' where empno=7800");
        stmt.addBatch("delete emp where empno=7900");
        stmt.executeBatch();
        System.out.println("Success");
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

PreparedStatement 批处理操作的示例代码如下:

```
public static void main(String[] args) {
    String sql = "delete emp where ename = ?";
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url = "jdbc:oracle:thin:@localhost:1521:ora9i";
        Connection conn = DriverManager.getConnection(url, "scott", "tiger");
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, "clarck");
        ps.addBatch();
        ps.setString(1, "night");
        ps.addBatch();
        ps.executeBatch();
        System.out.println("Success");
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

疑难点评

如果一个操作比较复杂,涉及多次的数据库更新操作,可以使用批处理功能提高效率。在使用批处理时,也可以与事务控制结合起来应用,这样可以保证一个操作的完整性。

注意:批处理不能用于查询语句。

知识链接

FAQ9.05 如何实现对数据库数据的增加、删除和修改?

FAQ9.19 如何实现 Oracle 字段值递增的功能?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

MySQL 和 SQLServer 数据库都支持字段值自动递增功能,但在 Oracle 数据库中却不支持。在 Oracle 中可以使用 Sequence 序列,实现字段值的自动递增。

❑ 创建序列的 SQL 语句的语法格式如下:

```
create sequence 序列名
[increment by 递增量] --设置递增量
[start with 起始值] --设置起始值
[maxvalue|nomaxvalue] --设置最大值
[nocycle] --设置累加,不循环
[cache 数量] --设置一次生成多少个序列值存入缓存
```


示例如下:

```
create sequence emp_seq  
increment by 1  
start with 100  
nomaxvalue  
nocycle  
cache 100
```

❑ 删除序列的 SQL 语句如下:

```
drop sequence emp_seq
```

❑ 序列的应用

在使用序列时,可以使用 nextval 和 currval 两个属性。currentval 表示序列当前值;而 nextval 表示在当前值基础之上递增之后的值。

获取序列值得查询语句如下:

```
select emp_seq.nextval from dual;
```

使用序列向表中插入记录的 SQL 语句如下:

```
insert into emp (empno,ename,sal,deptno) values (emp_seq.nextval,'张三',5000,20);
```

Java 执行插入操作的示例代码如下:

```
public static void add(){  
    Connection conn=null;  
    Statement stmt=null;  
    try{  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
        String url="jdbc:oracle:thin:@localhost:1521:ora9i";  
        conn=DriverManager.getConnection(url,"scott","tiger");  
        stmt=conn.createStatement();  
        String sql="insert into emp (empno,ename,sal,deptno) values (emp_seq.nextval,'张三',5000,20)";  
        stmt.executeUpdate(sql);  
        conn.close();  
    }catch(Exception e){  
        e.printStackTrace();  
    }finally{  
        try {  
            stmt.close();  
        } catch (SQLException e) {  
        }  
        try {  
            conn.close();  
        } catch (SQLException e) {  
        }  
    }  
}
```

疑难点评

Sequence 是数据库系统按照一定规则自动增加的数字序列。这个序列一般作为主键代理,因为其不会重复。Oracle、DB2 和 PostgreSQL 等数据库都有 Sequence,而 MySQL、SQLServer 和 Sybase 等数据库则没有。

知识链接

FAQ9.05 如何实现对数据库数据的增加、删除和修改?

FAQ9.20 如何处理数据表中 Date 类型的字段?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在对数据库进行操作时,经常会遇到日期类型,如果日期类型处理得不好,很容易出现 SQL 语句错误。

在 Oracle 数据库中对日期字段进行插入、更新操作或作为条件查询时,需要使用 `to_date()` 函数转换,否则 SQL 发生错误。例如下列 SQL 语句在 Oracle 中就会发生错误。

```
update emp set hiredate='2008-8-8' where empno=7800;  
select * from emp where hiredate='2008-8-8';
```

在 Oracle 中,日期默认格式为“dd-mm-yy”,即日月年格式,如果字符串符合该格式,数据库在执行 SQL 语句时可以将其自动转换为日期类型,否则转化失败,发生错误。

因此在 Oracle 中,上述 SQL 语句正确的写法应该如下:

```
update emp set hiredate=to_date('2008-8-8','yyyy-mm-dd') where empno=7800;  
select * from emp where hiredate= to_date('2008-8-8','yyyy-mm-dd');
```

执行上述 SQL 语句的 Java 程序请参考前面类似的问题示例。

疑难点评

`to_date()` 是 Oracle 数据库的转换函数,该函数不能在不同数据库之间通用。在 MySQL 数据库中,“2008-8-8”格式的字符串可以自动转换为 Date 类型,不需要使用转换函数。

知识链接

FAQ9.05 如何实现对数据库数据的增加、删除和修改?

FAQ9.21 如何向表中插入含有特殊字符的信息?

FAQ9.22 如何使用 BLOB 类型的字段存取图片?

FAQ9.23 如何使用 CLOB 类型的字段存取字符文件?

FAQ9.21 如何向表中插入含有特殊字符的信息?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在使用 Statement 执行 SQL 操作时, 需要以“字符串+变量”的形式拼写 SQL 语句。如果变量中包含一些特殊字符, 例如“'”, “,” 或“where”等特殊字符, 会引起 SQL 格式错误。

将含有“'”, “,” 或“where”等特殊字符的字符串写入或更新到数据库, 可以使用 PreparedStatement 工具。因为 PreparedStatement 是将 SQL 语句和参数值分开的, 所以不会发生上述错误。

示例代码如下:

```
public static void main(String[] args) {
    String sql = "update emp set ename=? where empno=?";
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url = "jdbc:oracle:thin:@localhost:1521:ora9i";
        Connection conn = DriverManager.getConnection(url, "scott", "tiger");
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, "tom', zhang");
        ps.setInt(12, 7800);
        ps.executeUpdate();
        System.out.println("Success");
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

疑难点评

在遇到特殊字符的时候, 使用 PreparedStatement 操作数据表比 Statement 要方便得多。上述功能如果使用 Statement 实现, 首先需要将“tom', zhang”字符串转为 ASCII 码, 然后再写入 SQL 语句。一般在对数据库操作时建议选用 PreparedStatement, 这样就可以避免因特殊字符使 SQL 混乱产生错误。

知识链接

- FAQ9.05 如何实现对数据库数据的增加、删除和修改?
- FAQ9.20 如何处理数据表中 Date 类型的字段?
- FAQ9.22 如何使用 BLOB 类型的字段存取图片?
- FAQ9.23 如何使用 CLOB 类型的字段存取字符文件?

FAQ9.22 如何使用 BLOB 类型的字段存取图片?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

BLOB 类型用于存储二进制的的数据，例如存取图片和多媒体等文件。

使用 Statement 需要数据库写入操作，需要将 BLOB 字段内容写入 SQL，因此不太适合应用。操作 BLOB 类型需要使用 PreparedStatement 类，通过 setBinaryStream()方法可实现将 BLOB 类型数据写入数据库。

创建数据表 images 的 SQL 语句如下：

```
create table images
(
    name varchar2(10),
    changdu number(10),
    content blob
);
```

将图片存入数据表的代码如下：

```
public static void main(String args[]) throws Exception
{
    String pic= "b.jpg";
    File f= new File(pic);
    //获取图片的字节流信息
    FileInputStream fis = new FileInputStream(pic);
    //注册驱动
    Class.forName("oracle.jdbc.driver.OracleDriver");
    //连接 Oracle 数据库
    Connection conn =
    DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:oracle9i","scott","tiger");
    //预编译插入操作
    PreparedStatement pstmt = conn.prepareStatement("insert into images values(?,?,?)");
    //设置图片名字段
    pstmt.setString(1,pic);
    //设置图片大小字段
    pstmt.setInt(2,(int)f.length());
    //设置 BLOB 字段
    pstmt.setBinaryStream(3,fis,(int)f.length());
    //执行插入操作
    pstmt.executeUpdate();
    pstmt.close();
    conn.close();
}
```

从数据库读取图片时，可以通过 Statement 或 PreparedStatement 类调用 getBinaryStream()方法实现 BLOB 内容的读取，示例代码如下：

```
public static void main(String args[]) throws Exception {
    // 注册驱动
    Class.forName("oracle.jdbc.driver.OracleDriver");
    // 连接 Oracle 数据库
    Connection conn = DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:oracle9i", "scott", "tiger");
```

```
//获取 Statement 对象
Statement stmt = conn.createStatement();
//执行查询语句
ResultSet rs = stmt
    .executeQuery("select * from images where name='b.jpg'");
if (rs.next()) {
    //获取文件名字段的值
    String name = rs.getString(1);
    //获取文件大小
    int length = rs.getInt(2);
    //获取 BLOB 的内容
    InputStream is = rs.getBinaryStream(3);
    //将 BLOB 的内容还原为图片文件
    byte[] b = new byte[length];
    is.read(b);
    FileOutputStream fos = new FileOutputStream(name);
    fos.write(b);
    fos.close();
}
conn.close();
}
```

疑难点评

在处理一些非纯文本类型的文件时,可以使用 BLOB 类型存取,例如*.doc、*.jpg 和*.mp3 等。利用 JDBC 对 BLOB 类型字段操作时,除了可以使用 `getBinaryStream()`和 `setBinaryStream()`方法读写之外,也可以使用 `setBlob()`和 `getBlob()`方法。

知识链接

- FAQ9.05 如何实现对数据库数据的增加、删除和修改?
- FAQ9.20 如何处理数据表中 Date 类型的字段?
- FAQ9.21 如何向表中插入含有特殊字符的信息?
- FAQ9.23 如何使用 CLOB 类型的字段存取字符文件?

FAQ9.23 如何使用 CLOB 类型的字段存取字符文件?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

CLOB 类型用于存储字符串类型的数据,例如字符文件。

操作 CLOB 类型需要使用 `PreparedStatement` 类,通过 `setAsciiStream()`方法可实现将 CLOB

类型数据写入数据库的功能。

创建数据表 files 的 SQL 语句如下：

```
create table files
(
    name varchar2(10),
    changdu number(10),
    content clob
);
```

将文件存入数据表的代码如下：

```
public static void main(String args[]) throws Exception
{
    String pic= "test.txt";
    File f= new File(pic);
    //获取图片的字节流信息
    FileInputStream fis = new FileInputStream(pic);
    //注册驱动
    Class.forName("oracle.jdbc.driver.OracleDriver");
    //连接 Oracle 数据库
    Connection conn =
    DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:oracle9i","scott","tiger");
    //预编译插入操作
    PreparedStatement pstmt = conn.prepareStatement("insert into files values(?,?,?)");
    //设置文件名字段
    pstmt.setString(1,pic);
    //设置文件大小字段
    pstmt.setInt(2,(int)f.length());
    //设置 CLOB 字段
    pstmt.setAsciiStream(3,fis,(int)f.length());
    //执行插入操作
    pstmt.executeUpdate();
    pstmt.close();
    conn.close();
}
```

从数据库读取文件时，可以通过 Statement 或者 PreparedStatement 类调用 getAsciiStream() 方法实现 CLOB 内容的读取，示例代码如下：

```
public static void main(String args[]) throws Exception {
    //注册驱动
    Class.forName("oracle.jdbc.driver.OracleDriver");
    //连接 Oracle 数据库
    Connection conn = DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:oracle9i", "scott", "tiger");
    //获取 Statement 对象
    Statement stmt = conn.createStatement();
    //执行查询语句
    ResultSet rs = stmt.executeQuery("select * from files where name='test.txt'");
    if (rs.next()) {
        //获取 CLOB 的内容
        InputStream is = rs.getAsciiStream(3);
    }
}
```



```

        //将 CLOB 的内容输出到控制台
        BufferedReader br = new BufferedReader(new
InputStreamReader(is));
        String line = null;
        //循环读取并输出
        while((line=br.readLine()) != null){
            System.out.println(line);
        }
        br.close();
        is.close();
    }
    conn.close();
}

```

疑难点评

在处理一些纯文本类型的字符文件时,可以使用 CLOB 类型存取,例如*.txt、*.csv、*.html 和*.java 等。利用 JDBC 对 BLOB 类型字段操作时,除了可以使用 getAsciiStream()和 setAsciiStream()方法读写之外,也可以使用 setClob()和 getClob()方法。

知识链接

FAQ9.05 如何实现对数据库数据的增加、删除和修改?

FAQ9.20 如何处理数据表中 Date 类型的字段?

FAQ9.21 如何向表中插入含有特殊字符的信息?

FAQ9.22 如何使用 BLOB 类型的字段存取图片?

FAQ9.24 如何通过程序创建和删除数据表?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

创建数据库的 SQL 语法格式如下:

```

CREATE TABLE 表名
(
    字段名 类型,
    字段名 类型,
    .....
    字段名 类型
)

```

下列 SQL 语句创建了一个名字为 person 的数据表,具体代码如下:

```

CREATE TABLE person
(

```

```

id number,
name varchar2(20),
age number(2),
address varchar(50)
)

```

上述 SQL 适用于 Oracle 数据库, 不同数据库的数据类型有些差异, 例如 Oracle 数值类型用 number, 而 MySQL 可以使用 int 或 double 等。MySQL 提供了布尔类型 boolean, 而 Oracle 和 DB2 不支持布尔类型。

删除 person 表的语句如下:

```
DROP TABLE person
```

创建和删表数据表的 SQL 语句, 可以通过 Statement 或者 PreparedStatement 类的 execute() 方法执行。

创建表的示例如下:

```

public void createTable() {
    Connection conn = null;
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url = "jdbc:oracle:thin:@localhost:1521:ORCL";
        conn = DriverManager.getConnection(url, "scott", "tiger");
        Statement stmt = conn.createStatement();
        //创建表
        stmt.execute("create table student("
            + "studentid varchar2(10),studentname char(20),"
            + "gender char(6),phone char(16))");

        //添加数据
        for (int i = 1; i < 45; i++) {
            stmt.executeUpdate("insert into student values(" + "ij2004"
                + (1000 + i) + "," + "tom" + (char) (48 + i) + ","
                + (i % 2 == 0 ? '男' : '女') + "," + (82622266 + i)
                + ")");
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

删除表的示例如下:

```

public void dropTable() {
    Connection conn = null;
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");

```

```
String url = "jdbc:oracle:thin:@localhost:1521:ORCL";
conn = DriverManager.getConnection(url, "scott", "tiger");
Statement stmt = conn.createStatement();
//创建表
stmt.execute("drop table student");

} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        conn.close();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

疑难点评

Statement 和 PreparedStatement 提供了很多执行 SQL 的方法。一般的使用习惯是 SELECT 语句使用 executeQuery() 方法执行, DELETE、UPDATE 和 INSERT 语句使用 executeUpdate() 方法执行, 而 CREATE 和 DROP 语句使用 execute() 方法执行, 当然也可以使用 executeUpdate() 方法。

知识链接

FAQ9.25 如何获取数据表的结构信息?

FAQ9.26 如何获取数据库中所有表名?

FAQ9.25 如何获取数据表的结构信息?

📖 难度系数: ★★★★★

📖 问题频率: 75%

核心解答

与数据库和数据库表相关的信息被称为数据库元数据。

(1) DatabaseMetaData

通过 Connection 的 getMetaData() 方法可以获得包含数据库元数据的 DatabaseMetaData 类的对象。DatabaseMetaData 提供了非常丰富的方法, 用于获取数据库的整体综合信息, 例如版本号、产品名称、驱动名称和列名称允许的最大字符数等等。

使用 DatabaseMetaData 获取数据库信息的示例如下:

```
public void testDatabaseMetaData() {
    try {
```



```
new oracle.jdbc.driver.OracleDriver();
String url = "jdbc:oracle:thin:@localhost:1521:orcl";
Connection conn = DriverManager
    .getConnection(url, "scott", "tiger");
DatabaseMetaData dmd = conn.getMetaData();

System.out.println("数据库名称: " + dmd.getDatabaseProductName());
System.out.println("数据库版本: " + dmd.getDatabaseProductVersion());
System.out.println("数据库用户名: " + dmd.getUserName());
conn.close();
} catch (SQLException e) {
    System.err.println(e);
}
}
```

(2) ResultSetMetaData

通过 `ResultSet` 的 `getMetaData()` 方法可以获取包含数据表元数据的 `ResultSetMetaData` 对象。`ResultSetMetaData` 提供了获取表名称、字段名称、字段类型和字段个数等信息的方法，具体有以下几种。

- ❑ `getTableName()`: 获取表名称。
- ❑ `getColumnCount()`: 获取字段的个数。
- ❑ `getColumnName()`: 获取字段的名称。
- ❑ `getColumnType()`: 获取字段的 SQL 类型。
- ❑ `isNullable()`: 判断字段值是否允许为空。

使用 `ResultSetMetaData` 获取表结构的示例如下：

```
public void testResultSetMetaData() {
    try {
        new oracle.jdbc.driver.OracleDriver();
        String url = "jdbc:oracle:thin:@localhost:1521:orcl";
        Connection conn = DriverManager
            .getConnection(url, "scott", "tiger");
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        ResultSetMetaData rsmd = rs.getMetaData();
        int columnCount = rsmd.getColumnCount();
        System.out.println("有" + columnCount + "字段");
        for (int i = 0; i < columnCount; i++) {
            System.out.println("第" + i + "个字段名称是: " + rsmd.getColumnName(i));
        }
        conn.close();
    } catch (SQLException e) {
        System.err.println(e);
    }
}
```

疑难点评

ResultSetMetaData 其实是获取 ResultSet 对象中的信息,如果 SELECT 查询语句使用了字段别名,那么 ResultSetMetaData 得到的也是字段的别名。因此如果需要得到某个表的真实信息,应该使用“select * from 表名”语句进行查询。

知识链接

FAQ9.24 如何通过程序创建和删除数据表?

FAQ9.26 如何获取数据库中所有表名?

FAQ9.26 如何获取数据库中所有表名?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

通过 Connection 类的 getMetaData()方法可以获取一个 DatabaseMetaData 对象,该对象包含了数据库的整体综合信息。利用 DatabaseMetaData 的 getTables()方法可以获取数据库中所有的表名。getTables()方法的定义如下:

ResultSet getTables(String catalog,String schemaPattern,String tableNamePattern,String[] types) throws SQLException

getTables()方法的参数描述如下。

- ❑ catalog: 类别名称。该参数为""时检索没有类别的描述;为 null 时则表示不充当过滤条件。
- ❑ schemaPattern: 模式名称。该参数为""时检索那些没有模式的描述,为 null 时则表示该模式名称不充当过滤条件。
- ❑ tableNamePattern: 表名称。该参数为%时表示检索所有表。
- ❑ types: 表类型的列表。该参数为 null 时表示返回所有类型。

getTables()方法可用于返回与类别、模式、表名称和类型标准匹配的表描述,每一行都是一个表描述。它们根据 TABLE_TYPE、TABLE_SCHEM 和 TABLE_NAME 进行排序。

每个表描述都包含以下信息项。

- ❑ TABLE_CAT String : 表类别 (可为 null)。
- ❑ TABLE_SCHEM String: 表模式 (可为 null)。
- ❑ TABLE_NAME String : 表名称。
- ❑ TABLE_TYPE String: 表类型。典型的类型是 "TABLE"、"VIEW"、"SYSTEM TABLE"、"GLOBAL TEMPORARY"、"LOCAL TEMPORARY"、"ALIAS" 和 "SYNONYM"。

- ❑ REMARKS String: 表的解释性注释。
- ❑ TYPE_CAT String: 类型的类别 (可为 null)。
- ❑ TYPE_SCHEM String: 类型模式 (可为 null)。
- ❑ TYPE_NAME String: 类型名称 (可为 null)。
- ❑ SELF_REFERENCING_COL_NAME String: 有类型表的指定 "identifier" 列的名称 (可为 null)。
- ❑ REF_GENERATION String: 指定在 SELF_REFERENCING_COL_NAME 中创建值的方式。这些值为 "SYSTEM"、"USER" 和 "DERIVED" (可能为 null)。

获取数据库所有表的示例代码如下:

```
public static void showTable() throws Exception{
    Class.forName("com.mysql.jdbc.Driver");
    String url = "jdbc:mysql:///test";
    Connection conn = DriverManager.getConnection(url, "root", "root");
    DatabaseMetaData dmd = conn.getMetaData();
    String[] types = {"TABLE"};
    ResultSet rs = dmd.getTables(null, null, "%", types);
    while(rs.next()){
        System.out.println(rs.getString(3));
    }
    rs.close();
}
```

疑难点评

在获取数据库所有表信息的时候,每一种数据库都有其各自的实现方法,例如 Oracle 也可以使用“select table_name from user_tables”语句获取所有表名。通过 DatabaseMetaData 类获取的方式对大部分数据库都适用,例如 Oracle、MySQL 和 SQLServer 等,但有些数据库可能不会返回所有表信息。

知识链接

FAQ9.25 如何获取数据表的结构信息?

FAQ9.27 如何用程序备份和恢复数据库?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

实现用 Java 程序备份和恢复数据库,最通用的方法是利用 Runtime 类的 exec()方法执行备份和恢复的命令语句。

下面以 MySQL 数据库为例, 介绍利用 Java 程序备份和恢复数据库的实现过程。

每一种数据库都提供了备份和恢复的工具, 在 CMD 界面中可以使用命令调用这些工具的功能。MySQL 数据库备份和恢复的命令格式如下:

```
//备份命令
mysqldump -u 用户名 -p 密码 --opt 库名 > 备份文件路径
//恢复命令
mysql -u 用户名 -p 密码 库名 < 恢复文件路径
```

使用示例如下:

```
//备份命令
mysqldump -uroot -proot --opt test > f:\db.sql
//恢复命令
mysql -uroot -proot test < f:\db.sql
```

为了方便上述命令的使用, 在使用之前需要设置系统的环境变量, 在 PATH 变量值后追加: “安装路径\MySQL\MySQL Server 5.0\bin”。

利用 Java 程序执行上述命令即可实现 MySQL 数据库的备份和恢复功能。备份功能的示例代码如下:

```
/**
 * 数据库备份
 */
public void backup() {
    String user = "root";           //数据库账号
    String password = "root";       //登录密码
    String database = "test";       //需要备份的数据库名
    String filepath = "f:\\db.sql"; //备份的路径地址

    String back = "mysqldump -u"+user+" -p"+password+" --opt "+database+" > "+filepath;
    System.out.println(back);
    try {
        Process p = Runtime.getRuntime().exec("cmd.exe /c "+back);
        try {
            p.waitFor();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("数据已导出到文件" + filepath + "中--"+p.exitValue());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

恢复功能的示例代码如下:

```
/**
 * 数据库恢复
 */
public void load() {
    String user = "root"; //数据库账号
```

```
String password = "root"; //登录密码
String database = "test"; //需要备份的数据库名
String filepath = "f:\\db.sql"; //备份的路径地址

String createDb = "mysqladmin -u"+user+" -p"+password+" create "+database;
String store = "mysql -u"+user+" -p"+password+" "+database+" < "+filepath;
System.out.println(createDb);
System.out.println(store);
try{
    Process p = Runtime.getRuntime().exec("cmd.exe /c "+createDb);
    p.waitFor();
    Runtime.getRuntime().exec("cmd.exe /c "+store);
    System.out.println("数据已从 "+ filepath + " 导入到数据库中");
} catch (Exception e) {
    e.printStackTrace();
}
}
```

疑难点评

上述内容介绍了如何利用 Java 备份、恢复 MySQL 数据库, 其他数据库的实现方式与之相似。不同数据库都有其各自的备份和恢复命令, 只需要将其命令写出, 然后通过 Java 执行即可。

知识链接

FAQ9.02 Java 与数据库的连接方式有哪些?

FAQ9.28 什么是事务? 如何使用 JDBC 事务控制?

📖 难度系数: ★★★

📖 问题频率: 95%

核心解答

数据库事务是指作为单个逻辑单元执行的一系列操作, 它可以由一个或多个 SQL 语句组成。如果事务中的 SQL 操作全部正确执行, 则向数据库提交更改所有数据; 如果一旦有错误发生, 则不会对数据库作任何修改或改变。使用事务操作数据库可以保障数据库数据的完整性。

在 JDBC 中, 使用 Statement 类的 executeUpdate() 方法对数据库操作时, 默认情况下, 事务自动提交功能是打开的, 即每次成功调用 executeUpdate() 后都会自动提交事务操作。因此如果当一个逻辑单元包含若干个 SQL 操作时, 即使后续操作出错, 前面的操作也已经提交到数据库了, 不能取消, 这样就破坏了系统数据的完整性。

利用 JDBC 事务封装若干 SQL 操作时, 需要将事务自动提交功能关闭, 然后通过代码方式

控制事务提交和事务回滚操作。示例代码如下:

```
public static void main(String args[]){
    Connection conn= null;
    try{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String url="jdbc:oracle:thin:@localhost:1521:ora9i";
        conn=DriverManager.getConnection(url,"scott","tiger");
        //关闭自动提交功能
        conn.setAutoCommit(false);
        Statement stmt=conn.createStatement();
        //执行 3 个 SQL 操作
        stmt.executeUpdate("insert into dept values(15,'dirk','new')");
        stmt.executeUpdate("insert into dept values(15,'dirk','new')");
        stmt.executeUpdate("update dept set dname='program' where deptno=10");
        //如果正确执行则事务提交
        conn.commit();
        stmt.close();
        conn.close();
    }catch(Exception e){
        try{
            //如果发生异常则事务回滚
            conn.rollback();
        } catch(Exception e1) {
            e1.printStackTrace();
        }
        e.printStackTrace();
    }
}
```

疑难点评

使用事务可以将一组 SQL 操作当作一个整体进行控制,保障逻辑和数据完整性。在数据库中使用 commit 和 rollback 命令也可以实现事务的提交和回滚操作。

知识链接

FAQ9.29 什么是 JTA? JTA 事务与 JDBC 事务有什么区别?

FAQ9.29 什么是 JTA? JTA 事务与 JDBC 事务有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

JTA (Java Transaction API) 是一种高层的、与实现无关的、与协议无关的 API,应用程序

和应用服务器可以使用 JTA 实现事务管理。

JTA 主要用于分布式的多个数据源的事务控制，而 JDBC 的 Connection 提供的是单个数据源的事务。JDBC 事务因为只涉及一个数据源，所以其事务可以由数据库自己单独实现，而 JTA 事务因为其分布式和多数据源的特性，不能由任何一个数据源实现事务管理，因此 JTA 中的事务由事务管理器实现，它会在多个数据源之间管理事务。一般 JTA 事务都是用于 EJB 中，因此一般的应用服务器都有自己的事务管理器用来管理 JTA 事务。

注意：如果使用 Tomcat 应用服务器，是不能使用 JTA 事务的。JTA 在使用时，一般会选用 Weblogic、JBoss 和 Websphere 等服务器。

疑难点评

JTA 也是用于管理事务的一套 API，与 JDBC 相比，JTA 主要用于管理分布式多个数据源的事务操作，而 JDBC 主要用于管理单个数据源的事务操作。

知识链接

FAQ9.30 如何使用 JTA 实现分布式事务控制？

FAQ9.30 如何使用 JTA 实现分布式事务控制？

📖 难度系数：★★★★

📖 问题频率：80%

核心解答

在分布式系统中，一个逻辑单元可能会涉及若干数据源的数据。JTA (Java Transaction API)，能够实现在网络环境中多个数据库在一个事务中进行操作，而 JDBC 事务只能在一个数据库中进行，因为 JDBC 中的事务是与连接相关的。

与 JTA 相关的 API 都在 javax.transaction 包中，在 J2EE 1.4 的 API 文档能找到相关的介绍。JTA 最常用是 UserTransaction 接口，它与 JDBC 的事务控制类似。在 JTA 中，事务开始时需要调用 UserTransaction 对象的 begin() 方法，而事务提交时则调用 commit() 方法，如果程序执行时发生异常需要取消操作，可以使用 rollback() 方法将事务回滚。

注意：UserTransaction 是一个接口类型，其实现类则根据使用的 Web 容器不同，由容器实现。

在使用 JTA 之前，需要在应用服务器上配置数据库源和事务管理器。然后通过 JNDI 获取。假如现在已经在 Weblogic 服务器上配置了 mysql 和 myoracle 两个数据源和一个 usertransaction 事务管理器，使用 JTA 控制两个数据源操作的示例代码如下：

```
public static void main(String[] args) {  
    Hashtable env = new Hashtable();  
    env.put(Context.INITIAL_CONTEXT_FACTORY,
```

```

        "weblogic.jndi.WLInitialContextFactory");
env.put(Context.PROVIDER_URL, "t3://localhost:7001");
Context ctx = null;
UserTransaction transaction = null;
try {
    ctx = new InitialContext(env);
    transaction = (UserTransaction) ctx.lookup("usertransaction");
    System.out.println("begin");
    //打开 JTA 事务
    transaction.begin();
    //操作 mysql 数据源
    DataSource ds = (DataSource) ctx.lookup("mysql");
    Connection conn1 = ds.getConnection();
    String sql = "insert into test1 values('1','2')";
    Statement stmt = conn1.createStatement();
    stmt.executeUpdate(sql);
    conn1.close();
    //操作 myoracle 数据源
    DataSource ds1 = (DataSource) ctx.lookup("myoracle");
    Connection conn2 = ds1.getConnection();
    sql = "insert into table1 values('h')";
    Statement stmt1 = conn2.createStatement();
    stmt1.executeUpdate(sql);
    //提交 JTA 事务
    transaction.commit();
    System.out.println("end");
} catch (Exception e) {
    try {
        //回滚 JTA 事务
        transaction.rollback();
    } catch (Exception e1) {
        e1.printStackTrace();
    }
    e.printStackTrace();
}
}

```

疑难点评

JTA 分布式事务控制与 JDBC 相似,难以实现的是服务器配置数据源和事务管理器。在配置数据源时需要选择 XA 类型,因为 XA 数据源产生的数据库连接可以参与 JTA 事务,不支持自动提交。Oracle、Sybase、DB2 和 SQL Server 等大型数据库都支持 XA,支持分布事务,MySQL 从 5.0 版本开始也增加了对 XA 的支持。

知识链接

FAQ9.29 什么是 JTA? JTA 事务与 JDBC 事务有什么区别?

FAQ9.31 什么是数据库连接池？工作原理如何？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

(1) 连接池概念及优点

连接池用于创建和管理数据库连接的缓冲池技术，缓冲池中的连接可以被任何需要它们的线程使用。当一个线程需要用 JDBC 对一个数据库操作时，将从池中请求一个连接。当这个连接使用完毕后，将返回到连接池中，等待为其他的线程服务，如图 9-8 所示。

连接池的优点主要有以下几个方面。

❑ 减少连接创建时间

连接池中的连接是已准备好的、可重复使用的，获取后可以直接访问数据库，因此减少了连接创建的次数和时间。

❑ 简化的编程模式

当使用连接池时，每一个单独的线程能够像创建了一个自己的 JDBC 连接一样操作，允许用户直接使用 JDBC 编程技术。

❑ 控制资源的使用

如果不使用连接池，每次访问数据库都需要创建一个连接，这样系统的稳定性受系统连接需求影响很大，很容易产生资源浪费和高负载异常。连接池能够使性能最大化，将资源利用控制在一定的水平之下。连接池能控制池中的连接数量，增强了系统在大量用户应用时的稳定性。

(2) 连接池原理

连接池技术的核心思想是连接复用，通过建立一个数据库连接池以及一套连接使用、分配和管理策略，使得该连接池中的连接可以得到高效、安全的复用，避免了数据库连接频繁建立、关闭的开销。

连接池的工作原理主要由 3 部分组成，分别为连接池的建立、连接池中连接的使用管理、连接池的关闭。

❑ 连接池的建立

一般在系统初始化时，连接池会根据系统配置建立，并在池中创建了几个连接对象，以便使用时能从连接池中获取。连接池中的连接不能随意创建和关闭，这样避免了连接随意建立和关闭造成的系统开销。Java 中提供了很多容器类可以方便的构建连接池，例如 Vector、Stack 等。

❑ 连接池的管理

连接池管理策略是连接池机制的核心，连接池内连接的分配和释放对系统的性能有很大的影响。其管理策略如下。

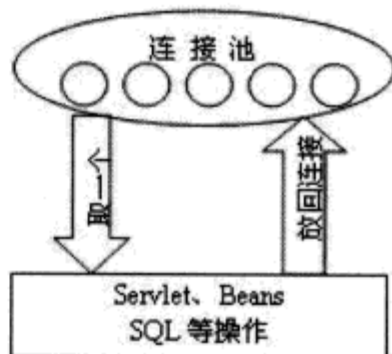


图 9-8 连接池模型

当客户请求数据库连接时, 首先查看连接池中是否有空闲连接, 如果存在空闲连接, 则将连接分配给客户使用; 如果没有空闲连接, 则查看当前所开的连接数是否已经达到最大连接数, 如果没达到就重新创建一个连接给请求的客户; 如果达到就按设定的最大等待时间进行等待, 如果超出最大等待时间, 则抛出异常给客户。

当客户释放数据库连接时, 先判断该连接的引用次数是否超过了规定值, 如果超过就从连接池中删除该连接, 否则保留为其他客户服务。

上述策略保证了数据库连接的有效复用, 避免频繁地建立、释放连接所带来的系统资源开销。

□ 连接池的关闭

当应用程序退出时, 关闭连接池中所有的连接, 释放连接池相关的资源, 该过程正好与创建相反。

疑难点评

在用户量和数据库访问量比较多的情况下, 运用连接池访问技术能提高数据库的访问效率, 增强系统性能和稳定性, 减少系统的开销, 从而提高整个应用系统的运行效率。在使用时可以选用第三方连接池组件, 也可以利用 Java 代码依据某些管理策略自定义。

FAQ9.32 如何提升 SQL 语句的查询性能?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

在对数据库进行操作时, 如果 SQL 语句书写不当, 对程序的效率也会造成很大影响。提高 SQL 语句的执行效率可以从以下几个方面入手。

1. 数据库设计与规划

- Primary Key 字段的长度尽量小, 能用 small integer 就不要用 integer。例如员工数据表, 如果能用员工编号作主键, 就不要用身份证号码。
- 字符字段如果长度固定, 就不要用 varchar 或 nvarchar 类型, 而应该用 char 或 nchar 类型。例如身份证号码、银行密码等字段。
- 字符字段如果长度不固定, 则应该使用 varchar 或 nvarchar 类型。除了可节省存储空间外, 存取硬盘时也会较有效率。例如地址、个人介绍等字段。
- 设计字段时, 如果其值可有可无, 最好也给一个默认值, 并设成“不允许 NULL”。因为有些数据库在存放和查询有 NULL 的数据表时, 会花费额外的运算动作, 例如 SQL Server 数据库。

2. 适当地创建索引

- ☐ Primary Key 字段可以自动创建索引, 而 Foreign Key 字段不可以。为 Foreign Key 字段手动创建索引, 即使是很少被 JOIN 的数据表也有必要这样做。
- ☐ 为经常被查询或排序的字段创建索引。
- ☐ 创建索引字段的长度不宜过长, 不要用超过 20 个字节的字段, 例如地址等。
- ☐ 不要为内容重复性高的字段创建索引, 例如性别等。
- ☐ 不要为使用率低的字段建立索引。
- ☐ 不宜为过多字段建立索引, 否则会影响到 INSERT、UPDATE 和 DELETE 语句的性能。
- ☐ 如果数据表存放的数据很少, 就不必刻意使用索引。

3. 使用索引功能

在查询数据表时, 使用索引查询可以极大提升查询速度, 但是如果 where 子句书写不当, 即使某些列存在索引, 也不能使用该索引查询, 而同样会使用全表扫描, 这就造成了查询速度的降低。在 where 语句中避免使用以下关键词:

NOT、!=、<>、!>、!<、EXISTS、IN、LIKE、||。

使用 LIKE 关键字做模糊查询时, 即使已经为某个字段建立索引, 但需要以常量字符开头才会使用到索引, 如果以 “%” 开头则不会使用索引。例如 “name LIKE ‘%To’” 不启用 name 字段上的索引; 而 “name LIKE ‘To%’” 会启用 name 字段上的索引。

4. 避免在 where 子句中对字段使用函数

对字段使用函数, 也等于对字段做运算或连接的动作, 调用函数的次数与数据表的记录成正比。如果数据表内记录很多时, 会严重影响查询性能。例如下列 SQL 语句:

```
SELECT * FROM Orders WHERE DATEPART(yyyy, OrderDate) = 1996 AND DATEPART(mm, OrderDate) = 7
```

上述 SQL 语句可改成以下格式:

```
SELECT * FROM Orders WHERE CustomerID LIKE 'D%'
```

使用 UPDATE 和 DELETE 语句时, 如果有 WHERE 子句, 也应符合上述原则。

5. AND 与 OR 的使用

在 AND 运算中, 只要有一个条件使用到索引, 即可大幅提升查询速度。但在 OR 运算中, 则要所有的条件都有使用到索引才能提升查询速度, 因此使用 OR 运算符时需要特别小心。

6. JOIN 与子查询

相对于子查询, 如果能使用 JOIN 完成的查询, 一般建议使用后者。原因除了 JOIN 的语法较容易理解外, 在多数情况下, JOIN 的性能也会比子查询高。

7. 其他查询技巧

DISTINCT、ORDER BY 语法, 会让数据库做额外的计算。如果没有要过滤重复记录的需求, 使用 UNION ALL 会比 UNION 更好, 因为后者会加入类似 DISTINCT 的算法。

8. 尽可能使用存储过程 (Stored Procedure)

Stored Procedure 除了经过事先编译、性能较好以外, 也可减少 SQL 语句在网络中的传递, 方便商业逻辑的重复使用。

9. 尽可能在数据源过滤数据

使用 SELECT 语法时, 尽量先用 SQL 条件或 Stored Procedure 过滤出所要的信息, 避免将大量冗余数据返回给程序, 然后由程序处理。

疑难点评

通过采取上述方法可以使 SQL 语句得到一定优化, 提高其执行效率。提高程序的效率除了优化 SQL 语句之外, 与程序的处理逻辑、数据库表设计等方面也有直接的关系。如果要开发出高效的程序需要开发者对各方面知识都有深入的了解。

知识链接

FAQ9.04 如何实现对数据库数据的查询?

FAQ9.05 如何实现对数据库数据的增加、删除和修改?

FAQ9.33 如何解决 MySQL 数据库插入乱码的问题?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在使用 JDBC 访问 MySQL 数据库时, 对数据表添加和更新时容易出现乱码问题。解决办法如下。

(1) 设置连接字符串编码

在数据库连接字符串后面追加参数, 指明向 MySQL 服务器发送 SQL 语句的编码格式, 格式如下:

```
jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf-8
```

(2) 设置数据表及其字段的编码

将数据表的存储类型、表中字符字段的存储类型都设置成与连接字符串一致的编码。依据上述连接字符串示例, 数据表的存储编码应该设置成 UTF-8 类型。

(3) 设置其他编码

如果是从 JSP 页面取值, 然后使用 SQL 写入数据库, 那么还要保障从 JSP 页面取值正常。具体步骤如下。

首先在 JSP 页面中设置以下代码:

```
<%@ page language="java" pageEncoding="utf-8"%>
<%@ page contentType="text/html; charset=utf-8"%>
```

然后将 JSP 页面中 <form> 标记的 method 属性值指定为 POST 方式, 具体如下:

```
<form action=" " method="post">
```



```
...  
</form>
```

最后在使用 request.getParameter()方法获取 JSP 页面值之前, 设置 request 取值的编码, 具体如下:

```
request.setCharacterEncoding("UTF-8");  
String name = request.getParameter("name");  
String psw = request.getParameter("password");
```

疑难点评

使用 MySQL 数据库在处理中文字符时经常会发生乱码问题, 过上述解决方案可以解决从 JSP 页面取值和将信息存储 MySQL 数据库过程中遇到的中文乱码问题。在上述解决方案中, 各步骤的操作都必须使用相同的编码, 对中文操作可使用 UTF-8 或 GBK 编码, 建议采用 UTF-8 格式。

知识链接

FAQ9.05 如何实现对数据库数据的增加、删除和修改?



第10章

Java Web 程序设计

由于互联网的兴起,当前 B/S 结构的程序非常流行,例如网上商城系统、网上办公系统等。JSP 和 Servlet 是用来开发动态网站的核心技术,用于实现 B/S 结构的系统。本章重点介绍了 JSP 和 Servlet 技术的原理以及一些常用功能的实现,内容主要涉及 JSP 概念及原理、JSP 常用内置对象、如何使用 Session 和 Cookie、Servlet 概念及原理、如何实现报表、图标、发送邮件和打印等功能。通过本章的学习,读者不仅可以全面掌握开发动态网站的核心技术、常用功能的实现,也可以为以后学习 Struts、Hibernate 和 Spring 等 Java EE 框架奠定基础。

FAQ10.01 什么是 JSP? JSP 的工作原理如何?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

JSP 是 Java Server Pages 的简称,是由 Sun 公司于 1999 年推出的一种动态网页技术标准,利用这一技术可以建立安全、跨平台的先进动态网站。JSP 网页在传统的 HTML 网页中加入了 Java 程序片段和 JSP 标记,可根据用户的不同操作显示不同的结果。

JSP 和 ASP 技术非常相似,ASP 的编程语言是 VBScript,而 JSP 使用的是 Java。与 ASP 相比,JSP 以 Java 技术为基础,又在许多方面做了改进,具有动态页面与静态页面分离、能够脱离硬件平台的束缚、以及先编译后运行等优点。

JSP 程序的工作方式为请求/响应模式,客户端发出 HTTP 请求,JSP 程序收到请求后进行处理,并返回处理的结果,如图 10-1 所示。

JSP 程序需要运行于特定的 Web 服务器中,例如 Tomcat、WebLogic 等。所有 JSP 文件,在执行的时候都会被服务器端的 JSP 引擎转换为 Servlet 程序 (Java 源文件),然后调用



图 10-1 请求/响应模式

Java 编译器将 Servlet 程序编译为 class 文件（字节码文件），并由 Java 虚拟机（JVM）解释执行。JSP 的运行原理如图 10-2 所示。

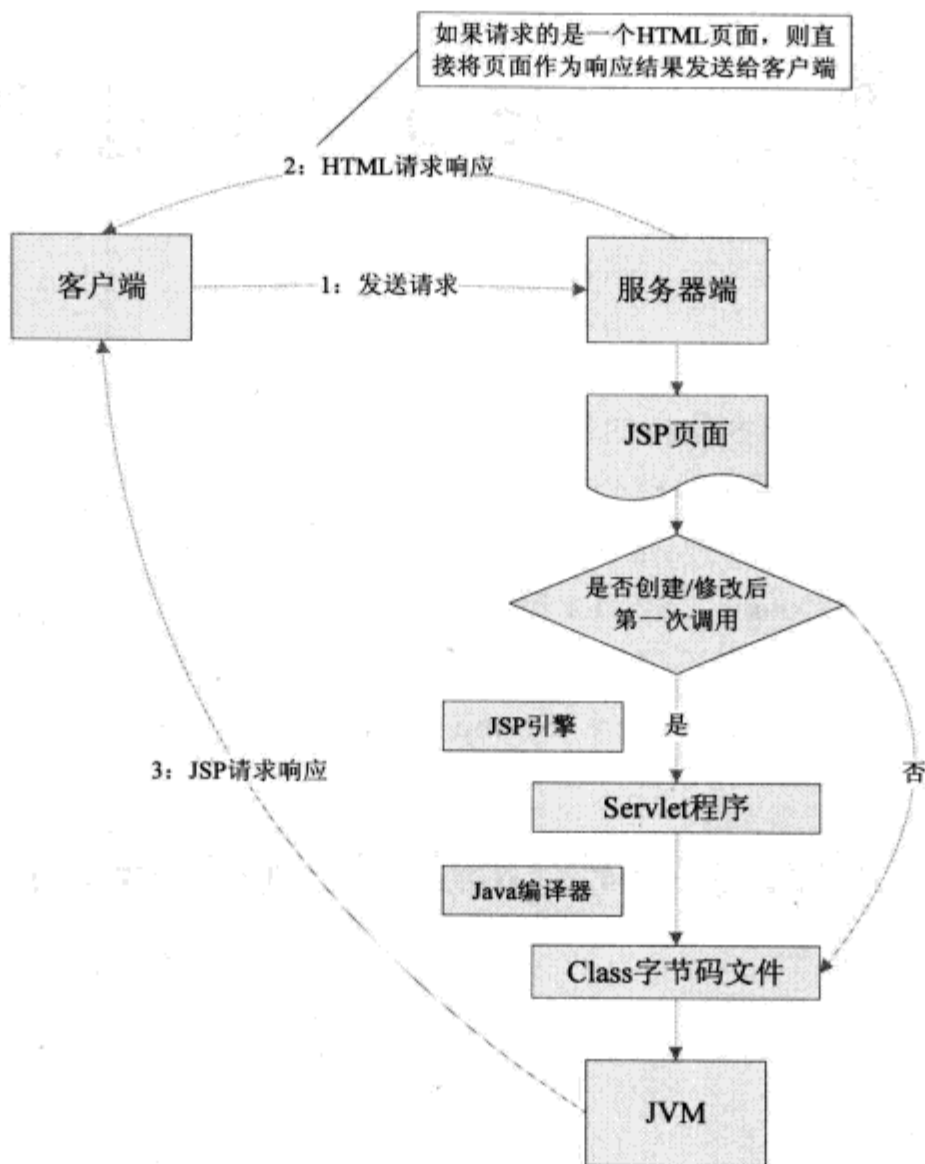


图 10-2 JSP 工作原理

当客户浏览器请求 Tomcat 服务器中的 JSP 页面（例如 Test.jsp）时，在 %CATALINA_HOME%\work\Catalina\ 目录下将生成两个文件，分别为 _Test_jsp.java 和 _Test_jsp.class，它们就是根据 JSP 页面产生的 Servlet 程序和 class 文件。

疑难点评

JSP 是开发动态网页的一种技术体系，用于开发基于 B/S 结构的 Java Web 程序。读者需要重点理解 JSP 程序的工作模式及其工作原理，这对后续深入学习 Java Web 编程有很大帮助。

知识链接

FAQ10.02 JSP、Java 和 JavaScript 有什么区别和联系？

FAQ10.03 JSP 程序开发和运行环境是什么？如何搭建？

FAQ10.02 JSP、Java 和 JavaScript 有什么区别和联系？

📖 难度系数：★★★

📖 问题频率：85%

核心解答

JSP 是由 Sun 公司倡导、许多公司参与一起建立的一种动态网页技术标准，用于编写动态网站程序。JSP 技术以 Java 语言作为脚本，嵌入到 JSP 页面中，由 Tomcat 等服务器负责解释运行。

Java 是 Sun 公司推出的新一代面向对象的程序设计语言，特别适合于 Internet 应用程序开发。Java 程序需要有 JRE 运行环境才能解释运行。

JavaScript 是由 Netscape 公司基于 Sun 的 Java 语法开发的。它是一种基于对象的脚本语言，可以在浏览器中直接运行，无需服务器的支持。该脚本可以直接嵌入在 HTML 代码中，用于增强网页特效，提高与终端用户之间的交互性能。

JavaScript 与 Java 是完全不同的两种语言，Java 属于面向对象的语言，而 JavaScript 是基于对象的函数式的语言，两者只是在结构和语法上相似而已。JavaScript 可以与 JSP、ASP 和 PHP 等动态网页技术结合，在客户端用于实现表单验证和网页特效功能。

疑难点评

JSP、Java 和 JavaScript 之间既有区别又有联系，因此很容易造成混淆。该问题将 JSP、Java 和 JavaScript 结合起来进行比较，这样有助于读者区分和理解这几个概念。

知识链接

FAQ10.01 什么是 JSP？JSP 工作原理如何？

FAQ10.03 JSP 程序开发和运行环境是什么？如何搭建？

📖 难度系数：★★★

📖 问题频率：95%

核心解答

JSP 程序开发和运行的基本环境需要有 JDK 和 Web 服务器（例如 Tomcat），在开发时，为了方便 JSP 程序的编写、调试、运行和部署，经常还会使用一些 IDE 集成开发工具，例如 MyEclipse、NetBean 和 JBuilder 等。

MyEclipse 和 Tomcat 服务器是最常用的两个工具,下面介绍如何搭建 JSP 的开发运行环境。

1. 安装 JDK

安装 JDK, 设置环境变量。过程请参考“如何安装 Java 基本开发环境 JDK”问题。

2. 安装 Tomcat

安装 Tomcat, 设置环境变量。具体过程如下。

下载 Tomcat 压缩文档 apache-tomcat-6.0.16.zip, 网址为 <http://tomcat.apache.org/index.html>。下载后将其解压到安装目录, 例如 D:\apache-tomcat-6.0.16。解压之后的目录结构如图 10-3 所示。

名称	修改日期	类型	大小
bin	2008/1/28 23:39	文件夹	
conf	2008/4/20 17:56	文件夹	
lib	2008/1/28 23:39	文件夹	
logs	2008/1/28 23:39	文件夹	
temp	2008/1/28 23:39	文件夹	
webapps	2008/1/28 23:39	文件夹	
work	2008/4/20 17:56	文件夹	
LICENSE	2008/1/28 23:39	文件	38 KB
NOTICE	2008/1/28 23:39	文件	1 KB
RELEASE-NOTES	2008/1/28 23:39	文件	8 KB
RUNNING	2008/1/28 23:39	文本文档	7 KB

图 10-3 Tomcat 的目录结构

配置与 Tomcat 相关的环境变量, 具体变量如表 10-1 所示。

表 10-1 环境变量

环境变量名称	环境变量值
TOMCAT_HOME	D:\apache-tomcat-6.0.16
CATALINA_HOME	%TOMCAT_HOME%

3. 安装 Eclipse 和 MyEclipse

MyEclipse 是一个插件集, 用于增强 Eclipse 开发平台。必须先安装 Eclipse 平台才能安装 MyEclipse 插件集。具体过程如下:

(1) 下载 eclipse-SDK-3.2.2-win32.zip 安装包, 并解压缩到某个目录 (例如 D 盘根目录)。网址为 <http://archive.eclipse.org/eclipse/downloads/drops/R-3.2.2-200702121330/index.php>。

(2) 下载 MyEclipse 安装程序 (MyEclipse_5.1.1GA_Installer.exe)。网址为 <http://www.myeclipseide.com/ContentExpress-display-ceid-10.html>。

(3) 双击 MyEclipse 的安装程序开始安装, 此时会显示“MyEclipse 介绍”对话框, 如图 10-4 所示。

(4) 从图 10-4 开始依次单击【Next】按钮, 直至出现“选择 Eclipse 安装目录”对话框, 如图 10-5 所示。

(5) 在图 10-5 中单击【Choose】按钮, 设定 Eclipse 解压的目录 (例如 D:\eclipse), 然后依次单击【Next】按钮并使用默认选项, 直至安装操作完成。

(6) 添加 Tomcat 配置。启动 Eclipse, 在菜单中依次选择【Window】→【Preferences...】

命令，弹出“Preferences”对话框，如图 10-6 所示。

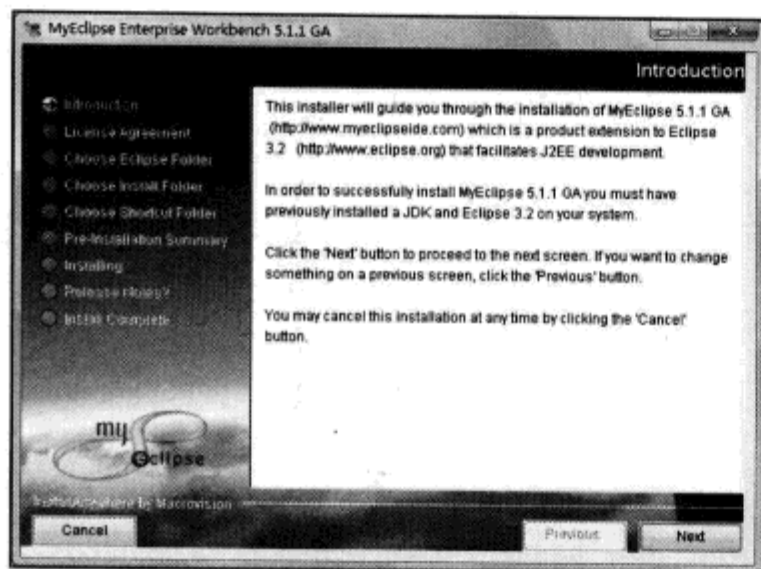


图 10-4 “MyEclipse 介绍”对话框

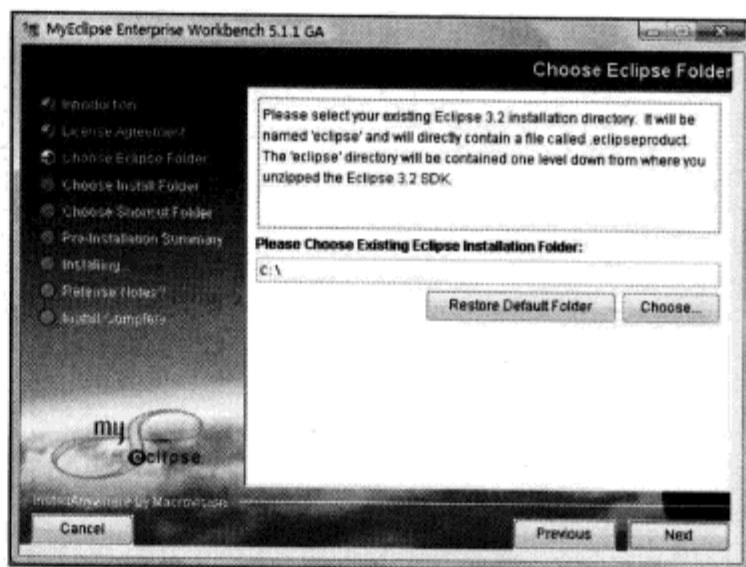


图 10-5 “选择 Eclipse 安装目录”对话框

注意：由于 Tomcat 启动需要 JDK，而 Eclipse 默认指向 JRE 路径，因此需要将 Eclipse 的 JRE 设置指向到 JDK 目录。

(7) 在图 10-6 中左侧列表，选择【Java】→【Installed JREs】命令，效果如图 10-7 所示。

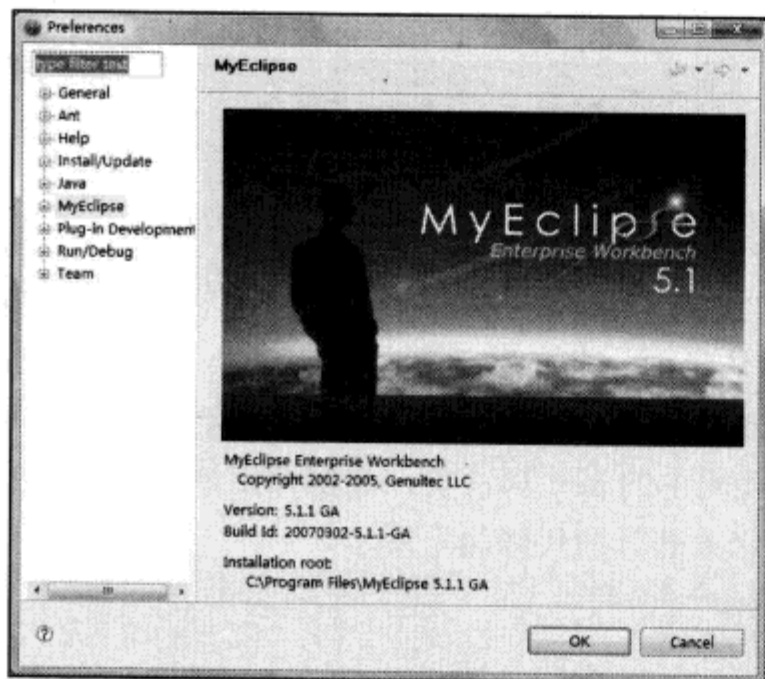


图 10-6 “Preferences”对话框

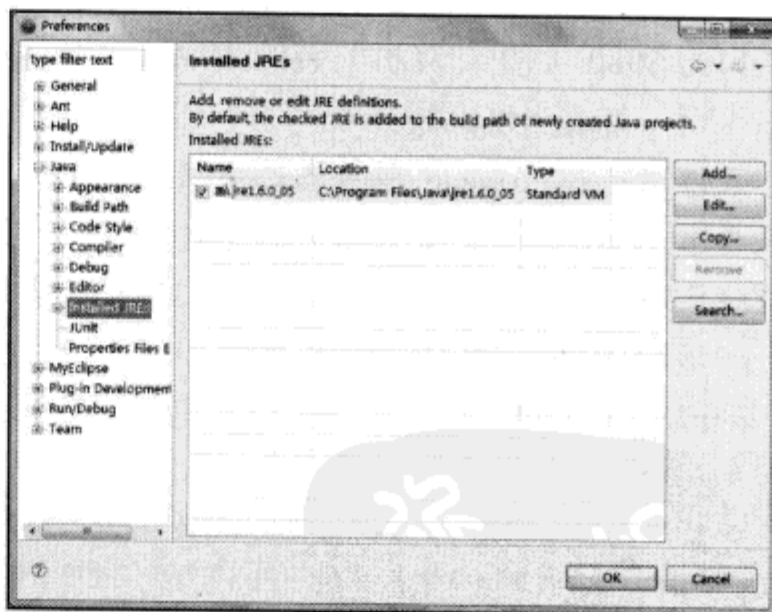


图 10-7 配置 JRE 对话框

(8) 在图 10-7 中，双击“jre1.6.0_05”，弹出“Edit JRE”对话框，将“JRE home directory”输入项指向 jdk1.6.0_05 所在目录，效果如图 10-8 所示。

(9) 单击图 10-8 和图 10-7 中的【OK】按钮，使 JRE 修改生效。

(10) 选择【Window】→【Preferences...】命令，弹出“Preferences”对话框。在左侧列表中选择【MyEclipse】→【Application Servers】→【Tomcat 6】命令，进入“设置 Tomcat”对话框。在对话框中选中“Enable”单选框，指定“Tomcat Home Directory”路径，效果如图 10-9 所示。

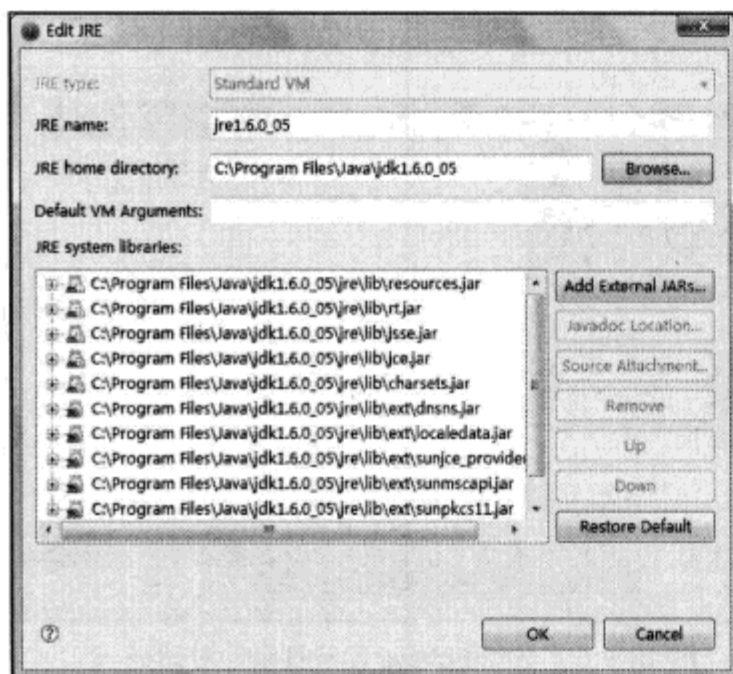


图 10-8 “Edit JRE” 对话框

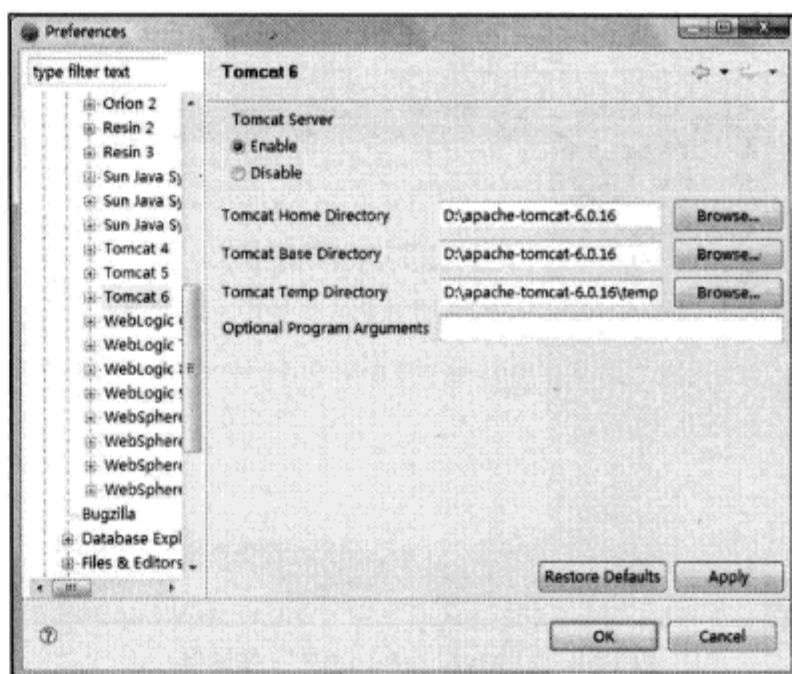


图 10-9 “设置 Tomcat” 对话框

(11) 单击对话框中的【OK】按钮，设置 Tomcat 完毕。

(12) 通过 Eclipse 工具栏按钮运行 Tomcat 服务器，进行测试，如图 10-10 所示。

在启动 Tomcat 服务器时，经常会出现“java.net.BindException: Address in use: JVM_Bind”异常。这是因为 Tomcat 服务器默认端口为 8080，如果计算机安装的其他软件将该端口占用，就会出现上述异常。

注意：在 Tomcat 服务器已启动情况下，如果再次启动 Tomcat 也会发生上述异常，因为端口被已启动的 Tomcat 占用，一个端口在某一时刻只能被一个应用程序占用。

解决上述异常的方法是更改 Tomcat 服务器的通信端口，修改 Tomcat 安装目录下的“/conf/server.xml”配置文件。打开该文件并修改代码中的 port 属性值，建议设定值大于 1 000，例如 6 666、7 777 或 8 888 等。server.xml 中需要修改部分的代码如下所示。



图 10-10 启动 Tomcat 服务器

```
<!-- A "Connector" represents an endpoint by which requests are received
and responses are returned. Documentation at :
Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
Java AJP Connector: /docs/config/ajp.html
APR (HTTP/AJP) Connector: /docs/apr.html
Define a non-SSL HTTP/1.1 Connector on port 8080
-->
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

通过上述方法可以修改 Tomcat 的通信端口，当浏览器访问服务器中的 JSP 页面时，需要使用修改后的端口号替换 URL 请求中的“8080”。

疑难点评

上面介绍了如何搭建 JSP 常用的开发和运行环境,在安装 MyEclipse 和 Eclipse 时需要注意版本问题,不同版本的 Eclipse 需要安装特定版本的 MyEclipse。另外目前较新版本的 MyEclipse 都包含了 Tomcat 工具,因此也可以不必再单独安装和配置 Tomcat 服务器。

知识链接

FAQ10.04 如何开发一款 JSP 程序?

FAQ10.04 如何开发一款 JSP 程序?

📖 难度系数: ★★★

📖 问题频率: 95%

核心解答

开发和运行 JSP 程序的基本环境为 JDK 和 Web 服务器,在编写 JSP 页面时,最简单的工具可以使用记事本等文本编辑器,也可以使用高效的 MyEclipse 等 IDE 工具。

开发 JSP 页面时,常用语法主要有以下几种。

- ☐ 编译器指引: `<%@ 编译器指引 %>`。
- ☐ 代码预定义: `<%! 预定义的变量或方法 %>`。
- ☐ 运算输出: `<%= 变量或运算表达式 %>`。
- ☐ 程序代码: `<% 程序代码 %>`。
- ☐ 注释: `<%-- 注释 --%>`。

下面分别介绍用记事本和 MyEclipse 工具开发 JSP 程序的具体过程。

1. 使用文本编辑器

文本编辑器种类很多,经常选用记事本、EditPlus 或 UltraEdit 等。开发 JSP 过程如下。

(1) 在 Tomcat 服务器的 webapps 目录下新建一个文件夹,作为 JSP 程序的工程目录。示例文件夹的名称为“first”。

(2) 在“first”文件夹中新建一个“WEB-INF”文件夹,并在该文件夹中加入一个名为 web.xml 配置文件。web.xml 文件的内容如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
</web-app>
```

(3) 在 first 文件夹下新建一个名为“hello.jsp”的文件,文件内容如下所示:

```
<%@ page language="java" pageEncoding="GBK"%>
<html>
  <head>
    <title>JSP 入门示例</title>
  </head>

  <body>
    <%
      //循环打印输出字符 A~Z
      for(char c='A';c<='Z';c++){
        //打印输出字符
        out.println(c);

      }
    %><br>
  </body>
</html>
```

注意：JSP 文件的扩展名必须为小写字符“.jsp”，否则 JSP 程序无法正常运行。

(4) 启动 Tomcat 服务器后，打开浏览器访问 hello.jsp 页面，请求 URL 的格式为“http://localhost:8080/first/hello.jsp”，效果如图 10-11 所示。如果服务器更改过端口，需要将 URL 中的“8080”替换为修改之后的端口。

2. 使用 MyEclipse 工具

使用 Eclipse 平台开发 JSP 程序的具体步骤如下。

(1) 打开 MyEclipse 开发平台，在菜单中依次选择【File】→【New】→【Project】命令，弹出“New Project”对话框，如图 10-12 所示。

(2) 在图 10-12 中选择【Web Project】命令，单击【Next】按钮将显示“New JZEE Web Project”对话框，如图 10-13 所示。

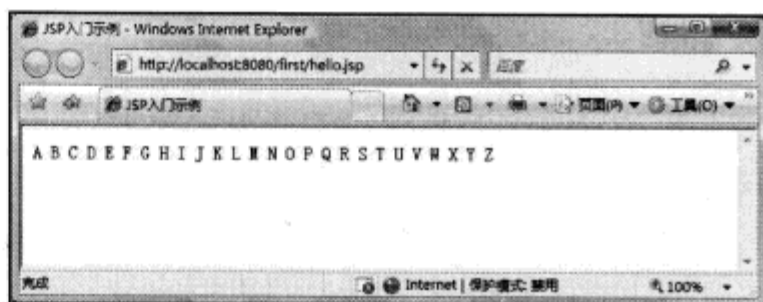


图 10-11 hello.jsp 页面运行效果图

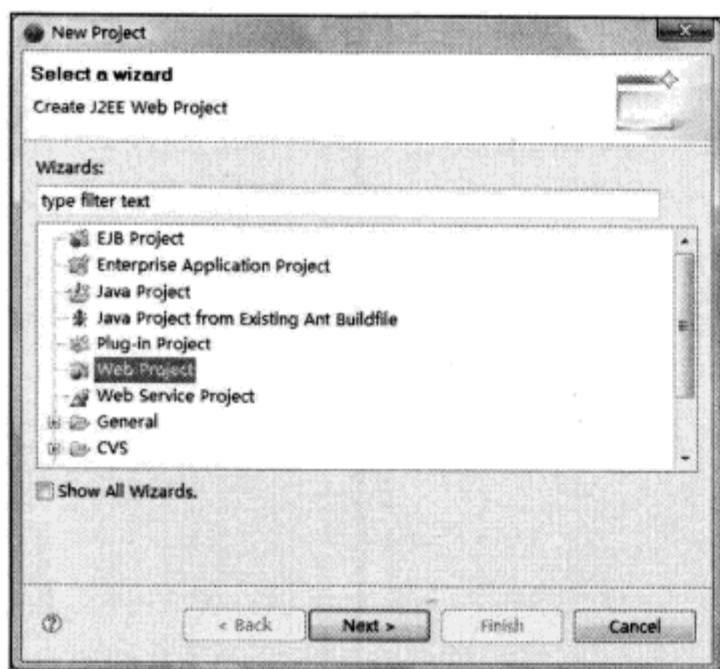


图 10-12 新建项目

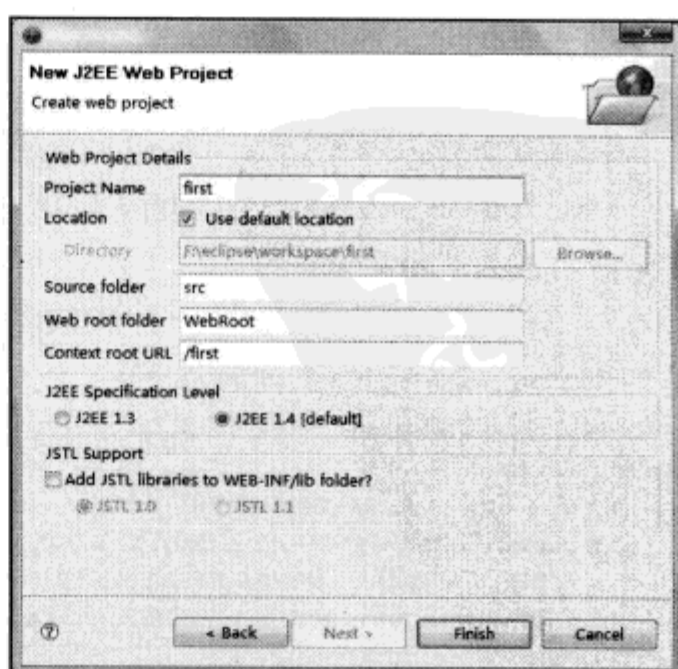


图 10-13 新建 Web Project

(3) 在图 10-13 中的“Project Name”文本框中输入 first, 其他采取默认值, 单击【Finish】按钮完成工程创建。

(4) 选中工程 WebRoot 目录, 单击鼠标右键, 依次选择菜单栏上的【New】→【JSP】命令, 如图 10-14 所示, 将会弹出“新建 JSP 文件”对话框, 如图 10-15 所示。

(5) 在图 10-15 中, 将“File Name”文本框的值设置为 hello.jsp, 单击【Finish】按钮完成 JSP 文件的创建。

(6) 修改 JSP 页面, 添加 Java 代码, 具体参见上面示例。

(7) 单击 Eclipse 的工具按钮【Deploy MyEclipse JAVA EE Project to Server...】, 显示“Project Deployments”对话框, 如图 10-16 所示。

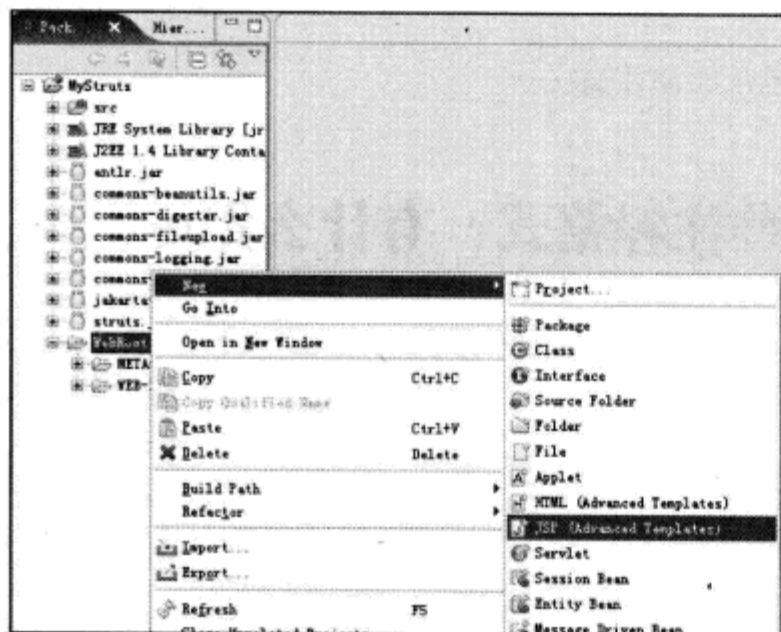


图 10-14 选择创建 JSP 命令效果图

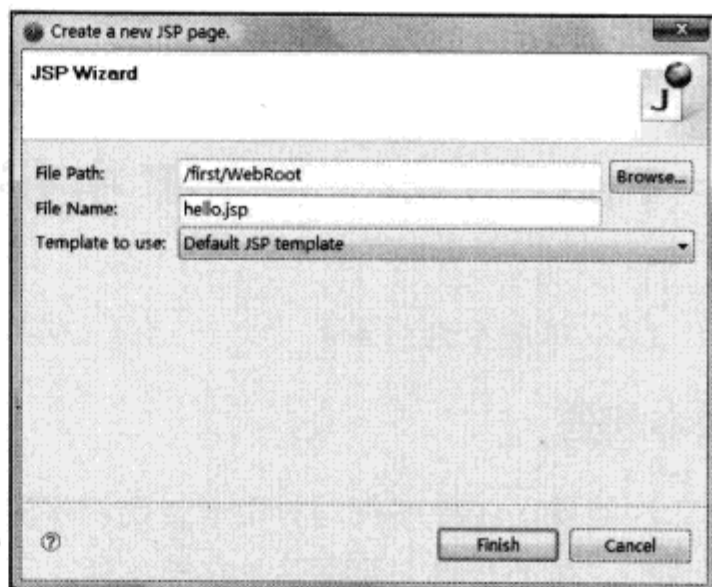


图 10-15 新建 JSP 文件对话框

(8) 在图 10-16 中单击【Add】按钮, 弹出“New Deployment”对话框, 在【Server】下拉列表框中选择 Tomcat 6, 单击【Finish】按钮, 如图 10-17 所示。

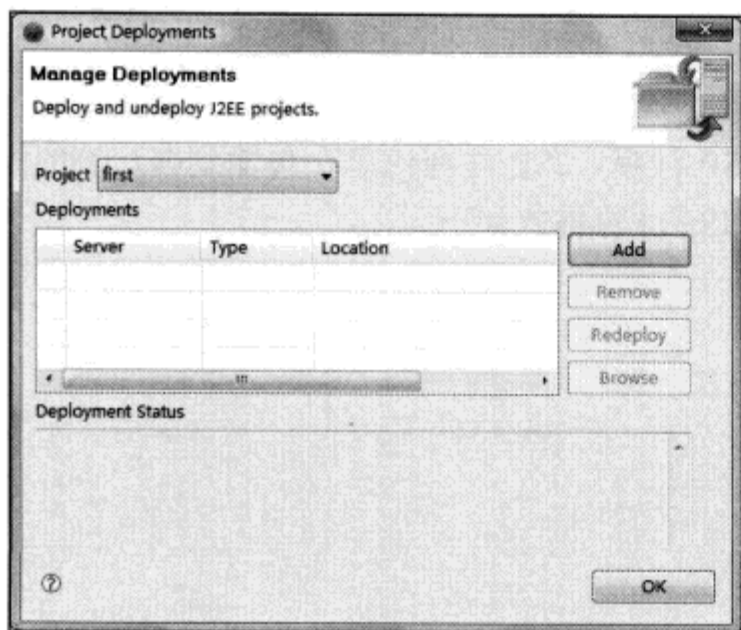


图 10-16 “Project Deployments”对话框

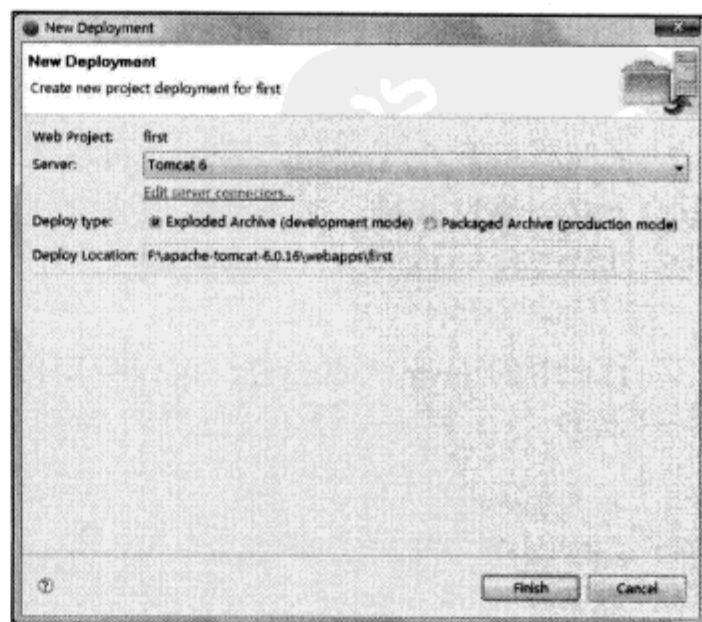


图 10-17 “New Deployment”对话框

(9) 在 Eclipse 工具按钮中依次选择【Run MyEclipse Server】→【Tomcat 6】→【Start】命令, 启动 Tomcat。

(10) 打开浏览器并输入网址 `http://localhost:8080/first/hello.jsp`, 运行效果与图 10-11 相同。

疑难点评

在企业中开发时, 为了提高软件的开发效率, 都会采用 MyEclipse 等集成环境。但读者在学习时, 建议先使用记事本等工具, 熟悉之后再使用集成工具, 这样可以摆脱对工具的依赖性, 对以后的学习和工作都会有很大的帮助。

知识链接

FAQ10.03 JSP 程序开发和运行环境是什么? 如何搭建?

FAQ10.05 在 JSP 中有哪些注释格式? 有什么作用?

📖 难度系数: ★★

📖 问题频率: 85%

核心解答

注释就是在程序代码中用来说明程序流程的语句, 注释语句可以帮助程序员识别和理解程序代码。在 JSP 中主要有以下几种注释。

(1) HTML 注释

HTML 注释信息将被发送到客户端, 但不直接显示, 用户单击浏览器主菜单【查看】→【源文件】, 在源代码中可以查看到。HTML 注释格式如下:

```
<-- 注释信息 -->
```

上述注释格式, 可以注释 HTML 代码。

(2) JSP 隐藏注释

如果注释信息不想让用户看到, 可以采用 JSP 注释。JSP 注释信息在传输到客户端的过程中会被 JSP 引擎过滤掉, 而不会发送到客户端。JSP 注释格式如下:

```
<%-- 注释信息 --%>
```

(3) 代码注释

在 JSP 中的 Java 代码部分, 可以使用以下格式注释:

```
//单行注释
```

```
/*
```

```
    多行注释
```

```
    多行注释
```

```
*/
```

上述单行和多行注释格式, 可用于注释 Java 代码, 也可用于注释 JavaScript 代码。

疑难点评

在 JSP 文件中包含了很多元素, 例如 HTML 代码、Java 代码和 JavaScript 代码等。通过上述注释格式, 可以实现代码的注释和说明等功能。

知识链接

FAQ10.04 如何开发一个 JSP 程序?

FAQ10.06 JSP 中有哪些内建对象? 分别有什么作用?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

在 JSP 页面中, 可以提供了 9 种内建对象, 具体说明如下:

(1) request 对象

request 代表请求对象, 主要用于接受客户端通过 HTTP 协议连接传输到服务器端的数据。它是 `HttpServletRequest` 类型的实例, 常用方法如下:

- ❑ `Object getAttribute(String name)` 返回指定属性的属性值;
- ❑ `Enumeration getAttributeNames()` 返回所有可用属性名的枚举;
- ❑ `String getCharacterEncoding()` 返回字符编码方式;
- ❑ `int getContentLength()` 返回请求体的长度 (以字节数);
- ❑ `String getContentType()` 得到请求体的 MIME 类型;
- ❑ `ServletInputStream getInputStream()` 得到请求体中一行的二进制流;
- ❑ `String getParameter(String name)` 返回 name 指定参数的参数值;
- ❑ `Enumeration getParameterNames()` 返回可用参数名的枚举;
- ❑ `String[] getParameterValues(String name)` 返回包含参数 name 的所有值的数组;
- ❑ `String getProtocol()` 返回请求用的协议类型及版本号;
- ❑ `String getScheme()` 返回请求用的计划名, 如: http.https 及 ftp 等;
- ❑ `String getServerName()` 返回接受请求的服务器主机名;
- ❑ `int getServerPort()` 返回服务器接受此请求所用的端口号;
- ❑ `BufferedReader getReader()` 返回解码过了的请求体;
- ❑ `String getRemoteAddr()` 返回发送此请求的客户端 IP 地址;
- ❑ `String getRemoteHost()` 返回发送此请求的客户端主机名;
- ❑ `void setAttribute(String key, Object obj)` 设置属性的属性值;

❑ `String getRealPath(String path)` 返回一虚拟路径的真实路径。

(2) response 对象

`response` 代表响应对象，主要用于向客户端发送数据。它是 `HttpServletResponse` 类型的实例，常用方法如下：

- ❑ `String getCharacterEncoding()` 返回响应用的是何种字符编码；
- ❑ `ServletOutputStream getOutputStream()` 返回响应的一个二进制输出流；
- ❑ `PrintWriter getWriter()` 返回可以向客户端输出字符的一个对象；
- ❑ `void setContentLength(int len)` 设置响应头长度；
- ❑ `void setContentType(String type)` 设置响应的 MIME 类型；
- ❑ `sendRedirect(java.lang.String location)` 重新定向客户端的请求。

(3) session 对象

`session` 代表客户端与服务器的会话，从客户连到服务器开始，直到客户端与服务器断开连接为止。`session` 对象主要用于保存用户信息，它是 `HttpSession` 类型的实例，常用方法如下：

- ❑ `long getCreationTime()` 返回 `session` 创建时间；
- ❑ `public String getId()` 返回 `session` 创建时 JSP 引擎为它设的惟一 ID 号；
- ❑ `long getLastAccessedTime()` 返回此 `session` 里客户端最近一次请求时间；
- ❑ `int getMaxInactiveInterval()` 返回两次请求间隔多长时间此 `session` 被取消，单位毫秒；
- ❑ `String[] getValueNames()` 返回一个包含此 `session` 中所有可用属性的数组；
- ❑ `void invalidate()` 取消 `session`，使 `session` 不可用；
- ❑ `boolean isNew()` 返回服务器创建的一个 `session` 客户端是否已经加入；
- ❑ `void removeValue(String name)` 删除 `session` 中指定的属性；
- ❑ `void setMaxInactiveInterval()` 设置两次请求间隔多长时间此 `session` 被取消，单位毫秒。

(4) out 对象

`out` 主要用于向客户端输出数据。它是 `JspWriter` 类型的实例，常用方法如下：

- ❑ `void clear()` 清除缓冲区的内容；
- ❑ `void clearBuffer()` 清除缓冲区的当前内容；
- ❑ `void flush()` 清空流；
- ❑ `int getBufferSize()` 返回缓冲区中字节数的大小，如不设缓冲区则为 0；
- ❑ `int getRemaining()` 返回缓冲区还剩余多少可用空间；
- ❑ `boolean isAutoFlush()` 返回缓冲区满时，是自动清空还是抛出异常；
- ❑ `void close()` 关闭输出流。

服务器端要输出到客户端的内容，不是直接写到客户端的，而是先写到一个输出缓冲区中，只有在下面 3 种情况下，才会把该缓冲区中的内容输出到客户端上。

- ❑ 该 JSP 网页已完成信息的输出。
- ❑ 输出缓冲区已满。

❑ JSP 中调用了 `out.flush()` 或 `response.flushbuffer()` 方法。

(5) page 对象

`page` 对象代表当前 JSP 页面本身, 有点类似于 `this` 关键字, 它是 `java.lang.Object` 类的实例, 平时很少使用。

(6) application 对象

`application` 对象实现了用户间数据的共享, 可存放全局变量。它开始于服务器的启动, 直到服务器的关闭, 在此期间, 此对象将一直存在。`application` 是 `ServletContext` 类型的实例, 常用方法如下:

- ❑ `Object getAttribute(String name)` 返回给定名的属性值;
- ❑ `Enumeration getAttributeNames()` 返回所有可用属性名的枚举;
- ❑ `void setAttribute(String name, Object obj)` 设定属性的属性值;
- ❑ `void removeAttribute(String name)` 删除一个属性及其属性值;
- ❑ `String getServerInfo()` 返回 JSP(SERVLET)引擎名及版本号;
- ❑ `String getRealPath(String path)` 返回一个虚拟路径的真实路径;
- ❑ `ServletContext getContext(String uripath)` 返回指定应用程序的 `application` 对象;
- ❑ `int getMajorVersion()` 返回服务器支持的 Servlet API 的最大版本号;
- ❑ `int getMinorVersion()` 返回服务器支持的 Servlet API 的最小版本号;
- ❑ `String getMimeType(String file)` 返回指定文件的 MIME 类型;
- ❑ `URL getResource(String path)` 返回指定资源(文件及目录)的 URL 路径;
- ❑ `InputStream getResourceAsStream(String path)` 返回指定资源的输入流;
- ❑ `RequestDispatcher getRequestDispatcher(String uripath)` 返回指定资源的 `RequestDispatcher` 对象;
- ❑ `Servlet getServlet(String name)` 返回指定名的 Servlet;
- ❑ `Enumeration getServlets()` 返回所有 Servlet 的枚举;
- ❑ `Enumeration getServletNames()` 返回所有 Servlet 名的枚举;
- ❑ `void log(String msg)` 把指定消息写入 Servlet 的日志文件;
- ❑ `void log(Exception exception, String msg)` 把指定异常的栈轨迹及错误消息写入 Servlet 的日志文件;
- ❑ `void log(String msg, Throwable throwable)` 把栈轨迹及给出的 `Throwable` 异常的说明信息写入 Servlet 的日志文件。

(7) exception 对象

`exception` 对象是一个异常对象, 当一个页面在运行过程中发生了异常时, 就产生该对象。`exception` 是 `java.lang.Throwable` 类型的实例, 常用方法如下:

- ❑ `String getMessage()` 返回描述异常的消息;
- ❑ `String toString()` 返回关于异常的简短描述消息;

- ☐ void printStackTrace() 显示异常及其栈轨迹;
- ☐ Throwable fillInStackTrace() 重写异常的执行栈轨迹。

如果一个 JSP 页面要应用此对象,就必须在 Page 指令中添加“isErrorPage=true”属性,否则无法编译。

(8) pageContext 对象

pageContext 对象提供了对 JSP 页面内所有的对象及名字空间的访问,使用它可以访问 session、request 和 application 等对象中的内容。pageContext 是 PageContext 类型的实例,常用方法如下:

- ☐ JspWriter getOut() 返回当前客户端响应被使用的 JspWriter 流 (out);
- ☐ HttpSession getSession() 返回当前页中的 HttpSession 对象 (session);
- ☐ Object getPage() 返回当前页的 Object 对象 (page);
- ☐ ServletRequest getRequest() 返回当前页的 ServletRequest 对象 (request);
- ☐ ServletResponse getResponse() 返回当前页的 ServletResponse 对象 (response);
- ☐ Exception getException() 返回当前页的 Exception 对象 (exception);
- ☐ ServletConfig getServletConfig() 返回当前页的 ServletConfig 对象 (config);
- ☐ ServletContext getServletContext() 返回当前页的 ServletContext 对象 (application);
- ☐ void setAttribute(String name, Object attribute) 设置属性及属性值;
- ☐ void setAttribute(String name, Object obj, int scope) 在指定范围内设置属性及属性值;
- ☐ public Object getAttribute(String name) 取属性的值;
- ☐ Object getAttribute(String name, int scope) 在指定范围内取属性的值;
- ☐ public Object findAttribute(String name) 寻找一个属性,返回其属性值或 NULL;
- ☐ void removeAttribute(String name) 删除某属性;
- ☐ void removeAttribute(String name, int scope) 在指定范围删除某属性;
- ☐ int getAttributeScope(String name) 返回某属性的作用范围;
- ☐ Enumeration getAttributeNamesInScope(int scope) 返回指定范围内可用的属性名枚举;
- ☐ void release() 释放 pageContext 所占用的资源;
- ☐ void forward(String relativeUrlPath) 使当前页面重导到另一页面;
- ☐ void include(String relativeUrlPath) 在当前位置包含另一文件。

(9) config 对象

config 对象是在一个 Servlet 初始化时, JSP 引擎向它传递信息用的,此信息包括 Servlet 初始化时所要用的参数以及服务器的有关信息。config 是 ServletConfig 类型的实例,常用方法如下:

- ☐ ServletContext getServletContext() 返回含有服务器相关信息的 ServletContext 对象;
- ☐ String getInitParameter(String name) 返回初始化参数的值;
- ☐ Enumeration getInitParameterNames() 返回 Servlet 初始化所需所有参数的枚举。

疑难点评

JSP 提供的 9 种内建对象,是为了方便开发者使用事先在 JSP 内部定义好的。读者在学习这几个对象时,可从每个对象的作用、对象类型及其常用方法等几个方面入手。其中 out、request、response 和 session 是使用次数最多的,可作为重点进行学习。

知识链接

FAQ10.07 page、request、session 和 application 有什么区别?

FAQ10.07 page、request、session 和 application 有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 98%

核心解答

page、request、session 和 application 对象的区别主要有以下几点:

(1) 类型不同

page 是 Object 类型; request 是 HttpServletRequest 类型; session 是 HttpSession 类型; 而 application 是 ServletContext 类型。

(2) 作用范围不同

上述 4 个对象都可以存储信息,但是它们的作用范围不同,具体如下:

application: 全局作用范围,整个应用程序共享。生命周期为从应用程序启动到停止。

session: 会话作用域,当用户首次访问时,产生一个新的会话,以后服务器就可以记住这个会话状态。生命周期为会话超时或者服务器端强制使会话失效。

request: 请求作用域,客户端的一次请求。生命周期为一次请求或使用 forward 方式执行请求转发。

page: 一个 JSP 页面有效。

page、request、session 和 application 对象作用范围是越来越大, request 和 page 的生命周期都是短暂的,它们之间的区别就是: 一个 request 可以包含多个 page 页(include、forward 以及 filter)。

疑难点评

该问题重点介绍了 page、request、session 和 application 对象作用范围的区别。

知识链接

FAQ10.06 JSP 中有哪些内建对象？分别有什么作用？

FAQ10.08 如何解决 request.getParameter()取值乱码问题？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

客户端向服务器端发送信息，如果发送的编码和服务器接收编码不一致时，使用 request.getParameter()方法获取的表单信息会产生乱码。

客户端接收服务器的响应信息，如果响应编码和客户浏览器的编码不一致时，会造成中文乱码显示。

为了解决中文正常显示，可以采取以下两种方案。

(1) POST 方式提交表单

❑ 设置页面的编码

在 JSP 或 HTML 页面中，有时会涉及编码的定义，可使用统一编码，例如 GBK 或 UTF-8。

❑ 设置表单提交方式

在页面中，将<form>标记的 method 属性值设置为 POST，该属性默认为 GET。GET 方式会将表单信息采取特殊编码，然后借助于 URL 发送给服务器。

❑ 设置服务器接收编码

在使用 request.getParameter()方法接收客户表单信息之前，使用以下代码设置接收编码，编码与页面编码保持一致。

```
request.setCharacterEncoding("GBK");
```

使用 setCharacterEncoding()方法指定编码后，可以通过 getParameter()方法按指定编码获得客户信息，如果不指定，则默认使用 ISO-8859-1 编码。

❑ 设置服务器响应信息编码

设置服务器向客户端响应的信息编码，告诉浏览器采用指定编码显示，可以使用下列指令或代码。

在响应的 JSP 中，使用如下指令设置：

```
<%@ page contentType="text/html; charset=GBK" %>
```

或者使用如下代码设置：

```
response.setContentType("text/html; charset=GBK");
```

(2) GET 方式提交表单

使用 GET 方式提交表单时,可以使用以下代码将获取的表单信息进行编码转换。代码如下:

```
String title = request.getParameter("title");  
String mytitle = new String(title.getBytes("ISO-8859-1"),("GBK"));
```

服务器响应编码的设置与 POST 方式介绍的相同。

疑难点评

在使用 request.getParameter()方法获取客户提交的表单信息时,根据客户提交表单信息的方式不同,需要采取不同的解决方法,在解决时,JSP 页面和代码中使用的编码要统一。

根据 JavaDoc 帮助文档提示,在执行 setCharacterEncoding()之前,不要使用 getParameter()方法,否则指定编码无效。而且,该方法只对 POST 方式提交表单有效,对 GET 方式无效。分析原因,应该是在执行第一个 getParameter()的时候,Java 将会按照编码分析所有的提交内容,而后续的 getParameter()不再进行分析,所以 setCharacterEncoding()无效。而对于 GET 方法提交表单,提交的内容在 URL 中,一开始就已经按照编码分析所有的提交内容,setCharacterEncoding()自然就无效,因此需要使用代码进行编码转换才能解决。

知识链接

FAQ10.11 如何解决 URL 传递中文时出现乱码的问题?

FAQ10.09 JSP 中 forward 和 redirect 有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

(1) forward 跳转

实现 forward 跳转的方法如下:

在 JSP 页面中可以使用<jsp:forward>指令,代码如下:

```
<jsp:forward page="a.jsp">
```

在 Java 代码中,使用 requestDispatcher 对象也可以实现 forward 跳转,代码如下:

```
request.getRequestDispatcher("/a.jsp").forward(request,response);
```

forward 可实现服务器端请求的转发,其工作原理如图 10-18 所示:

在上述图中,由 a.jsp 到 b.jsp 采用 forward 跳转时,客户浏览器不知道该跳转过程,因此客户浏览器中虽然显示的内容为 b.jsp,但是 URL 还是 a.jsp 的请求路径。

此外,使用 forward 由 a.jsp 跳转到 b.jsp 后,request 请求对象使用的是同一个,因此可以在 b.jsp 中使用的 request 对象获取前一次请求的客户信息,也可以利用 request 对象从 a.jsp 往 b.jsp 中传值。

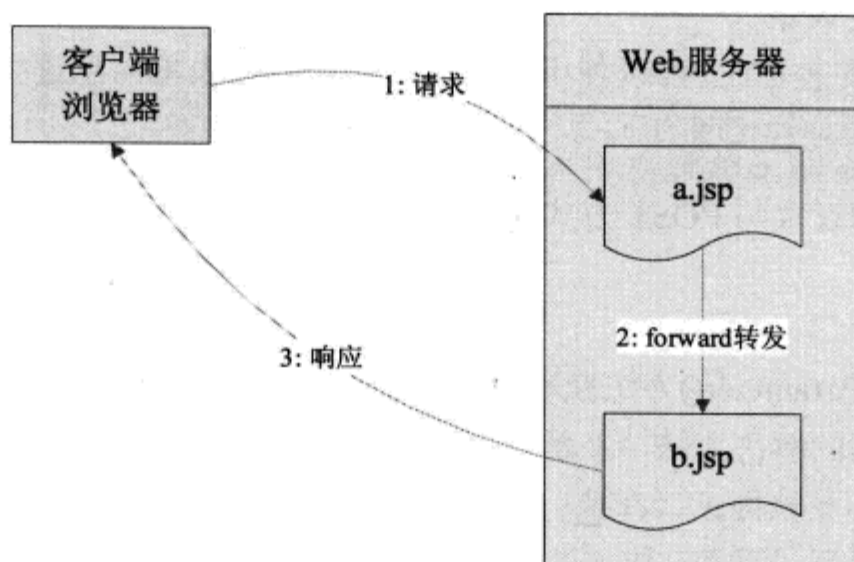


图 10-18 forward 工作原理

注意: forward 表示请求转发, 使用该方式实现页面跳转时, request 对象与前一次相同, request 对象无法跨站点使用, 因此 forward 方法无法实现跨站点跳转。要实现跨站点跳转, 读者可使用其它方法, 例如 Java 的 redirect 方式、JavaScript 或 HTML 的<meta>标记。

(2) redirect 跳转

在 Java 代码中, 使用 response 对象的 sendRedirect() 方法可以实现 redirect 跳转, 代码如下:

```
response.sendRedirect("/a.jsp");
```

redirect 可实现服务器端请求的重定向, 其工作原理如图 10-19 所示:

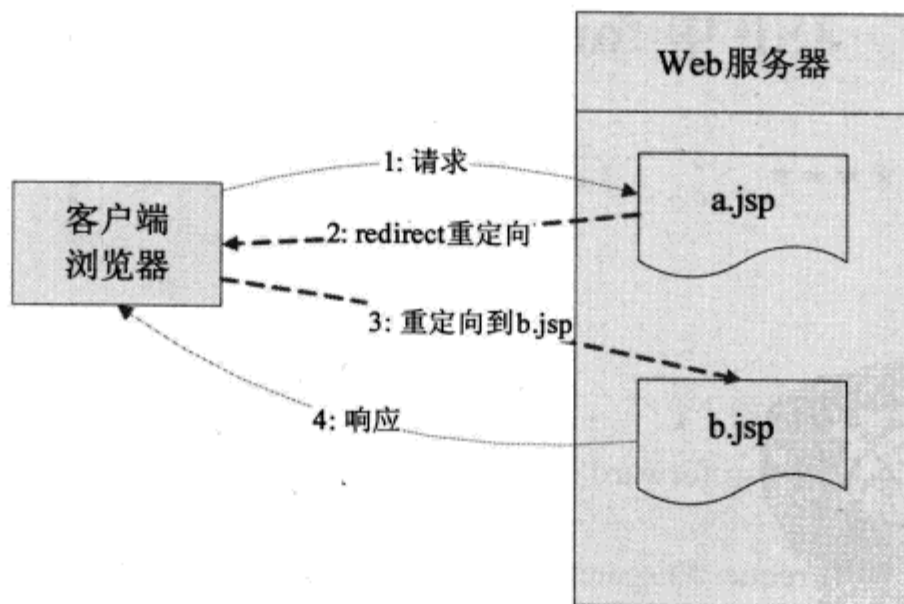


图 10-19 redirect 工作原理

在上述图中, 由 a.jsp 到 b.jsp 采用 redirect 跳转时, 服务器端根据逻辑向客户端发送一个状态码, 告诉浏览器重新去请求那个地址, 然后客户端浏览器重新发出请求地址, 因此浏览器的 URL 为 b.jsp 的请求路径, 即重定向之后的请求路径。

此外, 在实现 redirect 跳转过程中, 浏览器又重新发出了一次重定向请求, 服务器会将 a.jsp 使用的 request 对象销毁, 并重新创建一个 request 对象传递给 b.jsp。因此使用 redirect 跳转时,

无法使用 request 对象从 a.jsp 往 b.jsp 中传值。

疑难点评

forward 和 redirect 方式虽然都可以实现页面跳转,但是跳转时的工作原理不同。使用 forward 跳转时,浏览器请求 URL 不会改变,request 对象不会被销毁;使用 redirect 跳转时,浏览器请求的 URL 会改变,request 对象会被销毁并重新创建。在使用 request 对象在页面传值时,需要使用 forward 方式,而不能使用 redirect 方式。

知识链接

FAQ10.10 如何在多个 JSP 页面之间传递信息?

FAQ10.27 如何实现网站登录记忆跳转的功能?

FAQ10.10 如何在多个 JSP 页面之间传递信息?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

跨多个页面传值,主要有以下几种实现方案,可根据具体的实际情况进行选择。

(1) 使用 URL

在发送请求时,可以借助于 URL 进行传值。URL 示例格式如下:

```
http://localhost:8080/javafaq/hello.jsp?name=tom&age=11
```

在 URL 传递多个值时,使用&符号进行分割。如果传递中文会发生乱码问题,解决方法可参考 FAQ10.11 “如何解决 URL 传递中文时出现乱码的问题”。

获取上述 URL 中信息的代码如下:

```
String n = request.getParameter("name");  
String a = request.getParameter("age");
```

(2) 使用 request 对象

当多个页面之间使用 forward 方式跳转时,可以使用 request 对象传递。request 对象的使用代码如下所示:

```
//往 request 对象中设置信息  
request.setAttribute("name","tom");  
request.setAttribute("age",20);  
//获取 request 对象中的信息  
String n = request.getAttribute("name");  
Integer a = (Integer)request.getAttribute("age");
```

如果 setAttribute()方法是用了重复的属性名,那么后面的属性值会将前面的设置覆盖。任何类型的对象,在存入 request 对象后,都变为 Object 类型,因此使用 getAttribute()方法获取时,

需要类型强制转化。

(3) 使用 session 对象

使用 session 对象在多个页面之间传值,是最常用的方法。每个用户从访问系统开始到离开为止,服务器会为该用户创建一个 session 对象,用于跟踪保存该用户的信息。每个用户只能访问自己的 session 对象,不能相互访问。session 对象的使用代码如下所示:

```
//往 session 对象中设置信息
session.setAttribute("name","tom");
session.setAttribute("age",20);
//获取 session 对象中的信息
String n = session.getAttribute("name");
Integer a = (Integer)session.getAttribute("age");
```

session 的使用方式与 request 相似,在获取信息时也需要使用类型强制转化。

(4) 使用 application 对象

如果在多个页面中传递的信息需要多个用户之间共享,可以使用 application 对象。整个应用程序只有一个 application 对象,每个用户都可以使用该对象。application 对象的使用代码如下:

```
//往 application 对象中设置信息
application.setAttribute("name","tom");
application.setAttribute("age",20);
//获取 application 对象中的信息
String n = application.getAttribute("name");
Integer a = (Integer)application.getAttribute("age");
```

在应用 application 时,需要考虑并发问题。可以使用 synchronized 关键字进行同步,示例代码如下:

```
synchronized(application){
//使用 application 对象的代码
}
```

疑难点评

利用 URL、request、session 和 application 都可以实现跨页面传值,如果相邻页面传值,可以考虑使用 URL 或 request;如果是单用户跨多个页面传值,可以考虑使用 session;如果是多用户跨多个页面传值,可以考虑使用 application。

知识链接

FAQ10.09 JSP 中 forward 和 redirect 有什么区别?

FAQ10.11 如何解决 URL 传递中文时出现乱码的问题?

FAQ10.11 如何解决 URL 传递中文时出现乱码的问题?

📖 难度系数: ★★★★★

📖 问题频率: 98%

核心解答

在使用 URL 传递中文信息时,有时会出现接收方得到的是乱码的情况,具体解决方法如下。

□ 方案一

在发送请求时, URL 格式如下:

```
http://localhost:8080/javafaq/hello.jsp?name=张三
```

在接收时,进行编码转换,示例如下:

```
String name = request.getParameter("name");
String myname = new String(name.getBytes("ISO-8859-1"), "GBK");
```

□ 方案二

发送请求时,将中文先用 `URLEncoder` 类的 `encode()` 方法编码,示例如下:

```
http://localhost:8080/javafaq/hello.jsp?name=<%=URLEncoder.encode("张三","GBK")%>
```

在接收时,用 `URLDecoder` 类的 `decode()` 方法解码,示例如下:

```
String name = request.getParameter("name");
String myname = URLDecoder.decode(name, "GBK");
```

修改 Tomcat 服务器的 `server.xml` 配置文件,将配置文件中的 `URIEncoding` 属性设置为“GBK”,示例如下:

```
<Connector port="8080"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" redirectPort="8443" acceptCount="100"
debug="0" connectionTimeout="20000" useBodyEncodingForURI="true"
disableUploadTimeout="true" URIEncoding="GBK"/>
```

疑难点评

Tomcat 服务器默认使用的编码是 ISO-8859-1,利用 URL 传递中文或使用 GET 方式提交表单都会产生乱码问题。因此,利用 URL 传值时,不建议传递中文字符;另外表单提交方式也建议使用 POST,因为 GET 方式提交表单信息也是利用 URL 传值。

知识链接

FAQ10.08 如何解决 `request.getParameter()` 取值乱码问题?

FAQ10.10 如何在多个 JSP 页面之间传递信息?

FAQ10.12 动态 include 与静态 include 有什么区别?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

`<jsp:include>` 和 `<%@include%>` 都可以实现页面引入功能,一般情况下可以通用,但是两者

也存在一些区别,有时必须分情况使用。

□ `<%@include%>`

`<%@ include%>`被称为静态引入,仅在第一次运行的时候,将页面内容引入当前页面。如果被包含的文件发生改变,而当前 JSP 没有更新,那么就不会用到更新过的被包含文件。使用示例如下:

```
<%@include file="a.jsp"%>
```

`<%@ include%>`主要用于引入 HTML 静态页面和共通的 JSP 源代码,例如包含验证代码的 JSP 页面。

□ `<jsp:include/>`

`<jsp:include/>`被称为动态引入,每次运行的时候都加载,所以如果被包含文件改变了,这个文件就会使用已经更新过的文件。因为每次都要加载,所以会比前者有更大的性能负担。使用示例如下:

```
<jsp:include page="a.jsp"/>
```

`<jsp:include/>`主要用于引入动态变化的 JSP 页面,例如列表显示功能等。

疑难点评

`<jsp:include>`和`<%@include%>`都可以实现在当前 JSP 页面中引入另一个页面,`<%@ include%>`主要用于引入 HTML 静态页面和共通的 JSP 源代码;而`<jsp:include/>`主要用于引入动态变化的 JSP 页面。

FAQ10.13 什么是 JavaBean? 如何使用 JavaBean?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

JavaBean 就是一个符合 Javabeen 规范的 Java 类,可用于封装一些共通的业务逻辑,从而实现重复利用。JavaBean 必须符合以下 3 点。

- 放在一个包(package)中。
- JavaBean 类必须要提供一个无参的构造方法。在 JSP 中使用`<jsp:useBean/>`创建 JavaBean 对象时会使用无参的构造方法。
- JavaBean 类不要定义公共类型的属性,避免外界直接访问实例变量,变量名称首字母必须小写。
- JavaBean 类通过 `gettero/setter()`方法来读/写属性的值,并且将对应的属性首字母改成大写。注意使用 `setter()`时 `value` 属性的类型要匹配。

JavaBean 组件与 EJB (Enterprise JavaBean) 组件完全不同,EJB 是 JavaEE 的核心,是用于

创建分布式应用的组件模型, 比 JavaBean 要复杂得多, 需要专门的 EJB 容器支持。

JSP 提供了一些标记, 用于实现在 JSP 页面中使用 JavaBean, 示例代码如下:

```
<jsp:useBean id="db" scope="request" class="util.DBUtil"/>
<jsp:setProperty name="db" property="balance" value="0.0" [param="m_balance"]/>
<jsp:getProperty name="db" property="balance"/>
```

`<jsp:useBean/>` 标记用于创建一个 JavaBean 对象, 其中 `scope` 属性用于指定对象的存储范围, 可以使用 `page`、`request`、`session` 或 `application`; `class` 属性用于指定 JavaBean 的类型。

`<jsp:setProperty/>` 标记用于给 JavaBean 对象中的属性赋值, `name` 属性用于指定对象名; `property` 属性用于指定属性名; `value` 属性用于指定属性值; `param` 可选, 当表单组件的 `name` 属性值与 JavaBean 对象中的属性名不一致时, 可使用 `param` 属性指定组件的属性名。可以使用以下指令, 将表单提交的信息批量地为 JavaBean 对象的属性赋值。

```
<jsp:setProperty name="db" property="*" />
```

`<jsp:getProperty/>` 标记用于获取 JavaBean 属性值并显示, `name` 属性用于指定对象名; `property` 属性用于指定属性名。

如果 JavaBean 中有其他的业务方法, 可以在 Java 代码中使用 `<jsp:useBean/>` 标记的 `id` 属性值引用。示例代码如下:

```
<jsp:useBean id="db" scope="request" class="util.DBUtil"/>
<%
    db.execute("select * from emp");
    //其他处理代码
    con.close();
%>
```

在上述示例中, 通过使用 `<jsp:useBean/>` 标记创建了一个 JavaBean 对象。此外, 也可以使用代码方式创建 JavaBean 对象。示例代码如下:

```
<%
    util.DbUtil db = new util.DbUtil();
    db.execute("select * from emp");
    //其他处理代码
    con.close();
%>
```

疑难点评

JavaBean 可以实现重复利用, 将业务代码和 JSP 页面进行有效分离。在开发一些小型系统时, JSP+JavaBean 开发模式非常流行。

FAQ10.14 什么是 Session? 如何使用 Session?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

Session 的意思是“会话”，表示客户与服务器进行交互的时间间隔。Session 开始于用户进入网站，结束于关闭浏览器离开网站。在客户和服务器“会话”期间，可以利用 Session 保存用户的信息，实现用户信息在系统各页面之间的传递。

Session 对象存储于 Web 服务器内存中，并与用户相关，不同用户具有不同的 Session 对象，每个 Session 对象的信息只能由特定用户使用。

1. 使用 Session 存取信息

在 JSP 中 Session 可以通过 session 内建对象直接使用，而在 Servlet 中，可使用下列代码获取：

```
HttpSession session = request.getSession();
```

写入 Session 的示例代码如下：

```
session.setAttribute("name","tom");//写入一个 String 类型的值  
session.setAttribute("list",new ArrayList());//写入一个 ArrayList 对象
```

读取 Session 的示例代码如下：

```
String n = (String)session.getAttribute("name");//获取 String 类型的值  
ArrayList l = (ArrayList)session.getAttribute("list");//获取 ArrayList 对象
```

2. 设置 Session 有效期

Session 对象存储在服务器内存中，用于跟踪存储用户信息。Session 对象默认有效期一般在 30 分钟左右，如果用户在该时间内不做任何操作，服务器会将 Session 对象自动销毁。开发者可以对 Session 对象的有效期限进行设定，具体方法如下。

(1) 在 Tomcat 的 server.xml 中定义

通过 Tomcat 的 conf/server.xml 文件，可以修改服务器上所有程序的默认有效期，设置单位为毫秒，定义代码如下：

```
<Context path="/livsorder" docBase="/home/httpd/html/livsorder"  
    defaultSessionTimeout="3600" isWARExpanded="true"  
    isWARValidated="false" isInvokerEnabled="true"  
    isWorkDirPersistent="false"/>
```

(2) 在工程的 web.xml 中定义

通过工程的 web.xml 文件，可以修改当前程序的默认有效期，设置单位为分钟，定义代码如下：

```
<session-config>  
    <session-timeout>30</session-timeout>  
</session-config>
```

(3) 通过 Java 代码设定

通过 HttpSession 类的 setMaxInactiveInterval() 方法可以设定 Session 对象的有效期限，设置单位为秒，设置为 -1 永不过期。示例代码如下：

```
HttpSession ses = request.getSession();  
ses.setMaxInactiveInterval(10);
```

3. 判断 Session 是否失效

通过 request 对象的 getSession() 方法可以获取 Session 对象，getSession() 方法的定义如下：

```
HttpSession getSession()  
HttpSession getSession(boolean create)
```

getSession()方法的作用是返回一个 Session 对象, 如果 Session 不存在则创建一个并返回。而 getSession(boolean create)方法当参数值为 true 时, 如果 Session 不存在则创建一个并返回; 当参数值为 false 时, 如果 Session 不存在则返回 null。

通过 getSession(boolean create)方法的返回值是否为 null 来判断 Session 是否过期。示例代码如下:

```
if(request.getSession(false)==null)  
    System.out.println("Session has been invalidated!");  
else  
    System.out.println("Session is active!");
```

疑难点评

在用户进入和退出系统之间, 各页面都共享同一个 Session 对象。在写入 Session 时, 任何类型的信息都变为 Object 类型, 在读取 Session 时, 需要经过强制转化才能变为原来类型。

知识链接

FAQ10.15 如何在关闭页面时自动清除 Session?

FAQ10.17 如何在禁用 Cookie 的情况下使用 Session?

FAQ10.15 如何在关闭页面时自动清除 Session?

📖 难度系数: ★★

📖 问题频率: 80%

核心解答

在用户退出系统时, 为了系统的安全性, 希望尽快清除 Session 中保存的信息。

清除 Session 对象内容的方法如下。

❑ 使用 removeAttribute()方法

使用 removeAttribute()方法, 可以删除 Session 对象中保存的指定属性信息。示例代码如下:

```
session.setAttribute("name","tom");  
session.removeAttribute("name");
```

❑ 使用 invalidate()方法

使用 invalidate()方法可以清除 Session 对象中的所有信息, 示例代码如下:

```
session.invalidate();
```

一般情况下, 在关闭浏览器后, Session 信息需要等到 Session 对象失效才能清除, 如果需要实现关闭浏览器就清除 Session, 可以使用如下方法:

```
<body onbeforeunload="window.location='logout.jsp'">
```

在上述代码中,通过 `onbeforeunload` 事件属性捕获关闭事件,然后调用 `logout.jsp` 页面的代码进行处理。在 `logout.jsp` 里面可以销毁 Session 对象,其实现代码如下:

```
<%  
HttpSession session = request.getSession();  
session.invalidate();  
%>
```

疑难点评

在默认情况下,Session 对象在关闭浏览器后并不是立刻被销毁,因此,为了考虑系统的安全性,可以使用上述方法在用户退出时,清除 Session 对象,防止他人盗用 Session 对象中的信息。

知识链接

FAQ10.14 什么是 Session? 如何使用 Session?

FAQ10.17 如何在禁用 Cookie 的情况下使用 Session?

FAQ10.16 什么是 Cookie? 如何使用 Cookie?

📖 难度系数: ★★★

📖 问题频率: 85%

核心解答

Cookie 的原本意思是指搭配牛奶一起吃的小甜饼。但是在因特网中, Cookie 是指存储在客户浏览器目录下的文本文件,文件信息由 Web 服务器发送到客户浏览器并存储,下次该客户再次访问该 Web 服务器时,可从该浏览器读回此信息。例如 Windows XP 操作系统存储 Cookie 文件的目录“C:\Documents and Settings\用户名\Cookies”。

使用 Cookie, Web 服务器可以将一些客户的特定信息存储在客户计算机中,例如上次访问的位置、花费的时间或用户首选项(如样式表)等。Cookie 的信息应该是小量的、不影响系统安全性的信息,例如用户密码等不建议使用 Cookie 保存。

(1) JSP/Servlet 操作 Cookie

写入 Cookie 的示例代码如下:

```
Cookie c=new Cookie("username","tom");  
c.setMaxAge(120);  
response.addCookie(c);
```

在上述代码中,如果不使用 `setMaxAge()` 方法设置有效期, Cookie 信息将在客户关闭浏览器之后删除。

读取 Cookie 的示例代码如下:


```
Cookie[] cookies = request.getCookies();
for(int i=0;i<cookies.length;i++){
    if(cookies[i].getName().equals("username"){
        username = cookies[i].getValue();
    }
}
```

客户端的 Cookie 文件可以存储若干个 Cookie 对象的信息, 在读取时, request.getCookies() 返回一个 Cookie 数组, 可在该数组中遍历寻找指定的 Cookie 对象。

删除 Cookie 的示例代码如下:

```
Cookie killMyCookie = new Cookie("mycookie",null);
killMyCookie.setMaxAge(0);
killMyCookie.setPath("/");
response.addCookie(killMyCookie);
```

(2) JavaScript 操作 Cookie

写入 Cookie 的示例代码如下:

```
function setCookie(name,value,expiry,path,domain,secure){
    var nameString = name + "=" + value;
    var expiryString = (expiry == null) ? "" : ";expires = " + expiry.toGMTString();
    var pathString = (path == null) ? "" : ";path = " + path;
    var domainString = (domain == null) ? "" : ";domain = " + domain;
    var secureString = (secure) ? ";secure" : "";
    document.cookie = nameString + expiryString + pathString + domainString + secureString;
}
```

读取 Cookie 的示例代码如下:

```
function getCookie (name) {
    var CookieFound = false;
    var start = 0;
    var end = 0;
    var CookieString = document.cookie;
    var i = 0;

    while (i <= CookieString.length) {
        start = i;
        end = start + name.length;
        if (CookieString.substring(start, end) == name){
            CookieFound = true;
            break;
        }
        i++;
    }

    if (CookieFound){
        start = end + 1;
        end = CookieString.indexOf(";",start);
        if (end < start)
            end = CookieString.length;
        return unescape(CookieString.substring(start, end));
    }
}
```

```
}  
return "";  
}
```

删除 Cookie 的示例代码如下:

```
function deleteCookie(name){  
    var expires = new Date();  
    expires.setTime (expires.getTime() - 1);  
    setCookie( name , "Delete Cookie", expires,null,null,false);  
}
```

疑难点评

Cookie 信息在浏览器运行时存储在客户计算机的内存中, 如果客户从网站或 Web 服务器中退出, Cookie 也可存储在客户计算机的磁盘上。

Cookie 的用途非常广泛, 例如广告代理商用来追踪人口统计, 查看某个站点吸引了哪种消费者; 一些网站还用 Cookie 保存客户最近的账号信息; 也可以利用 Cookie 保存私人特色服务信息, 例如新浪股评板块利用 Cookie 保存客户自选股的股票编码信息等。

知识链接

FAQ10.17 如何在禁用 Cookie 的情况下使用 Session?

FAQ10.17 如何在禁用 Cookie 的情况下使用 Session?

📖 难度系数: ★★★★★

📖 问题频率: 88%

核心解答

Cookie 和 Session 之间有一定的关联, 如果客户端禁用了 Cookie, Session 使用会受到影响。

Cookie 信息是以文本文件的方式存储在客户端, 而 Session 信息却存储在服务器端内存, 当客户发送请求时, Cookie 信息会随请求一起发送到服务器, 服务器会根据 Cookie 中的 SessionID 寻找客户对应的 Session 对象, 因此如果 Cookie 被用户禁用后, 会影响 Session 的使用。

如果客户端禁用了 Cookie, 可以使用 URL 重写, 将 SessionID 添加到 URL 中, 利用 URL 传递给服务器。

HttpServletResponse 接口提供了重写 URL 的方法, 方法声明如下:

```
public java.lang.string encodeurl(java.lang.string url)
```

encodeurl()方法在使用时, 首先会判断 Session 是否启用, 如果未启用, 就直接返回参数 URL; 然后再判断客户端浏览器是否支持 Cookie, 如果支持, 就直接返回参数 URL; 如果不支持 Cookie, 就在参数 URL 中加入 SessionID 信息, 然后返回修改后的 URL。

为了解决以上问题, 可以对网页中的超链接稍作修改, 使用示例如下:

修改前:

```
<a href="maillogin.jsp">
```

修改后:

```
<a href="<%=response.encodeurl("maillogin.jsp")%">">
```

疑难点评

Session 对象的正常使用要依赖于 Cookie。如果客户端浏览器出于安全的考虑禁用了 Cookie, 则应该使用 URL 重写的方式使 Session 在客户端禁用 Cookie 的情况下继续生效。

知识链接

FAQ10.14 什么是 Session? 如何使用 Session?

FAQ10.16 什么是 Cookie? 如何使用 Cookie?

FAQ10.18 如何在 JSP 中避免表单的重复提交?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

在用户单击按钮提交表单后, 由于响应速度慢, 很多用户会习惯性地重复单击按钮, 这样会增加服务器处理负担, 影响服务器和客户浏览器之间的交互。

在 JSP 中避免 Form 表单重复提交, 可以使用以下 3 种方案。

(1) 使用 JavaScript 脚本, 设置一个变量, 只允许表单提交一次

在 JSP 或 HTML 页面的<head></head>之间添加 JavaScript 脚本函数, 示例代码如下:

```
<script language="javascript">
var checksubmitflg = false;
function checksubmit() {
    if (checksubmitflg == true) {
        return false;
    }
    checksubmitflg = true;
    return true;
}
document.ondoubleclick = function doondoubleclick() {
    window.event.returnValue = false;
}
document.onclick = function doonclick() {
    if (checksubmitflg) {
        window.event.returnValue = false;
    }
}
```



```
}  
</script>
```

在表单<form>标记中添加 onsubmit 属性, 示例代码如下:

```
<form action="myaction.do" method="post" onsubmit="return checksubmit();">
```

(2) 使用 JavaScript 脚本, 用户单击按钮后将按钮设置为 disable

```
<form action="myaction.do" method="post" onsubmit="getelbyid('submitinput').disabled = true; return true;">  
  <input type="submit" name="submitinput" value="提交" />  
</form>
```

(3) 利用 Struts 的同步令牌机制

Struts 框架提供了同步令牌机制来解决 Web 应用中重复提交的问题。令牌机制的基本原理如下所示。

服务器端在处理到达的请求之前, 会将请求中的令牌值与保存在用户 Session 中的令牌值进行比较, 看是否匹配。在请求处理完毕给客户端响应之前, 将会产生一个新的令牌, 该令牌除了发送给客户端之外, 也会将用户 Session 中保存的旧的令牌进行替换。如果用户回退到刚才的提交页面并再次提交, 那么客户端传过来的令牌就和服务器端的令牌不一致, 从而有效地防止了重复提交的发生。

利用 Struts 同步令牌机制防止表单重复提交, 具体使用方法如下:

① 生成令牌

在进入表单页面之前, 需要执行以下代码生成令牌, 示例代码如下。

```
saveToken(request);
```

执行上述代码后进入表单页面, 在表单页面将会多一个隐藏域, 用于保存令牌值, 具体如下:

```
<input type="hidden" name="org.apache.struts.taglib.html.TOKEN" value="6aa35341f25184fd996c4c918255c3ae">
```

隐藏域的 value 属性值即令牌值, 该值由 TokenProcessor 类的 generateToken() 方法获得, 是根据当前用户的 sessionid 和当前时间的 long 值来计算的。

② 验证令牌

在处理表单提交的方法中添加以下代码, 根据令牌值判断是否属于重复提交, 示例代码如下:

```
public ActionForward insert(ActionMapping mapping, ActionForm form,  
    HttpServletRequest request, HttpServletResponse response)  
{  
    if (isTokenValid(request, true)) {  
        // 表单不是重复提交  
        // 这里是保存数据的代码  
    } else {  
        // 表单重复提交  
        saveToken(request);  
        // 其它的处理代码  
    }  
}
```

疑难点评

用户重复提交会严重影响服务器和客户浏览器之间的交互,影响系统的正常工作。作为开发人员,可以通过上述方案避免用户重复提交,这样做不仅使系统更具人性化,同时还增强了健壮性。

FAQ10.19 如何实现 JSP 数据和 JavaScript 数据的交互使用?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

对于 Web 程序来说,前端 (JavaScript) 和后端 (JSP/Servlet) 是没法共用数据的,只能是后端程序 (JSP) 把数据输出,生成页面到前端,这时候生成的页面中的 JavaScript 代码才有可能得到所谓 JSP 的数据。同样的,只有把 JavaScript 里的数据提交给后端 JSP 代码, JSP 程序中才能得到 JavaScript 的数据。JSP 变量的数据与 JavaScript 变量的数据可以按以下方案进行交互。

(1) 将 JavaScript 数据提交给后台的 JSP 程序

将 JavaScript 数据提交给后台的 JSP 程序有两种方法,具体如下所示。

❑ 第 1 种方法

将 JavaScript 的数据以 `xxx.JSP?var1=aaa&var2=bbb` 的形式作为 URL 的参数传给 JSP 程序,在 JSP 中用 `<%String strVar1=request.getParameter("var1");%>` 就可以获取到 JavaScript 脚本传递过来的数据。

❑ 第 2 种方法

在表单里加入隐藏域信息,通过 JavaScript 将数据写入隐藏域,然后用表单提交的方式把数据传递给 JSP 程序。

(2) 将 JSP 数据提交给前台的 JavaScript 程序

将 JSP 数据提交给前台的 JavaScript 程序比较简单,只要在 JavaScript 脚本中使用 `<%%>` 和 `<%= %>` 标记获取 Java 变量值并输出即可。

疑难点评

为了提升用户体验,减少客户端与服务器的交互次数,JavaScript 在 Web 程序中的使用量越来越多,现在非常流行的 Ajax 技术也是以 JavaScript 为基础的。因此有越来越多的业务数据需要 Java 和 JavaScript 共同使用处理,此时可以采用上述方法将数据在 Java 和 JavaScript 程序

之间进行传递。

知识链接

FAQ10.02 JSP、Java 和 JavaScript 有什么区别和联系?

FAQ10.20 什么是 Servlet? Servlet 与 JSP 有什么区别?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

Servlet 是用 Java 编写的服务器端程序,它与协议和平台无关。Java Servlet 可以动态地扩展服务器端的处理能力,并采用请求/响应模式提供 Web 服务。Servlet 处理客户端请求的过程如下:

- ☐ 客户端发送请求至服务器端;
- ☐ 服务器将请求信息发送至 Servlet;
- ☐ Servlet 根据客户的请求信息生成响应内容,并将其传给服务器;
- ☐ 服务器将响应返回给客户端。

Sun 公司首先推出 Servlet 技术,其功能比较强大、体系设计也很先进。但是它依然采用了 CGI 方式逐句地输出 HTML 语句,所以编写和修改 HTML 非常不方便。后来, Sun 公司推出了与 ASP 类似的 JSP 技术,把 Java 代码嵌套在 HTML 语句中,这样就大大简化和方便了网页的设计和修改。

一个分布式系统应分为 3 层,即表示层、业务逻辑层和数据访问层。在 J2EE 体系结构中,由于 Servlet 是一个 Java 类,因此主要用于业务逻辑层,实现业务逻辑的处理;而 JSP 则主要是为了实现表示层而设计的,主要完成界面的显示逻辑。

疑难点评

JSP 和 Servlet 技术都可以处理客户端请求,各有优缺点。JSP 主要是在 HTML 中嵌入 Java 脚本;而 Servlet 则是在 Java 代码中输出 HTML 内容。为了使 JSP 更加简洁,提高程序的维护性和或扩展性,常常采用 JSP+JavaBean 或 JSP+Servlet+JavaBean 结构开发,将业务代码从 JSP 中分离出来。

知识链接

FAQ10.01 什么是 JSP? JSP 工作原理如何?

FAQ10.21 Servlet 容器的工作原理如何?

FAQ10.21 Servlet 容器的工作原理如何?

📖 难度系数: ★★

📖 问题频率: 60%

核心解答

Servlet 容器, 形象地说就是担当 Web 服务器和 Servlet 之间的中间人的角色。Web 服务器将被请求的 Servlet 的 URI 和 request 对象转交给 Servlet 容器, 然后由 Servlet 容器调用相应的 Servlet 程序处理该请求, 并将 Servlet 的请求结果返回给 web 服务器。

javax.servlet.Servlet 是 Servlet 程序的一个重要接口, 所有的 Servlet 必须实现这个接口或继承该接口的实现类 (例如 HttpServlet)。Servlet 接口定义了 5 个方法, 具体定义如下:

```
public void init(ServletConfig config) throws ServletException;  
public void service(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException;  
public void destroy();  
public ServletConfig getServletConfig();  
public java.lang.String getServletInfo();
```

为了更详细地了解 Servlet 容器的工作流程, 下面先介绍一下 Servlet 的生命周期, Servlet 的生命周期主要有以下几个过程。

(1) 加载和实例化

容器负责加载和实例化一个 Servlet。实例化和加载可以发生在引擎启动的时候, 也可以推迟到容器需要该 Servlet 为客户请求服务的时候。

(2) 初始化

init()方法用于初始化操作, 该方法在 Servlet 的整个生命周期中只被调用一次; 初始化的过程主要是读取永久的配置信息, 以及其他仅仅需要执行一次的任务。

(3) 处理请求

service()方法由 Servlet 容器调用, 以允许 Servlet 响应一个请求。Servlet 容器传递 javax.servlet.ServletRequest 对象和 javax.servlet.ServletResponse 对象。ServletRequest 对象包含客户端 HTTP 请求信息, ServletResponse 则封装 Servlet 响应。

(4) 移除实例

调用 destroy()方法 (在整个生命周期中只被调用一次); 服务器决定删除已经加载的 Servlet 实例之前将调用 Servlet 的 destroy()方法。

当 Servlet 容器接收到一个请求后, 首先到容器池内查找被请求的 Servlet 实例是否已经存在, 若不存在, 则会加载和实例化被请求的 Servlet, 并对它进行初始化; 若已存在, 则直接调用 Servlet 的 service()方法, 然后根据 HTTP 请求种类的不同, 在 service()内部调用 doGet()或 doPost()等方法处理相应的请求。当 Servlet 容器下一次接收到对该 Servlet 的请求时, 则只需重新创建一个

ServletRequest 对象和一个 ServletResponse 对象，并将它们作为参数传递给 service()方法，同时新建一个线程处理该请求。最后当不需要该 Servlet 时，通过调用 destroy()方法移除该实例。

Tomcat 是最常见的 Servlet 容器之一，由于它同时也能处理对 JSP 页面的请求，所以通常也被称为 JSP/Servlet 容器。

疑难点评

在编写 Servlet 时，一般都继承 HttpServlet，然后根据需要重写 init()、destroy()、doGet()和 doPost()方法。大多数开发者对 Servlet 容器和 Servlet 程序的运行机制没有太多关注，多了解一些运行机制可以加深对程序的理解，建议读者在学习时可以深入一些，不要只停留在应用层次。

知识链接

FAQ10.20 什么是 Servlet? Servlet 与 JSP 有什么区别?

FAQ10.22 如何在 Servlet 中使用 Session 和 Application?

📖 难度系数: ★★★

📖 问题频率: 90%

核心解答

在 JSP 中使用 Session 的示例代码如下:

```
session.setAttribute("name","tom");// Session 写入  
String name = session.getAttribute("name");//Session 读取
```

在 JSP 中使用 Application 的示例代码如下:

```
application.setAttribute("name","tom");// Application 写入  
String name = application.getAttribute("name");//Application 读取
```

在 Servlet 的 doPost()和 doGet()方法中，可以使用 request 和 response 参数，但不能直接使用 session、application 等其他 JSP 内建对象，需要使用代码显示定义和获取。示例代码如下:

```
HttpSession session = request.getSession();//获取 session 对象  
ServletContext application = this.getServletContext();//获取 application 对象
```

疑难点评

JSP 和 Servlet 使用 Session 和 Application 有一定的区别，在 JSP 中可以直接通过 session 和 application 使用；但在 Servlet 中需要通过代码实现获取。当获取 Session 和 Application 对象后，写入和读取的代码是一样的，而且 JSP 和 Servlet 使用的是同一个对象。

知识链接

FAQ10.14 什么是 Session? 如何使用 Session?

FAQ10.23 如何编写多线程安全的 Servlet 程序?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

Servlet 默认是以多线程模式执行的, 因此需要考虑多线程并发的问题。

(1) Servlet 的多线程机制

Servlet 体系结构是建立在 Java 多线程机制之上的, 它的生命周期是由 Web 容器负责的。当客户端第一次请求某个 Servlet 时, Servlet 容器将会根据 web.xml 配置文件实例化这个 Servlet 类。当再有其他客户请求该 Servlet 时, 一般不会再实例化该 Servlet 类, 也就会出现多个线程使用一个 Servlet 实例的情况。

当两个或多个线程同时访问同一个 Servlet 时, 可能会发生多个线程同时访问同一资源的情况, 数据可能会变得不一致。所以在使用 Servlet 构建 Web 应用时如果不注意线程安全的问题, 会使 Servlet 程序产生难以发现的错误。

(2) 如何编写线程安全的 Servlet

Servlet 的线程安全问题主要是由于成员变量使用不当而引起的, 下面针对该问题给出了 3 种解决方案。

❑ 实现 SingleThreadModel 接口

SingleThreadModel 接口指定了系统如何处理对同一个 Servlet 的调用。如果一个 Servlet 被这个接口指定, 那么在这个 Servlet 中的 service 方法将不会有两个线程被同时执行, 当然也就不存在线程安全的问题。示例代码如下:

```
public class ServletTest extends HttpServlet implements SingleThreadModel {  
    ...  
}
```

❑ 同步对共享数据的操作

使用 synchronized 关键字能保证一次只有一个线程可以访问被保护的区段, 在 Servlet 中可以通过同步块操作来保证线程的安全。示例代码如下:

```
public class ServletTest extends HttpServlet {  
    Username = request.getParameter("username");  
    synchronized (this){  
        tutput = response.getWriter();  
        try {  
            Thread.sleep(5000);  
        } catch (InterruptedException e){}  
        output.println("用户名:"+Username+"<BR>");  
    }  
}
```



```
}  
}
```

❑ 避免使用成员变量（也称为实例变量）

线程安全问题是成员变量造成的，只要在 Servlet 中不使用成员变量，尽量使用局部变量，那么该 Servlet 就是线程安全的。示例代码如下：

```
public class ServletTest extends HttpServlet {  
    PrintWriter output; //成员变量  
    String username; //成员变量  
    public void service (HttpServletRequest request, HttpServletResponse response) throws ServletException,  
        IOException {  
        PrintWriter output; //局部变量  
        String username; //局部变量  
        Response.setContentType ("text/html; charset=gb2312");  
        ...  
    }  
}
```

Servlet 的线程安全问题只有在大量的并发访问时才会显现出来，并且很难发现，因此在编写 Servlet 程序时要特别注意。线程安全问题主要是由成员变量造成的，因此在 Servlet 中应避免使用成员变量。如果应用程序设计无法避免使用成员变量，那么使用同步来保护要使用的成员变量，但为保证系统的最佳性能，应该同步可用性最小的代码路径。

注意：由于 JSP 在运行时也是被转换为 Servlet，因此也需要注意线程安全问题。JSP 内建对象 out、request、response、session、config、page 和 pageContext 是线程安全的，而 application 在整个系统内被使用，所以不是线程安全的。

疑难点评

使用上述 3 种方法都能设计出线程安全的 Servlet 程序。但是，如果一个 Servlet 实现了 SingleThreadModel 接口，Servlet 引擎将为每个新的请求创建一个单独的 Servlet 实例，这将引起大量的系统开销。SingleThreadModel 在 Servlet 2.4 中已不再提倡使用；同样如果在程序中使用同步来保护要使用的共享的数据，也会使系统的性能大大下降。这是因为被同步的代码块在同一时刻只能有一个线程执行它，使得其同时处理客户请求的吞吐量降低，而且很多客户处于阻塞状态。另外为保证主存内容和线程的工作内存中的数据的一致性，要频繁地刷新缓存，这也会很大地影响系统的性能。所以在实际的开发中也应避免或最小化 Servlet 中的同步代码；在 Servlet 中避免使用成员变量是保证 Servlet 线程安全的最佳选择。从 Java 内存模型也可以知道，方法中的局部变量是在栈上分配空间，而且每个线程都有自己私有的栈空间，所以它们不会影响线程的安全。

知识链接

FAQ10.21 Servlet 容器的工作原理如何？

FAQ10.24 如何在 Servlet 和 JSP 中获取工程文件的绝对路径?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

(1) JSP 应用

在 JSP 中获取工程资源的绝对路径, 示例代码如下:

```
out.println("根目录所对应的绝对路径:" + request.getRequestURI() + "<br/>");
String strPathFile = application.getRealPath(request.getRequestURI());
out.println("文件的绝对路径:" + strPathFile + "<br/>");
out.println(application.getRealPath(request.getRequestURI()));
String strDirPath = new File(application.getRealPath(request.getRequestURI())).getParent();
out.println("目录的绝对路径:" + strDirPath + "<br/>");
```

(2) Servlet 应用

在 Servlet 中获取工程资源的绝对路径, 示例代码如下:

```
System.out.println("根目录所对应的绝对路径:" + request.getServletPath() + "<br/>");
String strPathFile = request.getSession().getServletContext().getRealPath(request.getRequestURI());
System.out.println("文件的绝对路径:" + strPathFile + "<br/>");
String strDirPath = new File(request.getSession().getServletContext().getRealPath(request.getRequestURI())).getParent();
System.out.println("目录的绝对路径:" + strDirPath + "<br/>");
//获取 Web 项目的全路径
String strFullPath = getServletContext().getRealPath("/");
out.println(strFullPath);
//获得 Web 项目的上下文路径
String strContextPath = request.getContextPath();
out.println(strContextPath);
```

疑难点评

在 Servlet 和 JSP 中通过代码动态获取工程资源的绝对路径, 提高了程序的灵活性和可移植性。其中 request.getRealPath() 方式仅作为了解, 已不推荐使用。

知识链接

FAQ10.25 如何获取客户端浏览器和操作系统信息?

FAQ10.25 如何获取客户端浏览器和操作系统信息?

📖 难度系数: ★★★

📖 问题频率: 80%

核心解答

在服务器处理时，有时候需要获取客户端浏览器或操作系统的一些信息，例如客户计算机 IP 地址、计算机名等。

使用 request 对象的方法，可以获取客户浏览器和操作系统信息。示例代码如下：

```
String Agent = request.getHeader("User-Agent");
StringTokenizer st = new StringTokenizer(Agent,";");
st.nextToken();
String userbrowser = st.nextToken(); //得到用户的浏览器名
String useros = st.nextToken(); //得到用户的操作系统名
String h= getHeader(String name); //获得 http 协议定义的传送文件头信息
String m = request. getMethod(); //获得客户端向服务器端传送数据的方法有 GETPOST、PUT 等类型
String uri = request. getRequestURI(); //获得发出请求字符串的客户端地址
String path = request. getServletPath(); //获得客户端所请求的脚本文件的文件路径
String name = request. getServerName(); //获得服务器的名字
String port= request.getServerPort(); //获得服务器的端口号
String ip= request.getRemoteAddr(); //获得客户端的 IP 地址
String host= request.getRemoteHost(); //获得客户端电脑的名字，若失败，则返回客户端电脑的 IP 地址
```

取得服务器系统信息，示例代码如下：

```
System.getProperty("os.name");
System.getProperty("os.version");
System.getProperty("os.arch");
```

疑难点评

服务器在接收到客户请求信息后，会将客户发送的所有信息都封装到 request 对象中，通过 request 对象可以获取客户发送过来的所有信息。

知识链接

FAQ10.24 如何在 Servlet 和 JSP 中获取工程文件的绝对路径？

FAQ10.26 如何在 Web 程序中实现定时运行的功能？

📖 难度系数：★★★★

📖 问题频率：80%

核心解答

定时运行功能可以使用线程或定时器组件实现。在 Web 程序中，定时器的启动一般随 Web 服务器的启动而启动，主要有以下两种方法。

方法一：在 web.xml 里配置一个 Servlet，并设置其随 Web 服务器的启动而启动。然后在该 Servlet 的 init()方法里启动定时器，在 destory()方法里销毁定时器。

方法二：在 web.xml 里配置一个 Listener，然后在该 Listener 的初始化方法里启动定时器，在其销毁的方法销毁定时器。

下面介绍如何利用 Servlet 启动定时器。

□ 编写定时器代码

```
public class Task extends TimerTask{

    private ServletContext context;

    private static boolean isRunning = true;

    public Task(ServletContext context){
        this.context = context;
    }

    public void run() {
        if(isRunning){
            //业务处理
        }
    }

}
```

□ 编写 Servlet 程序

在 Servlet 的 init()方法中调用计时器，在 destroy()方法中释放定时器资源，示例代码如下：

```
public class ConvergeDataServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    private Timer timer1 = null;

    private Task task1;

    // init()方法启动定时器
    public void init() throws ServletException {

        ServletContext context = getServletContext();

        // (true 为用定时器刷新缓存)
        String startTask = getInitParameter("startTask");

        // 定时刷新时间(分钟)
        Long delay = Long.parseLong(getInitParameter("delay"));

        // 启动定时器
        if(startTask.equals("true")){
            timer1 = new Timer(true);
            task1 = new Task(context);
            timer1.schedule(task1, delay * 60 * 1000, delay * 60 * 1000);
        }
    }

}
```

```
    }  
}  
  
//释放定时器资源  
public void destroy() {  
    super.destroy();  
    if(timer1!=null){  
        timer1.cancel();  
    }  
}  
  
public void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
    //业务处理代码  
}  
  
public void doPost(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
    doGet(request, response);  
}  
}
```

□ 配置 Servlet

在 web.xml 中配置 Servlet，当服务启动时就调用该 Servlet 程序。配置代码如下：

```
<servlet>  
    <servlet-name>taskservlet</servlet-name>  
    <servlet-class>com.task</servlet-class>  
    <init-param>  
        <param-name>startTask</param-name>  
        <param-value>true</param-value>  
    </init-param>  
    <init-param>  
        <param-name>intervalTime</param-name>  
        <param-value>1</param-value>  
    </init-param>  
    <load-on-startup>9</load-on-startup>  
</servlet>
```

疑难点评

在使用 Servlet 启动定时器时，web.xml 配置信息中的<load-on-startup>元素必须定义，该元素能够使 Servlet 在服务器启动时加载 Servlet，否则需要等到客户请求时才能创建。使用 Listener 启动定时器的过程与使用 Servlet 相似，需要编写 Listener 实现类，在 web.xml 中添加 Listener 配置。

知识链接

FAQ10.23 如何编写多线程安全的 Servlet 程序？

FAQ10.27 如何实现网站登录记忆跳转的功能?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

记忆登录跳转是指当用户访问系统受保护页面时,会将页面转移到登录界面,要求用户登录,当用户登录后,将页面转移到登录前要访问的页面。记忆登录跳转功能可以为用户带来一定的方便性。

使用 `request.getHeader("Referer");` 可以方便地获取上次访问的 URL 请求,从而实现记忆跳转功能,但是该方法不能处理 POST 方式提交的参数信息。

为了将上次请求的所有信息都保存下来,可以写一个工具类,将请求和表单信息拼写成一个字符串,保存到表单中的一个隐藏域;等登录成功后,再将隐藏域信息获取并解析,然后发送请求完成页面跳转。实现示例如下:

❑ 处理请求的工具类

`RedirectUtil` 类用于处理请求 URL 和表单信息,完成 URL 转换。示例代码如下:

```
public class RedirectUtil
{
    //将请求信息转换为字符串
    public static String dealurl(HttpServletRequest request)
    {
        String url = "";
        url = request.getRequestURL()+"?";
        url +=param(request);
        if(url.indexOf("&")>-1)
            url=url.replaceAll("&","@#@");//实际上就是把有&的字符转化成了@#@
        return url;
    }

    //将字符串还原为请求
    public static String geturl(String url)
    {
        if(url.indexOf("@#@")>-1)
            url=url.replaceAll("@#@","&");
        return url;
    }

    //将 request 中的信息拼成字符串
    public static String param(HttpServletRequest request)
    {
        String url = "";
        Enumeration param = request.getParameterNames();//得到所有参数名
```



```

        while(param.hasMoreElements()){
            String pname = param.nextElement().toString();
            url += pname+"="+request.getParameter(pname)+"&";
        }
        if(url.endsWith("&")){
            url = url.substring(0,url.lastIndexOf("&"));
        }
        return url;
    }
}

```

□ 受保护页面的处理

在受保护的 JSP 页面中添加以下代码:

```

<%
String url = ""
if(session.getAttribute("username")==null)
{
    url = RedirectUtil.dealurl(request);//记录当前地址和请求参数, dealurl 将实际 url 处理了, 避免和要请求 url 有干扰,
    因为会有&字符
    response.sendRedirect("weblogin.jsp?url = "+url) ;//重定向到登录页面
}
%>

```

□ 登录页面代码

在登录页面的表单中需要添加一个隐藏域, 用于保存上次请求的信息。登录页面 welogin 的示例代码如下:

```

<%
String url = request.getParameter("url");
String userid =request.getParameter("userid");
String password =request.getParameter("password");
//如果是登录提交执行下面代码
if(userid!=null){
    if(登录成功){
        session.setAttribute("userid",userid);
        out.println("<script>alert('登录成功,谢谢光临');self.location.href='"+RedirectUtil.geturl(/url)+"';</script>");
        //转向到首次请求的 url, geturl(/url)是把转化过的地址转化回来变成真实 url
    }else{
        out.println("<script>alert('用户名或密码错误');history.back();</script>");
    }
}
}else{//如果不是登录提交显示现在登录界面
%>

<form name="loginfrm" action="weblogin.jsp" method="post">
账&nbsp;号:<input name="userid" type="text" size="12"><br>
账&nbsp;码:<input name="password" type="password" size="12"><br>
<input type="submit" name="Submit" value="登录">
<input name="url" type="hidden" value="<%=url%>">
</form>
<%

```

```
}  
%>
```

疑难点评

记忆登录跳转功能其实就是想办法将上一次请求的 URL 和表单参数保存下来,然后等到登录成功之后再取出并执行跳转。上述方法在实现时通过隐藏域保存请求信息,该方式使用比较普遍,当然也可以采用其它方法保存,例如 Session 等。

知识链接

FAQ10.09 JSP 中 forward 和 redirect 有什么区别?

FAQ10.28 如何将 JSP 动态页面转换为 HTML 静态页面?

📖 难度系数: ★★★★★

📖 问题频率: 75%

核心解答

在一些新闻系统中,为了提高响应速度,可以将 JSP 页面事先转换为 HTML 页面,这样用户在查看时可以直接浏览 HTML 静态页面。

上述功能的实现思路是:利用 URL 请求某个 JSP 页面,然后获取 JSP 的输出流,并将响应信息保存到文件。示例代码如下:

```
public void jspToHtml(String urlString) throws Exception{  
    URL url = new URL(urlString);  
    HttpURLConnection ts = (HttpURLConnection) url.openConnection();  
    InputStream socketInput = ts.getInputStream();  
  
    BufferedReader in = new BufferedReader(new InputStreamReader(  
        socketInput));  
    String fileName = url.getContent().toString() + ".html";  
    PrintWriter file = new PrintWriter(new FileWriter(fileName), true);  
    while (true) {  
        try {  
            String s = in.readLine();  
            if (s == null) {  
                break;  
            }  
            file.print(s);  
        } catch (Exception e) {  
            System.out.println(e);  
            break;  
        }  
    }  
}
```

```
}  
file.close();  
System.out.println("转换完成");  
}
```

疑难点评

使用上述方法, 可以将一个 URL 请求的响应信息转换成一个 HTML 页面。读者可以基于上述方法进行扩展, 比如实现批量转换和批量下载等。

FAQ10.29 如何实现数据分页显示的功能?

📖 难度系数: ★★★★★

📖 问题频率: 95%

核心解答

在 JSP 中实现分页显示的方法有很多, 常用方法如下:

(1) 基于 ResultSet 的 absolute() 方法

该方法的实现思路是: 当用户单击翻页按钮时, 根据当前页数和一页显示的记录数计算出当前页要显示记录的起始点, 然后通过 absolute() 方法在 ResultSet 结果集中进行定位, 最后获取记录显示。示例代码如下:

在 JSP 中添加 page 指令声明和 JavaBean 的引用, 代码如下:

```
<%@ page contentType="text/html;charset=GBK" %>  
<%@ page language="java" import="java.sql.*"%>  
<jsp:useBean id="conn" scope="page" class="com.DbUtil"/>
```

在 JSP 中使用 <%!%> 符号添加声明代码, 定义变量和分页函数, 代码如下:

```
<%!  
ResultSet rs = null;  
ResultSet rsTmp = null;  
String sql = "";  
int size = 6;  
int currentPage = 3;  
int totalPage = 1;  
String str = "";  
  
public String ShowOnePage(ResultSet rs, int currentPage, int size) {  
    str = "";  
    //先将记录指针定位到相应的位置  
    try {  
        rs.absolute( (currentPage-1) * size + 1);  
    } catch(SQLException e) {  
    }  
}
```



```

        for(int iPage=1; iPage<=size; iPage++) {
            str += RsToHtml(rs);
            try {
                if(!rs.next()) break;
            } catch(Exception e) {}
        }
        return str;
    }

    // 显示单行记录子模块
    public String RsToHtml( ResultSet rs ) {
        String tt = "";
        try {
            tt += "<TR>";
            tt += "<TD>" + rs.getString("studentid") + "</TD>";
            tt += "<TD>" + rs.getString("studentname") + "</TD>";
            tt += "<TD>" + rs.getString("gender") + "</TD>";
            tt += "<TD>" + rs.getString("phone") + "</TD>";
            tt += "</TR>";
        } catch(SQLException e) {}

        return tt;
    }
}
%>

```

在 JSP 中使用<%%>符号添加执行代码，实现数据的获取和显示，代码如下：

```

<%
    sql = "select * from student";
    try {
        rs = conn.executeQuery( sql );
    } catch(Exception e) {
        out.println("访问数据库出错！");
    }
%>
<html>

<head>
    <title>分页浏览数据库的技巧</title>
</head>

<body bgcolor="#FFFFFF">

<h2 ALIGN="CENTER">JSP 中的分页控制, Version 3</h2>

<hr>
<center>
<table border>
    <TR bgcolor=lightblue>
        <TH>学生编号</TH>
        <TH>学生姓名</TH>

```

```

        <TH>性别</TH>
        <TH>联系电话</TH>
    </TR>
<%
    rsTmp = conn.executeQuery("select count(*) from student");
    rsTmp.next();

    int totalrecord = rsTmp.getInt(1);
    if(totalrecord % size == 0) totalPage = totalrecord / size; //如果记录总条数是单页显示条目数的整数倍
    else totalPage = totalrecord / size + 1; //如果最后还空余一页

    if(totalPage == 0) totalPage = 1;
    rsTmp.close();

    try {
        if(request.getParameter("Page")==null || request.getParameter("Page").equals(""))
            currentPage = 1;
        else
            currentPage = Integer.parseInt(request.getParameter("Page"));
    } catch(java.lang.NumberFormatException e) { //捕获用户从浏览器地址栏直接输入 Page=sdfsdfsdf 所造成的
异常
        currentPage = 1;
    }

    if(currentPage < 1) currentPage = 1;
    if(currentPage > totalPage) currentPage = totalPage;

    out.println(ShowOnePage(rs, currentPage, size));
%>
</table>
<form Action="pagev3.jsp" Method="GET">
<%
    if(currentPage != 1) {
        out.println("<A HREF=pagev3.jsp?Page=1>第一页</A> ");
        out.println("<A HREF=pagev3.jsp?Page=" + (currentPage-1) + ">上一页</A> ");
    }
    if(currentPage != totalPage) {
        out.println("<A HREF=pagev3.jsp?Page=" + (currentPage+1) + ">下一页</A> ");
        out.println("<A HREF=pagev3.jsp?Page=" + totalPage + ">最后一页</A> ");
    }
    conn.close();
%>
    <p>输入页数: <input TYPE="TEXT" Name="Page" SIZE="3"> 页数:<font COLOR="Red"><%=currentPage%>
/<%=totalPage%></font>
    </p>
</form>
</center>
<hr>
</body>
</html>

```


在上述代码中,使用了一个 JavaBean 类 com.DbUtil, DbUtil 用于管理数据库连接、执行 SQL 语句。DbUtil 的实现代码如下:

```
package com;

import java.sql.*;

public class DbUtil {

    private static final String ORAURL = "jdbc:oracle:thin:@192.168.7.104:1521:ORCL";
    private static final String NAME = "scott";
    private static final String PASSWORD = "tiger";
    private static final int OK = 0;
    private static final int ERROR = 9;
    private static final String DRIVER = "oracle.jdbc.driver.OracleDriver";
    private Connection conn = null;
    private Statement stat = null;

    public DbUtil() {
        try {
            Class.forName(DRIVER);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public int executeUpdate(String sql) {
        try {
            if (conn == null || conn.isClosed()) {
                conn = DriverManager.getConnection(ORAURL, NAME, PASSWORD);
            }
            stat = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
            stat.executeUpdate(sql);
            return OK;
        } catch (Exception e) {
            e.printStackTrace();
            return ERROR;
        }
    }

    public ResultSet executeQuery(String sql) {
        try {
            if (conn == null || conn.isClosed()) {
                conn = DriverManager.getConnection(ORAURL, NAME, PASSWORD);
            }
            stat = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
            return stat.executeQuery(sql);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```



```

        e.printStackTrace();
        return null;
    }
}

public int close(){
    try {
        stat.close();
    } catch (Exception e) {
        e.printStackTrace();
        return ERROR;
    }
    try {
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
        return ERROR;
    }
    return OK;
}
}

```

注意：在上述代码中，为了使 ResultSet 类支持 absolute() 方法，在获取 Statement 对象时需要为 createStatement() 方法指定参数，否则在执行 absolute() 方法时将出现异常。

(2) 基于分页的 SQL 语句

该方法的实现思路是：当用户单击翻页按钮时，根据当前页数和一页显示的记录数计算出当前页要显示记录的起始点和结束点，然后拼写一个获取 n~m 位置之间记录的 SQL 语句，最后执行 SQL 语句获取数据并显示。

基于 SQL 语句实现分页，需要针对不同数据库拼写不同格式的 SQL，以 Oracle 为例，获取 n~m 位置之间记录的 SQL 语句如下：

```

select * from
(
    select empno,ename,sal,rownum row from
        (select * from emp order by sal desc)
    where rownum <= m
)
where row >= n

```

基于 SQL 语句实现分页的代码与使用 absolute() 方法有些相似，完整的实现代码如下：

```

<%@ page contentType="text/html; charset=GBK" %>
<%@ page language="java" import="java.sql.*" %>
<jsp:useBean id="conn" scope="page" class="com.DbUtil"/>

<%!
ResultSet rs = null;
ResultSet rsTmp = null;
String sql = "";

```

```

int size = 6;
int currentPage = 3;
int totalPage = 1;
String str = "";

//获取分页的 SQL 语句
public String getOnePageSql(int currentPage, int size) {
    str = "";
    // 先将记录指针定位到相应的位置
    int begin = (currentPage-1) * size + 1;
    int end = begin+size;
    StringBuffer sb = new StringBuffer();
    sb.append("select * from");
    sb.append("(");
    sb.append("select empno,ename,sal,rownum row from");
    sb.append("(select * from emp order by sal desc)");
    sb.append("where rownum <"+end);
    sb.append(")");
    sb.append("where row>"+begin);
    return sb.toString();
}

//将 ResultSet 结果集中的记录转换为 HTML 标记
public String rsToHtml( ResultSet rs ) {
    StringBuffer tt = new StringBuffer();
    try {
        while(rs.next()){
            tt.append("<TR>");
            tt.append("<TD>" + rs.getString("studentid") + "</TD>");
            tt.append("<TD>" + rs.getString("studentname") + "</TD>");
            tt.append("<TD>" + rs.getString("gender") + "</TD>");
            tt.append("<TD>" + rs.getString("phone") + "</TD>");
            tt.append("</TR>");
        }
    } catch (Exception e) {
    }
    return tt.toString();
}

%>

<%
    sql = "select * from student";
    try {
        rs = conn.executeQuery( sql );
    } catch (Exception e) {
        out.println("访问数据库出错! ");
    }
%>
<html>

```



```

<head>
    <title>分页浏览数据库的技巧</title>
</head>

<body bgcolor="#FFFFFF">

<h2 ALIGN="CENTER">JSP 中的分页控制, Version 3</h2>

<hr>
<center>
<table border>
    <TR bgcolor=lightblue>
        <TH>学生编号</TH>
        <TH>学生姓名</TH>
        <TH>性别</TH>
        <TH>联系电话</TH>
    </TR>
<%
    rsTmp = conn.executeQuery("select count(*) from student");
    rsTmp.next();

    int totalrecord = rsTmp.getInt(1);
    if(totalrecord % size == 0) totalPage = totalrecord / size; //如果记录总条数是单页显示条目数的整数倍
    else totalPage = totalrecord / size + 1; //如果最后还空余一页

    if(totalPage == 0) totalPage = 1;
    rsTmp.close();

    try {
        if(request.getParameter("Page") == null || request.getParameter("Page").equals(""))
            currentPage = 1;
        else
            currentPage = Integer.parseInt(request.getParameter("Page"));
    } catch(java.lang.NumberFormatException e) { //捕获用户从浏览器地址栏直接输入 Page=sdfsdfsdf 所造成的
        currentPage = 1;
    }

    if(currentPage < 1) currentPage = 1;
    if(currentPage > totalPage) currentPage = totalPage;

    //执行分页的 SQL 语句
    ResultSet rs = conn.executeQuery(getOnePageSql(currentPage, size));
    //获取记录并显示
    out.println(rsToHtml(rs));
%>
</table>
<form Action="pagev3.jsp" Method="GET">
<%
    if(currentPage != 1) {

```

异常

PDG


```

        out.println("<A HREF=pagev3.jsp?Page=1>第一页</A> ");
        out.println("<A HREF=pagev3.jsp?Page=" + (currentPage-1) + ">上一页</A> ");
    }
    if(currentPage != totalPage) {
        out.println("<A HREF=pagev3.jsp?Page=" + (currentPage+1) + ">下一页</A> ");
        out.println("<A HREF=pagev3.jsp?Page=" + totalPage + ">最后一页</A> ");
    }
    conn.close();
    %>
    <p>输入页数: <input TYPE="TEXT" Name="Page" SIZE="3"> 页数:<font COLOR="Red"><%=currentPage%>/
<%=totalPage%></font>
    </p>
</form>
</center>
<hr>
</body>
</html>

```

注意:不同数据库的分页 SQL 语句是不同的,示例介绍的是 Oracle 数据库,如果使用 MySQL 数据库,其分页 SQL 的格式应该为“select * from emp limit n,m”。

疑难点评

数据分页显示功能非常常见,其实现方法也有很多,每一种方法都有其各自的优点。例如基于 ResultSet 的 absolute() 方法,在显示每一页记录时,需要将大量数据取出然后通过 absolute() 方法定位,造成了资源的浪费;而基于 SQL 语句分页的方法,在显示每一页记录时,仅获取需要显示的记录,减少了查询的数据量,但是该方法需要针对不同数据库拼写不同的 SQL 语句,通用性和移植性方面较差。另外还有基于缓存实现分页的方法,思路是将数据全部取出,放入服务器缓存,每次显示记录从缓存中取出并显示,这种方法虽然减少了与数据库的交互次数但增加了服务器负担。在实际使用时,可以根据具体情况选择合适的实现方法。

知识链接

FAQ9.13 如何获取 ResultSet 中 n~m 位置区间的记录?

FAQ10.30 如何将 JSP 内容以 Excel 或 Word 格式输出?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

一般情况下, JSP 程序的处理结果都采用 HTML 格式为客户端浏览器响应输出,除 HTML 格式之外,还有 Excel、Word 或 Image 等格式。在 JSP 程序将响应结果输出之前,可以使用指

令或程序代码指定响应结果的输出格式，这样客户端在接收到响应结果后，会根据指定格式在浏览器中显示。

一般的 JSP 页面都是通过以下指令指定结果的输出格式。

```
<%@ page contentType="text/html; charset=GBK" language="java" %>
```

contentType 属性的作用是指定 JSP 的处理结果采用什么格式的编码输出到客户端浏览器。

如果 JSP 结果需要以 Excel 或 Word 格式输出，可以修改 contentType 属性值，示例如下：

□ Excel 格式

```
<%@ page contentType="application/vnd.ms-Excel; charset=GBK" language="java"%>
```

□ Word 格式

```
<%@ page contentType="application/vnd.ms-word; charset=GBK" language="java"%>
```

在 JSP 开始输出数据之前，加入以下代码：

□ Excel 格式

```
<%  
    response.setHeader("Content-disposition","attachment;filename=result.xls");  
%>
```

□ Word 格式

```
<%  
    response.setHeader("Content-disposition","attachment;filename=result.doc");  
%>
```

疑难点评

在 JSP 页面中，通过 page 指令的 contentType 属性可以指定响应结果的输出格式。如果响应结果以 Excel 或 Word 格式输出，需要将 JSP 页面中的 JavaScript 和 CSS 元素删除，另外客户端计算机需要安装 Office 软件才能正常显示。

FAQ10.31 如何在 JSP 中实现打印功能？

📖 难度系数：★★★★

📖 问题频率：90%

核心解答

实现 JSP 页面打印的方法有很多，主要是借助于 JavaScript。下面介绍几种实现方法，具体如下：

□ 方法一

在页面的<head></head>标记中，添加 JavaScript 脚本函数，代码如下：

```
function PrintTable(Id){  
    var mStr;  
    mStr = window.document.body.innerHTML;
```

```

var mWindow = window;
window.document.body.innerHTML = Id.innerHTML;
mWindow.print();
window.document.body.innerHTML = mStr;
}

```

在页面中, 将要打印的部分用<div>标记修饰, 代码如下:

```

<div id="dy">
...
</div>

```

打印按钮的代码如下:

```

<input type="button" value="打 印" onclick="return PrintTable(dy)">

```

❑ 方法二

在页面中添加如下代码:

```

<OBJECT id=WebBrowser classid=CLSID:8856F961-340A-11D0-A96B-00C04FD705A2 height=0 width=0>
</OBJECT>
<input type=button value=打印 onclick=document.all.WebBrowser.ExecWB(6,1)>
<input type=button value=直接打印 onclick=document.all.WebBrowser.ExecWB(6,6)>
<input type=button value=页面设置 onclick=document.all.WebBrowser.ExecWB(8,1)>
<input type=button value=打印预览 onclick=document.all.WebBrowser.ExecWB(7,1)>

```

在需要输出打印按钮时, 可以使用如下代码, 将打印工具栏输出显示:

```

<script>showPrintBar()</script>

```

❑ 方法三

在页面中利用 JavaScript 脚本的 execCommand() 方法, 打开打印对话框, 供用户使用。代码如下:

```

document.execCommand('Print');

```

或调用如下脚本直接打印:

```

window.print()

```

疑难点评

上述方法都是基于 JavaScript 和 HTML 元素, 不仅可以使用在 JSP 程序中, 也可以利用在 ASP.NET、PHP 等其他动态网页中。

FAQ10.32 如何实现图片验证码功能?

📖 难度系数: ★★★★★

📖 问题频率: 96%

核心解答

利用 JSP 或 Servlet 都可以实现验证码功能, 下面分别介绍。

(1) 利用 JSP 实现

首先新建一个 JSP 页面 imageJsp.jsp, 添加一行 page 指令, 用于指定返回的信息格式为

“image/jpeg”，代码如下：

```
<%@ page contentType="image/jpeg"
import="java.awt.* java.awt.image.* java.util.* javax.imageio.*"
pageEncoding="gb2312"%>
```

利用<%! %>标记声明一个方法，用于获取图片的随机颜色，示例代码如下：

```
<%!Color getRandColor(int fc, int bc) {
//给定范围获得随机颜色
Random random = new Random();
if (fc > 255)
fc = 255;
if (bc > 255)
bc = 255;
int r = fc + random.nextInt(bc - fc);
int g = fc + random.nextInt(bc - fc);
int b = fc + random.nextInt(bc - fc);
return new Color(r, g, b);
}%>
```

生成随机数，添加干扰线，以 JPEG 格式将动态生成的图片输出，示例代码如下：

```
<%
//设置页面缓存
response.setHeader("Pragma", "No-cache");
response.setHeader("Cache-Control", "no-cache");
response.setDateHeader("Expires", 0);
//在内存中创建图像
int width = 60, height = 20;
BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
//获取图形上下文
Graphics g = image.getGraphics();
//生成随机类
Random random = new Random();
//设定背景色
g.setColor(getRandColor(200, 250));
g.fillRect(0, 0, width, height);
//设定字体
g.setFont(new Font("Times New Roman", Font.PLAIN, 18));
//随机产生 155 条干扰线，使图像中的认证码不易被其它程序探测到
g.setColor(getRandColor(160, 200));
for (int i = 0; i < 155; i++) {
int x = random.nextInt(width);
int y = random.nextInt(height);
int xl = random.nextInt(12);
int yl = random.nextInt(12);
g.drawLine(x, y, x + xl, y + yl);
}
//取随机产生的认证码(4 位数字)
String sRand = "";
for (int i = 0; i < 4; i++) {
```

```

String rand = String.valueOf(random.nextInt(10));
sRand += rand;
//将验证码显示到图像中
g.setColor(new Color(20 + random.nextInt(110), 20 + random
.nextInt(110), 20 + random.nextInt(110)));
//调用函数出来的颜色相同, 可能是因为种子太接近, 所以只能直接生成
g.drawString(rand, 13 * i + 6, 16);
}
//将验证码存入 Session
session.setAttribute("rand", sRand);
//图像生效
g.dispose();
//输出图像到页面
ImageIO.write(image, "JPEG", response.getOutputStream());
%>

```

在登录或注册页面, 添加 imageJsp.jsp 的使用代码, 示例代码如下:

```

```

(2) 使用 Servlet 实现

创建一个 Servlet 类 ImageServlet, 重写 doGet() 方法, 示例代码如下:

```

public class ImageServlet extends HttpServlet {
    //处理 GET 方式的请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType(request.getContentType());
        response.setContentType("image/jpeg"); //必须设置 ContentType 为 image/jpeg
        int length = 4; //设置默认生成 4 个数字
        Date d = new Date();
        long lseed = d.getTime();
        java.util.Random r = new Random(lseed); //设置随机种子
        if (request.getParameter("length") != null) {
            try {
                length = Integer.parseInt(request.getParameter("length"));
            } catch (NumberFormatException e) {
            }
        }
        StringBuffer str = new StringBuffer();
        for (int i = 0; i < length; i++) {
            str.append(r.nextInt(9)); //生成随机数字
        }
        //可以在此加入保存验证码的代码
        HttpSession session = request.getSession();
        session.setAttribute("rand", str);
        //创建内存图像
        BufferedImage bi = new BufferedImage(40, 16, BufferedImage.TYPE_INT_RGB);
        Graphics2D g = bi.createGraphics();
        g.clearRect(0, 0, 16, 40);
        g.setColor(Color.RED);
        g.drawString(str.toString(), 4, 12);
    }
}

```



```
try {  
    //使用 JPEG 编码, 输出到 response 的输出流  
    JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(response  
        .getOutputStream());  
    JPEGEncodeParam param = encoder.getDefaultJPEGEncodeParam(bi);  
    param.setQuality(1.0f, false);  
    encoder.setJPEGEncodeParam(param);  
    encoder.encode(bi);  
} catch (Exception ex) {  
}  
}
```

在 web.xml 文件中添加 Servlet 配置, 示例代码如下:

```
<servlet>  
    <servlet-name>ImageServlet</servlet-name>  
    <servlet-class>faq46.ImageServlet</servlet-class>  
</servlet>  
  
<servlet-mapping>  
    <servlet-name>ImageServlet</servlet-name>  
    <url-pattern>/imageServlet</url-pattern>  
</servlet-mapping>
```

在登录或注册页面, 添加 ImageServlet 的使用代码, 示例代码如下:

```

```

疑难点评

在登录和注册功能中, 使用验证码可以避免别人使用软件恶意登录和注册, 增强了系统的安全性。上面介绍了 JSP 和 Servlet 两种实现方式, 在实现过程中将随机数存入 Session, 然后在登录判断或注册逻辑中应将 Session 保存的信息取出与用户输入的比较。读者也可以在此功能基础上进行扩展, 比如实现汉字或特殊效果的数字的验证码。

FAQ10.33 如何实现饼状图、柱状图和曲线图?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

在实现一些数据分析功能时, 为了方便用户使用, 通常将数据分析之后, 利用一些饼状图、柱状图或曲线图表现。

JFreeChart 是一个开源的 Java 图形开发包, 它可以实现柱形图、饼形图、蜡烛图以及曲线图等功能, 可以应用在 C/S 和 B/S 结构的系统中。

1. JFreeChart 应用介绍

JFreeChart 的下载网址为 <http://www.jfree.org/jfreechart/>, 开发包下载之后, 使用过程如下。

- ☐ 创建 Dataset。创建一个数据源对象 Dataset, 用于包含图形中显示的数据。
- ☐ 创建 JFreeChart。创建图形对象 JFreeChart, 将 Dataset 中的数据导入到 JFreeChart 中。
- ☐ 设置 JFreeChart 的显示属性。也可以使用默认的 JFreeChart 显示属性。
- ☐ 生成图片。
- ☐ 在 JSP 页面中显示图片。

JFreeChart 开发包包含的重要类和接口如下。

- ☐ org.jfree.data.general.Dataset: 所有数据源类都要实现的接口。
- ☐ org.jfree.chart.ChartFactory: 由它来产生 JFreeChart 对象。
- ☐ org.jfree.chart.JFreeChart: 代表图形, 对所有图形的调整都要经过它。
- ☐ org.jfree.chart.plot.Plot: 通过 JFreeChart 对象获得它, 然后再通过它对图形外部部分进行调整, 例如坐标轴。
- ☐ org.jfree.chart.renderer.AbstractRenderer: 通过 JFreeChart 对象获取, 然后通过它对图形内部部分进行调整, 例如折线的类型。

2. JFreeChart 应用示例

(1) 曲线图

新建一个类 LineChart, 用于根据时间信息生成曲线图的图片。示例代码如下:

```
public class LineChart
{
    /**
     * 返回生成图片的文件名
     *
     * @param session
     * @param pw
     * @return 生成图片的文件名
     */
    public String getLineXYChart(HttpSession session, PrintWriter pw)
    {
        XYDataset dataset = this.createDataSet();//建立数据集
        String fileName = null;
        //建立 JFreeChart
        JFreeChart chart = ChartFactory.createTimeSeriesChart(
            "JFreeChart 时间曲线序列图", // title
            "Date", // x-axis label
            "Price", // y-axis label
            dataset, // data
            true, // create legend
            true, // generate tooltips
            false // generate URLs
        );
    }
}
```

```
);  
//设置 JFreeChart 的显示属性, 对图形外部部分进行调整  
chart.setBackgroundPaint(Color.red); //设置曲线图背景色  
//设置字体大小, 形状  
Font font = new Font("宋体", Font.BOLD, 16);  
TextTitle title = new TextTitle("JFreeChart 时间曲线序列图", font);  
chart.setTitle(title);  
  
XYPlot plot = (XYPlot) chart.getPlot(); //获取图形的画布  
plot.setBackgroundPaint(Color.lightGray); //设置网格背景色  
plot.setDomainGridlinePaint(Color.green); //设置网格竖线(Domain 轴)颜色  
plot.setRangeGridlinePaint(Color.white); //设置网格横线颜色  
plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 5.0)); //设置曲线图与 xy 轴的距离  
  
plot.setDomainCrosshairVisible(true);  
plot.setRangeCrosshairVisible(true);  
  
XYItemRenderer r = plot.getRenderer();  
if (r instanceof XYLineAndShapeRenderer)  
{  
    XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) r;  
    renderer.setDefaultShapesVisible(true);  
    renderer.setDefaultShapesFilled(true);  
    renderer.setShapesVisible(true); //设置曲线是否显示数据点  
}  
  
//设置 y 轴  
NumberAxis numAxis = (NumberAxis) plot.getRangeAxis();  
NumberFormat numFormater = NumberFormat.getNumberInstance();  
numFormater.setMinimumFractionDigits(2);  
numAxis.setNumberFormatOverride(numFormater);  
  
//设置提示信息  
StandardXYToolTipGenerator tipGenerator = new StandardXYToolTipGenerator(  
    "历史信息:({1} 16:00,{2})", new SimpleDateFormat("MM-dd"),  
    numFormater);  
r.setToolTipGenerator(tipGenerator);  
  
//设置 x 轴 (日期轴)  
DateAxis axis = (DateAxis) plot.getDomainAxis();  
axis.setDateFormatOverride(new SimpleDateFormat("MM-dd"));  
  
ChartRenderingInfo info = new ChartRenderingInfo(  
    new StandardEntityCollection());  
try  
{  
    fileName = ServletUtilities.saveChartAsPNG(chart, 500, 300, info, session); //生成图片  
    //将图片信息写入 PrintWriter 对象  
    ChartUtilities.writeImageMap(pw, fileName, info, false);  
}
```



```

        catch (IOException e)
        {
            e.printStackTrace();
        }
        pw.flush();

        return fileName;//返回生成图片的文件名
    }

    /**
     * 建立生成图形所需的数据集
     *
     *
     * @return 返回数据集
     */
    private XYDataset createDateSet()
    {
        TimeSeriesCollection dataset = new TimeSeriesCollection();//时间曲线数据集
        //创建时间数据源, 每一个 TimeSeries 在图上是一条曲线
        TimeSeries s1 = new TimeSeries("历史曲线", Day.class);
        //s1.add(new Day(day,month,year),value), 添加数据点信息
        s1.add(new Day(1, 2, 2006), 123.51);
        s1.add(new Day(2, 2, 2006), 122.1);
        s1.add(new Day(3, 2, 2006), 120.86);
        s1.add(new Day(4, 2, 2006), 122.50);
        s1.add(new Day(5, 2, 2006), 123.12);
        s1.add(new Day(6, 2, 2006), 123.9);
        s1.add(new Day(7, 2, 2006), 124.47);
        s1.add(new Day(8, 2, 2006), 124.08);
        s1.add(new Day(9, 2, 2006), 123.55);
        s1.add(new Day(10, 2, 2006), 122.53);

        dataset.addSeries(s1);
        dataset.setDomainIsPointsInTime(true);
        return dataset;
    }
}

```

在 web.xml 中添加配置, 使用专用的 Servlet 显示生成的图片, 示例代码如下:

```

<servlet>
    <servlet-name>DisplayChart</servlet-name>
    <servlet-class>
        org.jfree.chart.servlet.DisplayChart
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DisplayChart</servlet-name>
    <url-pattern>/servlet/DisplayChart</url-pattern>
</servlet-mapping>

```


新建 JSP 页面 lineChar.jsp, 显示曲线图, 示例代码如下:

```
<body>
<%
    LineChart xyChart=new LineChart();
    String fileName=xyChart.getLineXYChart(session,new PrintWriter(out));
    String graphURL = request.getContextPath() + "/servlet/DisplayChart?filename=" + fileName;
%>
">
</body>
```

启动服务器, 运行 lineChar.jsp 程序, 效果如图 10-20 所示:

(2) 饼状图

生成饼状图图片的示例代码如下:

```
//创建饼图
public void createPieDemo(String jpgname) {
    DefaultPieDataset dataset = getPieDataset();
```

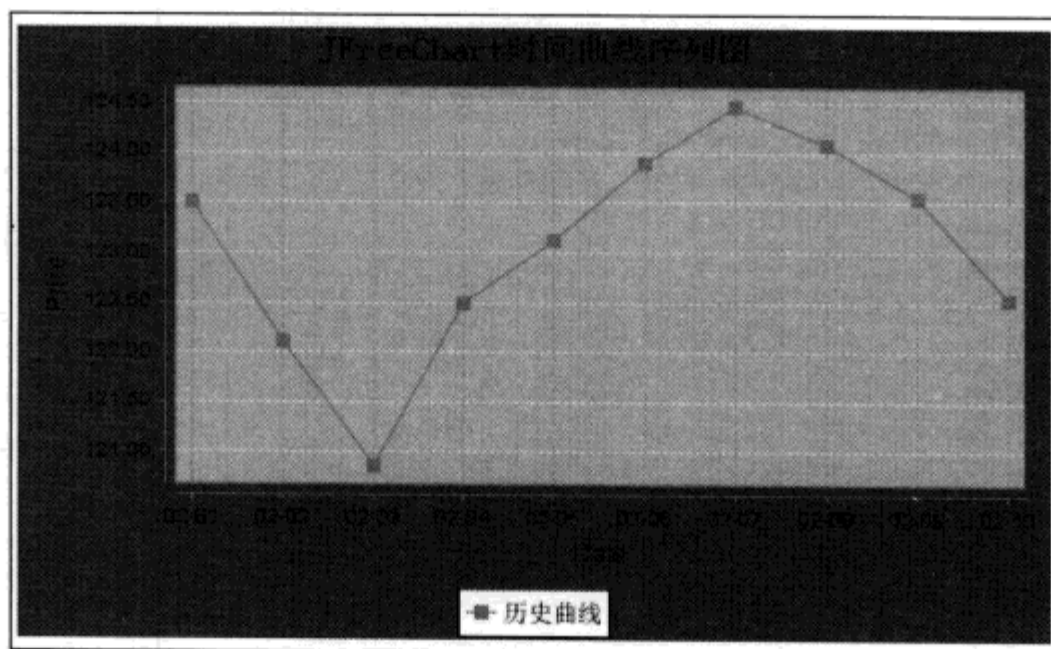


图 10-20 曲线图示例页面

```
JFreeChart chart = ChartFactory.createPieChart3D("水果产量", dataset, true,
    true, true);

FileOutputStream jpg = null;
try {
    jpg = new FileOutputStream(jpgname);
    ChartUtilities.writeChartAsJPEG(jpg, 0.5f, chart, 400, 300, null);

} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        jpg.close();
    } catch (IOException e) {
        // TODO 自动生成 catch 块
    }
}
```

```

        e.printStackTrace();
    }
}

//获取饼图数据
private DefaultPieDataset getPieDataset() {
    DefaultPieDataset dataset = new DefaultPieDataset();
    dataset.setValue("苹果", 100);
    dataset.setValue("梨子", 200);
    dataset.setValue("葡萄", 300);
    dataset.setValue("荔枝", 400);
    dataset.setValue("香蕉", 500);
    dataset.setValue("枣子", 600);
    return dataset;
}

```

运行上述代码, 效果如图 10-21 所示:

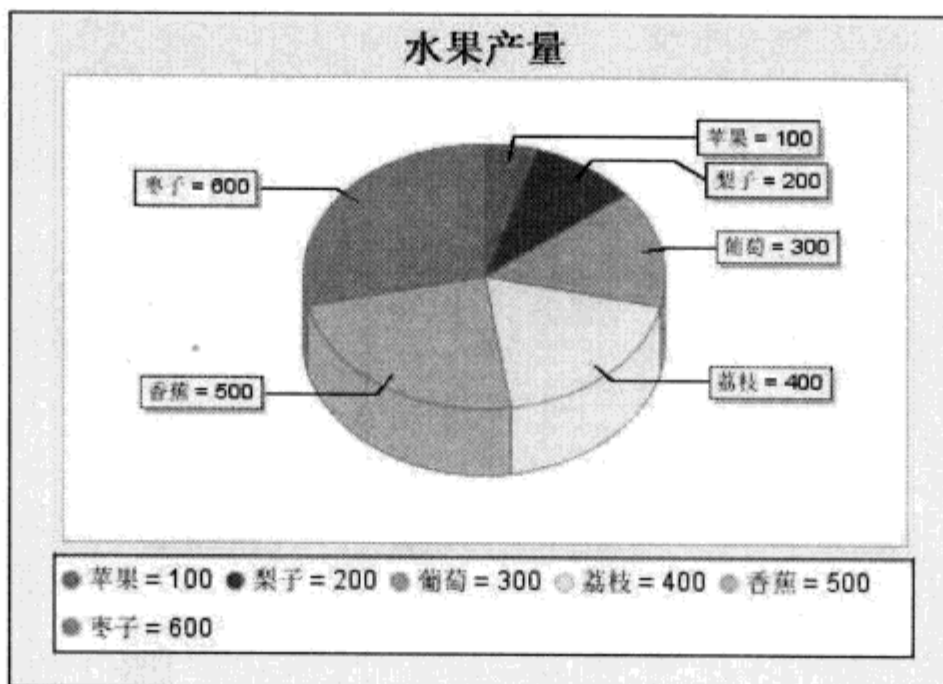


图 10-21 饼状图示例页面

(3) 柱状图

生成柱状图图片的示例代码如下:

```

//创建柱状图
public void createBarDemo(String jpgname) {
    CategoryDataset dataset = getBarDataset();
    JFreeChart chart = ChartFactory.createBarChart3D("水果产量图", "水果", "产量", dataset, PlotOrientation.
    VERTICAL, true, false, false);

    FileOutputStream jpg = null;
    try {
        jpg = new FileOutputStream(jpgname);
        ChartUtilities.writeChartAsJPEG(jpg, 0.5f, chart, 400, 300, null);
    } catch (Exception e) {
    }
}

```

```
e.printStackTrace();
} finally {
    try {
        jpg.close();
    } catch (IOException e) {
        // TODO 自动生成 catch 块
        e.printStackTrace();
    }
}

// 获取柱状图数据
private CategoryDataset getBarDataset() {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    dataset.addValue(100, "北京", "苹果");
    dataset.addValue(200, "上海", "梨子");
    dataset.addValue(300, "南昌", "葡萄");
    dataset.addValue(400, "海南", "香蕉");
    dataset.addValue(500, "北京", "荔枝");
    dataset.addValue(-250, "上海", "荔枝");
    return dataset;
}
```

运行上述代码，效果如图 10-22 所示：

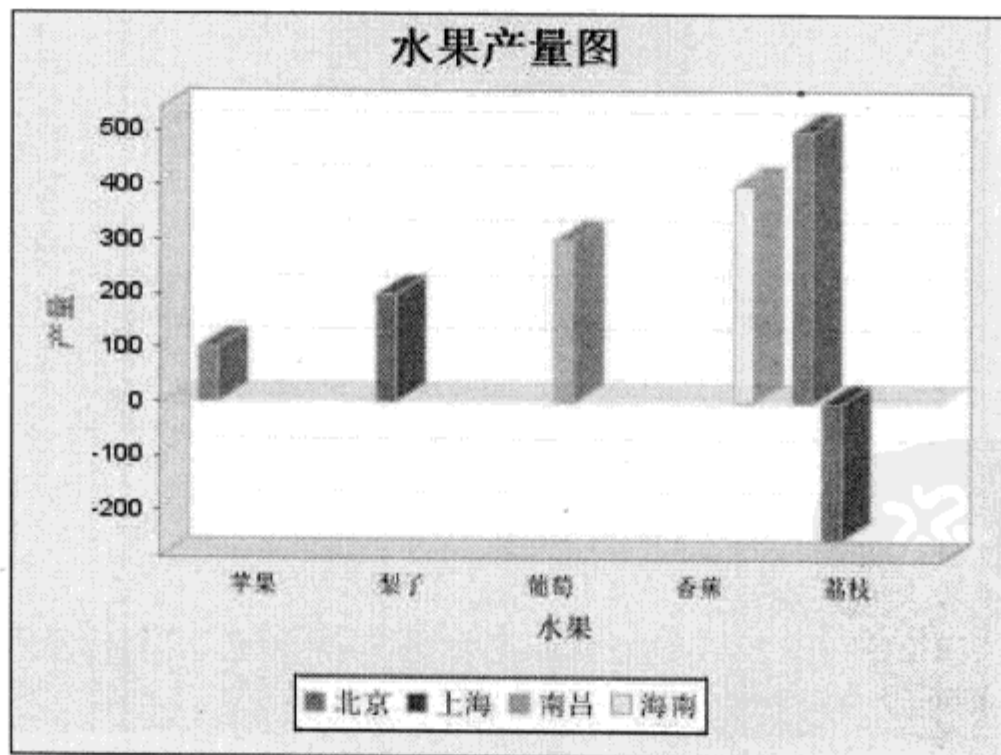


图 10-22 饼状图示例页面

疑难点评

依据数据动态生成图片，现在使用非常普遍，例如股票数据的曲线图、统计报表数据的饼状图和柱状图等。利用 JFreeChart 开发包可以很方便地生成各种类型的图片，既降低了开发的

复杂度,又丰富了系统的交互性。

FAQ10.34 如何实现进度条显示功能?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

在应用程序的安装或下载过程中,进度条的使用非常普遍。进度条可以用来标识项目的完成进度,可以用百分比或数字表示,可以水平放置也可以垂直放置。

进度条在实现时,可以使用 JSP+JavaScript 技术实现,实现时需要频繁地刷新 JSP 页面,因此引入了 AJAX 技术改善该功能,这样用户在使用时没有页面刷新感,界面更加友好。下面介绍如何基于 JSP+AJAX 技术实现进度条。

(1) 服务器代码

服务器代码 progress.jsp, 主要功能是依据客户端的请求信息, 返回相应百分比数字。示例代码如下:

```
<%@ page contentType="text/html; charset=gb2312" language="java"%>
<%!int counter = 1; //注意: 多用户将共享此变量, 此进度条只适合单用户%>
<%
    String task = request.getParameter("task");
    String res = "";
    if (task.equals("create")) {
        res = "1";
        counter = 1;
    } else {
        String percent = "";
        switch (counter) {
            case 1:
                percent = "10";
                break;
            case 2:
                percent = "23";
                break;
            case 3:
                percent = "35";
                break;
            case 4:
                percent = "51";
                break;
            case 5:
                percent = "64";
                break;
            case 6:
                percent = "77";
                break;
            case 7:
                percent = "89";
                break;
            case 8:
                percent = "95";
                break;
            case 9:
                percent = "99";
                break;
            case 10:
                percent = "100";
                break;
        }
        res = percent;
        counter++;
    }
%>
```



```

        xmlhttp.onreadystatechange = goCallback;
        xmlhttp.send(null);
    }
    //服务器处理完一次请求后, 过 2 秒继续发送请求
    function goCallback() {
        if (xmlhttp.readyState == 4) {
            if (xmlhttp.status == 200) {
                setTimeout("pollServer()", 2000);
            }
        }
    }
    //发送一次请求, 请求 progress.jsp
    function pollServer() {
        createXMLHttpRequest();
        var url = "progress.jsp?task=poll";
        xmlhttp.open("GET", url, true);
        xmlhttp.onreadystatechange = pollCallback;
        xmlhttp.send(null);
    }

    //获取 progress.jsp 返回的进度值, 显示进度
    function pollCallback() {
        if (xmlhttp.readyState == 4) {
            if (xmlhttp.status == 200) {
                var percent_complete = xmlhttp.responseXML.getElementsByTagName("percent")[0].first
Child.data;

                var index = processResult(percent_complete);
                for (var i = 1; i <= index; i++) {
                    var elem = document.getElementById("block" + i);
                    elem.innerHTML = clear; elem.style.backgroundColor = bar_color;
                    var next_cell = i + 1;
                    if (next_cell > index && next_cell <= 9) {
                        document.getElementById("block" + next_cell).innerHTML = percent_complete
+ "%";
                    }
                }
                if (index < 9) {
                    setTimeout("pollServer()", 2000);
                } else {
                    document.getElementById("complete").innerHTML = "网站已完成加载!";
                }
            }
        }
    }
    //处理返回的进度值
    function processResult(percent_complete) {
        var ind;
        if (percent_complete.length == 1) {
            ind = 1;
        } else if (percent_complete.length == 2) {

```



```

        ind = percent_complete.substring(0, 1);
    } else {
        ind = 9;
    }
    return ind;
}
//修改进度条的显示
function checkDiv() {
    var progress_bar = document.getElementById("progressBar");
    if (progress_bar.style.visibility == "visible") {
        clearBar();
        document.getElementById("complete").innerHTML = "";
    } else {
        progress_bar.style.visibility = "visible"
    }
}
//清空进度条
function clearBar() {
    for (var i = 1; i < 10; i++) {
        var elem = document.getElementById("block" + i);
        elem.innerHTML = clear;elem.style.backgroundColor = "white";
    }
}
</script>

```

调用上述的主入口方法 go(), 显示进度条的代码如下:

```

<body onload="go();">
    <h1 align=center>
        网站正在加载中, 请稍候
    </h1>
    <p>
    <table align="center">
        <tbody>
            <tr>
                <td>
                    <div id="progressBar"
                        style="padding:2px;border:solid yellow 2px;visibility:hidden">
                        <span id="block1">&nbsp;&nbsp;&nbsp;</span>
                        <span id="block2">&nbsp;&nbsp;&nbsp;</span>
                        <span id="block3">&nbsp;&nbsp;&nbsp;</span>
                        <span id="block4">&nbsp;&nbsp;&nbsp;</span>
                        <span id="block5">&nbsp;&nbsp;&nbsp;</span>
                        <span id="block6">&nbsp;&nbsp;&nbsp;</span>
                        <span id="block7">&nbsp;&nbsp;&nbsp;</span>
                        <span id="block8">&nbsp;&nbsp;&nbsp;</span>
                        <span id="block9">&nbsp;&nbsp;&nbsp;</span>
                    </div>
                </td>
            </tr>
        </tbody>
    </table>

```

```

<td align="center" id="complete"></td>
</tr>
</tbody>
</table>
</body>

```

JavaScript 函数 `createXMLHttpRequest()` 主要用来创建 `XMLHttpRequest` 对象；`go()` 函数完成向服务器端发送异步请求，该函数在网页加载时被调用，其主要作用就是通知服务器端，并在客户端开始运行进度条；`goCallback()` 函数主要处理服务器端响应，并每隔 2 秒调用 `pollServer()` 函数；`pollServer()` 函数也是向服务器端发送异步请求，主要请求服务器端响应的百分比数字；`pollCallback()` 函数主要处理服务器端响应，即依据服务器端的返回数字，指定进度条的显示状态。这里需要注意的是 `goCallback()` 函数只执行一次，而 `pollCallback()` 函数可以执行多次；其余的三个函数都是实现进度条的辅助函数。

上述示例的运行效果如图 10-23 所示：

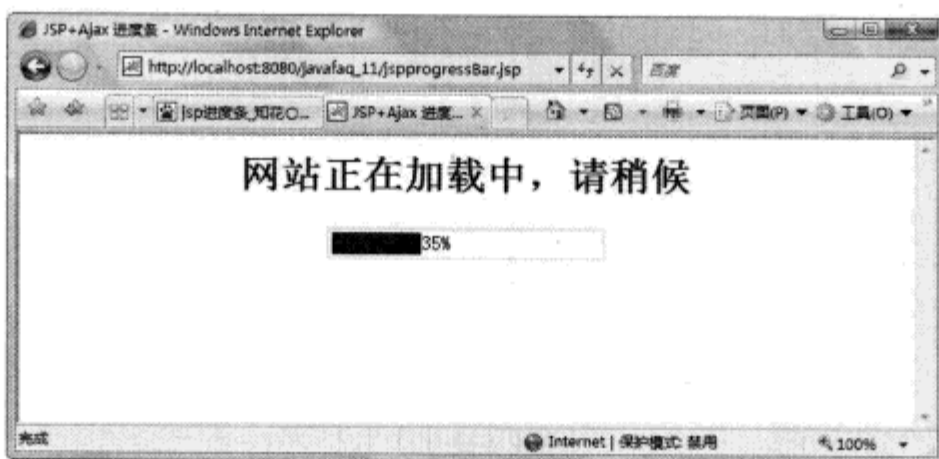


图 10-23 进度条示例页面

疑难点评

进度条功能可以与上传、下载和其他长时间的操作结合起来，这样增强了在服务器处理过程中与用户的交互。进度条使得用户界面更加友好，它可以不断地告诉用户当前操作的执行进度，避免用户频繁地刷新当前页面。

FAQ10.35 如何实现网站计数器功能?

📖 难度系数：★★★★

📖 问题频率：80%

核心解答

在 JSP 页面中实现网站计数器的方法有很多，其中比较普遍的做法是利用 `application` 和 `session` 对象。`application` 对象可被所有用户共享；`session` 是单用户独享，用户从访问系统开始到退出系统，都可以使用 `session`。网站计数器的实现思路是：当系统创建一个 `session` 对象时，

将 application 中保存的计数变量加 1。

示例代码如下：

```
<%@ page language="java" contentType="text/html; charset=GBK" pageEncoding="GBK"%>
<%!
    synchronized void countPeople()//将计数器加 1 函数
    {   ServletContext  application=getServletContext();
        Integer number=(Integer)application.getAttribute("Count");
        if(number==null)    //如果是第 1 个访问本站
        {   number=new Integer(1);
            application.setAttribute("Count",number);
        }
        else
        {   number=new Integer(number.intValue()+1);
            application.setAttribute("Count",number);
        }
    }
%>
<% if(session.isNew())//如果是一个新的会话
{
    countPeople();
}
Integer yourNumber=(Integer)application.getAttribute("Count");
%>
<P><P>欢迎访问本站，您是第<%=yourNumber%>个访问用户。
```

在上述代码中，为了避免使用 application 对象出现并发问题，因此使用了 synchronized 关键字。为了避免页面刷新问题，添加了 session.isNew() 判断，如果 session 对象是新建的，表明用户开始访问系统，将计数器加 1；当用户刷新当前页面时，由于 session 对象不是新建的，因此避免了重复加 1 的问题。

疑难点评

在实现计数器功能时，需要注意一些细节问题，例如频繁刷新和并发访问等。由于 application 对象的生命周期与服务器启动和停止相关，因此如果服务器由于出现问题重启，application 保存的计数器将清空。要解决该问题可以将计数器的值保存在文件中，利用读写文件的功能再对计数器加 1。

FAQ10.36 如何发送 HTML 格式和带附件的邮件？

📖 难度系数：★★★★★

📖 问题频率：85%

核心解答

JavaMail 是 Sun 公司发布的用来处理 E-mail 的 API，它可以方便地执行一些常用的邮件传输。虽然 JavaMail 是 Sun 公司的 API 之一，但目前还没有被加入到 JDK 中，因此实现发送邮

件功能之前, 需要另外下载 JavaMail 开发包。除此以外, 还需要有 Sun 的 JavaBeans Activation Framework (即 JAF), JavaMail 的运行必须得依赖于它的支持。

下面介绍如何使用 JavaMail 发送各种形式的邮件。

(1) 准备工作

在使用 JavaMail 之前, 需要获取 mail.jar 和 activation.jar 开发包。从网址 <http://java.sun.com/products/javamail/downloads/> 下载最新版 JavaMail, 解压缩后将 mail.jar 复制到应用程序的 WEB-INF、lib 目录下。

从网址 <http://java.sun.com/products/javabeans/jaf/> 下载 JAF, 解压缩后将 activation.jar 复制到应用程序的 WEB-INF、lib 目录下。

(2) 发送文本格式的邮件

发送文本格式的邮件, 示例代码如下:

```
public class SendMail{
    public static void main(String[] args){
        try{
            //用于设置系统属性
            Properties props = new Properties();

            //创建专用于发送 E-mail 的 Session 对象
            Session sendMailSession;

            //Transport 是专用于发送邮件的类
            Transport transport;

            //得到一个 Session 类型的对象
            sendMailSession = Session.getInstance(props, null);

            //在系统属性中设置发送邮件服务器
            props.put("mail.smtp.host", "127.0.0.1");

            //Message 类是专用于描述发送邮件信息的类, Message 是抽象类,
            Message newMessage = new MimeMessage(sendMailSession);

            //设置发件人
            newMessage.setFrom(new InternetAddress("system@zhangsan.com"));

            //设置收件人, Message.RecipientType.CC 则是设置抄送者
            newMessage.setRecipient(Message.RecipientType.TO, new InternetAddress("ljq@163.com"));

            newMessage.setSubject("问候"); //邮件主题
            newMessage.setSentDate(new Date()); //发送时间
            newMessage.setText("这是一个测试! \n 哈哈, 你好! \n 呵呵! "); //内容
            //得到 Transport 类的实例
            transport = sendMailSession.getTransport("smtp");
            transport.send(newMessage);
        } catch(MessagingException m) {
```

```
        m.printStackTrace();
    }
}
```

(3) 发送 HTML 格式的邮件

发送 HTML 格式的邮件，首先需要获取 HTML 页面信息，然后通过 `MimeBodyPart` 类的 `setContent()` 方法设置邮件内容及类型。主要示例代码如下：

```
<%
    InetAddress[] address = null;
    String mailserver = "smtp.moochi.com";//发出邮箱的服务器
    String From = "moochi@moochi.com";//发出的邮箱
    String to = "zhang-xinjie@163.com";//发到的邮箱
    String Subject = "嗨，亲爱的";//标题
    String type = "text/html";//发送邮件格式为 html
    String messageText = "<html><head>.....</head></html>";//获取要发送的页面的信息
    try {

        //设定所要用的 Mail 服务器和所使用的传输协议
        java.util.Properties props = System.getProperties();
        props.put("mail.host", mailserver);
        props.put("mail.transport.protocol", "smtp");
        props.put("mail.smtp.auth", "true");//指定是否需要 SMTP 验证

        //产生新的 Session 服务
        Session mailSession = javax.mail.Session.getDefaultInstance(props, null);
        Message msg = new MimeMessage(mailSession);

        // 设定发邮件的人
        msg.setFrom(new InetAddress(From));

        // 设定收信人的信箱
        address = InetAddress.parse(to, false);
        msg.setRecipients(Message.RecipientType.TO, address);

        // 设定信中的主题
        msg.setSubject(Subject);

        // 设定送信的时间
        msg.setSentDate(new Date());

        Multipart mp = new MimeMultipart();
        MimeBodyPart mbp = new MimeBodyPart();

        // 设定邮件内容的类型为 text/plain 或 text/html
        mbp.setContent(messageText, type + ";charset=utf8");
        mp.addBodyPart(mbp);
        msg.setContent(mp);
    }
}
```



```

        Transport transport = mailSession.getTransport("smtp");
        //请填入你的邮箱用户名和密码,千万别用我的^_^
        transport.connect(mailserver, "moochi", "moochi123");//设置发出邮箱的用户名、密码
        transport.sendMessage(msg, msg.getAllRecipients());
        transport.close();
        //Transport.send(msg);
        out.println("邮件已顺利发送");

    } catch (MessagingException mex) {
        mex.printStackTrace();
        out.println(mex);
    }
    try {
        response.sendRedirect("../indexSelf.jsp");//转向某页
    } catch (Exception e) {
        e.printStackTrace();
    }
}
%>

```

在上述代码中, messageText 变量用于存放要发送的 HTML 页面的信息, HTML 页面应事先存在, 可以使用 URL 或 I/O 流读取并以字符串形式返回。

(4) 发送带附件的邮件

发送带附件的邮件, 示例代码如下:

```

public class MyMail {
    String[] to;
    String[] cc;
    String[] bcc;
    String from;
    String host="";
    String subject="";
    String body="";
    Vector files = new Vector();

    public MyMail (String[] to,String from,String smtpServer,String subject,String body){
        this.to=to;
        this.from=from;
        this.host=smtpServer;
        this.subject=subject;
        this.body = body;
    }

    public void setCc(String[] cc){//抄送
        this.cc = cc;
    }

    public void setBcc(String[] bcc){//暗送
        this.bcc = bcc;
    }
}

```



```
public void attachfile(String fname){
    files.addElement(fname);
}

public boolean send(){
    Properties props = System.getProperties();
    props.put("mail.smtp.host", host);
    Session session=Session.getDefaultInstance(props, null);

    try {
        MimeMessage msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress(from));

        //设置收件人
        InternetAddress[] address= new InternetAddress[to.length];
        for(int i=0;i<to.length;i++){
            address[i] = new InternetAddress(to[i]);
        }
        msg.setRecipients(Message.RecipientType.TO,address);

        //设置抄送人
        if(cc != null){
            address= new InternetAddress[cc.length];
            for(int i=0;i<cc.length;i++){
                address[i] = new InternetAddress(cc[i]);
            }
            msg.setRecipients(Message.RecipientType.CC,address);
        }

        //设置暗送人
        if(bcc != null){
            address= new InternetAddress[bcc.length];
            for(int i=0;i<bcc.length;i++){
                address[i] = new InternetAddress(bcc[i]);
            }
            msg.setRecipients(Message.RecipientType.BCC,address);
        }

        //设置邮件主题
        msg.setSubject(subject);

        //创建复合邮件体, 后面的 BodyPart 将加入到此处创建的 Multipart 中
        Multipart mp = new MimeMultipart();

        MimeBodyPart mbp = null;
        mbp = new MimeBodyPart();
        mbp.setText(body);
        mp.addBodyPart(mbp);

        for(int i=0;i<files.size();i++){
```

```
String fileName = (String)files.elementAt(i);
MimeBodyPart mbp=new MimeBodyPart();
FileDataSource fds=new FileDataSource(fileName);
mbp.setDataHandler(new DataHandler(fds));
mbp.setFileName(fds.getName());
mp.addBodyPart(mbp);
}
files.removeAllElements();
msg.setContent(mp);
msg.setSentDate(new Date());
Transport.send(msg);
} catch (MessagingException mex) {
    mex.printStackTrace();
    return false;
}
return true;
}

public static void main(String[] args){
    String[] to = {"test1@163.com"};
    String[] cc = {"test2@163.com","test3@163.com"};
    String[] bcc = {"test4@163.com"};
    String body = "最后一次想你! \nbye-bye,world!";
    MyMail mail = new MyMail(to,"scott@zhangsan.com","127.0.0.1","测试带附件的邮件",body);
    mail.setCc(cc);
    mail.setBcc(bcc);
    mail.attachfile("E:\\ex\\love.pps");
    mail.send();
}
```

疑难点评

目前发送邮件功能非常普遍,在Java中,利用JavaMail开发包可以方便地发送各种形式的邮件,例如文本格式、HTML格式和带附件的邮件等。JavaMail不仅可以应用于JSP中,在Servlet和其他Java应用程序中同样适用。

FAQ10.37 如何实现文件的上传和下载?

📖 难度系数: ★★★★★

📖 问题频率: 85%

核心解答

在JSP程序中,使用第三方组件可以方便地实现上传和下载功能。jspSmartUpload是由www.jspsmart.com网站开发的可免费使用的组件,可用于实现文件的上传和下载,适用于嵌入

执行上传和下载操作的 JSP 程序中。

下面介绍如何使用 jspSmartUpload 组件实现 JSP 中的上传和下载。

1. 组件获取

jspSmartUpload 组件可以从 www.jspsmart.com 网站上自由下载，压缩包的名字是 jspSmartUpload.zip。下载后进行解压缩，为了方便应用，在引入 JSP 工程之前可通过 JAR 命令：`jar cvf jspSmartUpload.jar com` 将组件程序打包。（也可以打开资源管理器，切换到当前目录，用 WinZip 等压缩软件将 com 目录下的所有文件压缩成 jspSmartUpload.zip，然后将 jspSmartUpload.zip 的扩展名改为 jar。）

2. 工具类介绍

将 jspSmartUpload.jar 引入 JSP 工程之后，就可以在 JSP 程序使用该组件的工具类。各工具类的使用说明如下：

(1) File 类

File 类包含了一个上传文件的所有信息，通过它可以得到上传文件的文件名、文件大小、扩展名、文件数据等，主要方法如下。

- ☐ `saveAs()`: 将文件换名另存。
- ☐ `isMissing()`: 用于判断用户是否选择了文件，即对应的表单项是否有值。用户在表单中选择文件时，该方法返回 `false`；未选择文件时，则返回 `true`。
- ☐ `getFileName()`: 取文件名（不含目录信息）。
- ☐ `getFilePathName()`: 取文件全名（带目录）。
- ☐ `getFileExt()`: 取文件扩展名（后缀）。
- ☐ `getSize()`: 取文件长度（以字节计）。
- ☐ `getBinaryData()`: 取文件数据中指定位移处的一个字节，用于检测文件等处理。

(2) Files 类

Files 类表示所有上传文件的集合，通过它可以得到上传文件的数目、大小等，主要方法如下：

- ☐ `getCount`: 取得上传文件的数目。
- ☐ `getFile`: 取得指定位移处的文件对象 File（这是 `com.jspsmart.upload.File`，不是 `java.io.File`，注意区分）。
- ☐ `getSize`: 取得上传文件的总长度，可用于限制一次性上传的数据量大小。

(3) Request 类

Request 类的功能等同于 JSP 内置的对象 `request`。之所以提供这个类，是因为对于文件上传表单，通过 `request` 对象无法获得表单项的值，必须通过 jspSmartUpload 组件提供的 Request 对象来获取。具体使用方法与 JSP 内置对象 `request` 相似。

(4) SmartUpload

SmartUpload 类用于完成上传和下载功能，主要方法如下。

- ☐ `initialize()`: 执行上传下载的初始化工作，必须第一个执行。

- ❑ `upload()`: 上传文件数据。
- ❑ `save()`: 将全部上传文件保存到指定目录下, 并返回保存的文件个数。
- ❑ `getSize()`: 取上传文件数据的总长度。
- ❑ `getFiles()`: 取全部上传文件, 以 `Files` 对象形式返回, 可以利用 `Files` 类的操作方法来获得上传文件的数目等信息。
- ❑ `getRequest()`: 取得 `Request` 对象, 以便由此对象获得上传表单参数之值。
- ❑ `setAllowedFilesList()`: 设定允许上传带有指定扩展名的文件, 当上传过程中有文件名不允许时, 组件将抛出异常。允许上传的文件扩展名列表, 各个扩展名之间以逗号分隔。如果想允许上传那些没有扩展名的文件, 可以用两个逗号表示。 `SetAllowedFilesList("doc,txt,")`。
- ❑ `setDeniedFilesList()`: 用于限制上传那些带有指定扩展名的文件。若有文件扩展名被限制, 则上传时组件将抛出异常, 该方法的使用与 `setAllowedFilesList()` 相似。
- ❑ `setMaxFileSize()`: 设定每个文件允许上传的最大长度。
- ❑ `setTotalMaxFileSize()`: 设定允许上传的文件的总长度, 用于限制一次性上传的数据量大小。
- ❑ `setContentDisposition()`: 将数据追加到 MIME 文件头的 `CONTENT-DISPOSITION` 域。`jspSmartUpload` 组件会在返回下载的信息时自动填写 MIME 文件头的 `CONTENT-DISPOSITION` 域, 如果用户需要添加额外信息, 请用此方法。
- ❑ `downloadFile()`: 下载文件。

3. 实现上传

上传页面的表单 `<form>` 标记必须满足以下两个要求。

- ❑ `method` 属性必须 `POST`, 即 `method="POST"`。
- ❑ 增加 `enctype` 属性, 即 `enctype="multipart/form-data"`。

上传页面表单的示例代码如下:

```
<FORM method="POST" enctype="multipart/form-data" action="/upload.jsp">
<INPUT TYPE="FILE" NAME="MYFILE">
<INPUT TYPE="SUBMIT">
</FORM>
```

`upload.jsp` 中实现上传的代码如下:

```
<%@ page contentType="text/html; charset=GBK" import="com.jspsmart.upload.*"%>
<html>
<head>
<title>文件上传处理页面</title>
<meta http-equiv="Content-Type" content="text/html; charset=GBK">
</head>
<body>
<%
//新建一个 SmartUpload 对象
SmartUpload su = new SmartUpload();
```

```

//上传初始化
su.initialize(pageContext);
//设定上传限制
//1.限制每个上传文件的最大长度
su.setMaxFileSize(10000);
//2.限制总上传数据的长度
su.setTotalMaxFileSize(20000);
//3.设定允许上传的文件（通过扩展名限制），仅允许 doc，txt 文件
su.setAllowedFilesList("doc,txt");
//4.设定禁止上传的文件（通过扩展名限制），禁止上传带有 exe，bat，jsp，htm，html 扩展名的文件和没有扩展名
的文件
su.setDeniedFilesList("exe,bat,jsp,htm,html,");
//上传文件
su.upload();
//将上传文件全部保存到指定目录
int count = su.save("/upload");
out.println(count+"个文件上传成功！<br>");

//利用 Request 对象获取参数之值
out.println("TEST="+su.getRequest().getParameter("TEST" + "<BR><BR>"));
%>
</body>
</html>

```

4. 实现下载

下载页面表单的示例代码如下：

```

<html>
  <head>
    <title>下载</title>
  </head>
  <body>
    <a href="/download.jsp">单击下载</a>
  </body>
</html>

```

download.jsp 中实现下载的代码如下：

```

<%@ page contentType="text/html;charset=gbk" import="com.jspsmart.upload.*" %>
//新建一个 SmartUpload 对象
SmartUpload su = new SmartUpload();
//初始化
su.initialize(pageContext);
//设定 contentDisposition 为 null 以禁止浏览器自动打开文件，保证单击链接后是下载文件
//若不设定，则下载的文件扩展名为 doc 时，浏览器将自动用 Word 打开它。扩展名为 pdf 时，浏览器将用 acrobat
打开
su.setContentDisposition(null);
//下载文件
su.downloadFile("/upload/为什么呢.doc");
%>

```

注意：在 download.jsp 页面中，除 Java 脚本外（即<% ... %>），不要包含 HTML 代码、空

格、回车或换行等字符,否则将会影响下载代码的执行。

在下载过程中,如果下载文件名是中文会产生乱码,为了解决该问题,可以在下载前将中文文件名采用 UTF-8 编码。转换编码的代码如下:

```
/**
 * 将文件名中的汉字转为 UTF-8 编码的串,以便下载时能正确显示另存的文件名
 * @param s 原文件名
 * @return 重新编码后的文件名
 */
public static String StringtoUtf8(String s) {
    StringBuffer sb = new StringBuffer();
    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (c >= 0 && c <= 255) {
            sb.append(c);
        } else {
            byte[] b;
            try {
                b = Character.toString(c).getBytes("utf-8");
            } catch (Exception ex) {
                System.out.println(ex);
                b = new byte[0];
            }
            for (int j = 0; j < b.length; j++) {
                int k = b[j];
                if (k < 0)
                    k += 256;
                sb.append("%" + Integer.toHexString(k).toUpperCase());
            }
        }
    }
    return sb.toString();
}
```

疑难点评

jspSmartUpload 组件是 JSP 程序中经常使用的上传和下载组件,它使用简单、方便。在实现下载功能时,通过编码转换可以解决中文名字文件下载时文件名乱码的问题,增强了对中文的支持,大大提高了 jspSmartUpload 组件实用性。在使用时开发者还可以将其和进度条功能结合起来,实现更加友好的界面效果。

FAQ10.38 如何禁止浏览器缓存页面内容?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

浏览器为了避免频繁地与服务器交互, 提供了缓存功能, 当频繁地访问同一 JSP 页面时, 浏览器会将缓存中的结果取出为用户响应, 并没有向服务器发送请求, 因此经常出现不能反映服务器最新处理结果的情况。

如果某个 JSP 或 Servlet 程序不需要客户浏览器缓存其相应内容, 可以添加如下代码:

```
response.setHeader("Pragma","No-cache");  
response.setHeader("Cache-Control","no-cache");  
response.setDateHeader("Expires",-10);
```

疑难点评

禁用浏览器缓存 JSP 或 Servlet 的响应结果, 在用户频繁地访问同一网页时, 保障每次都向服务器发送请求, 这样可以将服务器最新结果显示给用户。在 Servlet 中使用上述代码时, 需要用在 response.getWriter() 方法之前, 否则无效。

FAQ10.39 如何在网页中在线播放音乐和视频?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

在网页中播放多媒体文件, 主要使用 HTML 中的 <object> 和 <embed> 标记实现。可用于播放 mp3 和 wma 格式的音乐, 示例代码如下:

```
<object id="MPPlay" width="30%" height="20%"  
  classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">  
  <param name="url" value="6.rm" >  
  <param name="AudioStream" value="-1" >  
  <!--是否自动调整播放大小-->  
  <param name="AutoSize" value="-1" >  
  <!--是否自动播放-->  
  <param name="AutoStart" value="-1" >  
  <param name="AnimationAtStart" value="-1">  
  <param name="AllowScan" value="-1">  
  <param name="AllowChangeDisplaySize" value="-1">  
  <param name="AutoRewind" value="0">  
  <!--左右声道平衡, 最左-9640, 最右 9640-->  
  <param name="Balance" value="0">  
  <param name="BaseURL" value="">  
  <!--缓冲时间-->  
  <param name="BufferingTime" value="15">  
  <param name="CaptioningID" value>
```

```

<param name="ClickToPlay" value="-1">
<param name="CursorType" value="0">
<!--播放速率控制, 1 为正常, 允许小数, 1.0 ~ 2.0-->
<param name="rate" value="1">
<!--控件设置: 当前位置-->
<param name="currentPosition" value="0">
<!--当前播放进度-1 表示不变, 0 表示开头单位是秒, 比如 10 表示从第 10 秒处开始播放, 值必须是-1.0 或大于等于 0-->
<param name="currentMarker" value="0">
<!--显示默认框架-->
<param name="defaultFrame" value="">
<param name="DisplayBackColor" value="0">
<param name="DisplayForeColor" value="16777215">
<param name="DisplayMode" value="0">
<!--视频 1-50%, 0-100%, 2-200%, 3-全屏, 其他的值做 0 处理, 小数则采用四舍五入然后按前面的处理-->
<param name="DisplaySize" value="0">
<param name="Enabled" value="-1">
<!--是否用右键弹出菜单控制-->
<param name="EnableContextMenu" value="-1">
<param name="EnablePositionControls" value="-1">
<param name="EnableFullScreenControls" value="-1">
<!--是否允许拉动播放进度条到任意地方播放-->
<param name="EnableTracker" value="-1">
<!--脚本命令设置: 是否调用 URL-->
<param name="InvokeURLs" value="-1">
<param name="Language" value="-1">
<!--默认声音大小 0%-100%, 50 则为 50%-->
<param name="volume" value="50">
<!--是否静音-->
<param name="Mute" value="0">
<!--重复播放次数, 0 为始终重复-->
<param name="PlayCount" value="10">
<param name="PreviewMode" value="-1">
<!--播放速率控制, 1 为正常, 允许小数-->
<param name="Rate" value="1">
<!--SAMI 样式-->
<param name="SAMISyle" value="">
<!--SAMI 语言-->
<param name="SAMILang" value="">
<!--字幕 ID-->
<param name="SAMIFilename" value="">
<!--是否显示字幕, 为一块黑色, 下面会有一大块黑色, 一般不显示-->
<param name="ShowCaptioning" value="0">
<!--是否显示控制, 比如播放, 停止, 暂停-->
<param name="ShowControls" value="-1">
<!--是否显示音量控制-->
<param name="ShowAudioControls" value="-1">
<!--显示节目信息, 比如版权等-->
<param name="ShowDisplay" value="0">
<!--是否启用上下文菜单-->

```

```
<param name="ShowGotoBar" value="0">
<!--是否显示往前往后及列表, 如果显示一般也都是灰色不可控制-->
<param name="ShowPositionControls" value="-1">
<!--当前播放信息, 显示是否正在播放, 及总播放时间和当前播放到的时间-->
<param name="ShowStatusBar" value="-1">
<!--是否显示当前播放跟踪条, 即当前的播放进度条-->
<param name="ShowTracker" value="-1">
<param name="TransparentAtStart" value="-1">
<!--指定显示部分的宽度如果小于视频宽, 则最小为视频宽, 或者加大到指定值, 并自动加大高度。此改变只改变四周的黑框大小, 不改变视频大小-->
<param name="VideoBorderWidth" value="0">
<!--指定黑色框的颜色, 为 RGB 值, 比如 fff000 为黄色-->
<param name="VideoBorderColor" value="0">
<param name="VideoBorder3D" value="0">
<!--音量大小, 负值表示是当前音量的减值, 值自动会取绝对值, 最大为 0, 最小为 -9640-->
<param name="Volume" value="0">
<!--如果是 0 可以允许全屏, 否则只能在窗口中查看-->
<param name="WindowlessVideo" value="0">
</object>
```

在使用上述代码时, 客户端还需要安装相应的播放器才可以, 例如 Media Player 和 Real One 等。

疑难点评

网页中除了可以嵌入图片之外, 还可以嵌入各种类型的播放器, 用于播放不同类型的文件。由于多媒体文件类型非常繁多, 因此在使用时, 需要根据要播放的文件类型选择不同的实现代码。客户计算机操作系统一般都内置安装了 Media Player 播放器, 因此播放 mp3 和 wma 类型文件不会有什么问题; 但是如果遇到 rm 格式的, 需要客户计算机安装 Real One 播放器才能正常使用。

FAQ10.40 如何处理 JSP 页面的错误?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

根据 JSP 对错误的处理方式不同可以将其分为局部异常处理和全局异常处理。局部异常处理适用于个别 JSP 页面, 当这些页面发生错误后, 采取特殊的处理方式; 全局异常处理适用于所有 JSP 页面, 当所有页面发生某些指定错误后, 采取统一方式处理。

(1) 局部异常处理

局部异常处理主要涉及 JSP 页面 page 指令的 errorPage 和 isErrorPage 属性。

errorPage 属性用于设置错误处理的 JSP 页面, 如果当前 JSP 页面内产生了未被捕获的异常,

则跳转到 `errorPage` 指定的 JSP 页面进行处理。

`isErrorPage` 属性用于错误处理页面, 只有将 JSP 页面中的 `isErrorPage` 属性设置为 `true`, 此 JSP 页面才能被用作错误处理页面, 在该 JSP 的代码中才可以使用 `exception` 隐式对象。

错误处理页面 `dealError.jsp`, 示例代码如下:

```
<%@ page isErrorPage="true"%>
<%@ page import="java.io.PrintWriter" %>
<%
    out.println(" out of deal error!");
    exception.printStackTrace(new PrintWriter(out));
%>
```

业务处理页面 `divide.jsp`, 示例代码如下:

```
<%@ page errorPage="dealError.jsp"%>
<%
    out.println("before exception!");
    int x=1/0;
    out.println("after exception");
%>
```

(2) 全局异常处理

通过在 `web.xml` 文件中添加 `<error-page>` 配置, 可以为整个 Web 应用程序设置异常处理页面。每个 `<error-page>` 元素用于设置一种异常或者一个 HTTP 错误状态码的处理页面。

`web.xml` 中的示例代码如下:

```
<!--依据 HTTP 错误状态码设置错误处理页面-->
<error-page>
    <error-code>404</error-code>
    <location>/errorhandler.jsp</location>
</error-page>
<!--依据异常类型设置错误处理页面-->
<error-page>
    <exception-type>javax.servlet.ServletException</exception-type>
    <location>/errorhandler.jsp</location>
</error-page>
```

在上述代码中, `<error-page>` 元素中的子元素 `<exception-type>` 和 `<error-code>`, 两者是二选一的关系。

错误处理页面 `errorhandler.jsp`, 示例代码如下:

```
<%@page contentType="text/html; charset=GBK" isErrorPage="true"%>
<html>
<head><title>错误与例外处理页面</title></head>
<body>
    错误码: <%=request.getAttribute("javax.servlet.error.status_code")%> <br>
    讯息: <%=request.getAttribute("javax.servlet.error.message")%> <br>
    例外: <%=request.getAttribute("javax.servlet.error.exception_type")%> <br>
</body>
</html>
```

疑难点评

上述的异常处理方法可以对 JSP 页面的错误进行特殊或统一的处理, 通过对 JSP 页面应用错误处理, 可以增强系统的稳定性和友好性。除了使用上述方法外, 也可以使用 try-catch-finally 语句对异常捕获处理。

知识链接

FAQ10.04 如何开发一个 JSP 程序?

FAQ10.41 如何利用过滤器实现权限验证功能?

📖 难度系数: ★★★★★

📖 问题频率: 90%

核心解答

为了增强系统的安全性, 有些页面要求只有登录的用户或者具有操作权限的用户才能访问。在对系统进行权限控制时, 实现方法有很多, 但推荐使用过滤器方式实现, 它可以将权限控制和业务处理分开, 提高了程序的扩展性和维护性。

过滤器可用于过滤客户请求, 当用户请求服务器的 JSP 或 Servlet 文件时, 可以使用过滤器对请求进行拦截处理, 其作用类似于系统的防火墙。用过滤器实现权限验证功能, 可以对客户请求进行过滤, 符合验证条件的允许继续访问 JSP 或 Servlet, 否则拦截请求, 将页面跳转到错误提示页面。

下面介绍一个登录验证的过滤器, 即只有登录用户才能访问系统页面。

❑ 过滤器代码

过滤器的实现类需要实现 javax.servlet.Filter 接口, 示例代码如下:

```
public class CheckFilter implements Filter {
    private FilterConfig filterConfig;

    //登录页面, 当用户没有登录时, 将会首先转到这个页面
    private String loginPage = "/login.jsp";

    public void init(FilterConfig config) throws ServletException {
        //通过 FilterConfig 获得 web.xml 中设置的初始化参数
        filterConfig = config;
        if (filterConfig.getInitParameter("loginPage") != null)
            loginPage = filterConfig.getInitParameter("loginPage");
    }

    public void doFilter(ServletRequest request, ServletResponse response,
```

```

        FilterChain chain) throws IOException, ServletException {
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse res = (HttpServletResponse) response;
    //通过判断 session 中是否具有 auth 参数来判断用户是否已经登录
    HttpSession session = req.getSession(true);
    //如果已经登录
    if (session.getAttribute("auth") != null) {
        chain.doFilter(req, res);
        return;
    }
    //尚未登录
    else {
        ServletContext ctx = filterConfig.getServletContext();
        //跳转到登录页面
        ctx.getRequestDispatcher(loginPage).forward(req, res);
    }
}

public void destroy() {
    filterConfig = null;
}
}

```

javax.servlet.Filter 接口定义了 init()、doFilter()和 destroy()方法, init()方法用于初始化, 在服务器启动时调用; destroy()方法用于资源释放, 在服务器关闭时调用; doFilter()方法用于实现请求过滤, 每次客户发出与过滤器匹配的请求时, 都会调用 doFilter()方法处理。可以在 doFilter()方法中添加处理逻辑, 实现请求的过滤。

❑ 过滤器配置

```

<filter>
    <filter-name>CheckFilter</filter-name>
    <filter-class>faq54.CheckFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>CheckFilter</filter-name>
    <url-pattern>/manager/*</url-pattern>
</filter-mapping>

```

在上述配置中, 符合 http://localhost:8080/manager/*格式的请求, 将调用 CheckFilter 过滤器处理, 即调用 doFilter()方法。

注意: 登录页面需要和权限控制的页面分开存放, 否则会发生死循环。

疑难点评

过滤器可以实现代码的重复利用, 降低程序的耦合性, 使程序变得灵活。过滤器不仅可以实现权限控制, 其他共同处理也可以使用过滤器封装, 例如解决中文乱码的代码每个页面都要使用, 因此可以利用过滤器解决整个系统页面的乱码问题。

FAQ10.42 如何实现 JSP 防盗链功能?

📖 难度系数: ★★★★★

📖 问题频率: 80%

核心解答

在 Web 系统中, 盗链的问题时有发生, 即复制一个 URL 地址, 在另一个地方也能访问。

在 JSP 中通过 request 对象可以获取客户请求信息和表单信息, 在客户请求头信息中, “Referer” 属性的信息是上一次请求的 URL, 利用 “Referer” 属性值可以解决盗链的问题。

例如 index.jsp 页面的示例代码如下:

```
<html>
  <head><title>Simple jsp page</title></head>
  <body>Place your content here

  here is index.jsp
    get header info
    <a href="a.jsp">a.jsp</a>
  </body>
</html>
```

防止 a.jsp 页面发生盗链问题, a.jsp 的示例代码如下:

```
<html>
  <head><title>Simple jsp page</title></head>
  <body>Place your content here
  here is a.jsp
    get header info
    <%=request.getHeader("Referer")%>
    <%if(null == request.getHeader("Referer") || request.getHeader("Referer").indexOf("yourdomain.com") < 0){%>
      欢迎! 欢迎! 热烈欢迎!
    <%}else{%>
      合法访问
    <%}%>
  </body>
</html>
```

在上述代码中, “yourdomain.com” 部分可根据实际情况改为站点的域名。上述的 a.jsp 页面, 允许从站点内部访问, 而粘贴 URL 地址到另一个浏览器里访问的方式将被禁止。

疑难点评

上面介绍的防盗链方法, 主要是通过读取请求中的 Referer 信息来判断连接来源。使用该方法解决一般性的盗链问题, 既简洁又有效, 但如果遇到模拟包含 Referer 信息的请求就会失效。为了系统的安全性, 可以与权限验证功能结合使用。