



新浪微博: Javamrbook
腾讯QQ: 4006751066

Java学习路线图



Java经典编程 300例

快速服务：微博、QQ在线服务

自学视频：40集大型多媒体自学视频

海量资源：模块库、案例库、素材库、题库



明日科技 编著

本书提供了内容丰富的配套资源，可以登录www.tup.com.cn，找到本书后，在该页面的“网络资源”超链接处下载。也可以访问本书的新浪微博，根据提示链接下载。



清华大学出版社

Java学习路线图

Java的学习整体解决方案

Java学习路线图，为读者朋友提供了从入门到实际项目开发所需要的各方面必备知识，提供了较为完善的学习整体解决方案，搭起了从学校走向社会的桥梁。各个品种既有前后关联，也可以独立使用。从而避免了像以前那样，学完一本书之后，仍然无所适从，既不会做项目也不知道接下来该学什么，以至于半途而废的困惑。

Java开发入门及项目实战

49.80元

ISBN 978-7-302-27662-3



Java经典编程300例

49.80元

ISBN 978-7-302-27670-8



Java典型模块精解

49.80元

ISBN 978-7-302-27666-1



Java必须知道的300个问题

49.80元

ISBN 978-7-302-27669-2



Java项目案例分析

39.80元

ISBN 978-7-302-27661-6



清华大学出版社数字出版网站

WQBook 文水网
www.wqbook.com

ISBN 978-7-302-27670-8



9 787302 276708 >

定价：49.80元

Java 学习路线图

Java 经典编程 300 例

明日科技 编著

（模块库、案例库、素材库、题库）

（微博、QQ、论坛技术支持）

清华大学出版社

北 京

内 容 简 介

本书以基础知识为框架,介绍了各部分知识所对应的常用开发实例,并进行透彻解析。本书内容包括 Java 语言概述、Eclipse 开发工具、Java 语言基础、流程控制、数组及其常用操作、面向对象入门、面向对象进阶、字符串与包装类、Java 集合类框架、常用数学工具类、错误处理、输入/输出、枚举类型与泛型、Swing 入门、多线程、网络通信和数据库操作。

本书所精选的实例都是一线开发人员在实际项目中所积累的,并进行了技术上的解析,给出了详细的实现过程。读者通过对本书的学习,能够提高开发的能力。

本书提供了大量的源程序、素材,提供了相关的模块库、案例库、素材库、题库等多种形式辅助学习资料,还提供迅速及时的微博、QQ、论坛等技术支持。

本书内容详尽,实例丰富,非常适合作为零基础学习人员的学习用书和大中专院校师生的学习教材,也适合作为相关培训机构的师生和软件开发人员的参考资料。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

Java 经典编程 300 例/明日科技编著. —北京:清华大学出版社,2012.1
(Java 学习路线图)

ISBN 978-7-302-27670-8

I. ①J… II. ①明… III. ①Java 语言-程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字(2011)第 267882 号

责任编辑:赵洛育

版式设计:文森时代

责任校对:王 云

责任印制:何 芊

出版发行:清华大学出版社

地 址:北京清华大学学研大厦 A 座

<http://www.tup.com.cn>

邮 编:100084

社 总 机:010-62770175

邮 购:010-62786544

投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈:010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者:北京季蜂印刷有限公司

装 订 者:北京市密云县京文制本装订厂

经 销:全国新华书店

开 本:185×260 印 张:28.25 字 数:819 千字

版 次:2012 年 1 月第 1 版 印 次:2012 年 1 月第 1 次印刷

印 数:1~5000

定 价:49.80 元

产品编号:045373-01

前言

Preface



学会站在巨人的肩膀上！

软件开发的终极目标是满足用户需求，一个软件往往包含复杂的功能，作为一名程序员，需要在有限的时间内实现它们。对于一名新手，显然并不容易。为何有开发经验的程序员编程效率非常高？答案就是他们做过类似的程序，通过适当的修改以前的代码就可以满足现在的要求。因此如何快速加强编程经验的积累就成了新手的当务之急。显然，单单依靠项目来积累速度是非常慢的。

本书图文并茂、难易并举，汇集了 300 个日常开发中广泛使用的实例，内容涵盖了 Java SE 编程的方方面面，每个实例分成实例说明、实现过程和技术要点 3 部分进行讲解。通过对本书的学习，不但能快速掌握相关知识点，还能逐步提升编程能力。

本书内容

本书以基础知识结构为框架，给出了每部分知识中可能遇到的疑难问题或者是开发技巧。本书共 17 章，主要包括 Java 语言概述、Eclipse 开发工具、Java 语言基础、流程控制、数组及其常用操作、面向对象入门、面向对象进阶、字符串与包装类、Java 集合类框架、常用数学工具类、错误处理、输入/输出、枚举类型与泛型、Swing 入门、多线程、网络通信和数据库操作。

为了更清晰地阐述问题和给出问题的解决方案，本书设置了以下栏目：

- ☑ 实例说明：详细描述本实例的用途，并给出实例的运行效果截图。
- ☑ 实现过程：逐步讲解如何解决本实例的问题，并给出关键代码、注意事项等。
- ☑ 技术要点：对本实例使用的关键技术进行总结，方便日后使用。

本书特色

- ☑ 贴近应用。本书精选的实例都是真正来自开发一线。以实例的形式来进行讲解，使其更容易被读者接受。
- ☑ 横向链接。本书知识框架与《Java 开发入门及项目实战》一书相对应，可以在使用《Java 开发入门及项目实战》一书进行基础学习之后，再用本书丰富并提高技能。
- ☑ 解析透彻。本书对每个问题的相关知识进行细致地讲解，并进行知识拓展，使读者不仅知其然，而且知其所以然。
- ☑ 授人以渔。本书在讲解技术的同时，还注重对读者能力的培养，使读者掌握分析问题与解决问题的能力。



本书配套资源

本书提供了内容丰富的配套资源，包括源程序、素材，以及模块库、案例库、题库、素材库等多项辅助内容，读者朋友可以通过如下方式获取。

第 1 种方式：

(1) 登录 www.tup.com.cn，在网页右上角的搜索文本框中输入本书书名（注意区分大小写和留出空格），或者输入本书关键字，或者输入本书 ISBN 号（注意去掉 ISBN 号间隔线“-”），单击“搜索”按钮。

(2) 找到本书后单击超链接，在该书的网页下侧单击“网络资源”超链接，即可下载。

第 2 种方式：

访问本书的新浪微博 Javamrbook，找到配套资源的链接地址进行下载。

读者人群

本书非常适合以下人员阅读：

- ☒ 从事 Java 语言编程行业的开发人员
- ☒ 有一定语言基础，想进一步提高技能的人员
- ☒ 大中专院校的老师和学生
- ☒ 即将走向工作岗位的大学毕业生
- ☒ 相关培训机构的老师和学员
- ☒ Java 语言编程爱好者

读者服务&本书勘误

读者在使用本书过程中遇到的所有问题，均可通过以下方式联系我们。

1. 新浪微博：Javamrbook。

及时发布读者答疑、本书勘误、配套资料更新等内容。

2. 腾讯 QQ：4006751066。

3. 登录网站：www.mingribook.com，在论坛、勘误发布、读者纠错、技术支持、读者之家等栏目中的相关模块中提问、留言或查看。

本书作者

本书由明日科技组织编写，参加编写的人员有陈丹丹、王国辉、张振坤、李伟、沈博、孙秀梅、曹飞飞、王雪、朱晓、赵永发、李鑫、潘凯华、刘欣、李慧、高春艳、王小科、赵会东、李继业、赛奎春、杨丽、李丽、刘龄龄、王明招、孙茜、陈英、肖鑫等。由于作者水平有限，疏漏和不足之处在所难免，敬请广大读者朋友批评指正。

编 者

目 录

Contents



第 1 章 Java 语言概述	1	实例 026 使用嵌套循环在控制台上 输出九九乘法表	35
实例 001 输出“Hello World”	2	实例 027 使用 while 循环计算 $1+1/2!+1/3!+\cdots+1/20!$	36
实例 002 输出控制台传递的参数	2	实例 028 使用 for 循环输出空心的菱形	38
实例 003 输出由“*”组成的三角形	3	实例 029 终止循环体	39
实例 004 输出符号表情	5	实例 030 循环体的过滤器	41
第 2 章 Eclipse 开发工具	6	第 5 章 数组及其常用操作	43
实例 005 下载并运行 Eclipse 工具	7	实例 031 获取一维数组的最小值	44
实例 006 为 Eclipse 安装汉化包	8	实例 032 将二维数组中的行列互换	45
实例 007 使用 Eclipse 注释代码	10	实例 033 利用数组随机抽取幸运观众	47
实例 008 使用 Eclipse 格式化代码	11	实例 034 用数组设置 JTable 表格的 列名与列宽	49
实例 009 安装 WindowBuilder 插件	12	实例 035 使用按钮控件数组实现 计算器界面	51
实例 010 开发计算器界面	14	实例 036 通过复选框控件数组实现 添加多个复选框控件	52
第 3 章 Java 语言基础	15	实例 037 使用选择排序法对数组排序	53
实例 011 输出错误信息与调试信息	16	实例 038 使用冒泡排序法对数组排序	55
实例 012 从控制台接收输入字符	16	实例 039 使用快速排序法对数组排序	57
实例 013 重定向输出流实现程序日志	17	实例 040 使用直接插入法对数组排序	59
实例 014 自动类型转换与强制类型转换	19	实例 041 使用 sort() 方法对数组排序	61
实例 015 加密可以这样简单(位运算)	20	实例 042 反转数组中元素的顺序	63
实例 016 用三元运算符判断奇数和偶数	21	第 6 章 面向对象入门	65
实例 017 不用乘法运算符实现 2×16	22	实例 043 自定义图书类	66
实例 018 实现两个变量的互换 (不借助第 3 个变量)	23	实例 044 温度单位转换工具	67
第 4 章 流程控制	25	实例 045 成员变量的默认初始值	68
实例 019 判断某一年是否为闰年	26	实例 046 单例模式的应用	69
实例 020 验证登录信息的合法性	27	实例 047 汉诺塔问题求解	70
实例 021 为新员工分配部门	28	实例 048 编写同名的方法	71
实例 022 用 switch 语句根据消费 金额计算折扣	29	实例 049 构造方法的应用	72
实例 023 判断用户输入月份的季节	31	实例 050 统计图书的销售量	73
实例 024 使用 while 循环语句与自增 运算符循环遍历数组	33	实例 051 两只完全相同的宠物	74
实例 025 使用 for 循环输出杨辉三角形	34		



实例 052	重新计算对象的哈希码	76
实例 053	使用字符串输出对象	77
实例 054	Java 对象的假克隆	78
实例 055	Java 对象的浅克隆	80
实例 056	Java 对象的深克隆	82
实例 057	序列化与对象克隆	84
实例 058	深克隆效率的比较	87

第 7 章 面向对象进阶

实例 059	经理与员工的差异	90
实例 060	重写父类中的方法	92
实例 061	计算几何图形的面积	93
实例 062	简单的汽车销售商场	95
实例 063	使用 Comparable 接口自定义排序	96
实例 064	策略模式的简单应用	98
实例 065	适配器模式的简单应用	100
实例 066	普通内部类的简单应用	102
实例 067	局部内部类的简单应用	103
实例 068	匿名内部类的简单应用	104
实例 069	静态内部类的简单应用	105
实例 070	实例化 Class 类的几种方式	107
实例 071	查看类的声明	108
实例 072	查看类的成员	110
实例 073	查看内部类信息	112
实例 074	动态设置类的私有域	113
实例 075	动态调用类中方法	115
实例 076	动态实例化类	116
实例 077	创建长度可变的数组	117
实例 078	利用反射重写 toString() 方法	119

第 8 章 字符串与包装类

实例 079	将数字格式化为货币字符串	122
实例 080	货币金额大写格式	123
实例 081	String 类格式化当前日期	125
实例 082	字符串大小写转换	126
实例 083	字符与 Unicode 码的转换	128
实例 084	判断用户名是否正确	129
实例 085	用户名排序	130
实例 086	判断网页请求与 FTP 请求	132
实例 087	判断文件类型	133
实例 088	判断字符串是否为数字	135
实例 089	验证 IP 地址的有效性	136

实例 090	鉴别非法电话号码	137
实例 091	将字符串转换成整数	139
实例 092	整数进制转换器	140
实例 093	获取字符串中汉字的个数	141
实例 094	批量替换某一类字符串	142
实例 095	查看数字的取值范围	144
实例 096	ASCII 编码查看器	145
实例 097	判断手机号的合法性	146
实例 098	用字符串构建器追加字符	147
实例 099	去掉字符串中的所有空格	148
实例 100	Double 类型的比较	149

第 9 章 Java 集合类框架

范例 101	用动态数组保存学生姓名	152
实例 102	用 List 集合传递学生信息	153
实例 103	Map 集合二级联动	155
实例 104	不重复随机数组排序	157
实例 105	for 循环遍历 ArrayList	159
实例 106	Iterator 遍历 ArrayList	159
实例 107	ListIterator 逆序遍历 ArrayList	160
实例 108	制作电子词典	161
实例 109	制作手机电话簿	162

第 10 章 常用数学工具类

实例 110	角度和弧度的转换	165
实例 111	三角函数的使用	166
实例 112	反三角函数的使用	167
实例 113	双曲函数的使用	168
实例 114	指数与对数运算	169
实例 115	高精度整数运算	170
实例 116	高精度浮点运算	171
实例 117	七星彩号码生成器	173
实例 118	大乐透号码生成器	174

第 11 章 错误处理

实例 119	算数异常	178
实例 120	数组下标越界异常	179
实例 121	空指针异常	180
实例 122	类未发现异常	181
实例 123	非法访问异常	182
实例 124	文件未发现异常	183
实例 125	数据库操作异常	184
实例 126	方法中抛出异常	185
实例 127	方法上抛出异常	186



实例 128	自定义异常类	187	实例 168	解压缩 RAR 压缩包	233
实例 129	捕获单个异常	188	实例 169	为 RAR 压缩包添加注释	234
实例 130	捕获多个异常	189	实例 170	获取压缩包详细文件列表	235
第 12 章 输入/输出	191		实例 171	从 RAR 压缩包中删除文件	237
实例 131	显示指定类型的文件	192	实例 172	在压缩文件中查找字符串	238
实例 132	以树结构显示文件路径	193	实例 173	重命名 RAR 压缩包中文件	239
实例 133	查找替换文本文件内容	194	实例 174	创建自解压 RAR 压缩包	240
实例 134	设置 Windows 系统的文件 属性	195	第 13 章 枚举类型与泛型	242	
实例 135	文件批量重命名	196	实例 175	查看枚举类型的定义	243
实例 136	快速批量移动文件	197	实例 176	枚举类型的基本特性	244
实例 137	删除文件夹中的 tmp 文件	198	实例 177	增加枚举元素的信息	245
实例 138	将图片文件保存到数据库	199	实例 178	选择合适的枚举元素	246
实例 139	从数据库读取图片文件	200	实例 179	高效的枚举元素集合	248
实例 140	窗体动态加载磁盘文件	201	实例 180	高效的枚举元素映射	249
实例 141	删除文件夹中所有文件	202	实例 181	使用枚举接口遍历元素	250
实例 142	创建磁盘索引文件	203	实例 182	使用泛型实现栈结构	251
实例 143	控制台记录器	205	实例 183	自定义泛型化数组类	253
实例 144	防止创建多个字符串对象	206	实例 184	泛型方法与数据查询	254
实例 145	合并多个文本文件	207	实例 185	使用通配符增强泛型	256
实例 146	对大文件实现分割处理	208	实例 186	泛型化的折半查找法	257
实例 147	将分割后的文件重新合并	209	第 14 章 Swing 入门	259	
实例 148	读取属性文件单个属性值	210	实例 187	从上次关闭位置启动窗体	260
实例 149	向属性文件中添加信息	211	实例 188	始终在桌面最顶层显示窗体	261
实例 150	在复制文件时使用进度条	212	实例 189	设置窗体大小	262
实例 151	从 XML 文件中读取数据	213	实例 190	根据桌面大小调整窗体大小	263
实例 152	读取 Jar 文件属性	214	实例 191	自定义最大化、最小化和 关闭按钮	265
实例 153	电子通讯录	215	实例 192	禁止改变窗体的大小	267
实例 154	批量复制指定扩展名文件	217	实例 193	指定窗体标题栏图标	267
实例 155	分类保存文件	218	实例 194	设置闪烁的标题栏	269
实例 156	搜索指定文件夹中的文件	219	实例 195	实现带背景图片的窗体	270
实例 157	实现文件锁定功能	220	实例 196	背景为渐变色的主界面	271
实例 158	简单的投票软件	221	实例 197	随机更换窗体背景	273
实例 159	压缩所有文本文件	222	实例 198	椭圆形窗体界面	275
实例 160	将压缩包解压到指定文件夹	223	实例 199	钻石形窗体	276
实例 161	压缩所有子文件夹	225	实例 200	创建透明窗体	277
实例 162	深层文件夹压缩包的释放	226	实例 201	信息提示对话框	278
实例 163	解决压缩包中文乱码	227	实例 202	设置信息提示对话框的图标	279
实例 164	Apache 实现文件解压缩	228	实例 203	指定打开对话框的文件类型	280
实例 165	把窗体压缩成 ZIP 文件	229	实例 204	为保存对话框设置默认文件名	282
实例 166	解压缩 Java 对象	230	实例 205	支持图片预览的文件选 择对话框	283
实例 167	文件压缩为 RAR 文档	231			



实例 206	颜色选择对话框	285
实例 207	信息输入对话框	286
实例 208	定制信息对话框	287
实例 209	拦截事件的玻璃窗格	289
实例 210	简单的每日提示信息	290
实例 211	震动效果的提示信息	292
实例 212	制作圆形布局管理器	293
实例 213	制作阶梯布局管理器	295
实例 214	密码域控件简单应用	296
实例 215	文本域设置背景图片	297
实例 216	文本区设置背景图片	298
实例 217	简单的字符统计工具	299
实例 218	能预览图片的复选框	300
实例 219	简单的投票计数软件	301
实例 220	单选按钮的简单应用	302
实例 221	能显示图片的组合框	303
实例 222	使用滑块来选择日期	305
实例 223	模仿记事本的菜单栏	308
实例 224	自定义纵向的菜单栏	309
实例 225	复选框与单选按钮菜单项	311
实例 226	包含图片的弹出菜单	312
实例 227	工具栏的实现与应用	314
实例 228	修改列表项显示方式	315
实例 229	列表项与提示信息	316
实例 230	表头与列的高度设置	317
实例 231	调整表格各列的宽度	319
实例 232	设置表格的选择模式	321
实例 233	为表头增添提示信息	323
实例 234	单元格的粗粒度排序	325
实例 235	实现表格的查找功能	326
实例 236	应用网格布局设计计算 器窗体	327

第 15 章 多线程

实例 237	查看线程的运行状态	330
实例 238	查看 JVM 中的线程名	331
实例 239	查看和修改线程优先级	333
实例 240	休眠当前线程	335
实例 241	终止指定线程	336
实例 242	线程的插队运行	337
实例 243	使用方法实现线程同步	339
实例 244	使用特殊域变量实现线程同步	341
实例 245	简单的线程通信	342

实例 246	新建有返回值的线程	344
实例 247	使用线程池优化多线程编程	346
实例 248	哲学家的就餐问题	348

第 16 章 网络通信

实例 249	获得内网的所有 IP 地址	351
实例 250	获取网络资源的大小	352
实例 251	解析网页中的内容	354
实例 252	网络资源的单线程下载	355
实例 253	网络资源的多线程下载	357
实例 254	下载网络资源的断点续传	359
实例 255	建立服务器套接字	362
实例 256	建立客户端套接字	363
实例 257	设置等待连接的超时时间	364
实例 258	获得 Socket 信息	365
实例 259	接收和发送 Socket 信息	367
实例 260	关闭 Socket 缓冲	369
实例 261	使用 Socket 通信	371
实例 262	防止 Socket 传递汉字乱码	375
实例 263	使用 Socket 传递对象	377
实例 264	使用 Socket 传输图片	379
实例 265	使用 Socket 传输音频	381
实例 266	使用 Socket 传输视频	384
实例 267	一个服务器与一个客户端 通信	385
实例 268	一个服务器与多个客户端 通信	387
实例 269	客户端一对多通信	389
实例 270	客户端一对一通信	391
实例 271	基于 Socket 的数据库编程	393
实例 272	使用 Proxy 创建代理服务器	396
实例 273	使用 ProxySelector 选择 代理服务器	397
实例 274	聊天室服务器端	399
实例 275	聊天室客户端	401

第 17 章 数据库操作

实例 276	JDBC 连接 MySQL 数据库	406
实例 277	连接 SQL Server 2005 数据库	407
实例 278	JDBC 连接 Oracle 数据库	408
实例 279	获取 SQL Server 指定数据 库中的数据表信息	409
实例 280	获取 MySQL 指定数据库 中的数据表名称	411



实例 281	查看数据表结构	412	实例 291	添加数据时使用数据验证	431
实例 282	动态维护投票数据库	414	实例 292	插入用户登录日志信息	432
实例 283	SQL Server 数据备份	416	实例 293	生成有规律的编号	433
实例 284	SQL Server 数据恢复	419	实例 294	生成无规律的编号	435
实例 285	MySQL 数据备份	422	实例 295	插入数据时过滤危险字符	436
实例 286	MySQL 数据恢复	424	实例 296	复选框保存到数据库	437
实例 287	动态附加数据库	425	实例 297	把数据复制到另一张表中	438
实例 288	生成 SQL 数据库脚本	426	实例 298	批量插入数据	439
实例 289	表中字段的描述信息	429	实例 299	更新指定记录	440
实例 290	将员工信息添加到数据表	430	实例 300	在删除数据时给出提示信息 ..	442

第1章

Java 语言概述

本章读者可以学到如下实例：

- » 实例 001 输出“Hello World”
- » 实例 002 输出控制台传递的参数
- » 实例 003 输出由“*”组成的三角形
- » 实例 004 输出符号表情



实例 001 输出“Hello World”

(实例位置: 配套资源\SL\01\001)

实例说明

学习 Java 编程首先要下载和配置 JDK。完成下载和配置之后,可以编写一个简单的程序来检验配置的效果。本实例将在 DOS 控制台上输出“Hello World”,其运行效果如图 1.1 所示。

实现过程

(1) 打开文本编辑软件,如 Windows 系统的记事本,并在其中输入如下代码:

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

(2) 保存文件到 D 盘,并将其命名为 Test,扩展名是.java。对于记事本,可以按 Ctrl+S 键保存文件,并将文件名写成“Test.java”。

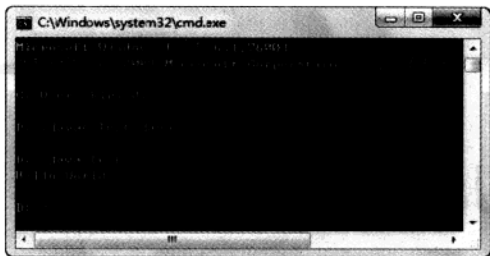


图 1.1 输出“Hello World”

脚下留神:

记事本默认会对文件增加.txt 扩展名,因此“Test.java”中的双引号并不能省略。

(3) 打开 DOS 控制台并切换路径到 D 盘,输入“javac Test.java”命令编译源代码,输入“java Test”命令运行 class 文件。

脚下留神:

javac 和 java 命令后面都需要一个空格。

技术要点

本实例主要使用了两个 Java 命令,即 javac 和 java。其中, javac 命令用于编译源文件,后面的文件要带扩展名.java,此时将生成一个 class 文件; java 命令用于运行 class 文件,并在 DOS 控制台上显示运行的结果。

实例 002 输出控制台传递的参数

(实例位置: 配套资源\SL\01\002)

实例说明

在使用 java 命令运行程序时,可以同时传递多个参数。本实例将输出用户传递的参数,其



运行效果如图 1.2 所示。

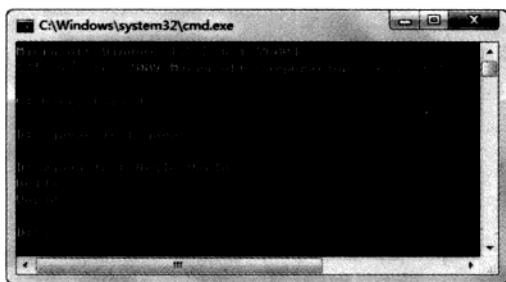


图 1.2 输出用户传递的参数



Note

实现过程

(1) 打开文本编辑软件，如 Windows 系统的记事本，并在其中输入如下代码：

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

(2) 保存文件到 D 盘，并将其命名为 Test，扩展名是.java。对于记事本，可以按 Ctrl+S 键保存文件，并将文件名写成“Test.java”。

脚下留神：

记事本默认会对文件增加.txt 扩展名，因此“Test.java”中的双引号并不能省略。

(3) 打开 DOS 控制台并切换路径到 D 盘，输入“javac Test.java”命令编译源代码，输入“java Test Hello World”命令运行 class 文件。

脚下留神：

javac 和 java 命令后面都需要一个空格。

技术要点

使用 java 命令时，如果传递多个参数，它们之间需要使用空格进行分隔。传递的参数保存在一个 String 类型的数组中并传递给 main() 方法。在 main() 方法中，可以使用其方法参数调用传递的值。

实例 003 输出由“*”组成的三角形

(实例位置：配套资源\SL\01\003)

实例说明

在学习了基本的输出语句后，就可以使用它来输出一些简单的图形。本实例将输出一个由



“*”组成的三角形，其运行效果如图 1.3 所示。

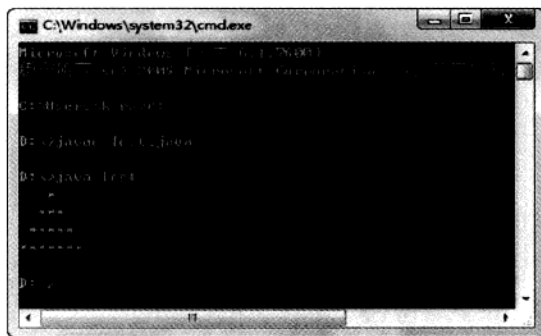


图 1.3 输出三角形

实现过程

(1) 打开文本编辑软件，如 Windows 系统的记事本，并在其中输入如下代码：

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

多学两招：

也可以使用循环语句来输出三角形，这样编写的代码更加简洁。

(2) 保存文件到 D 盘，并将其命名为 Test，扩展名是.java。对于记事本，可以按 Ctrl+S 键保存文件，并将文件名写成“Test.java”。

脚下留神：

记事本默认会对文件增加.txt 扩展名，因此“Test.java”中的双引号并不能省略。

(3) 打开 DOS 控制台并切换路径到 D 盘，输入“javac Test.java”命令编译源代码，输入“java Test”命令运行 class 文件。

脚下留神：

javac 和 java 命令后面都需要一个空格。

技术要点

使用 javac 命令在编译源代码时，需要指明代码的类型，即需包含.java 扩展名。使用 java 命令在运行 class 文件时，不需要指明扩展名，即不能包括 class。



实例 004 输出符号表情

(实例位置: 配套资源\SL\01\004)

实例说明

除了可以输出字符串和简单的几何图形外, 还可以通过字符串的适当组合, 输出符号表情。本实例将在控制台上输出一个猪头, 其运行效果如图 1.4 所示。

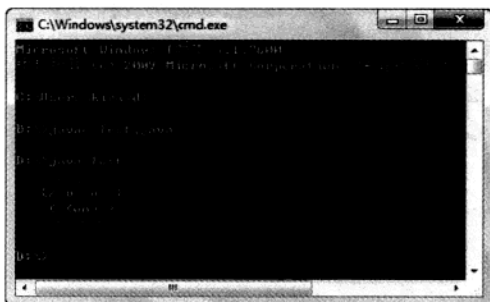


图 1.4 输出猪头

实现过程

(1) 打开文本编辑软件, 如 Windows 系统的记事本, 并在其中输入如下代码:

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println("OoOoOoOo");  
        System.out.println("{/o o/}");  
        System.out.println("((oo))");  
        System.out.println("_");  
    }  
}
```

(2) 保存文件到 D 盘, 并将其命名为 Test, 扩展名是 .java。对于记事本, 可以按 Ctrl+S 键保存文件, 并将文件名写成 “Test.java”。

脚下留神:

记事本默认会对文件增加 .txt 扩展名, 因此 “Test.java” 中的双引号并不能省略。

(3) 打开 DOS 控制台并切换路径到 D 盘, 输入 “javac Test.java” 命令编译源代码, 输入 “java Test” 命令运行 class 文件。

脚下留神:

javac 和 java 命令后面都需要一个空格。

技术要点

Java 中使用 System.out.println(); 语句完成在控制台上的输出, 该语句在输出完毕后会自动换行。类似地还有 System.out.print(); 语句, 它在输出完毕后会不会换行。

第 2 章

Eclipse 开发工具

本章读者可以学到如下实例：

- » 实例 005 下载并运行 Eclipse 工具
- » 实例 006 为 Eclipse 安装汉化包
- » 实例 007 使用 Eclipse 注释代码
- » 实例 008 使用 Eclipse 格式化代码
- » 实例 009 安装 WindowBuilder 插件
- » 实例 010 开发计算器界面



实例 005 下载并运行 Eclipse 工具

实例说明

在开发 Java 程序时,首先要编写源代码。虽然可以使用简单的文本编辑器完成该工作,但是效率非常低。专业的开发人员会选择一款适合自己风格的集成开发环境。本实例讲述如何下载和运行 Eclipse 工具,其运行效果如图 2.1 所示。

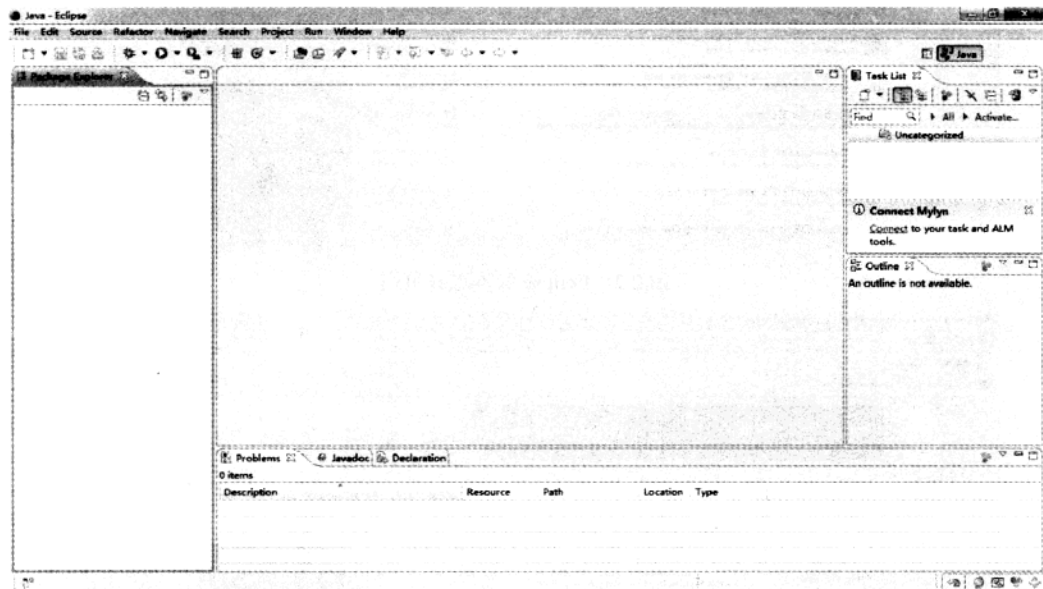


图 2.1 Eclipse 开发环境

实现过程

(1) 打开浏览器,在地址栏中输入“<http://www.eclipse.org/downloads/>”,按 Enter 键将打开 Eclipse 的主页。单击页面中的 Eclipse Downloads 超链接,如图 2.2 所示。

(2) 选择 Eclipse IDE for Java Developers 中的 Windows 32 Bit 下载。如果读者使用的是 64 位计算机,可以选择 Windows 64 Bit 下载。然后跳转到如图 2.3 所示的页面。

(3) 单击[China] Actuate Shanghai (http)超链接开始下载。保存压缩包到磁盘,并解压缩。运行其中的 eclipse.exe 文件,得到的效果如图 2.1 所示。

技术要点

在图 2.2 中,提供了多个 Eclipse 版本,如适合进行 Java EE 开发的版本、适合进行 PHP 开发的版本等。读者可以根据自己的需求选择合适的版本。



Note



图 2.2 Eclipse 版本选择页面

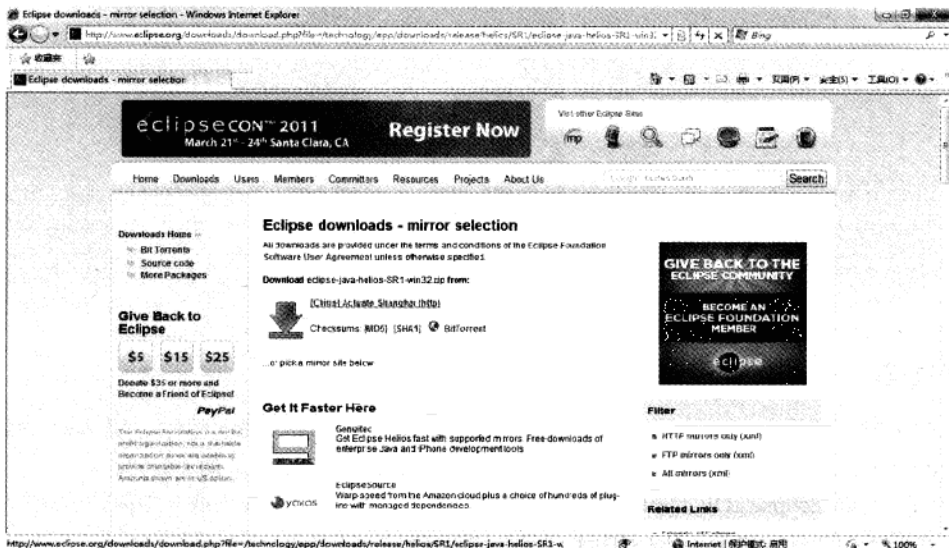


图 2.3 选择 Eclipse 下载链接

实例 006 为 Eclipse 安装汉化包

实例说明

为了方便中国用户的使用, Eclipse 提供了汉化包。本实例将演示如何下载并安装汉化包, 汉化后的 Eclipse 如图 2.4 所示。



Note

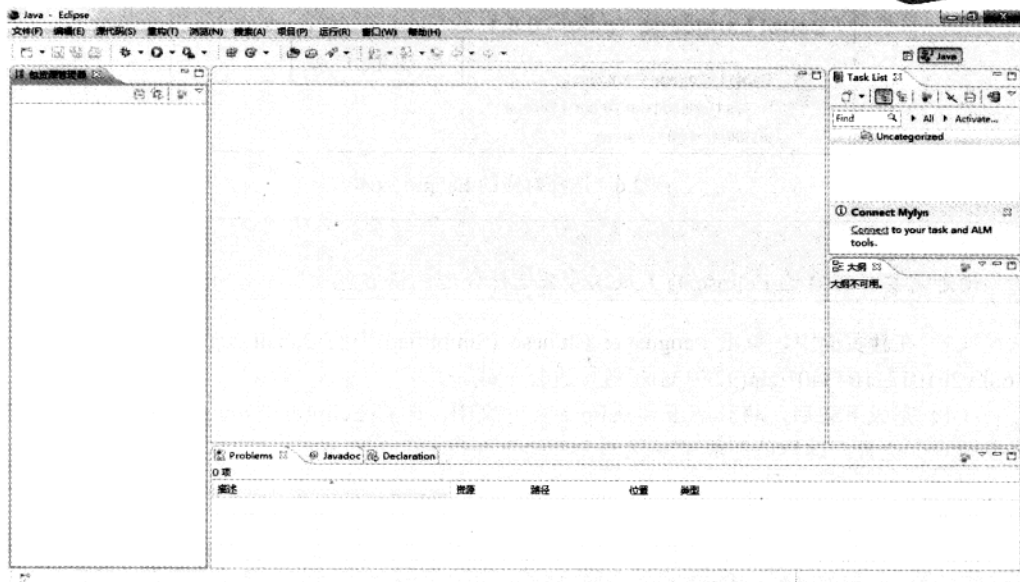


图 2.4 汉化后的 Java 开发环境界面

实现过程

(1) 在浏览器的地址栏中输入“<http://www.eclipse.org/babel/>”，按 Enter 键将打开 Babel 项目的网页，如图 2.5 所示。单击 Downloads 超链接。

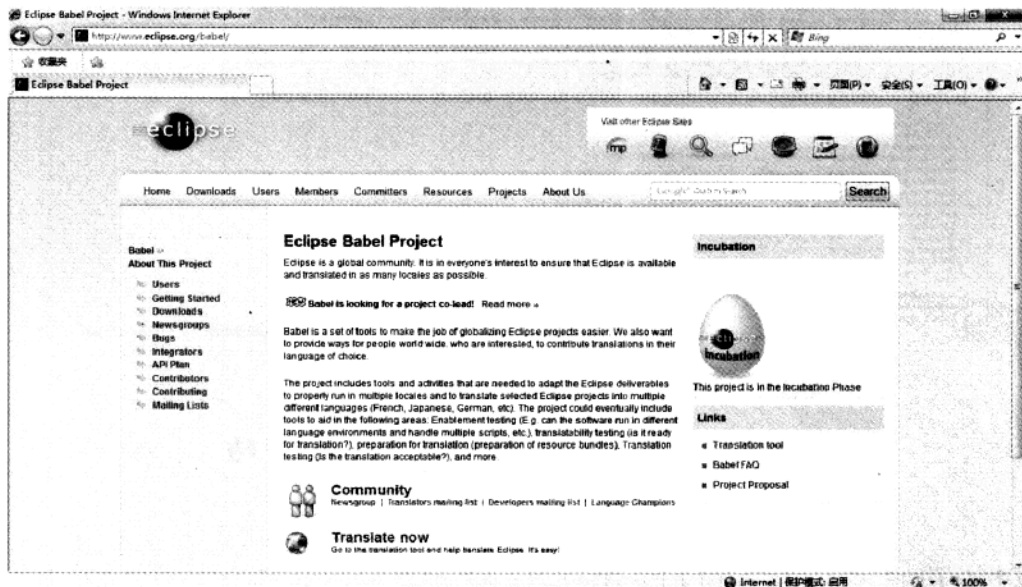


图 2.5 Eclipse Babel 项目主页

(2) 在打开的 Eclipse Babel 下载页面中单击 Helios 超链接，如图 2.6 所示。



Note

Babel Language Packs - 0.8.1



Babel Language Pack Zips

Helios | Galileo | Ganymede | Europa

Installation instructions

单击该超链接

图 2.6 选择对应的 Eclipse 版本

指点迷津:

读者需要根据自己 Eclipse 的主版本号来选择合适的语言包。

(3) 在新页面中, 单击 Language: Chinese (Simplified) 中的 BabelLanguagePack-eclipse-zh_3.6.0.v20101211043401.zip(92.91%)超链接进行下载。

(4) 完成下载后, 将其解压到 eclipse 文件夹中, 即将 Eclipse 和 Eclipse 汉化的压缩包解压缩在同一个位置。这样就完成了汉化。

技术要点

Eclipse Babel 项目中提供了多种语言包, 如图 2.7 所示。读者可以根据自己的需求进行选择。

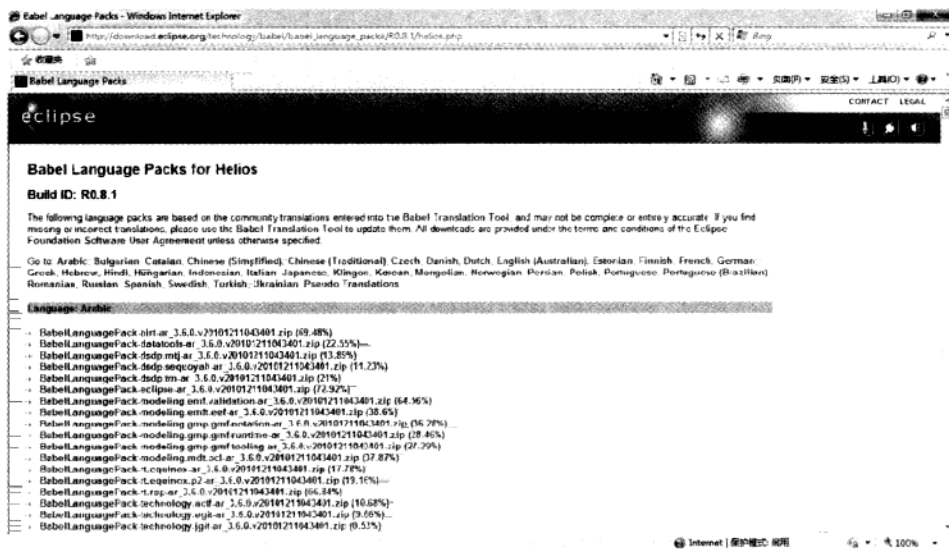


图 2.7 语言选择界面

实例 007 使用 Eclipse 注释代码

(实例位置: 配套资源\SL\02\007)

实例说明

在编写代码的过程中, 需要经过反复的调试以达到最佳的效果。通过注释代码可以直观地看到这段代码所起的作用。为了方便用户使用, Eclipse 提供了很多快捷键。本实例将演示注释



代码的快捷键的使用，其运行效果如图 2.8 所示。

```

1 package com.mingrisoft;
2
3 public class Test {
4     public static void main(String[] args) {
5         //      System.out.println("《Java编程词典》真好用!");
6     }
7 }

```

图 2.8 注释代码的效果



Note

实现过程

- (1) 在 Eclipse 中创建项目 007，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建类文件，名称为 Test。在类中输入如下代码：

```

public class Test {
    public static void main(String[] args) {
        System.out.println("《Java 编程词典》真好用!");
    }
}

```

- (3) 将光标停留在“《Java 编程词典》真好用!”一行上，按 Ctrl+/键，可以看到该行代码已经被注释了，效果如图 2.8 所示。

技术要点

Java 中支持 3 种注释格式，即单行注释、多行注释和文档注释。使用 Eclipse 时，可以同时选择多行代码，然后使用 Ctrl+/快捷键来同时注释。

实例 008 使用 Eclipse 格式化代码

(实例位置：配套资源\SL\02\008)

实例说明

在编写代码的过程中可能要反复地修改以达到最佳效果，此时可能造成代码格式混乱。Eclipse 提供了格式化代码的快捷键，可以帮助程序员完成单调的格式化任务。本实例将演示该快捷键的使用，格式化之前的效果如图 2.9 所示，格式化之后的效果如图 2.10 所示。

```

1 package com.mingrisoft;
2
3 public class Test {
4     public
5     static
6     void
7     main(String[]
8     args) {
9         System.out.println("《Java编程词典》真好用!");
10    }
11 }

```

图 2.9 格式化代码前的效果

```

1 package com.mingrisoft;
2
3 public class Test {
4     public static void main(String[] args) {
5         System.out.println("《Java编程词典》真好用!");
6     }
7 }

```

图 2.10 格式化代码后的效果

实现过程

- (1) 在 Eclipse 中创建项目 008，并在该项目中创建 com.mingrisoft 包。



(2) 在 com.mingrisoft 包中创建类文件, 名称为 Test。在类中输入如下代码:

```
public class Test {  
    public  
    static  
    void  
    main(String[]  
        args) {  
        System.out.println("《Java 编程词典》真好用!");  
    }  
}
```



Note

(3) 将光标放在编辑器窗体中, 按 Shift+Ctrl+F 键, 可以看到代码已经被格式化了。

技术要点

除了上面介绍的快捷键外, Eclipse 还提供了大量的快捷键。在菜单栏中选择“窗口”/“首选项”/“常规”/“键”命令, 可以看到各种预定义的快捷键的功能。读者可以根据自己的习惯进行修改。

实例 009 安装 WindowBuilder 插件

实例说明

默认的 Eclipse 工具并没有提供图形化的 Swing 程序开发工具。本实例将演示如何安装 WindowBuilder 插件, 它是 Google 公司提供的图形化 Swing 程序开发插件。使用它来开发 Swing 程序的界面如图 2.11 所示。

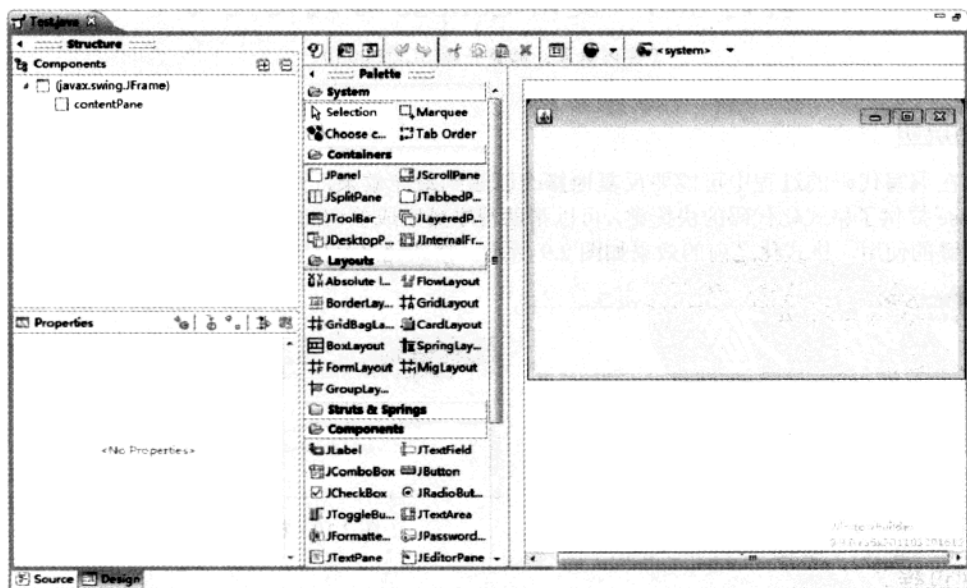


图 2.11 使用 WindowBuilder 插件开发 Swing 程序



实现过程

(1) 在浏览器的地址栏中输入网址“<http://code.google.com/intl/zh-CN/javadevtools/download-wbpro.html>”，按 Enter 键进入 WindowBuilder 插件的下载页面，根据 Eclipse 的版本，选择对应的更新网址。

(2) 打开 Eclipse，选择“帮助”/Install New Software 命令，如图 2.12 所示。

(3) 在打开的增加更新仓库对话框的 Name 文本框中输入“WindowBuilder”，在 Location 文本框中输入在第(1)步中输入的网址，最后单击“确定”按钮，如图 2.13 所示。

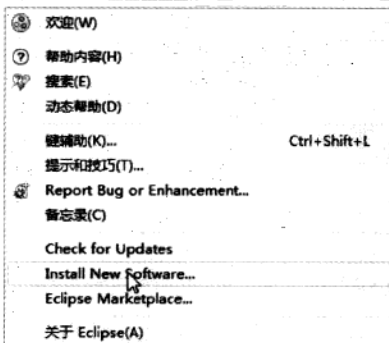


图 2.12 “帮助”菜单项

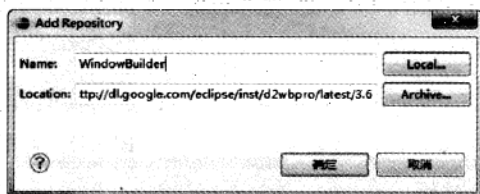


图 2.13 增加更新仓库对话框

(4) 开始进行安装，中间需要重启一次。在安装完毕后，可以在菜单栏中选择“文件”/“新建”/“其他”命令，会显示如图 2.14 所示的对话框。其中包括了 WindowBuilder 文件夹。

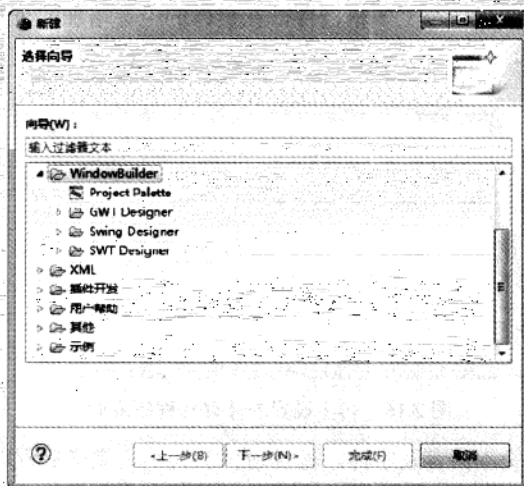


图 2.14 “新建”对话框

技术要点

对于其他的 Eclipse 插件，也可以通过类似的方式完成安装。例如，安装开发 Perl 语言的插件，可以在 www.epic-ide.org 上找到相关的说明。



实例 010 开发计算器界面

(实例位置: 配套资源\SL\02\010)



Note

实例说明

计算器是各种操作系统提供的经典程序之一, 在安装完 WindowBuilder 插件后, 将演示如何使用它来开发计算器界面。实例的运行效果如图 2.15 所示。



图 2.15 计算器程序界面

实现过程

- (1) 在 Eclipse 中创建项目 010, 并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建 JFrame 类型的文件, 名称为 Calculator, 并在屏幕下方选择 Design 选项卡切换到设计界面。
- (3) 在窗体的顶部增加一个文本框控件, 在窗体的中央增加一个面板, 然后将该面板设置成 4 行 4 列的网格布局, 并增加 16 个按钮, 效果如图 2.16 所示。

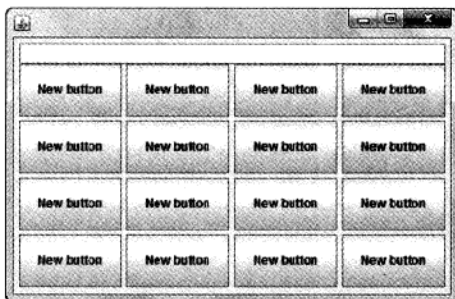


图 2.16 初步设置的计算器程序界面

- (4) 修改按钮上的文本内容及字体, 调整窗体的大小, 完毕后程序的运行效果如图 2.15 所示。

技术要点

Swing 中控件上的文本非常灵活, 不仅能够设置字体, 还能够设置颜色。此外, 还可以在按钮控件中使用 HTML 代码实现复杂的样式。

第 3 章

Java 语言基础

本章读者可以学到如下实例：

- ▶▶ 实例 011 输出错误信息与调试信息
- ▶▶ 实例 012 从控制台接收输入字符
- ▶▶ 实例 013 重定向输出流实现程序日志
- ▶▶ 实例 014 自动类型转换与强制类型转换
- ▶▶ 实例 015 加密可以这样简单（位运算）
- ▶▶ 实例 016 用三元运算符判断奇数和偶数
- ▶▶ 实例 017 不用乘法运算符实现 2×16
- ▶▶ 实例 018 实现两个变量的互换（不借助第 3 个变量）



实例 011 输出错误信息与调试信息

(实例位置: 配套资源\SL\03\011)

实例说明

程序开发中对于业务代码的部分功能需要配合调试信息以确定代码执行流程和数据的正确性,当程序出现严重问题时还要输出警告信息,这样可以在调试中完成程序开发,本实例将介绍如何输出调试信息与错误提示信息。实例的运行效果如图 3.1 所示。

实现过程

(1) 在 Eclipse 中新建项目 011,在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 PrintErrorAndDebug 类,并完成该类的 main()主方法,在该方法中分别输出调试信息与错误信息。关键代码如下:

```
public class PrintErrorAndDebug {
    public static void main(String[] args) {
        System.out.println("main()方法开始运行了。");
        //输出错误信息
        System.err.println("在运行期间手动输出一个错误信息:");
        System.err.println("\t 该软件没有买保险, 请注意安全");
        System.out.println("PrintErrorAndDebug.main()");
        System.out.println("main()方法运行结束。");
    }
}
```

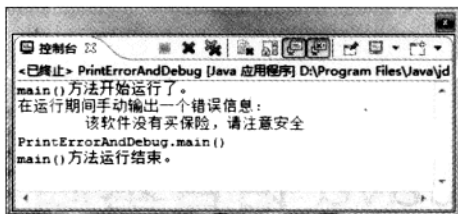


图 3.1 输出错误信息与调试信息

技术要点

本实例使用 System 类的 out 和 err 两个成员变量来完成调试信息与错误信息的输出,它们两个都是 System 的类变量,也就是说使用 static 关键字修饰的。out 是标准调试信息的输出流, err 是标准错误信息的输出流。实例中调用了两个输出流通用的 println()方法来输出一行数据。该方法的声明如下:

```
public void println(String x)
```

参数说明

x: 被输出到控制台的字符串。

实例 012 从控制台接收输入字符

(实例位置: 配套资源\SL\03\012)

实例说明

System 类除了包含 out 和 err 两个输出流之外,还有 in 输入流的实例对象作为类成员,它



可以接收用户的输入。本实例通过该输入流实现从控制台接收用户输入文本，并提示该文本的长度信息。实例的运行效果如图 3.2 所示。

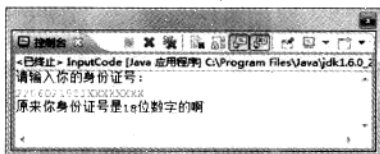


图 3.2 从控制台接收输入字符

实现过程

(1) 在 Eclipse 中新建项目 012，在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 InputCode 类，在该类的主方法中创建 Scanner 扫描器来封装 System 类的 in 输入流，然后提示用户输入身份证号码，并输出用户身份证号码的位数。关键代码如下：

```
import java.util.Scanner;
public class InputCode {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);           //创建输入流扫描器
        System.out.println("请输入你的身份证号: ");         //提示用户输入
        String line = scanner.nextLine();                     //获取用户输入的一行文本
        //打印对输入文本的描述
        System.out.println("原来你身份证号是" + line.length() + "位数字的啊");
    }
}
```

技术要点

本实例的关键技术就是用到了 System 类的输入流也就是类变量 in，它可以接收用户的输入信息，并且是标准的输入流实例对象。另外，Scanner 类是 Java 的扫描器类，它可以从输入流中读取指定类型的数据或字符串。本实例使用 Scanner 类封装了输入流对象，并使用 nextLine() 方法从输入流中获取用户输入的整行文本字符串。该方法的声明如下：

```
public String nextLine()
```

返回值：从扫描器封装的输入流中获取一行文本字符串作为方法的返回值。

实例 013 重定向输出流实现程序日志

(实例位置：配套资源\SL\03\013)

实例说明

System 类中的 out 成员变量是 Java 的标准输出流，程序常用它来输出调试信息。out 成员变量被定义为 final 类型的，无法直接重新复制，但是可以通过 setOut() 方法来设置新的输出流。本实例利用该方法实现了输出流的重定向，把它指向一个文件输出流，从而实现了日志功能。程序运行后控制台提示运行结束信息，如图 3.3 所示，但是在运行过程中的步骤都保存到了日志文件中，如图 3.4 所示。

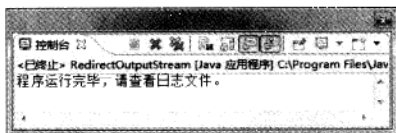


图 3.3 在控制台提示程序运行完毕

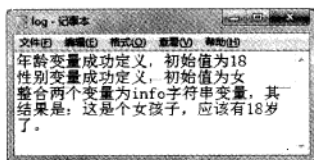


图 3.4 重定向输出流实现程序日志的效果

实现过程

(1) 在 Eclipse 中新建项目 013，在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 RedirectOutputStream 类，编写该类的 main() 主方法，在该方法中保存 System 类的 out 成员变量为临时变量，然后创建一个新的文件输出流，并把这个输出流设置为 System 类新的输出流。在程序关键位置输出调试信息，这些调试信息将通过新的输出流保存到日志文件中。最后恢复原有输出流并输出程序运行结束信息。关键代码如下：

```
import java.io.FileNotFoundException;
import java.io.PrintStream;
public class RedirectOutputStream {
    public static void main(String[] args) {
        try {
            PrintStream out = System.out;           //保存原输出流
            PrintStream ps = new PrintStream("./log.txt"); //创建文件输出流
            System.setOut(ps);                       //设置使用新的输出流
            int age = 18;                             //定义整型变量
            System.out.println("年龄变量成功定义，初始值为 18");
            String sex = "女";                         //定义字符串变量
            System.out.println("性别变量成功定义，初始值为女");
            //整合两个变量
            String info = "这是个" + sex + "孩子，应该有" + age + "岁了。";
            System.out.println("整合两个变量为 info 字符串变量，其结果是：" + info);
            System.setOut(out);                       //恢复原有输出流
            System.out.println("程序运行完毕，请查看日志文件。");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

技术要点

本实例的关键技术是调用了 System 类的 setOut() 方法改变了输出流，System 类的 out、err 和 in 成员变量是 final 类型的，不能直接赋值，要通过相应的方法来改变流。下面分别介绍改变这 3 个成员变量的方法。

1. setOut() 方法

该方法用于重新分配 System 类的标准输出流。方法的声明如下：

```
public static void setOut(PrintStream out)
```

参数说明

out: 新的 PrintStream 输出流对象。



2. setErr()方法

该方法将重新分配 System 类的标准错误输出流。方法的声明如下:

```
public static void setErr(PrintStream err)
```

参数说明

err: 新的 PrintStream 输出流对象。

3. setIn()方法

该方法将重新设置 System 类的 in 成员变量, 即标准输入流。方法的声明如下:

```
public static void setIn(InputStream in)
```

参数说明

in: 新的 InputStream 输入流对象。

实例 014 自动类型转换与强制类型转换

(实例位置: 配套资源\SL\03\014)

实例说明

Java 基本数据类型之间存在自动类型转换与强制类型转换两种转换方法。本实例将演示这两种类型转换的用法, 实例的运行效果如图 3.5 所示。注意 long 类型向低类型 short 转换时发生的数据丢失。

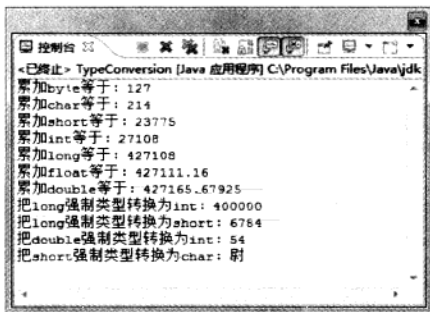


图 3.5 自动类型转换与强制类型转换的输出结果

实现过程

(1) 在 Eclipse 中新建项目 014, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 TypeConversion 类, 在该类的主方法中创建各种基本类型的变量, 在输出语句中分别输出所有变量累加值。注意每次累加值的数据类型, 所有整数运算都被自动转换为 int 类型再进行运算, 所有浮点数值都被自动转换为 double 类型进行运算。最后把高类型数据向低类型数据进行强制类型转换, 并注意运算结果是否丢失数据。关键代码如下:

```
public class TypeConversion {
    public static void main(String[] args) {
        byte b = 127;
```



Note

```
char c = 'W';
short s = 23561;
int i = 3333;
long l = 400000L;
float f = 3.14159F;
double d = 54.523;
//低类型向高类型自动转换
System.out.println("累加 byte 等于: " + b);
System.out.println("累加 char 等于: " + (b + c));
System.out.println("累加 short 等于: " + (b + c + s));
System.out.println("累加 int 等于: " + (b + c + s + i));
System.out.println("累加 long 等于: " + (b + c + s + i + l));
System.out.println("累加 float 等于: " + (b + c + s + i + l + f));
System.out.println("累加 double 等于: " + (b + c + s + i + l + f + d));
//高类型到低类型的强制转换
System.out.println("把 long 强制类型转换为 int: " + (int) l);
//高类型到低类型转换会丢失数据
System.out.println("把 long 强制类型转换为 short: " + (short) l);
//实数到整数转换将舍弃小数部分
System.out.println("把 double 强制类型转换为 int: " + (int) d);
//整数到字符类型的转换将获取对应编码的字符
System.out.println("把 short 强制类型转换为 char: " + (char) s);
}
```

技术要点

本实例的关键技术是强制类型转换。强制类型转换的实现方法是使用一对小括号，将其他类型转换为指定的类型，语法如下：

转换后的类型 变量 = (转换后的类型)被转换的变量

例如：

```
long value = 100L;
byte btValue = (byte)value;
```

指点迷津：

上面代码将 long 型变量的值强制转换为 byte 类型。

实例 015 加密可以这样简单（位运算）

（实例位置：配套资源\SL\03\015）

实例说明

本实例通过位运算的异或运算符“^”把字符串与一个指定的值进行异或运算，从而改变字符串中每个字符的值，这样就可以得到一个加密后的字符串，如图 3.6 所示。当把加密后的字符串作为程序输入内容，异或运算会把加密后的字符串还原为原有字符串的值，如图 3.7 所示。

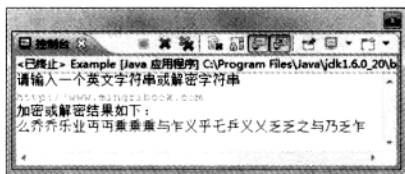


图 3.6 使用异或加密字符串

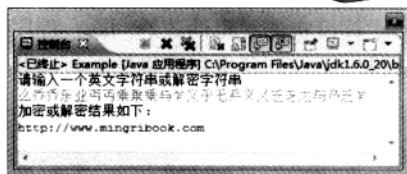


图 3.7 使用异或解密字符串



Note

实现过程

(1) 在 Eclipse 中新建项目 015，在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 Example 类，在该类的主方法中创建 System 类的标准输入流的扫描器对象，提示用户输入一个英文的字符串或者要解密的字符串，然后通过扫描器获取用户输入的字符串，经过加密或解密后，把字符串通过错误流输出到控制台。关键代码如下：

```
import java.util.Scanner;
public class Example {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("请输入一个英文字符串或解密字符串");
        String password = scan.nextLine();           //获取用户输入
        char[] array = password.toCharArray();       //获取字符数组
        for (int i = 0; i < array.length; i++) {     //遍历字符数组
            array[i] = (char) (array[i] ^ 20000);    //对每个数组元素进行异或运算
        }
        System.out.println("加密或解密结果如下: ");
        System.err.println(new String(array));       //输出密钥
    }
}
```

指点迷津：

程序最后使用的标准错误输出流不是用于输出错误信息，而是利用了其在 Eclipse 控制台以红色显示的特性来突出显示。

技术要点

本实例的关键技术是异或运算。如果某个字符（或数值） x 与一个数值 m 进行异或运算得到 y ，则再用 y 与 m 进行异或运算就可以还原为 x ，因此应用这个原理可以实现加密和解密功能。

实例 016 用三元运算符判断奇数和偶数

（实例位置：配套资源\SL\03\016）

实例说明

三元运算符是 `if...else` 条件语句的简写格式，它可以完成简单的条件判断。本实例利用这



个三元运算符实现了奇偶数的判断，程序要求用户输入一个整数，然后程序判断是奇数还是偶数，并输出到控制台中。实例的运行效果如图 3.8 所示。



Note

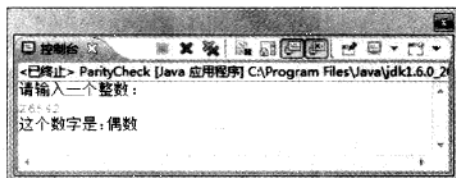


图 3.8 判断奇数和偶数的结果

实现过程

(1) 在 Eclipse 中新建项目 016，在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 ParityCheck 类，在该类的主方法中创建标准输入流的扫描器对象，提示用户输入一个整数，并通过扫描器的方法来接收一个整数，通过三元运算符判断该数字与 2 的余数，如果余数为 0 说明其是偶数，否则是奇数。关键代码如下：

```
import java.util.Scanner;
public class ParityCheck {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);           //创建输入流扫描器
        System.out.println("请输入一个整数：");
        long number = scan.nextLong();                    //获取用户输入的整数
        String check = (number % 2 == 0) ? "这个数字是：偶数" : "这个数字是：奇数";
        System.out.println(check);
    }
}
```

技术要点

本实例的关键技术就是以三元运算符实现简单的条件判断。其语法格式如下：

条件运算? 运算结果 1; 运算结果 2;

如果条件运算结果为 true，返回值就是运算结果 1，否则返回运算结果 2。

另外，本实例使用了扫描器的 nextLong() 方法直接获取了整型数据，避免了类型转换等业务代码。该方法的声明如下：

```
public long nextLong()
```

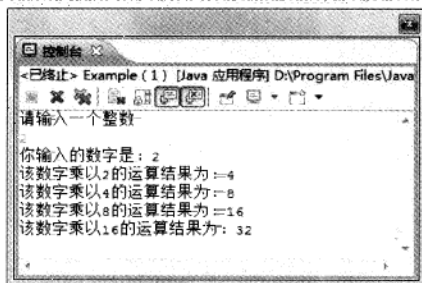
返回值：该方法返回一个 long 类型的数值，这个数是从扫描器封装的输入流中获取的。

实例 017 不用乘法运算符实现 2×16

(实例位置：配套资源\SL\03\017)

实例说明

程序开发中常用的乘法运算是通过“*”运算符或者 BigDecimal 类的 multiply() 方法实现的。但是本实例将介绍在这两种方法之外如何实现乘法，而且实现的运算效率非常高。实例的运行效果如图 3.9 所示。

图 3.9 不用乘法运算实现 2×16

实现过程

- (1) 在 Eclipse 中新建项目 017, 在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建 Example 类, 在该类的主方法中接收用户输入的一个整数, 然后对该整数执行位运算中的左移操作, 并输出运算结果。关键代码如下:

```
import java.util.Scanner;
public class Example {
    public static void main(String[] args) {
        Scanner scan=new Scanner(System.in);           //创建扫描器
        System.out.println("请输入一个整数");
        long number = scan.nextLong();                  //获取输入的整数
        System.out.println("你输入的数字是: "+number);
        System.out.println("该数字乘以 2 的运算结果为: "+(number<<1));
        System.out.println("该数字乘以 4 的运算结果为: "+(number<<2));
        System.out.println("该数字乘以 8 的运算结果为: "+(number<<3));
        System.out.println("该数字乘以 16 的运算结果为: "+(number<<4));
    }
}
```

技术要点

本实例的关键技术就是左移运算。如果一个整数左移运算 n 位, 就相当于这个整数乘以 2 的 n 次方。

例如, $2 \ll 4$ (即 2 左移 4 位) 相当于 2^4 , 也就是 16。

实例 018 实现两个变量的互换 (不借助第 3 个变量)

(实例位置: 配套资源\SL\03\018)

实例说明

变量的互换常见于数组排序算法中, 当判断两个数组元素需要交互时, 将创建一个临时变量来共同完成互换, 临时变量的创建增加了系统资源的消耗。如果需要交换的是两个整数类型的变量, 那么可以使用更高效的方法。本实例演示了如何不借助临时变量 (第 3 个变量) 实现两个整数类型变量的高效互换。实例的运行效果如图 3.10 所示。



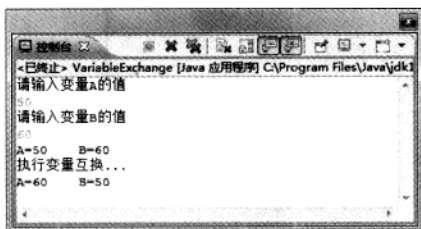


图 3.10 不借助第 3 个变量实现两个变量互换

实现过程

(1) 在 Eclipse 中新建项目 018，在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 VariableExchange 类，在该类的主方法中创建扫描器对象接收用户输入两个变量值，然后通过位运算中的异或运算符“^”实现两个变量的互换。代码如下：

```
import java.util.Scanner;
public class VariableExchange {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);           //创建扫描器
        System.out.println("请输入变量 A 的值");
        long A = scan.nextLong();                         //接收第 1 个变量值
        System.out.println("请输入变量 B 的值");
        long B = scan.nextLong();                         //接收第 2 个变量值
        System.out.println("A=" + A + "\tB=" + B);
        System.out.println("执行变量互换...");
        A = A ^ B;                                         //执行变量互换
        B = B ^ A;                                         //执行变量互换
        A = A ^ B;                                         //执行变量互换
        System.out.println("A=" + A + "\tB=" + B);
    }
}
```

技术要点

本实例的关键技术是异或运算，可以参考实例 015。

第 4 章

流程控制

本章读者可以学到如下实例：

- » 实例 019 判断某一年是否为闰年
- » 实例 020 验证登录信息的合法性
- » 实例 021 为新员工分配部门
- » 实例 022 用 switch 语句根据消费金额计算折扣
- » 实例 023 判断用户输入月份的季节
- » 实例 024 使用 while 循环语句与自增运算符循环遍历数组
- » 实例 025 使用 for 循环输出杨辉三角形
- » 实例 026 使用嵌套循环在控制台上输出九九乘法表
- » 实例 027 使用 while 循环计算 $1+1/2!+1/3!+\cdots+1/20!$
- » 实例 028 使用 for 循环输出空心的菱形
- » 实例 029 终止循环体
- » 实例 030 循环体的过滤器



实例 019 判断某一年是否为闰年

(实例位置: 配套资源\SL\04\019)



Note

实例说明

地球绕太阳一圈称之为一年, 所用时间是 365 天 5 小时 48 分 46 秒, 取 365 天为一年, 4 年将多出 23 小时 15 分 6 秒, 将近一天, 所以 4 年设一个闰日 (2 月 29 日), 这年称为闰年。本实例将要求用户输入年份, 并判断输入的年份是否为闰年。实例的运行效果如图 4.1 所示。

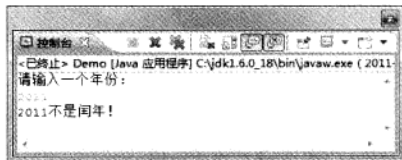


图 4.1 判断某一年是否为闰年

实现过程

(1) 在 Eclipse 中创建项目 019, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Demo。在该类的主方法中接收用户输入的一个整数年份, 然后通过闰年计算公式, 判断这个年份是否为闰年, 并在控制台输出判断结果。关键代码如下:

```
package com.mingrisoft;
import java.util.Scanner;

public class Demo {
    /**
     * @param args
     */
    public static void main(String[] args) { //主方法
        Scanner scan = new Scanner(System.in);
        System.out.println("请输入一个年份: "); //向控制台输出一个提示信息
        long year;
        try {
            year = scan.nextLong();
            if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) { //是闰年
                System.out.print(year + "是闰年!");
            } else { //不是闰年
                System.out.print(year + "不是闰年!");
            }
        } catch (Exception e) {
            System.out.println("您输入的不是有效的年份!");
        }
    }
}
```

指点迷津:

java.util 包的 Scanner 类是一个用于扫描输入文本的简单文本扫描器, 可以用这个类从控制台写入数据。该类的 nextLong() 方法可以将输入信息扫描为一个 long 型的数据, 如果输入的信息不能被成功转换为 long 型, 将抛出 java.util.InputMismatchException 异常。



多学两招:

三元运算符(?:)是if...else语句的一个简洁写法,可以根据需求来决定使用哪种。前者常用于赋值判断,后者常用于业务流程。

技术要点

本实例主要的技术就是应用if语句判断闰年。判断一个年份是否为闰年,要满足两个条件:一个是能被4整除但不能被100整除,另一个是能被400整除。

判断闰年的公式用Java语法实现的格式如下:

```
year % 4 == 0 && year % 100 != 0 || year % 400 == 0
```

实例 020 验证登录信息的合法性

(实例位置: 配套资源\SL\04\020)

实例说明

大多系统的登录模块都会接收用户通过键盘输入的登录信息,这些登录信息将会被登录模块验证,如果使用的是指定的用户名与密码,则允许用户登录;否则将用户拒之门外。本实例将通过if...else语句进行多条件判断实现登录信息验证。实例的运行效果如图4.2所示。

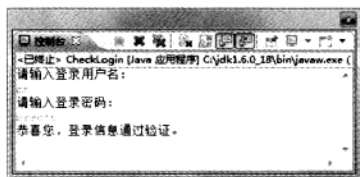


图 4.2 输入合法登录信息的效果

实现过程

(1) 在Eclipse中创建项目020,并在该项目中创建com.mingrisoft包。

(2) 在com.mingrisoft包中创建类文件,名称为CheckLogin,并在该类的主方法中接收用户输入的登录用户名与登录密码,然后通过if条件语句分别判断用户名与密码,并输出登录验证结果。关键代码如下:

```
import java.util.Scanner;
public class CheckLogin {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);           //创建扫描器
        System.out.println("请输入登录用户名: ");
        String username = scan.nextLine();                 //接收用户输入登录名
        System.out.println("请输入登录密码: ");
        String password = scan.nextLine();                 //接收用户输入登录密码
        if (!username.equals("mr")) {                     //判断用户名合法性
            System.out.println("用户名非法。");
        } else if (!password.equals("mrsoft")) {          //判断密码合法性
            System.out.println("登录密码错误。");
        } else {                                           //通过以上两个条件判断则默认通过登录验证
            System.out.println("恭喜您, 登录信息通过验证。");
        }
    }
}
```





多学两招：

字符串属于对象而非基本数据类型，不能使用“==”来判断两个字符串是否相当，所以需要通过 equals() 方法来判断两个字符串内容是否相同，正如本实例对用户名和密码的判断那样。使用“==”判断的将是两个字符串对象的内存地址，而非字符串内容。

技术要点

本实例应用的主要技术点就是 if...else 语句。if...else 语句的语法格式比较灵活，可以是简单的 if...else 格式，也可以是以下格式：

```
if(表达式){
    若干语句
}else if{
    若干语句
}else{
    若干语句
}
```

本实例中应用的就是这种格式。

实例 021 为新员工分配部门

(实例位置：配套资源\SL\04\021)

实例说明

本实例根据用户输入的信息确定员工应该分配到哪个部门。实例中需要根据用户输入的信息进行多条件判断，所以采用了 switch 语句。实例的运行效果如图 4.3 所示。

实现过程

(1) 在 Eclipse 中创建项目 021，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Example，并在该类的主方法中创建标准输入流的扫描器，通过扫描器获取人事部门输入的姓名与应聘编程语言，然后根据每个语言对应的哈希码来判断分配部门。关键代码如下：

```
import java.util.Scanner;
public class Example {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);           //创建扫描器
        System.out.println("请输入新员工的姓名：");
        String name = scan.nextLine();                     //接收员工名称
        System.out.println("请输入新员工应聘的编程语言：");
        String language = scan.nextLine();                 //接收员工应聘的编程语言
        //根据编程语言确定员工分配的部门
        Switch (language.hashCode()) {
            case 3254818:                                   //java 的哈希码
            case 2301506:                                   //Java 的哈希码
```

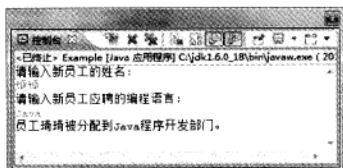


图 4.3 为新员工分配部门



Note

```

case 2269730:                                //JAVA 的哈希码
    System.out.println("员工"+name+"被分配到 Java 程序开发部门。");
    break;
case 3104:                                    //c#的哈希码
case 2112:                                    //C#的哈希码
    System.out.println("员工"+name+"被分配到 C#项目维护组。");
    break;
case -709190099:                             //asp.net 的哈希码
case 955463181:                             //Asp.net 的哈希码
case 9745901:                                / ASP.NET 的哈希码
    System.out.println("员工"+name+"被分配到 Asp.net 程序测试部门。");
    break;
default:
    System.out.println("本公司不需要" + language + "语言的程序开发人员。");
}
}
}

```

多学两招:

在 switch 语法中每个 case 关键字可以作为一个条件分支,但是对于多个条件采取相同业务处理的情况,可以把多个 case 分支关联在一起,省略它们之间的 break 语句,而在最后一个相同的 case 分支中实现业务处理并执行 break 语句,就像本实例中应用的那样。

技术要点

本实例的技术要点在于 switch 多分支语句的使用。该语句只支持对常量的判断,而常量又只能是 Java 的基本数据类型,虽然在以后的 JDK 版本中可能支持对 String 类的字符串对象进行判断,但是就目前项目的需求也有很多需要对字符串进行多条件判断的情况。本实例采取的是对字符串的哈希码进行判断,也就是把 String 类的 hashCode()方法返回值作为 switch 语法的表达式,case 关键字之后跟随的是各种字符串常量的哈希码整数值。

实例 022 用 switch 语句根据消费金额计算折扣

(实例位置: 配套资源\SL\04\022)

实例说明

俗话说“商场如战场”,各大商家为了笼络有限的顾客,经常会打出各种各样的促销手段。例如,会员折扣制度,即对会员的消费金额进行累加,当超过一定数额时,可以享受相应的折扣。累计消费金额越高,享受的折扣越多。本实例将应用 switch 语句计算累计消费金额达到一定数额时,享受不同的折扣价格。实例的运行效果如图 4.4 所示。

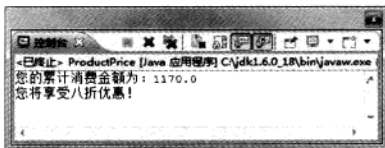


图 4.4 根据消费金额计算折扣

实现过程

(1) 在 Eclipse 中创建项目 022,并在该项目中创建 com.mingrisoft 包。



(2) 在 com.mingrisoft 包中创建类文件, 名称为 ProductPrice, 并在该类的主方法中实现本实例的业务代码。在该方法中, 首先假设一个用户消费总额的变量 money, 并初始化一个折扣变量 rebate, 然后经过运算来获得用户等级, 对不同的等级给予不同的折扣优惠。关键代码如下:

```
public class ProductPrice {
    public static void main(String[] args) {
        float money = 1170;           //金额
        String rebate = "";           //折扣
        if (money > 200) {
            int grade = (int) money / 200; //等级
            switch (grade) {           //根据等级计算折扣比例
                case 1:
                    rebate = "九五折";
                    break;
                case 2:
                    rebate = "九折";
                    break;
                case 3:
                    rebate = "八五折";
                    break;
                case 4:
                    rebate = "八三折";
                    break;
                case 5:
                    rebate = "八折";
                    break;
                case 6:
                    rebate = "七八折";
                    break;
                case 7:
                    rebate = "七五折";
                    break;
                case 8:
                    rebate = "七三折";
                    break;
                case 9:
                    rebate = "七折";
                    break;
                case 10:
                    rebate = "六五折";
                    break;
                default:
                    rebate = "六折";
            }
        }
        System.out.println("您的累计消费金额为: " + money); //输出消费金额
        System.out.println("您将享受" + rebate + "优惠!"); //输出折扣比例
    }
}
```



Note



脚下留神:

在程序开发中经常使用的都是正数,负数因为使用得少,常常被忽略,例如“ $N\%2==1$ ”本来是用来计算数字N是否为奇数的,但是开发者没有考虑到负数的情况,从而导致这个算法的失败,因为任何负数应用这个算法都会等于-1。

技术要点

本实例主要应用 switch 语句实现。switch 语句是多分支选择语句,常用来根据表达式的值选择要执行的语句。switch 语句的基本语法格式如下:

```
switch(表达式){
    case 常量表达式 1: 语句序列 1
        [break;]
    case 常量表达式 2: 语句序列 2
        [break;]
    ...
    case 常量表达式 n: 语句序列 n
        [break;]
    default: 语句序列 n+1
        [break;]
}
```

switch 语句的参数说明如表 4.1 所示。

表 4.1 switch 语句的参数说明

参数名称	参数描述
表达式	必要参数。可以是任何 byte、short、int 和 char 类型的变量
常量表达式 1	如果有 case 出现,则为必要参数。该常量表达式的值必须是一个与表达式数据类型相兼容的值
语句序列 1	可选参数。一条或多条语句,但不需要大括号。当表达式的值与常量表达式 1 的值匹配时执行;如果不匹配则继续判断其他值,直到常量表达式 n
常量表达式 n	如果有 case 出现,则为必要参数。该常量表达式的值必须是一个与表达式数据类型相兼容的值
语句序列 n	可选参数。一条或多条语句,但不需要大括号。当表达式的值与常量表达式 n 的值匹配时执行
break	可选参数。用于跳出 switch 语句
default	可选参数。如果没有该参数,则当所有匹配不成功时,将不会执行任何操作
语句序列 n+1	可选参数。如果没有与表达式的值相匹配的 case 常量时,将执行语句序列 n+1

实例 023 判断用户输入月份的季节

(实例位置: 配套资源\SL\04\023)

实例说明

一年有四季,每季 3 个月。其中,12 月、1 月和 2 月为冬季,3 月、4 月和 5 月为春季,6 月、7 月和 8 月为夏季,9 月、10 月和 11 月为秋季。本实例将根据用户输入的月份来判断季节,





这是一个最典型的演示 switch 语法的例子,通过这个例子,读者可以完全掌握 switch 语句的用法与技巧。实例的运行效果如图 4.5 所示。

实现过程

(1) 在 Eclipse 中创建项目 023,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 JudgeMonth,并在该类的主方法中创建扫描器接收用户输入的月份数字,然后判断该月份属于哪个季节并输出到控制台,对于非法月份也要给出提示。关键代码如下:

```
import java.util.Scanner;
public class JudgeMonth {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);           //创建扫描器
        //提示用户输入月份
        System.out.println("请输入一个月份,我能告诉你它属于哪个季节。");
        int month = scan.nextInt();                       //接收用户输入
        switch (month) {                                  //判断月份属于哪个季节
            case 12:
            case 1:
            case 2:
                System.out.print("您输入的月份属于冬季。");
                break;
            case 3:
            case 4:
            case 5:
                System.out.print("您输入的月份属于春季");
                break;
            case 6:
            case 7:
            case 8:
                System.out.print("您输入的月份属于夏季");
                break;
            case 9:
            case 10:
            case 11:
                System.out.print("您输入的月份属于秋季");
                break;
            default:
                System.out.print("你那有" + month + "月份吗? ");
        }
    }
}
```

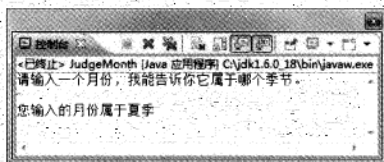


图 4.5 判断用户输入月份的季节

脚下留神:

switch 语句的每个 case 关键字都用于判断一个常量并做出相应的业务处理,熟练掌握 switch 语句之后可以组合多个 case 来完成多条件的处理,就是多个常量结果执行相同的业务处理,如本实例中应用的格式。



技术要点

本实例应用的主要技术是 switch 语句, 关于 switch 语句的详细介绍请参见实例 022。

实例 024 使用 while 循环语句与自增运算符循环遍历数组

(实例位置: 配套资源\SL\04\024)



Note

实例说明

在多数情况下, 遍历数组都是使用 for 循环语句实现。其实应用 while 循环语句和自增运算符也可实现遍历数组。本实例将利用自增运算符结合 while 循环语句实现数组的遍历, 并将遍历结果输出到控制台。实例的运行效果如图 4.6 所示。

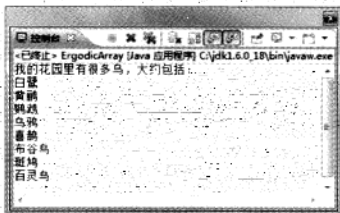


图 4.6 输出数组的遍历结果

实现过程

(1) 在 Eclipse 中创建项目 024, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 ErgodicArray, 并在该类的主方法中创建一个鸟类数组, 然后创建一个索引变量, 这个变量用于指定数组下标, 随着该索引的递增, while 循环会逐步获取每个数组的元素并输出到控制台中。关键代码如下:

```
public class ErgodicArray {
    public static void main(String[] args) {
        //创建鸟类数组
        String[] aves = new String[] { "白鹭", "黄鹂", "鸚鵡", "乌鸦", "喜鹊", "布谷鸟", "斑鸠", "百灵鸟" };

        int index = 0; //创建索引变量
        System.out.println("我的花园里有很多鸟, 大约包括: ");
        while (index < aves.length) { //遍历数组
            System.out.println(aves[index++]); //自增索引值
        }
    }
}
```

指点迷津:

自增自减运算符分前置和后置两种。其中, 前置运算如 “++index”, 会先将 index 的值递增, 然后再使用递增后的值; 而后置运算如 “index++”, 会首先使用该变量的值, 然后再把变量值递增。

技术要点

本实例应用的主要技术是 while 循环语句和自增运算符。while 循环语句的基本语法格式如下:

```
while(条件表达式){
    语句序列
}
```




参数说明

- ☑ 条件表达式：决定是否进行循环的表达式，其结果为 boolean 类型，也就是其结果只能是 true 或 false。
- ☑ 语句序列：也就是循环体，在条件表达式的结果为 true 时，重复执行。



Note

实例 025 使用 for 循环输出杨辉三角形

(实例位置：配套资源\SL\04\025)

实例说明

杨辉三角形由数字排列，可以把它看作一个数字表，其基本特性是两侧数值均为 1，其他位置的数值是其正上方的数值与左上角数值之和。本实例将使用 for 循环输出包括 10 行内容的杨辉三角形。实例的运行效果如图 4.7 所示。

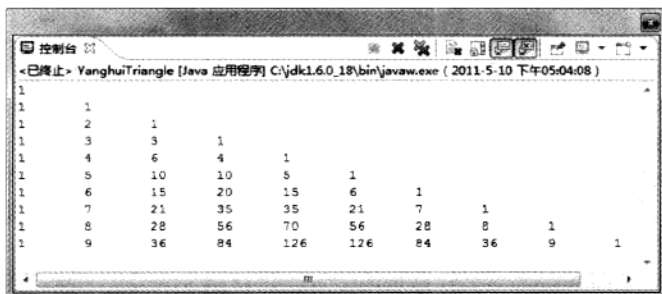


图 4.7 使用 for 循环输出杨辉三角形

实现过程

(1) 在 Eclipse 中创建项目 025，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 YanghuiTriangle，并在该类的主方法中创建一个二维数组，并指定二维数组的第一维长度，这个数组用于存放杨辉三角形的数值表，通过双层 for 循环来实现第二维数组的长度，然后计算整个数组的每个元素的值。关键代码如下：

```
public class YanghuiTriangle {
    public static void main(String[] args) {
        int triangle[][]=new int[10][];           //创建二维数组
        //遍历二维数组的第一层
        for (int i = 0; i < triangle.length; i++) {
            triangle[i]=new int[i+1];             //初始化第二层数组的大小
            //遍历第二层数组
            for(int j=0;j<=i;j++){
                if(i==0||j==0||j==i){              //将两侧的数组元素赋值为 1
                    triangle[i][j]=1;
                }else{                               //其他数值通过公式计算
                    triangle[i][j]=triangle[i-1][j]+triangle[i-1][j-1];
                }
            }
            System.out.print(triangle[i][j]+"\\t"); //输出数组元素
        }
    }
}
```



```
    }  
    System.out.println();           //换行  
}  
}  
}
```

指点迷津:

在创建二维数组时,其第一维的长度即是要输出杨辉三角形的行数。例如,在本实例中,要输出 10 行的杨辉三角形就可以将其第一维的长度设置为 10。

指点迷津:

Java 语言中的二维数组其实是每个元素都是一个一维数组的一维数组,所以第二维的长度可以任意,就像本实例中那样。这比其他语言的数组更灵活,而且多维数组也是如此。

技术要点

本实例应用的主要技术是应用 for 循环语句根据杨辉三角形的公式遍历二维数组。杨辉三角形的公式包括两部分,一部分是两侧数值都是 1,也就是说二维数组的 `triangle[0][0]`、`triangle[i][0]` 或者 `triangle[i][i]` 的元素值为 1,另一部分是其他位置的数值是其正上方的数值与左上角数值之和,也就是 `triangle[i][j]=triangle[i-1][j]+triangle[i-1][j-1]`。

指点迷津:

在二维数组中,第 1 个下标值代表的是行数,第 2 个下标值代表的是列数,即 `triangle[i][0]` 代表的是第 *i* 行第 0 列的元素。

实例 026 使用嵌套循环在控制台上输出九九乘法表

(实例位置: 配套资源\SL\04\026)

实例说明

对于九九乘法表读者都不会陌生,几乎每个人都能倒背如流。那么如何通过嵌套循环,在控制台上输出九九乘法表呢?这个内容在掌握了 for 循环的嵌套后,就不难实现。本实例将应用嵌套的 for 循环实现在控制台上输出九九乘法表。实例的运行效果如图 4.8 所示。



图 4.8 应用嵌套的 for 循环输出九九乘法表



实现过程

(1) 在 Eclipse 中创建项目 026，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 MultiplicationTable，并在该类的主方法中通过双层 for 循环输出九九乘法表。关键代码如下：

```
public class MultiplicationTable {
    public static void main(String[] args) {
        for(int i=1;i<=9;i++){           //循环控制变量从 1 遍历到 9
            for(int j=1;j<=i;j++){        //第二层循环控制变量与第一层最大索引相等
                System.out.print(j+"*"+i+"="+i*j+"\t"); //输出计算结果但不换行
            }
            System.out.println();         //在外层循环中换行
        }
    }
}
```

脚下留神：

在上面的代码中，在内层循环输出计算结果时，应用的是 System.out 对象的 print() 方法输出，应用该方法不换行，在内层循环结束后，再应用 System.out 对象的 println() 方法换行。

指点迷津：

循环语句可用于完成复杂的运算，也可以用于控制程序的递归流程，而多层循环可以实现更加复杂的业务逻辑，是学习编程必须掌握的一种应用。在处理有规则的大量数据时，应该考虑使用多层循环来优化程序代码，但是建议添加详细的代码注释，便于以后的维护与修改工作。

技术要点

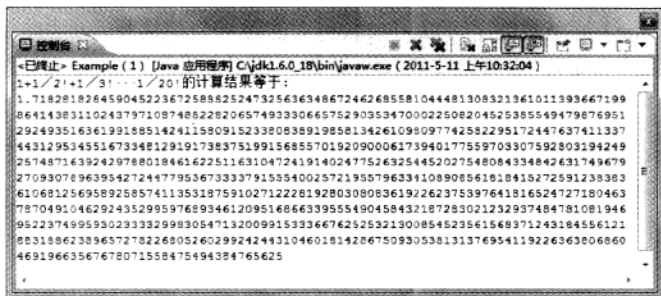
要在控制台上输出九九乘法表，可以通过嵌套的 for 循环来实现，即创建双层的 for 循环。第一层 for 循环，也称外循环，用于控制表格的行；第二层 for 循环，也称内循环，用于控制表格的列。其中，第一层 for 循环的控制变量的最大值是 9，第二层 for 循环的控制变量的最大值要等于行数的最大值。然后输出内层与外层循环控制变量的乘积，即可实现九九乘法表。

实例 027 使用 while 循环计算 $1+1/2!+1/3!+\cdots+1/20!$

(实例位置：配套资源\SL\04\027)

实例说明

本实例在计算阶乘的算法之上应用 while 循环语句计算 $1+1/2!+1/3!+\cdots+1/20!$ 的和。如果使用基本数据类型 double 是无法精确地显示运算结果的，所以本实例使用了 BigDecimal 类的实例来完成这个运算。实例的运行效果如图 4.9 所示。

图 4.9 用 while 循环计算 $1+1/2!+1/3!+\dots+1/20!$

指点迷津:

由于本实例的运行结果精度非常高,小数位数过长,默认情况下,运行结果只能显示单行的数字,所以需要控制台进行折行设置。具体方法是,在控制台中单击鼠标右键,在弹出的快捷菜单中选择“首选项”命令,在弹出的“首选项”对话框中选中“固定宽度控制台”复选框,并设置每行的最大字符数,可以采用默认的 80 个,然后单击“确定”按钮即可。这时,再运行本实例,将显示如图 4.9 所示的运行效果。

实现过程

(1) 在 Eclipse 中创建项目 027,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 Example,并在该类的主方法中保存总和的 sum 变量和计算阶乘的 factorial 变量,为保证计算结果的精度,这两个变量都是 BigDecimal 类的实例对象,然后通过 while 实现 20 次循环,并完成计算。关键代码如下:

```
import java.math.BigDecimal;

public class Example {
    public static void main(String args[]) {
        BigDecimal sum = new BigDecimal(0.0);           //和
        BigDecimal factorial = new BigDecimal(1.0);      //阶乘项的计算结果
        int i = 1;                                       //循环增量
        while (i <= 20) {
            sum = sum.add(factorial);                    //累加各项阶乘的和
            ++i;                                         //i 加 1
            factorial = factorial.multiply(new BigDecimal(1.0 / i)); //计算阶乘项
        }
        System.out.println("1+1 / 2!+1 / 3! ... 1 / 20!的计算结果等于: \n" + sum); //输出计算结果
    }
}
```

脚下留神:

对于高精度要求或者运算数较大的计算,应该使用 BigDecimal 类实现,否则 Java 基本类型的数据无法保证浮点数的精度,也无法对超出其表示范围的数字进行运算。

技术要点

本实例主要应用了 while 循环和 BigDecimal 类的实例对象的相关方法实现计算 $1+1/2!+1/3!+\dots$



1/20!的和。BigDecimal 类型的数字可以用来做超大的浮点数的运算,如加、减、乘、除等。使用 BigDecimal 对象的 add()方法可以实现加法运算,使用 multiply()方法可以实现乘法运算。



Note

实例 028 使用 for 循环输出空心的菱形

(实例位置: 配套资源\SL\04\028)

实例说明

本实例在输出菱形的基础上加大难度,输出空心的菱形图案,这在等级考试与公司面试时也出现过类似题目,实例目的在于要求熟练掌握 for 循环的嵌套使用。实例的运行效果如图 4.10 所示。

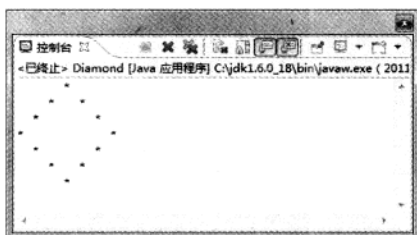


图 4.10 使用 for 循环输出空心的菱形

实现过程

(1) 在 Eclipse 中创建项目 028,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 Diamond,并在该类的主方法中调用 printHollowRhombus()方法完成 7 行的空心菱形输出。其中 printHollowRhombus()方法是实例中自定义的,该方法使用两个双层 for 循环分别输出菱形的上半部分与下半部分。关键代码如下:

```
public class Diamond {
    public static void main(String[] args) {
        printHollowRhombus(7); //输出 7 行的菱形
    }
    public static void printHollowRhombus(int size) {
        if (size % 2 == 0) {
            size++; //计算菱形大小
        }
        for (int i = 0; i < size / 2 + 1; i++) {
            for (int j = size / 2 + 1; j > i + 1; j--) {
                System.out.print(" "); //输出左上角位置的空白
            }
            for (int j = 0; j < 2 * i + 1; j++) {
                if (j == 0 || j == 2 * i) {
                    System.out.print("* "); //输出菱形上半部边缘
                } else {
                    System.out.print(" "); //输出菱形上半部空心
                }
            }
        }
    }
}
```



```

        System.out.println("");           //换行
    }
    for (int i = size / 2 + 1; i < size; i++) {
        for (int j = 0; j < i - size / 2; j++) {
            System.out.print(" ");         //输出菱形左下角空白
        }
        for (int j = 0; j < 2 * size - 1 - 2 * i; j++) {
            if (j == 0 || j == 2 * (size - i - 1)) {
                System.out.print("*");     //输出菱形下半部边缘
            } else {
                System.out.print(" ");     //输出菱形下半部空心
            }
        }
        System.out.println("");           //换行
    }
}

```



Note

脚下留神:

for 循环中有 3 个表达式, 这 3 个表达式都是可选的, 也就是说 for 循环可以没有表达式。例如 for(;;) 这样的 for 循环将是一个无限循环, 读者在使用 for 循环时应注意避免无限循环。

技术要点

本实例应用的主要技术是 for 循环语句的嵌套和 if...else 语句。其中, if...else 语句用于控制菱形中心位置不输出*号, 即输出空心菱形。

实例 029 终止循环体

(实例位置: 配套资源\SL\04\029)

实例说明

循环用于复杂的业务处理, 可以提高程序的性能和代码的可读性, 但是循环中也有特殊情况, 例如由于某些原因需要立刻中断循环去执行下面的业务逻辑, 这时就可以使用 break 语句实现。本实例将实现应用 break 语句中断单层循环和中断双层 for 循环。实例的运行效果如图 4.11 所示。

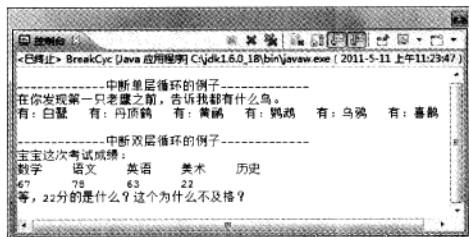


图 4.11 终止循环体



实现过程

(1) 在 Eclipse 中创建项目 029, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 BreakCyc。在该类的主方法中, 首先创建一个字符串数组, 然后在使用 for 循环遍历时判断如果发现数组中包含字符串“老鹰”则立刻中断循环。最后再创建一个整数类型的二维数组, 并使用双层 for 循环遍历, 当发现第一个小于 60 的数组元素时, 立刻中断整个双层循环, 而不是内层循环。关键代码如下:

```
public class BreakCyc {
    public static void main(String[] args) {
        System.out.println("\n-----中断单层循环的例子-----");
        //创建数组
        String[] array = new String[] { "白鹭", "丹顶鹤", "黄鹂", "鹦鹉", "乌鸦", "喜鹊",
            "老鹰", "布谷鸟", "老鹰", "灰纹鸟", "老鹰", "百灵鸟" };
        System.out.println("在你发现第一只老鹰之前, 告诉我都有什么鸟。");
        for (String string : array) {
            if (string.equals("老鹰"))
                break;
            System.out.print("有: " + string + " ");
        }

        System.out.println("\n\n-----中断双层循环的例子-----");
        //创建成绩数组
        int[][] myScores = new int[][] { { 67, 78, 63, 22, 66 }, { 55, 68, 78, 95, 44 }, { 95, 97, 92, 93,
            81 } };
        System.out.println("宝宝这次考试成绩: \n 数学\t 语文\t 英语\t 美术\t 历史");
        No1: for (int[] is : myScores) {
            for (int i : is) {
                System.out.print(i + "\t");
                if (i < 60) {
                    System.out.println("\n 等, " + i + "分的是什么? 这个为什么不及格? ");
                    break No1;
                }
            }
            System.out.println();
        }
    }
}
```

指点迷津:

充分利用循环可以提高程序的开发与执行效率, 但是如果不注重循环中的算法很容易导致程序的死循环, 所以在循环体中要对可能出现的特殊情况使用 break 语句中断循环。

技术要点

本实例应用的主要技术是 break 语句。break 语句用于强行退出循环。使用带标签的 break 语句, 可以强行退出多层循环。例如, 要退出双层的 for 循环, 可以使用下面的语法:

```
lable:
for(元素变量 x: 遍历对象 obj){
```



```
for(元素变量 x1: 遍历对象 obj1){
    引用了 x 的 Java 语句;
    if(条件表达式) {
        break label;
    }
}
```

在上面的语法中, 如果执行到 **break**, 正常情况下应该结束内层循环去执行外层循环, 但是由于 **break** 后带有标签, 所以程序将结束标签处的外层循环。



Note

实例 030 循环体的过滤器

(实例位置: 配套资源\SL\04\030)

实例说明

循环体中可以通过 **break** 语句中断整个循环, 这增加了循环的控制能力, 但是对于特殊情况还是不够, 例如某些条件下需要放弃部分循环处理, 而不是整个循环体。Java 提供了 **continue** 语句来实现这一功能, **continue** 语句可以放弃本次循环体的剩余代码, 不执行它们而开始下一轮的循环。本实例利用 **continue** 语句实现了循环体过滤器, 可以过滤“老鹰”字符串, 并做相应的处理, 但是放弃 **continue** 语句之后的所有代码。实例的运行效果如图 4.12 所示。

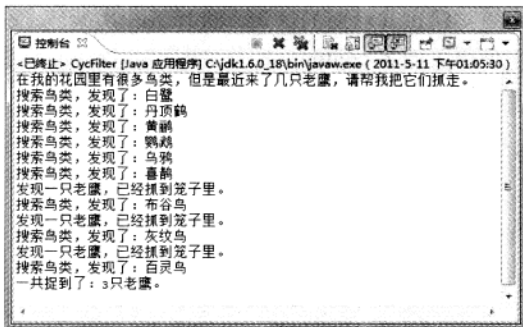


图 4.12 过滤“老鹰”字符串的结果

实现过程

(1) 在 Eclipse 中创建项目 030, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 CycFilter, 并在该类的主方法中创建鸟类名称的字符串数组, 其中包含多个“老鹰”字符串, 然后通过 for 循环遍历该数组, 在循环过程中如果遍历的数组元素是“老鹰”字符串, 则输出发现老鹰的信息并过滤循环体之后的所有代码。关键代码如下:

```
package com.mingrisoft;
public class CycFilter {
    public static void main(String[] args) {
        //创建数组
        String[] array = new String[] { "白鹭", "丹顶鹤", "黄鹌", "鸚鵡", "乌鸦", "喜鹊",
```



Note

```
"老鹰", "布谷鸟", "老鹰", "灰纹鸟", "老鹰", "百灵鸟"};
System.out.println("在我的花园里有很多鸟类，但是最近来了几只老鹰，请帮我把它们
    抓走。");
int eagleCount = 0;
for (String string : array) {                                //for 循环遍历数组
    if (string.equals("老鹰")) {                              //如果遇到老鹰
        System.out.println("发现一只老鹰，已经抓到笼子里。");
        eagleCount++;
        continue;                                             //中断循环
    }
    System.out.println("搜索鸟类，发现了：" + string);        //否则输出数组元素
}
System.out.println("一共捉到了：" + eagleCount + "只老鹰。");
}
```

多学两招：

`break` 语句和 `continue` 语句都是对循环体的控制语句，它们不仅应用于 `for` 循环，在任何循环体中都可以使用这些语句，灵活使用可以让循环实现更加复杂的运算和业务处理。

技术要点

本实例应用的主要技术是 `continue` 语句。`continue` 语句只能应用在 `for`、`while` 和 `do...while` 循环语句中，用于让程序直接跳过其后面的语句，进行下一次循环。`continue` 语句的语法比较简单，只需要在循环体中使用关键字 `continue` 加分号即可。具体的语法格式如下：

```
continue;
```

第 5 章

数组及其常用操作

本章读者可以学到如下实例：

- ▶▶ 实例 031 获取一维数组的最小值
- ▶▶ 实例 032 将二维数组中的行列互换
- ▶▶ 实例 033 利用数组随机抽取幸运观众
- ▶▶ 实例 034 用数组设置 JTable 表格的列名与列宽
- ▶▶ 实例 035 使用按钮控件数组实现计算器界面
- ▶▶ 实例 036 通过复选框控件数组实现添加多个复选框控件
- ▶▶ 实例 037 使用选择排序法对数组排序
- ▶▶ 实例 038 使用冒泡排序法对数组排序
- ▶▶ 实例 039 使用快速排序法对数组排序
- ▶▶ 实例 040 使用直接插入法对数组排序
- ▶▶ 实例 041 使用 sort() 方法对数组排序
- ▶▶ 实例 042 反转数组中元素的顺序



实例 031 获取一维数组的最小值

(实例位置: 配套资源\SL\05\031)

实例说明

一维数组常用于保存线性数据,如数据库中的单行数据就可以使用一维数组保存。本实例将接收用户在文本框中输入的单行数据(其中数据都是整数数字,以不同数量的空格分割),这个数据将被程序分解成一维数组,并从数组中提取最小值显示在界面中。实例的运行效果如图 5.1 所示。

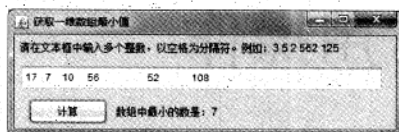


图 5.1 获取一维数组的最小值

指点迷津:

程序经过特殊判断,数字之间的空格可以使用多个。

实现过程

(1) 在 Eclipse 中创建项目 031,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建 JFrame 窗体类,名称为 ArrayMinValue。在窗体中添加一个文本框和一个“计算”按钮以及多个标签控件。

(3) 编写“计算”按钮的事件处理方法,在该方法中首先获取用户的输入,并通过 trim() 方法去除左右空格,然后对字符串内容进行检测,排除非法输入,并把字符串转换为整型数组,最后在遍历数组的同时提取最小值并显示到窗体的标签控件中。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String arrayStr = textField.getText().trim();           //去除左右空格
    if(arrayStr.equals("")){                                //对空值进行处理
        JOptionPane.showMessageDialog(null, "请输入数字内容");
        return;
    }
    for (int i = 0; i < arrayStr.length(); i++) {           //过滤非法输入
        char charAt = arrayStr.charAt(i);
        if (!Character.isDigit(charAt) && charAt != ' ') {
            JOptionPane.showMessageDialog(null, "输入包含非数字内容");
            textField.setText("");
            return;
        }
    }
    String[] numStrs = arrayStr.split(" {1,}");            //分割字符串
    int[] numArray = new int[numStrs.length];             //创建整型数组
    //转换输入为整型数组
```



```

for (int i = 0; i < numArray.length; i++) {
    numArray[i] = Integer.valueOf(numStrs[i]);
}
int min = numArray[0]; //创建最小数变量
for (int j = 0; j < numArray.length; j++) {
    if (min > numArray[j]) { //提取最小整数
        min = numArray[j];
    }
}
label.setText("数组中最小的数是: " + min); //显示最小值到指定的标签中
}

```



Note

多学两招:

for 语句用于程序的循环流程控制。该语句有 3 个表达式用于循环变量的控制, 其完整语法格式为:

```
for(int i=0; i<100; i++){
```

```
...
```

```
}
```

for 语句中的 3 个表达式不是完全必备的, 可以根据情况部分省略, 甚至完全省略。例如下面代码就以最简单的格式实现了无限循环。

```
for (;;) {
```

```
...
```

```
}
```

技术要点

本实例主要应用 String 类的 split() 方法将输入的字符串分割为字符串数组, 再将其转换为整型数组, 最后通过 for 循环遍历该数组, 并通过 if 语句提取最小值。其中, split() 方法可以使字符串按指定的分割字符或字符串进行分割, 并将分割后的结果存放在字符串数组中。其语法格式如下:

```
string.split(string sign)
```

参数说明

- ☒ string: 为字符串对象。
- ☒ sign: 为分割字符串的分割符, 也可以使用正则表达式。

实例 032 将二维数组中的行列互换

(实例位置: 配套资源\SL\05\032)

实例说明

数组是程序开发中最常用的, 其中二维数组使用最频繁, 它可以存储表格数据, 还可以根据数组下标索引加入各种运算。另外, 图片的关键运算方法也是以二维数组为基础进行矩阵运算的。作为数组知识的巩固, 本实例实现数组模拟表格行与列数据交换, 这在程序开发中常用。



于表格数据的整理。实例的运行效果如图 5.2 所示。

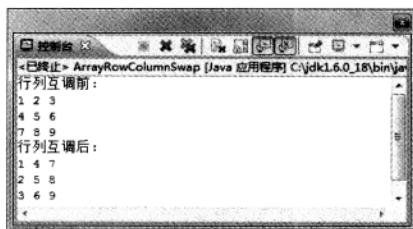


图 5.2 将二维数组中的行列互换

实现过程

(1) 在 Eclipse 中创建项目 032，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 ArrayRowColumnSwap，并在该类的主方法中定义一个二维数组，输出该数组的内容，这次输出是为了与交换数据后的数组进行对比。新创建一个同样大小的二维数组，利用双层 for 循环遍历数组时，把新数组与原数组的行列索引交换进行元素赋值，然后再输出新数组内容。关键代码如下：

```
public class ArrayRowColumnSwap { //创建类
    public static void main(String[] args) {
        //创建二维数组
        int arr[][] = new int[][] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
        System.out.println("行列互调前: ");
        //输出二维数组
        printArray(arr);
        int arr2[][] = new int[arr.length][arr.length];
        for (int i = 0; i < arr.length; i++) { //调整数组行列数据
            for (int j = 0; j < arr[i].length; j++) {
                arr2[i][j] = arr[j][i];
            }
        }
        System.out.println("行列互调后: ");
        //输出行列互调后的二维数组
        printArray(arr2);
    }
}
```

(3) 编写输出数组内容的 printArray() 方法。输出数组内容的业务在程序中出现两次，根据代码重用的原则，如果相同的业务代码在程序中出现两次以上，就应该把它们提取成一个独立的方法，在这个方法中简单地通过双层 for 循环遍历数组元素。关键代码如下：

```
private static void printArray(int[][] arr) {
    for (int i = 0; i < arr.length; i++) { //遍历数组
        for (int j = 0; j < arr[i].length; j++) {
            System.out.print(arr[i][j] + " "); //不换行输出数组元素
        }
        System.out.println(); //换行
    }
}
```



脚下留神:

在 Java 语言中定义数组变量时,不能声明其长度,只能在使用 new 关键字创建数组时指定,例如 `int[9] array =` 是错误的写法,应该是 `int[] array = new int[9]`。

技术要点

本实例应用的主要技术就是创建两个相同大小的二维数组,并使用双层的 for 循环遍历这两个二维数组,同时把新数组与原数组的行列索引交换进行元素赋值,从而实现将二维数组中的行列互换的操作。

实例 033 利用数组随机抽取幸运观众

(实例位置: 配套资源\SL\05\033)

实例说明

在电视节目,经常看到随机抽取幸运观众。如果观众抽取的范围较少,可以让程序使用数组实现,而且效率很高。具体的实现方法是:首先将所有观众姓名生成数组,然后获得数组元素的总数量,最后再随机抽取元素的下标,根据抽取的下标获得幸运观众。本实例将利用数组实现随机抽取幸运观众的程序。实例的运行效果如图 5.3 所示。

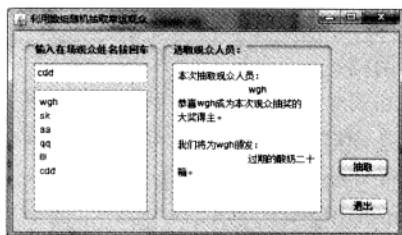


图 5.3 利用数组随机抽取幸运观众

实现过程

- (1) 在 Eclipse 中创建项目 033,并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类,名称为 ArrayExample。在窗体中添加两个文本域、一个文本框和两个按钮,其中两个按钮分别用于抽取幸运观众和退出程序。
- (3) 为文本框添加按键事件监听器,并编写事件处理方法,当用户在文本框中输入观众姓名并按下回车键时,事件处理方法将观众姓名添加到文本域中并以回车换行作为分割符,然后选择文本框中所有文本准备接收用户的下一次输入。关键代码如下:

```
protected void do_textField_keyPressed(KeyEvent e) {
    if (e.getKeyChar() != '\n') //不是回车字符不做处理
        return;
    String name = nameField.getText();
    if (name.isEmpty()) //如果文本框没有字符串不做处理
        return;
    personnelArea.append(name + "\n"); //把输入人名与回车符添加到人员列表
    nameField.selectAll(); //选择文本框所有字符
}
```

- (4) 编写“抽取”按钮的事件处理方法,在该方法中把文本域保存的所有观众名称分割成字符串数组,然后通过随机数生成数组下标,当然这个下标是不固定的,然后在另一个文本





域控件中输出抽取幸运观众的颁奖信息。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {  
    String perstring = personnelArea.getText();           //获取人员列表文本  
    String[] personnelArray = perstring.split("\n{1,}");   //获取人员数组  
    int index = (int) (Math.random() * personnelArray.length); //生成随机数组索引  
    //定义包含格式参数的中奖信息  
    String formatArg = "本次抽取观众人员: \n\t%1$s\n 恭喜%1$s 成为本次观众抽奖的大奖得主。"  
        + "\n\n 我们将为%1$s 颁发: \n\t 过期的酸奶二十箱。";  
    //为中奖信息添加人员参数  
    String info = String.format(formatArg, personnelArray[index]);  
    resultArea.setText(info);                             //在文本域显示中奖信息  
}
```

指点迷津：

在上面的代码中应用了 String 类的 format() 方法按指定的方式格式化字符串。其中该方法第一个参数用于指定格式字符串，第二个参数用于指定格式字符串中由格式说明符引用的参数。这里为幸运观众的名称。

多学两招：

在创建与初始化数组时，通常是先定义指定类型的数组变量，然后使用 new 关键字创建数组，再分别对数组元素进行赋值。例如下面的代码：

```
int[] array = new int[3];  
array[0]=1;  
array[1]=2;  
array[2]=3;
```

另外，Java 还支持静态数组初始化，也就是在定义数组的同时为数组分配空间并赋值。例如下面的代码：

```
int[] array = { 1, 2, 3, 4};
```

技术要点

本实例的重点是把字符串中的人员名单分割为数组，以及随机生成数组的下标索引，这需要用到 String 类的 split() 方法和 Math 类的 random() 方法。

1. 将字符串分割为数组

String 类的 split() 方法可以根据指定的正则表达式对字符串进行分割，并返回分割后的字符串数组，例如 “a, b, c”，如果以 “,” 作为分隔符，返回值就是包含 “a”、“b” 和 “c” 3 个字符串的数组。该方法的声明如下：

```
public String[] split(String regex)
```

参数说明

regex: 分割字符串的定界正则表达式。

2. 生成随机数

抽奖当然是随机抽取的，这就需要用到随机数，Java 在 Math 类中提供了静态方法 random()



可以生成 0~1 之间的 double 类型的随机数值。该方法的声明如下：

```
public static double random()
```

由于该方法生成的是 0~1 之间的小数，而数组下标是整数而且又要根据数组长度来生成随机数，所以要把生成的随机数与数组长度相乘，就像本实例中的算法那样。关键代码如下：

```
int index = (int) (Math.random() * personnelArray.length); //生成随机数组索引
```

此代码把随机数与数组长度的乘积转换为整型作为随机数组下标索引。



Note

实例 034 用数组设置 JTable 表格的列名与列宽

(实例位置：配套资源\SL\05\034)

实例说明

数组在程序开发中被广泛应用，使用数组可以使程序代码更加规范，更易于维护。例如，字符串数组可用于定义表格控件的列名称，而整型数组可以用来定义列对应的宽度，本实例就通过这两个数组实现了对表格控件中表头列的设置。实例的运行效果如图 5.4 所示。

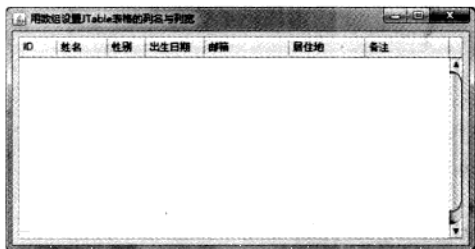


图 5.4 用数组设置 JTable 表格的列名与列宽

实现过程

(1) 在 Eclipse 中创建项目 034，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 ArrayCreateTable。在窗体中添加一个滚动面板。

(3) 编写 getTable() 方法来创建表格，在该方法中首先声明字符串数组 columns 作为表格的列名，并声明 int 类型的数组来定义每个表格列的宽度，然后创建表格的数据模型并遍历所有表格列对象，根据 int 类型数组的索引来设置表格列的宽度。关键代码如下：

```
private JTable getTable() {
    if (table == null) {
        table = new JTable(); //创建表格
        //定义列名数组
        String[] columns = { "ID", "姓名", "性别", "出生日期", "邮箱", "居住地", "备注" };
        //定义列宽数组
        int[] columnWidth = { 10, 30, 10, 40, 70, 60, 70 };
        //创建表格数据模型
        DefaultTableModel model = new DefaultTableModel(columns, 15);
        table.setModel(model); //设置表格数据模型
        TableColumnModel columnModel = table.getColumnModel(); //获取列模型
        int count = columnModel.getColumnCount(); //获取列数量
    }
}
```



```
for (int i = 0; i < count; i++) {  
    TableColumn column = columnModel.getColumn(i);  
    column.setPreferredWidth(columnWidth[i]);  
}  
}  
return table;  
}
```

//遍历列
//获取列对象
//以数组元素设置列的宽度

脚下留神:

如果直接将表格控件添加到滚动面板以外的容器中, 首先应该通过 `JTable` 类的 `getTableHeader()` 方法获取表格的 `JTableHeader` 表头类的对象, 然后再将该对象添加到容器相应的位置, 否则表格将没有表头, 无法显示任何列名称。

技术要点

本实例的关键技术在于设置表格的数据模型和访问列模型。其中表格的数据模型可以采用 `DefaultTableModel` 类创建数据模型对象, 而创建过程中可以把字符串数组作为参数来创建表格列的名称。

1. 创建表格数据模型

`DefaultTableModel` 类的构造方法有很多, 其中一个可以把字符串数组作为参数来生成列名称, 同时接收 `int` 类型的参数来设置表格添加多少行空白数据。这个构造方法的声明如下:

```
public DefaultTableModel(Object[] columnNames, int rowCount)
```

参数说明

- ☒ `columnNames`: 存放列名的数组。
- ☒ `rowCount`: 指定创建多少行空白数据。

2. 设置表格数据模型

`JTable` 类是表格控件, 它提供了 `setModel()` 方法来设置表格的数据模型。设置数据模型以后, 表格控件可以从数据模型中提取表头所有列名称和所有行数据, 这个数据模型将负责表格所有数据的维护。该设置表格模型的方法的声明如下:

```
public void setModel(TableModel dataModel)
```

参数说明

`dataModel`: 此表的新数据模型。

3. 获取表格列模型

表格中所有列对象都存放在列模型中, 它们用于定义表格的每个列的名称及宽度等信息。表格的列模型可以通过 `getColumnModel()` 方法来获取。其方法的声明如下:

```
public TableColumnModel getColumnModel()
```

4. 设置列宽度

列对象存放在列模型中, 并且列的宽度需要通过列对象的 `setPreferredWidth()` 方法来设置。该方法的声明如下:

```
public void setPreferredWidth(int preferredWidth)
```

参数说明

`preferredWidth`: 列对象的首选宽度参数。



实例 035 使用按钮控件数组实现计算器界面

(实例位置: 配套资源\SL\05\035)

实例说明

控件数组的应用范围非常广泛,合理使用控件数组可以提高程序开发效率。本实例将应用按钮控件数组来管理界面中的所有按钮控件,从而使用最少的代码实现模拟的计算器界面。实例的运行效果如图 5.5 所示。

实现过程

(1) 在 Eclipse 中创建项目 035,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建 JFrame 窗体类,名称为 ButtonArrayExample。在窗体中添加一个文本框控件用于模拟计算器的液晶屏。

(3) 在构造方法中设置窗体标题,布局管理器,并创建 JButton 控件的二维数组,其中每个数组元素都初始化为一个按钮控件,同时再声明一个按钮名称的字符串数组,这两个数组共同初始化界面中的所有按钮控件。关键代码如下:



图 5.5 按钮控件数组实现计算器界面

```
public ButtonArrayExample() {
    super(); //继承父类的构造方法
    BorderLayout borderLayout = (BorderLayout) getContentPane().getLayout();
    borderLayout.setHgap(20);
    borderLayout.setVgap(10);
    setTitle("按钮数组实现计算器界面 "); //设置窗体的标题
    setBounds(100, 100, 290, 282); //设置窗体的显示位置及大小
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置窗体关闭按钮的动作作为退出
    textField = new JTextField();
    textField.setHorizontalAlignment(SwingConstants.TRAILING);
    textField.setPreferredSize(new Dimension(12, 50));
    getContentPane().add(textField, BorderLayout.NORTH);
    textField.setColumns(10);
    final GridLayout gridLayout = new GridLayout(4, 0); //创建网格布局管理器对象
    gridLayout.setHgap(5); //设置组件的水平间距
    gridLayout.setVgap(5); //设置组件的垂直间距
    JPanel panel = new JPanel(); //获得容器对象
    panel.setLayout(gridLayout); //设置容器采用网格布局管理器
    getContentPane().add(panel, BorderLayout.CENTER);
    String[][] names = { { "1", "2", "3", "+" }, { "4", "5", "6", "-" },
        { "7", "8", "9", "x" }, { "0", "=", "÷" } };
    JButton[][] buttons = new JButton[4][4];
    for (int row = 0; row < names.length; row++) {
        for (int col = 0; col < names.length; col++) {
            buttons[row][col] = new JButton(names[row][col]); //创建按钮对象
        }
    }
}
```





Note

```
panel.add(buttons[row][col]);
```

//将按钮添加到面板中

技术要点

本实例的关键点在于 GridLayout 布局管理器的应用,通过它可以自动完成控件的布局与大小控制,否则,程序还要单独创建控制每个控件位置与大小的代码,其代码复杂度可想而知。通过 GridLayout 布局管理器,只需要指定布局的行列数量即可。下面介绍一下 GUI 如何使用 GridLayout 布局管理器。

1. 创建指定行列数量的布局管理器

可以在 GridLayout 类的构造方法中传递两个 int 类型的参数分别指定布局的行数与列数。其方法的声明如下:

```
public GridLayout(int rows, int cols)
```

参数说明

- ☑ rows: 布局的行数。
- ☑ cols: 布局的列数。

2. 设置容器的布局管理器

创建容器布局管理器后,可以把它添加到某个容器的 layout 属性中,这需要调用容器的设置布局管理器的方法来实现。其方法的声明如下:

```
public void setLayout(LayoutManager mgr)
```

参数说明

mgr: 布局管理器对象。

实例 036 通过复选框控件数组实现添加多个复选框控件

(实例位置: 配套资源\SL\05\036)

实例说明

复选框控件在 GUI 程序界面设计时经常使用,例如选择用户爱好的程序界面中要添加很多选项,这些选项如果通过 GUI 界面设计器来录入非常费时,而且生成的代码臃肿,不方便维护。不过可以通过复选框控件数组实现在窗体中添加多个复选框。本实例将通过复选框控件数组实现选择用户爱好信息的复选框,并且界面中的复选框数量可以根据指定复选框名称的字符串数组的长度来自动调节。实例的运行效果如图 5.6 所示。

实现过程

(1) 在 Eclipse 中创建项目 036,并在该项目中创建 com.mingrisoft 包。



图 5.6 通过复选框控件数组实现添加多个复选框控件



(2) 在 com.mingrisoft 包中新建 JFrame 窗体类, 名称为 CheckBoxArray。在窗体中添加一个标签显示“你的爱好有哪些:”。

(3) 编写 getPanel() 方法创建面板并在面板中通过控件数组来创建爱好复选框。其中所有复选框的文本都是由字符串数组定义的, 复选框的数量也是根据字符串数组长度确定的。关键代码如下:

```
private JPanel getPanel() {
    if (panel == null) {
        panel = new JPanel();                //创建面板对象
        panel.setLayout(new GridLayout(0, 4)); //设置网格布局管理器
        //创建控件文本数组
        String[] labels = { "足球", "篮球", "魔术", "乒乓球", "看电影", "魔兽世界", "CS 战队",
            "羽毛球", "游泳", "旅游", "爬山", "唱歌", "写博客", "动物世界", "拍照", "弹吉他",
            "读报纸", "飙车", "逛街", "逛商场", "麻将", "看书", "上网看资料", "新闻", "军事",
            "八卦", "养生", "饮茶" };
        JCheckBox[] boxes = new JCheckBox[labels.length]; //创建控件数组
        for (int i = 0; i < boxes.length; i++) {           //遍历控件数组
            boxes[i] = new JCheckBox(labels[i]);           //初始化数组中的复选框组件
            panel.add(boxes[i]);                           //把数组元素（即每个复选框）添加到面板中
        }
    }
    return panel;
}
```



Note

多学两招:

在编写一个方法时, 要考虑到方法的通用性, 尽量对方法进行抽象让方法适合更多的模块调用, 例如本实例中的 getPanel() 方法完全可以把控件标签文本数组作为方法的参数, 这样其他模块就可以为该方法传递参数创建爱好选择面板了。

技术要点

本实例中应用的主要技术是在 JPanel 面板中应用复选框控件数组添加复选框控件。应用复选框控件数组添加复选框控件的具体实现步骤如下:

(1) 定义一个字符串数组, 内容为复选框的标题文本。

(2) 创建 JCheckBox 类型的控件数组, 即复选框控件数组, 其长度与步骤(1)中创建的字符串数组的长度相同。

(3) 通过 for 循环遍历刚刚创建的复选框控件数组, 并将数组元素(即每个复选框)添加到面板中。

实例 037 使用选择排序法对数组排序

(实例位置: 配套资源\SL\05\037)

实例说明

选择排序 (Selection Sort) 是一种简单直观的排序算法。本实例演示如何使用选择排序法



对一维数组进行排序, 运行本实例, 首先单击“生成随机数组”按钮, 生成一个随机数组, 并显示在上方的文本域控件中; 然后单击“排序”按钮, 使用选择排序法对生成的一维数组进行排序, 并将排序后的一维数组显示在下方的文本域控件中。实例的运行效果如图 5.7 所示。



图 5.7 使用选择排序法对一维数组排序

实现过程

(1) 在 Eclipse 中创建项目 037, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建 JFrame 窗体类, 名称为 SelectSort。在窗体中添加两个文本域控件和“生成随机数组”、“排序”两个按钮。

(3) 编写“生成随机数组”按钮的事件处理方法, 在该方法中创建 Random 随机数对象, 初始化数组元素值时, 通过该对象为每个数组元素生成随机数。关键代码如下:

```
private int[] array = new int[10];
protected void do_button_actionPerformed(ActionEvent e) {
    Random random = new Random();           //创建随机数对象
    textArea1.setText("");                  //清空文本域
    for (int i = 0; i < array.length; i++) { //初始化数组元素
        array[i] = random.nextInt(50);      //生成 50 以内的随机数
        textArea1.append(array[i] + " ");   //把数组元素显示在文本域控件中
    }
}
```

(4) 编写“排序”按钮的事件处理方法, 在该方法中使用排序算法对生成的随机数组进行排序, 然后把排序后的数组元素显示到文本域控件中。关键代码如下:

```
protected void do_button_1_actionPerformed(ActionEvent e) {
    textArea2.setText("");                 //清空文本域
    int index;
    for (int i = 1; i < array.length; i++) {
        index = 0;
        for (int j = 1; j <= array.length - i; j++) {
            if (array[j] > array[index]) {
                index = j;                 //查找最大值
            }
        }
        //交换在 array.length-i 和 index (最大值) 位置的两个数
        int temp = array[array.length - i];
        array[array.length - i] = array[index];
        array[index] = temp;
    }
    for (int i = 0; i < array.length; i++) {
        textArea2.append(array[i] + " "); //把排序后的数组元素显示到文本域中
    }
}
```



多学两招:

利用选择排序法从数组中挑选最大值并放在数组最后,而遇到重复的相等值不会做任何处理,所以如果程序允许数组有重复值的情况,建议使用选择排序方法,因为它的数据交换次数较少,相对速度也会略微提升,这取决于数组中重复值的数量。



Note

技术要点

本实例应用的主要技术点就是选择排序算法。选择排序算法的基本思想如下:

每一趟从待排序的数据元素中选出最小(或最大)的一个元素,顺序放在已排好序的数列的最后,直到全部待排序的数据元素排完。例如,对一个一维数组采用选择排序算法进行排序的过程如图 5.8 所示。

举例:
初始数组资源 【63 4 24 1 3 15】
第一趟排序后 【15 4 24 1 3】 63
第二趟排序后 【15 4 3 1】 24 63
第三趟排序后 【1 4 3】 15 24 63
第四趟排序后 【1 3】 4 15 24 63
第五趟排序后 【1】 3 4 15 24 63

图 5.8 对一个一维数组采用选择排序算法进行排序的过程

实例 038 使用冒泡排序法对数组排序

(实例位置: 配套资源\SL\05\038)

实例说明

本实例演示如何使用冒泡排序法对一维数组进行排序。运行本实例,首先单击“生成随机数组”按钮,生成一个随机数组,并显示在上方的文本域控件中;然后单击“排序”按钮,使用冒泡排序法对生成的一维数组进行排序,并将排序过程中一维数组的变化显示在下方的文本域控件中。实例的运行效果如图 5.9 所示。



图 5.9 使用冒泡排序法对一维数组排序



实现过程



Note

- (1) 在 Eclipse 中创建项目 038，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 BubbleSort。在窗体中添加两个文本域控件和“生成随机数组”、“排序”两个按钮。
- (3) 编写“生成随机数组”按钮的事件处理方法，在该方法中创建 Random 随机数对象，初始化数组元素值时，通过该对象为每个数组元素生成随机数。关键代码如下：

```
private int[] array = new int[10];
protected void do_button_actionPerformed(ActionEvent e) {
    Random random = new Random();           //创建随机数对象
    textArea1.setText("");                  //清空文本域
    for (int i = 0; i < array.length; i++) { //初始化数组元素
        array[i] = random.nextInt(50);      //生成 50 以内的随机数
        textArea1.append(array[i] + " ");   //把数组元素显示在文本域控件中
    }
}
```

- (4) 编写“排序”按钮的事件处理方法，在该方法中使用排序算法对生成的随机数组进行排序，然后把排序后的数组元素显示到文本域控件中。关键代码如下：

```
protected void do_button_1_actionPerformed(ActionEvent e) {
    textArea2.setText("");                 //清空文本域
    for (int i = 1; i < array.length; i++) {
        //比较相邻两个元素，较大的数往后冒泡
        for (int j = 0; j < array.length - i; j++) {
            if (array[j] > array[j + 1]) {
                int temp = array[j];        //把第一个元素值保存到临时变量中
                array[j] = array[j + 1];    //把第二个元素值保存到第一个元素单元中
                array[j + 1] = temp;        //把临时变量也就是第一个元素原值保存到第二个元素中
            }
            textArea2.append(array[j] + " "); //把排序后的数组元素显示到文本域中
        }
        textArea2.append("【"];
        for (int j = array.length - i; j < array.length; j++) {
            textArea2.append(array[j] + " "); //把排序后的数组元素显示到文本域中
        }
        textArea2.append("】\n");
    }
}
```

多学两招：

实际上，初始化数组时可以省略 new 运算符和数组的长度，编译器将根据初始值的数量来自动计算数组长度，并创建数组。例如 `int[] array = {1, 2, 3, 4, 5};`。

技术要点

在实现本实例时，主要用到了冒泡排序算法。冒泡排序的基本思想是对比相邻的元素值，如果满足条件就交换元素值，把较小的元素移动到数组前面，把大的元素移动到数组后面（也就是交换两个元素的位置），这样数组元素就像气泡一样从底部上升到顶部。



冒泡算法在双层循环中实现，其中外层循环控制排序轮数，是要排序数组长度-1次。而内层循环主要是用于对比临近元素的大小，以确定是否交换位置，对比和交换次数依排序轮数而减少。例如一个拥有6个元素的数组，在排序过程中每一次循环的排序过程和结果如图5.10所示。

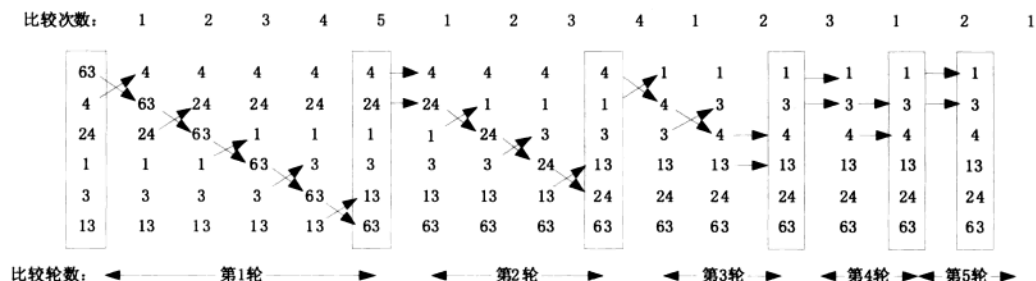


图 5.10 排序过程和结果

第一轮外层循环时把最大的元素值63移动到了最后面（相应地比63小的元素向前移动，类似气泡上升），第二轮外层循环不再对比最后一个元素值63，因为它已经确认为最大（不需要上升），应该放在最后，需要对比和移动的是其他剩余元素，这次将元素24移动到了63的前一个位置。其他循环将依此类推，继续完成排序任务。

实例 039 使用快速排序法对数组排序

（实例位置：配套资源\SL\05\039）

实例说明

快速排序（Quick Sort）是对气泡排序的一种改进，其排序速度相对较快。本实例演示如何使用快速排序法对一维数组进行排序，运行本实例，首先单击“生成随机数组”按钮，生成一个随机数组，并显示在上方的文本框中；然后单击“排序”按钮，使用快速排序法对生成的一维数组进行排序，并将排序后的一维数组显示在下方的文本框中。实例的运行效果如图5.11所示。



图 5.11 使用快速排序法对数组排序





实现过程

(1) 在 Eclipse 中创建项目 039，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 Quick Sort。在窗体中添加一个文本框、一个文本域控件和“生成随机数组”、“排序”两个按钮。

指点迷津：

由于“生成随机数组”按钮的事件处理方法与实例 038 中的“生成随机数组”按钮的事件处理方法相同，都是用于生成一个由多个随机数组组成的一维数组，所以这里不给出具体的实现方法。

(3) 编写“排序”按钮的事件处理方法，在该方法中利用快速排序算法对生成的随机数组进行排序，并将排序过程输出到文本域控件中。关键代码如下：

```
protected void do_button_1_actionPerformed(ActionEvent e) {  
    textArea2.setText(""); //清空文本域  
    quickSort(array, 0, array.length - 1); //调用快速排序算法  
}
```

(4) 编写快速排序方法 quickSort()，这个方法将被按钮的事件处理方法调用，该方法在实现快速排序的同时，把排序过程显示到文本域控件中。关键代码如下：

```
private void quickSort(int sortarray[], int lowIndex, int highIndex) {  
    int lo = lowIndex; //记录最小索引  
    int hi = highIndex; //记录最大索引  
    int mid; //记录分界点元素  
    if (highIndex > lowIndex) {  
        mid = sortarray[(lowIndex + highIndex) / 2]; //确定中间分界点元素值  
        while (lo <= hi) {  
            while ((lo < highIndex) && (sortarray[lo] < mid))  
                ++lo; //确定不大于分界元素值的最小索引  
            while ((hi > lowIndex) && (sortarray[hi] > mid))  
                --hi; //确定大于分界元素值的最大索引  
            if (lo <= hi) {  
                swap(sortarray, lo, hi); //交换两个索引的元素  
                ++lo; //递增最小索引  
                --hi; //递减最大索引  
            }  
        }  
        if (lowIndex < hi) //递归排序没有未分解元素  
            quickSort(sortarray, lowIndex, hi);  
        if (lo < highIndex) //递归排序没有未分解元素  
            quickSort(sortarray, lo, highIndex);  
    }  
}
```

(5) 由于快速排序方法中频繁地交换数组元素，而且在程序代码中出现的次数较多，所以应该把数组元素交换单独提炼为一个 swap() 方法，以实现代码重用，并且可以在该方法中掌



握排序过程并显示到文本域控件。关键代码如下：

```
private void swap(int swapArray[], int i, int j) {
    int temp = swapArray[i];           //交换数组元素
    swapArray[i] = swapArray[j];
    swapArray[j] = temp;
    for (int k = 0; k < array.length; k++) {           //把数组元素显示到文本域
        textArea2.append(array[k] + " ");
    }
    textArea2.append("\n");           //追加换行符
}
```



Note

技术要点

本实例主要用到了快速排序算法，下面对快速排序算法的实现原理进行详细讲解。

快速排序算法是对冒泡排序算法的一种改进，它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据小，然后再按此方法对这两部分数据分别进行快速排序。整个排序过程可以递归进行，以此使整个数据变成有序序列。

假设要排序的数组是 $A[1] \cdots A[N]$ ，首先任意选取一个数据（通常选用第一个数据）作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序，递归调用此过程，即可实现数组的快速排序。

使用快速排序法的排序过程如图 5.12 所示。

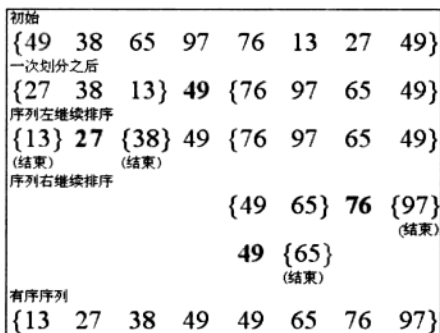


图 5.12 快速排序法的排序过程

实例 040 使用直接插入法对数组排序

(实例位置：配套资源\SL\05\040)

实例说明

本实例演示如何使用直接插入排序法对一维数组进行排序。运行本实例，首先单击“随机生成数组”按钮，生成一个随机数组，并显示在左边的文本框中；然后单击“排序”按钮，使用直接插入排序法对生成的一维数组进行排序，并将排序后的一维数组显示在右边的文本框中。实例的运行效果如图 5.13 所示。

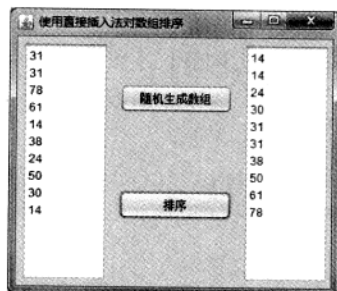


图 5.13 使用直接插入法对一维数组进行排序

实现过程

- (1) 在 Eclipse 中创建项目 040，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类，名称为 InsertSort。在窗体中添加两个文本域控件和“随机生成数组”、“排序”两个按钮。
- (3) 编写“随机生成数组”按钮的事件处理方法，在该方法中利用 Random 类的实例对象的 nextInt() 方法生成随机数，并为数组设置初始值。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {  
    Random random = new Random();           //创建随机数对象  
    textArea1.setText("");  
    for (int i = 0; i < array.length; i++) { //初始化数组元素  
        array[i] = random.nextInt(90);      //生成 50 以内的随机数  
        textArea1.append(array[i] + "\n");  //把数组元素显示在文本域控件中  
    }  
}
```

- (4) 编写“排序”按钮的事件处理方法，在该方法中使用直接插入排序算法对数组进行排序，并将排序后的数组显示到界面中。关键代码如下：

```
protected void do_button_1_actionPerformed(ActionEvent e) {  
    int tmp;                                //定义临时变量  
    int j;  
    for (int i = 1; i < array.length; i++) {  
        tmp = array[i];                    //保存临时变量  
        for (j = i - 1; j >= 0 && array[j] > tmp; j--) {  
            array[j + 1] = array[j];      //数组元素交换  
        }  
        array[j + 1] = tmp;                //在排序位置插入数据  
    }  
    textArea2.setText("");  
    for (int i = 0; i < array.length; i++) { //初始化数组元素  
        textArea2.append(array[i] + "\n"); //把数组元素显示在文本域控件中  
    }  
}
```

技术要点

本实例主要用到了直接插入排序算法。下面对直接插入排序算法的实现原理进行详细讲解。插入排序是将一个记录插入到有序序列中，使得到的新序列仍然有序。插入排序算法的思



想是：将 n 个有序数存放在数组 a 中，要插入的数为 x ，首先确定 x 插在数组中的位置 p ，数组中 p 之后的元素都向后移一个位置，空出 $a(p)$ ，将 x 放入 $a(p)$ 。这样即可实现插入后数列仍然有序。

使用插入排序法排序的过程如图 5.14 所示。

	0	1	2	3	4	5	6	
k=2	20	40	90	30	80	70	50	n=7
	0	1	2	3	4	5	6	
k=3	20	30	40	90	80	70	50	n=7
	0	1	2	3	4	5	6	
k=4	20	30	40	80	90	70	50	n=7
	0	1	2	3	4	5	6	
k=5	20	30	40	70	80	90	50	n=7
	0	1	2	3	4	5	6	
k=6	20	30	40	50	70	80	90	n=7

图 5.14 直接插入排序法排序过程

实例 041 使用 sort() 方法对数组排序

(实例位置：配套资源\SL\05\041)

实例说明

实际开发项目时，经常需要在程序中对数组进行排序，而且，在计算机常用算法中，也提供了很多种对数组进行排序的算法，如冒泡排序法、直接插入法和选择排序法等，但在使用排序算法时，开发人员必须手动编写一堆代码，而且有的实现起来比较麻烦。Java 的 `Arrays` 类提供了一个 `sort()` 方法，使用该方法可以很方便地对各种数组进行排序，大大降低了数组排序的难度，本实例就将使用该方法对数组进行快速排序。实例的运行效果如图 5.15 所示。

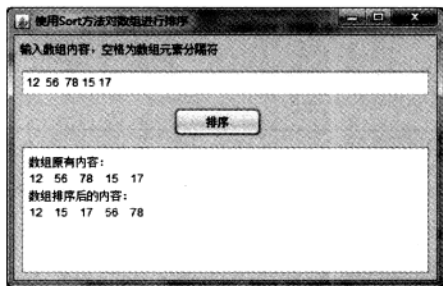


图 5.15 使用 `sort()` 方法对数组进行快速排序

实现过程

- (1) 在 Eclipse 中创建项目 041，并在该项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中新建 `JFrame` 窗体类，名称为 `InsertSort`。在窗体中添加一个文本框、一个文本域控件和一个“排序”按钮。





(3) 为“排序”按钮编写事件处理方法,在该方法中要接收用户的输入字符串,并以字符串中的空格为分割符,将字符串转换为字符串数组,再把字符串数组转换为整数数组,然后调用 Arrays 类的 sort() 方法对其进行排序,并显示到窗体中。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = arrayField.getText();           //获取用户输入
    text=text.trim();                             //去除首尾空格
    if("".equals(text)){
        JOptionPane.showMessageDialog(null,"请输入要排序的数组内容");
        return;
    }
    String[] arrayStr = text.split(" {1,}");      //拆分输入为数组
    int[] array = new int[arrayStr.length];       //创建整数类型数组
    sortArea.setText("数组原有内容: \n");
    for (String string : arrayStr) {              //输出原有数组内容
        sortArea.append(string + " ");
    }
    for (int i = 0; i < array.length; i++) {      //初始化整型数组
        array[i] = Integer.parseInt(arrayStr[i]);
    }
    sortArea.append("\n");
    Arrays.sort(array);                           //使用 sort() 方法对整型数组进行排序
    sortArea.append("数组排序后的内容: \n");
    for (int value : array) {                     //输出排序后的数组内容
        sortArea.append(value + " ");
    }
}
```

(4) 编写文本框的按键事件处理方法,通过该方法的编写来限制文本框可输入的字符,当用户按下非数字与空格字符时,可取消本次输入的有效性。关键代码如下:

```
protected void do_arrayField_keyPressed(KeyEvent e) {
    char key = e.getKeyChar();                   //获取用户按键字符
    String mask = "0123456789 " + (char) 8;     //定义规范化字符模板
    if (mask.indexOf(key) == -1) {               //判断按键字符是否属于规范化字符范围
        e.consume();                             //取消非规范化字符的输入有效性
    }
}
```

指点迷津:

Arrays 类提供了创建、操作、搜索和排序数组的方法。在程序开发中有效利用 Arrays 类的各种方法来完成数组操作将大幅度提升程序开发效率,并且 Arrays 类的方法是经过测试的,可以减少程序开发中错误代码的出现。

技术要点

本实例在对数组进行快速排序时,主要用到了 Arrays 类的 sort() 方法,下面对其进行详细讲解。

Arrays 类位于 java.util 包,它是数组的一个工具类,包含很多方法,其中 sort() 方法就是 Arrays 类提供的对数组进行排序的方法,它有很多重载格式,可以接收任何数据类型的数组并



执行不同类型的排序。本实例使用 sort()方法的 int 参数类型的重载实现,其方法声明如下:

```
public static void sort(int[] array)
```

参数说明

array: 要排序的 int 类型的一维数组。

实例 042 反转数组中元素的顺序

(实例位置: 配套资源\SL\05\042)

实例说明

反转数组就是以相反的顺序把原有数组的内容重新排序。本实例在 GUI 窗体中演示反转数组中元素的顺序。运行本实例,首先在界面的文本框内输入数组元素,每个元素使用空格分隔,然后单击界面上的“反转数组元素”按钮,程序将对数组进行反转运算,并把运算过程中对数组的改变显示在窗体中。实例的运行效果如图 5.16 所示。

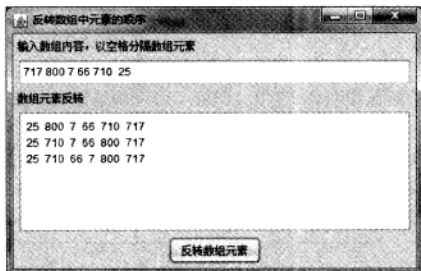


图 5.16 反转数组中元素的顺序

实现过程

- (1) 在 Eclipse 中创建项目 042,并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中新建 JFrame 窗体类,名称为 ReverseSort。在窗体中添加一个文本框、一个文本域控件和“反转数组元素”按钮。
- (3) 编写“反转数组元素”按钮的事件处理方法,在该方法中获取用户输入的字符串,然后以空格为分隔符,把字符串拆分成字符串数组,最后再通过反转算法排序数组的同时,把反转过程中数组的变化输出到文本域控件中。关键代码如下:

```
protected void do_button_1_actionPerformed(ActionEvent e) {
    String inText = textField.getText();           //获取用户输入
    String[] array = inText.split(" {1,}");        //把字符串分割为数组
    int len = array.length;                        //获取数组长度
    textArea.setText("");                          //清空文本域控件内容
    for (int i = 0; i < len / 2; i++) {            //反转数组元素
        String temp = array[i];
        array[i] = array[len - 1 - i];
        array[len - 1 - i] = temp;
        for (String string : array) {              //在文本域中显示数组排序过程
            textArea.append(string + " ");
        }
    }
}
```





Note

脚下留神:

“反转”并不等于“倒序排序”。“反转”只是将数组元素的顺序进行颠倒，并不对其执行排序操作；而“倒序排序”则是对数组中的元素进行从大到小的排序。

技术要点

本实例的核心技术是使用了数组反转算法。反转算法的基本思想比较简单，也很好理解，思路就是：把数组最后一个元素与第一个元素替换，倒数第二个元素与第二个元素替换，依此类推，直到把所有数组元素反转替换。

反转数组元素是对数组两边的元素进行替换，所以只需要循环数组长度的半数。例如，对长度为 6 的一维数组进行反转，只需要循环 $6/2$ ，即 3 次，具体的反转过程如图 5.17 所示。

举例：

初始数组资源 【10 20 30 40 50 60】

第一趟排序后	60	【20	30	40	50】	10
第二趟排序后	60	50	【30	40】	20	10
第三趟排序后	60	50	40	30	20	10

图 5.17 对长度为 6 的数组进行反转的过程

第 6 章

面向对象入门

本章读者可以学到如下实例：

- » 实例 043 自定义图书类
- » 实例 044 温度单位转换工具
- » 实例 045 成员变量的默认初始化值
- » 实例 046 单例模式的应用
- » 实例 047 汉诺塔问题求解
- » 实例 048 编写同名的方法
- » 实例 049 构造方法的应用
- » 实例 050 统计图书的销售量
- » 实例 051 两只完全相同的宠物
- » 实例 052 重新计算对象的哈希码
- » 实例 053 使用字符串输出对象
- » 实例 054 Java 对象的假克隆
- » 实例 055 Java 对象的浅克隆
- » 实例 056 Java 对象的深克隆
- » 实例 057 序列化与对象克隆
- » 实例 058 深克隆效率的比较



实例 043 自定义图书类

(实例位置: 配套资源\SL\06\043)

实例说明

在使用 Java 语言进行开发时,时刻要以面向对象的思想考虑问题。面向对象的基础就是类,本实例将演示如何自定义类,它用于表示图书。实例的运行效果如图 6.1 所示。

实现过程

(1) 在 Eclipse 中创建项目 043,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 Book。在 Book 类中定义 3 个成员变量,分别表示书名、作者和价格,同时提供构造方法和成员方法来修改成员变量。关键代码如下:

```
public class Book {  
    private String title;           //定义书名  
    private String author;         //定义作者  
    private double price;          //定义价格  
    public Book(String title, String author, double price) {           //利用构造方法初始化域  
        this.title = title;  
        this.author = author;  
        this.price = price;  
    }  
    public String getTitle() {           //获得书名  
        return title;  
    }  
    public String getAuthor() {           //获得作者  
        return author;  
    }  
    public double getPrice() {           //获得价格  
        return price;  
    }  
}
```

(3) 在 com.mingrisoft 包中再创建类文件,名称为 Test。在该类的 main()方法中,创建一个 Book 对象并输出其属性。关键代码如下:

```
public class Test {  
    public static void main(String[] args) {  
        Book book = new Book("《Java 从入门到精通 (第 2 版)》", "明日科技", 59.8); //创建对象  
        System.out.println("书名: " + book.getTitle());           //输出书名  
        System.out.println("作者: " + book.getAuthor());           //输出作者  
        System.out.println("价格: " + book.getPrice() + "元");       //输出价格  
    }  
}
```

(4) 编译并运行 Test 文件,在 Eclipse 控制台上的输出效果如图 6.1 所示。

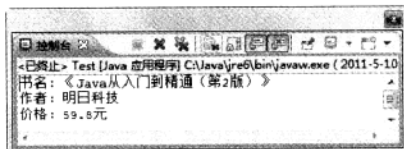


图 6.1 输出自定义的图书类对象



技术要点

在 Java 中,使用 `class` 关键字来定义类。一个类,通常包括域和方法两部分。域表示对象的状态,方法表示对象的行为。使用 `new` 关键字可以创建类的对象。通常情况下,不同的对象状态是有差别的。可以使用构造方法在创建对象时就设置状态,也可以使用方法在创建对象后修改对象的状态。

实例 044 温度单位转换工具

(实例位置:配套资源\SL\06\044)

实例说明

目前有两种广泛使用的温度单位,即摄氏度和华氏度。在标准大气压下,沸腾的水可以表示成 100 摄氏度和 212 华氏度。本实例将编写一个简单的温度单位转换工具,它可以将用户输入的摄氏度转换成华氏度,其运行效果如图 6.2 所示。

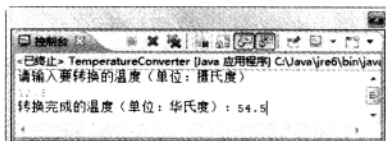


图 6.2 将输入的摄氏度温度转换成华氏度

实现过程

(1) 在 Eclipse 中创建项目 044,并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件,名称为 `CelsiusConverter`。在 `CelsiusConverter` 类中定义了两个方法,`getFahrenheit()`方法用于将摄氏温度转换成华氏温度,`main()`方法用于测试。关键代码如下:

```
public class CelsiusConverter {
    public double getFahrenheit(double celsius) {
        double fahrenheit = 1.8 * celsius + 32;           //计算华氏温度
        return fahrenheit;                               //返回华氏温度
    }
    public static void main(String[] args) {
        System.out.println("请输入要转换的温度 (单位: 摄氏度)");
        Scanner in = new Scanner(System.in);             //获得控制台输入
        double celsius = in.nextDouble();                //获得用户输入的摄氏温度
        CelsiusConverter converter = new CelsiusConverter(); //创建类的对象
        double fahrenheit = converter.getFahrenheit(celsius); //转换温度为华氏度
        System.out.println("转换完成的温度 (单位: 华氏度): " + fahrenheit); //输出转换结果
    }
}
```

技术要点

本实例中定义了一个普通方法 `getFahrenheit()`,它完成了将摄氏度转换成华氏度的功能。在 Java 中使用普通方法需要先创建一个方法所在类的对象,然后通过这个对象调用这个方法。这点与静态方法有明显不同。后者可以直接使用类名调用。



实例 045 成员变量的默认初始化值

(实例位置: 配套资源\SL\06\045)

实例说明

一般来说, 变量在使用之前需要对其进行初始化。Java 中的成员变量有些特殊, 虚拟机会自动为其进行初始化。为了了解默认的初始化是否符合编程的需求, 本实例将输入 Java 中各种类型的成员变量的初始化值, 其运行效果如图 6.3 所示。

实现过程

(1) 在 Eclipse 中创建项目 045, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Initialization。在 Initialization 方法中定义了 8 种基本类型的成员变量和一个引用类型的成员变量, 在未初始化的情况下, 直接在 main() 方法中将其输出。关键代码如下:

```
public class Initialization {  
    private byte b;  
    private short s;  
    private int i;  
    private long l;  
    private float f;  
    private double d;  
    private boolean bl;  
    private char c;  
    private String string;  
    public static void main(String[] args) {  
        Initialization init = new Initialization();  
        System.out.println("比特类型的初始值: " + init.b);  
        System.out.println("短整型类型的初始值: " + init.s);  
        System.out.println("整型类型的初始值: " + init.i);  
        System.out.println("长整型类型的初始值: " + init.l);  
        System.out.println("单精度浮点类型的初始值: " + init.f);  
        System.out.println("双精度浮点类型的初始值: " + init.d);  
        System.out.println("布尔类型的初始值: " + init.bl);  
        System.out.println("字符类型的初始值: " + init.c);  
        System.out.println("引用类型的初始值: " + init.string);  
    }  
}
```

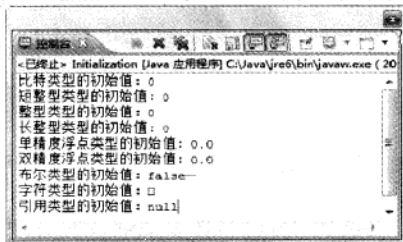


图 6.3 成员变量的默认初始化值

脚下留神:

对于引用类型的变量, 在使用之前需要进行初始化, 否则会抛出 NullPointerException 异常。



技术要点

除了成员变量, Java 中还可以定义局部变量。局部变量在使用之前必须要对其进行初始化, 虚拟机不会进行初始化工作; 否则会报告错误。

实例 046 单例模式的应用

(实例位置: 配套资源\SL\06\046)



Note

实例说明

中国历史上的皇帝通常仅有一人。为了保障其唯一性, 古人采用增加“防伪标识”的办法, 如玉玺。更简单的方法是限制皇帝的创建。本实例使用单例模式来保证皇帝的唯一性。实例的运行效果如图 6.4 所示。

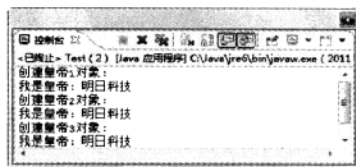


图 6.4 使用单例模式保证皇帝的唯一性

实现过程

(1) 在 Eclipse 中创建项目 046, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Emperor。在 Emperor 类中, 将构造方法设置成私有, 并提供一个静态方法用于获得该类的实例。关键代码如下:

```
public class Emperor {  
    private static Emperor emperor = null;           //声明一个 Emperor 类的引用  
    private Emperor() {                             //将构造方法私有  
    }  
    public static Emperor getInstance() {            //实例化引用  
        if (emperor == null) {  
            emperor = new Emperor();  
        }  
        return emperor;  
    }  
    public void getName() {                          //使用普通方法输出皇帝的名字  
        System.out.println("我是皇帝: 明日科技");  
    }  
}
```

(3) 在 com.mingrisoft 包中再创建类文件, 名称为 Test。在该类的 main()方法中, 创建 3 个 Emperor 对象并输出了其名字。关键代码如下:

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println("创建皇帝 1 对象: ");  
        Emperor emperor1 = Emperor.getInstance(); //创建皇帝对象  
        emperor1.getName();                       //输出皇帝的名字  
        System.out.println("创建皇帝 2 对象: ");  
        Emperor emperor2 = Emperor.getInstance(); //创建皇帝对象  
        emperor2.getName();                       //输出皇帝的名字  
        System.out.println("创建皇帝 3 对象: ");  
    }  
}
```



```
Emperor emperor3 = Emperor.getInstance();
emperor3.getName();
```

```
//创建皇帝对象
//输出皇帝的名字
```

技术要点

单例模式的特点在于仅能获得一个对象。为了防止其他用户创建对象，需要将构造方法设置成 `private` 的，然后提供一个静态方法，该方法返回这个类的对象。

实例 047 汉诺塔问题求解

(实例位置: 配套资源\SL\06\047)

实例说明

汉诺塔问题的描述如下: 有 A、B 和 C 3 根柱子, 在 A 上从下往上按照从小到大的顺序放着 64 个圆盘, 以 B 为中介, 把盘子全部移动到 C 上。移动过程中, 要求任意盘子的下面要么没有盘子, 要么只能有比它大的盘子。本实例将演示如何求解 3 阶汉诺塔问题, 其运行效果如图 6.5 所示。

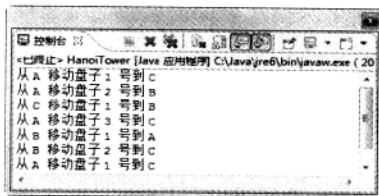


图 6.5 3 阶汉诺塔解决步骤

实现过程

- (1) 在 Eclipse 中创建项目 047, 并在该项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中创建类文件, 名称为 `HanoiTower`。在 `HanoiTower` 类中定义了一个 `moveDish()` 方法, 它使用递归算法完成汉诺塔问题的求解; 一个 `main()` 方法用于测试。关键代码如下:

```
public class HanoiTower {
    public static void moveDish(int level, char from, char inter, char to) {
        if (level == 1) {
            System.out.println("从 " + from + " 移动盘子 1 号到 " + to);
        } else {
            moveDish(level - 1, from, to, inter);
            System.out.println("从 " + from + " 移动盘子 " + level + " 号到 " + to);
            moveDish(level - 1, inter, from, to);
        }
    }

    public static void main(String[] args) {
        int nDisks = 3;
        moveDish(nDisks, 'A', 'B', 'C');
    }
}
```

技术要点

为了将 N 个盘子从 A 移动到 C, 需要先将第 N 个盘子上面的 $N-1$ 个盘子移动到 B 上, 这



样才能将第 N 个盘子移动到 C 上。同理,为了将第 N-1 个盘子从 B 移动到 C 上,需要将 N-2 个盘子移动到 A 上,这样才能将第 N-1 个盘子移动到 C 上。通过递归就可以实现汉诺塔问题的求解。

实例 048 编写同名的方法

(实例位置: 配套资源\SL\06\048)

实例说明

在 C 语言中,如果要在一个文件内定义两个同名的方法会报告错误。这对于编写具有相似功能不同参数的方法非常不利。在 Java 中,可以利用重载技术来完成这个需求。本实例将在一个类中定义两个同名的方法,它们的参数不同,其运行效果如图 6.6 所示。

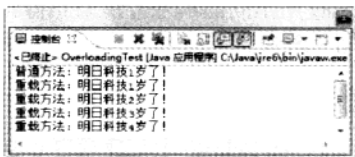


图 6.6 调用普通方法和重载方法的输出

实现过程

(1) 在 Eclipse 中创建项目 048,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 OverloadingTest。在 OverloadingTest 类中,定义了两个同名方法 info(),但是参数不同;一个 main()方法用于测试。关键代码如下:

```
public class OverloadingTest {  
    public void info() {                                //定义没有参数的 info()方法  
        System.out.println("普通方法: 明日科技 1 岁了!");  
    }  
    public void info(int age) {                          //定义包含整型参数的 info()方法  
        System.out.println("重载方法: 明日科技" + age + "岁了!");  
    }  
    public static void main(String[] args) {  
        OverloadingTest ot = new OverloadingTest();    //创建 OverloadingTest 类对象  
        ot.info();                                       //测试无参数 info()方法  
        for (int i = 1; i < 5; i++) {                   //测试有参数 info()方法  
            ot.info(i);  
        }  
    }  
}
```

技术要点

在 Java 中,可以通过重载 (overloading) 来减少方法名称的个数。当对象在调用方法时,可以根据方法参数的不同来确定执行哪个方法。方法参数的不同包括参数类型不同、参数个数不同和参数顺序不同。需要注意的是,不能通过方法的返回值来区分方法,即不能有两个方法签名相同但返回值不同的方法。

指点迷津:

方法名称和方法参数统称为方法签名。



实例 049 构造方法的应用

(实例位置: 配套资源\SL\06\049)

实例说明

Java 程序的各种功能是通过对象调用相关方法完成的, 因此必须先获得对象。使用构造方法来获得对象是一种常见方式。构造方法也支持重载, 本实例将演示使用不同的构造方法来获得对象, 其运行效果如图 6.7 所示。

实现过程

(1) 在 Eclipse 中创建项目 049, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Person。在 Person 类中, 定义了 3 个成员变量分别表示人的姓名、性别和年龄。此外还提供了构造方法和普通方法来修改成员变量的值。关键代码如下:

```
public class Person {  
    private String name;           //定义姓名  
    private String gender;         //定义性别  
    private int age;               //定义年龄  
    public Person() {              //定义没有参数的构造方法  
        System.out.println("使用无参构造方法创建对象");  
    }  
    public Person(String name, String gender, int age) { //利用构造方法初始化域  
        this.name = name;  
        this.gender = gender;  
        this.age = age;  
        System.out.println("使用有参构造方法创建对象");  
    }  
    public String getName() {       //获得姓名  
        return name;  
    }  
    public String getGender() {     //获得性别  
        return gender;  
    }  
    public int getAge() {           //获得年龄  
        return age;  
    }  
}
```

(3) 在 com.mingrisoft 包中再创建类文件, 名称为 Test。在该类的 main() 方法中, 创建了两个 Person 对象并输出了其属性。代码如下:

```
public class Test {  
    public static void main(String[] args) {
```

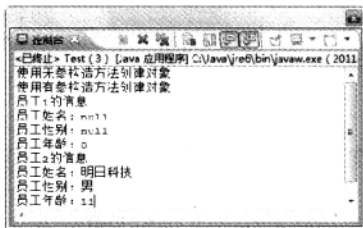


图 6.7 输出创建的两个对象



```

Person person1 = new Person();           //创建对象
Person person2 = new Person("明日科技", "男", 11); //创建对象
System.out.println("员工 1 的信息");
System.out.println("员工姓名: " + person1.getName()); //输出姓名
System.out.println("员工性别: " + person1.getGender()); //输出性别
System.out.println("员工年龄: " + person1.getAge()); //输出年龄
System.out.println("员工 2 的信息");
System.out.println("员工姓名: " + person2.getName()); //输出姓名
System.out.println("员工性别: " + person2.getGender()); //输出性别
System.out.println("员工年龄: " + person2.getAge()); //输出年龄
}

```



Note

技术要点

构造方法是一种特殊类型的方法，它可以用来实现成员变量的初始化操作。在声明时必须遵守如下规定：

- ☑ 构造方法的名称与类名相同。
- ☑ 构造方法没有返回值，并不是返回 void。
- ☑ 构造方法总是与 new 操作符一起使用，即不能用对象调用构造方法。

此外，在构造方法中，还可以使用 this 来调用其他构造方法，使用 super 调用超类构造方法。

实例 050 统计图书的销售量

(实例位置：配套资源\SL\06\050)

实例说明

在商品（类的实例）的销售过程中，需要对销量进行统计。本实例将在类的构造方法中增加计数器来实现该功能，其运行效果如图 6.8 所示。

实现过程

(1) 在 Eclipse 中创建项目 050，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Book。在 Book 类中，定义了一个静态的成员变量用于保存实例化的次数。在构造方法中实现了计数器的功能，关键代码如下：

```

public class Book {
    private static int counter = 0;           //定义一个计数器
    public Book(String title) {
        System.out.println("售出图书: " + title); //输出书名
        counter++;                             //计数器加 1
    }
    public static int getCounter() {           //获得计数器的结果
        return counter;
    }
}

```

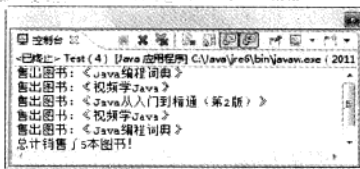


图 6.8 输出图书销售信息



(3) 在 com.mingrisoft 包中再创建类文件, 名称为 Test。在 Test 类中定义 main() 方法用于创建 Book 类对象并输出创建对象的个数。关键代码如下:

```
public class Test {
    public static void main(String[] args) {
        //创建书名数组
        String[] titles = { "《Java 从入门到精通 (第 2 版)》", "《Java 编程词典》", "《视频学 Java》" };
        for (int i = 0; i < 5; i++) {
            new Book(titles[new Random().nextInt(3)]); //利用书名数组创建 Book 对象
        }
        System.out.println("总计销售了" + Book.getCounter() + "本图书!"); //输出创建对象的个数
    }
}
```



Note

指点迷津:

new Random().nextInt(3) 代码用于获得 0~2 的整数。

技术要点

对于非 static 成员变量, 不同的对象可以对其任意修改而不会对其他对象产生影响。对于 static 成员变量, 则是所有对象所共享的。任何一个对象的修改会影响其他对象。

实例 051 两只完全相同的宠物

(实例位置: 配套资源\SL\06\051)

实例说明

由于生命的复杂性, 寻找两只完全相同的宠物 (类的对象) 是不可能的。在 Java 中, 可以通过比较对象的成员变量来判断对象是否相同。本实例将创建 3 只宠物猫, 通过比较它们的名字、年龄、重量和颜色属性来看它们是否相同。实例的运行效果如图 6.9 所示。

实现过程

(1) 在 Eclipse 中创建项目 051, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Cat。在 Cat 类中定义 4 个成员变量分别表示猫咪的名字、年龄、重量和颜色, 并提供构造方法来设置这些属性值。本类的重点内容在于重写 equals() 和 toString() 方法。重写 equals() 方法可以比较两个对象是否相同, 重写 toString() 方法可以直接输出对象。关键代码如下:

```
public class Cat {
    private String name; //表示猫咪的名字
    private int age; //表示猫咪的年龄
    private double weight; //表示猫咪的重量
```



图 6.9 输出对象信息和比较结果



Note

```

private Color color; //表示猫咪的颜色
public Cat(String name, int age, double weight, Color color) { //初始化猫咪的属性
    this.name = name;
    this.age = age;
    this.weight = weight;
    this.color = color;
}
@Override
public boolean equals(Object obj) { //利用属性来判断猫咪是否相同
    if (this == obj) { //如果两个猫咪是同一个对象则相同
        return true;
    }
    if (obj == null) { //如果两个猫咪有一个为 null 则不同
        return false;
    }
    if (getClass() != obj.getClass()) { //如果两个猫咪的类型不同则不同
        return false;
    }
    Cat cat = (Cat) obj;
    return name.equals(cat.name) && (age == cat.age)
        && (weight == cat.weight) && (color.equals(cat.color)); //比较猫咪的属性
}
@Override
public String toString() { //重写 toString()方法
    StringBuilder sb = new StringBuilder();
    sb.append("名字: " + name + "\n");
    sb.append("年龄: " + age + "\n");
    sb.append("重量: " + weight + "\n");
    sb.append("颜色: " + color + "\n");
    return sb.toString();
}
}

```

(3) 在 com.mingrisoft 包中再创建类文件, 名称为 Test。在该类的 main()方法中创建 3 只猫咪, 并为其初始化, 然后输出猫咪对象和比较的结果。关键代码如下:

```

public class Test {
    public static void main(String[] args) {
        Cat cat1 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 1 号
        Cat cat2 = new Cat("C++", 12, 21, Color.WHITE); //创建猫咪 2 号
        Cat cat3 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 3 号
        System.out.println("猫咪 1 号: " + cat1); //输出猫咪 1 号
        System.out.println("猫咪 2 号: " + cat2); //输出猫咪 2 号
        System.out.println("猫咪 3 号: " + cat3); //输出猫咪 3 号
        System.out.println("猫咪 1 号是否与猫咪 2 号相同: " + cat1.equals(cat2)); //比较是否相同
        System.out.println("猫咪 1 号是否与猫咪 3 号相同: " + cat1.equals(cat3)); //比较是否相同
    }
}

```

脚下留神:

本实例为了简单, 没有重写 hashCode()方法, 在实际编程中, 必须同时重写该方法。



技术要点

Java 中的类都是 Object 类的直接或间接子类。在 Object 类中定义了 equals() 方法用于比较类的两个对象是否相同。该方法的默认实现仅能比较两个对象是否是同一个对象。通常在定义类时推荐重写这个方法。



Note

实例 052 重新计算对象的哈希码

(实例位置: 配套资源\SL\06\052)

实例说明

Java 中创建的对象是保存在堆中的, 为了提高查找的速度而使用了散列查找。散列查找的基本思想是定义一个键来映射对象所在的内存地址。当需要查找对象时, 直接查找键即可, 这样就不用遍历整个堆来查找对象了。本实例将查看不同对象的散列值, 实例的运行效果如图 6.10 所示。

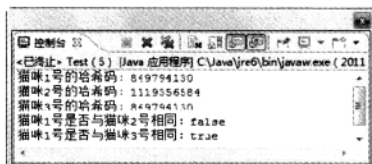


图 6.10 输出对象哈希码和比较结果

实现过程

(1) 在 Eclipse 中创建项目 052, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Cat。在 Cat 类中定义 4 个成员变量分别表示猫咪的名字、年龄、重量和颜色, 并提供构造方法来设置这些属性值。本类的重点内容在于重写 equals() 和 hashCode() 方法。重写 equals() 方法可以比较两个对象是否相同, 重写 hashCode() 方法可以让相同的对象保存在相同的位置。关键代码如下:

```
public class Cat {  
    private String name;           //表示猫咪的名字  
    private int age;               //表示猫咪的年龄  
    private double weight;         //表示猫咪的重量  
    private Color color;           //表示猫咪的颜色  
    public Cat(String name, int age, double weight, Color color) { //初始化猫咪的属性  
        this.name = name;  
        this.age = age;  
        this.weight = weight;  
        this.color = color;  
    }  
    @Override  
    public boolean equals(Object obj) { //利用属性来判断猫咪是否相同  
        if (this == obj) { //如果两个猫咪是同一个对象则相同  
            return true;  
        }  
        if (obj == null) { //如果两个猫咪有一个为 null 则不同  
            return false;  
        }  
        if (getClass() != obj.getClass()) { //如果两个猫咪的类型不同则不同  
            return false;  
        }  
    }  
}
```




```

    }
    Cat cat = (Cat) obj;
    return name.equals(cat.name) && (age == cat.age)
           && (weight == cat.weight) && (color.equals(cat.color)); //比较猫咪的属性
}
@Override
public int hashCode() {                //重写 hashCode()方法
    return 7 * name.hashCode() + 11 * new Integer(age).hashCode() + 13
           * new Double(weight).hashCode() + 17 * color.hashCode();
}
}

```



Note

(3) 在 com.mingrisoft 包中再创建类文件, 名称为 Test。在该类的 main() 方法中创建 3 只猫咪, 并为其初始化, 然后输出猫咪的哈希码和比较的结果。关键代码如下:

```

public class Test {
    public static void main(String[] args) {
        Cat cat1 = new Cat("Java", 12, 21, Color.BLACK);           //创建猫咪 1 号
        Cat cat2 = new Cat("C++", 12, 21, Color.WHITE);            //创建猫咪 2 号
        Cat cat3 = new Cat("Java", 12, 21, Color.BLACK);           //创建猫咪 3 号
        System.out.println("猫咪 1 号的哈希码: " + cat1.hashCode()); //输出猫咪 1 号的哈希码
        System.out.println("猫咪 2 号的哈希码: " + cat2.hashCode()); //输出猫咪 2 号的哈希码
        System.out.println("猫咪 3 号的哈希码: " + cat3.hashCode()); //输出猫咪 3 号的哈希码
        System.out.println("猫咪 1 号是否与猫咪 2 号相同: " + cat1.equals(cat2)); //比较是否相同
        System.out.println("猫咪 1 号是否与猫咪 3 号相同: " + cat1.equals(cat3)); //比较是否相同
    }
}

```

技术要点

一种简单的计算哈希码的方式是将重写 equals() 方法时使用到的成员变量, 乘以不同的质数然后求和, 以此作为新的哈希码。

实例 053 使用字符串输出对象

(实例位置: 配套资源\SL\06\053)

实例说明

在日常开发中, 经常需要输出对象的描述信息, 此时可以通过重写 toString() 方法来完成。本实例将演示其实现过程, 其运行效果如图 6.11 所示。

实现过程

(1) 在 Eclipse 中创建项目 053, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Cat。在 Cat 类中定义 4 个成员变量分别表示猫咪的名字、年龄、重量和颜色, 并提供构造方法来



图 6.11 输出对象信息



设置这些属性值。本类的重点内容在于重写 toString()方法,这样可以在输出对象时提供有意义的信息。关键代码如下:



Note

```
public class Cat {
    private String name;           //表示猫咪的名字
    private int age;               //表示猫咪的年龄
    private double weight;         //表示猫咪的重量
    private Color color;           //表示猫咪的颜色
    public Cat(String name, int age, double weight, Color color) { //初始化猫咪的属性
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.color = color;
    }
    @Override
    public String toString() {      //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append("名字: " + name + "\n");
        sb.append("年龄: " + age + "\n");
        sb.append("重量: " + weight + "\n");
        sb.append("颜色: " + color + "\n");
        return sb.toString();
    }
}
```

(3) 在 com.mingrisoft 包中再创建类文件, 名称为 Test。在该类的 main()方法中创建 3 只猫咪, 并为其初始化, 然后输出猫咪对象。关键代码如下:

```
public class Test {
    public static void main(String[] args) {
        Cat cat1 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 1 号
        Cat cat2 = new Cat("C++", 12, 21, Color.WHITE); //创建猫咪 2 号
        Cat cat3 = new Cat("Java", 12, 21, Color.BLACK); //创建猫咪 3 号
        System.out.println("猫咪 1 号: " + cat1); //输出猫咪 1 号
        System.out.println("猫咪 2 号: " + cat2); //输出猫咪 2 号
        System.out.println("猫咪 3 号: " + cat3); //输出猫咪 3 号
    }
}
```

技术要点

重写 toString()方法时, 为了给用户提供更多的信息, 通常会包括类中成员变量和成员方法的介绍等。本实例中类比较简单, 没有包含方法, 因此简单地返回了各个成员变量的含义。为了让 toString()方法可以有更好的通用性, 可以使用反射来获得域和方法的信息。

实例 054 Java 对象的假克隆

(实例位置: 配套资源\SL\06\054)

实例说明

对象的克隆是 Java 中的一项高级技术, 它可以根据给定的对象, 获得与其完全相同的另一



个对象。在克隆过程中，有些注意事项。本实例将演示常见的错误，其运行效果如图 6.12 所示。

实现过程

(1) 在 Eclipse 中创建项目 054，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Employee。在 Employee 类中，定义两个成员变量分别表示员工的名字和年龄，并提供了 get()和 set()方法用于获得和设置变量值。重写 toString()方法来方便对象的输出。关键代码如下：

```
public class Employee {
    private String name;           //表示员工的名字
    private int age;               //表示员工的年龄
    //省略 get()和 set()方法的相关代码
    @Override
    public String toString() {      //重写 toString()方法
        return "姓名: " + name + ", 年龄: " + age;
    }
}
```

(3) 在 com.mingrisoft 包中再创建类文件，名称为 Test。在该类中，首先创建 employee1 对象并修改其属性，再利用“=”将其赋值给 employee2，最后分别输出两个对象。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        System.out.println("克隆之前:");
        Employee employee1 = new Employee();           //创建 Employee 对象 employee1
        employee1.setName("张 XX");                     //为 employee1 设置姓名
        employee1.setAge(30);                           //为 employee1 设置年龄
        System.out.println("员工 1 的信息:");
        System.out.println(employee1);                  //输出 employee1 的信息
        System.out.println("克隆之后:");
        Employee employee2 = employee1;                 //将 employee1 赋值给 employee2
        employee2.setName("李 XX");                     //为 employee2 设置姓名
        employee2.setAge(24);                           //为 employee2 设置年龄
        System.out.println("员工 1 的信息:");
        System.out.println(employee1);                  //输出 employee1 的信息
        System.out.println("员工 2 的信息:");
        System.out.println(employee2);                  //输出 employee2 的信息
    }
}
```

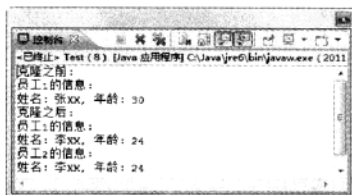


图 6.12 输出克隆前后对象信息



Note

技术要点

Java 中，对于基本类型可以使用“=”来进行克隆，此时两个变量除了相等是没有任何关系的。而对于引用类型却不能简单地使用“=”进行克隆，这与 Java 的内存空间使用有关。Java 将内存空间分成两块，即栈和堆。在栈中保存基本类型和引用变量；在堆中保存对象。对于引用变量而言，使用“=”将修改引用，而不是复制堆中的对象。此时两个引用变量将指向同一个对象。因此，如果一个变量对其进行修改则会改变另一个变量。



实例 055 Java 对象的浅克隆

(实例位置: 配套资源\SL\06\055)

实例说明

在克隆对象时, 如果对象的成员变量是基本类型, 则使用浅克隆即可完成。如果对象的成员变量包括可变引用类型, 则需要使用深克隆。本实例将演示如何使用浅克隆, 其运行效果如图 6.13 所示。

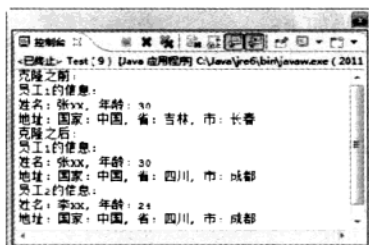


图 6.13 输出克隆前后对象信息

多学两招:

如果引用类型是不可变的, 如 `String` 类的对象, 则不必进行深克隆。

实现过程

(1) 在 Eclipse 中创建项目 055, 并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件, 名称为 `Address`。在 `Address` 类中定义 3 个成员变量, 分别表示地址中的国家、省和市, 使用构造方法可以对它们进行赋值。此外, 可以使用 `get()` 和 `set()` 方法来获取和修改成员变量的值, 最后重写 `toString()` 方法。关键代码如下:

```
public class Address {  
    private String state;  
    private String province;  
    private String city;  
    public Address(String state, String province, String city) {  
        this.state = state;  
        this.province = province;  
        this.city = city;  
    }  
    //省略 get()和 set()方法  
    @Override  
    public String toString() {  
        StringBuilder sb = new StringBuilder();  
        sb.append("国家: " + state + ", ");  
        sb.append("省: " + province + ", ");  
        sb.append("市: " + city);  
        return sb.toString();  
    }  
}
```

//表示员工所在的国家
//表示员工所在的省
//表示员工所在的市
//利用构造方法进行初始化

//重写 toString()方法



(3) 在 com.mingrisoft 包中再创建类文件, 名称为 Employee。在 Employee 中定义 3 个成员变量, 分别表示员工的姓名、年龄和地址。使用构造方法可以对它们进行赋值。此外, 可以使用 get() 和 set() 方法来获取和修改成员变量的值, 最后重写 toString() 和 clone() 方法。关键代码如下:

```
public class Employee implements Cloneable {
    private String name;           //表示员工的姓名
    private int age;               //表示员工的年龄
    private Address address;       //表示员工的地址
    public Employee(String name, int age, Address address) { //利用构造方法进行初始化
        this.name = name;
        this.age = age;
        this.address = address;
    }
    //省略 get() 和 set() 方法
    @Override
    public String toString() {      //重写 toString() 方法
        StringBuilder sb = new StringBuilder();
        sb.append("姓名: " + name + ", ");
        sb.append("年龄: " + age + "\n");
        sb.append("地址: " + address);
        return sb.toString();
    }
    @Override
    public Employee clone() {       //实现浅克隆
        Employee employee = null;
        try {
            employee = (Employee) super.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return employee;
    }
}
```

(4) 在 com.mingrisoft 包中再创建类文件, 名称为 Test。在该类中, 首先创建 address 对象并对其初始化, 然后创建 employee1 对象并对其初始化。使用 employee1 的克隆方法创建 employee2 对象。修改 employee2 的地址属性, 最后将两个对象输出。关键代码如下:

```
public class Test {
    public static void main(String[] args) {
        System.out.println("克隆之前: ");
        Address address = new Address("中国", "吉林", "长春"); //创建 address 对象
        Employee employee1 = new Employee("张 XX", 30, address); //创建 employee1 对象
        System.out.println("员工 1 的信息: ");
        System.out.println(employee1); //输出 employee1 对象
        System.out.println("克隆之后: ");
        Employee employee2 = employee1.clone(); //使用克隆创建 employee2 对象
        employee2.getAddress().setState("中国"); //修改员工地址
        employee2.getAddress().setProvince("四川"); //修改员工地址
        employee2.getAddress().setCity("成都"); //修改员工地址
        employee2.setName("李 XX"); //修改员工名字
    }
}
```





```

employee2.setAge(24); //修改员工年龄
System.out.println("员工 1 的信息: ");
System.out.println(employee1); //输出 employee1 对象
System.out.println("员工 2 的信息: ");
System.out.println(employee2); //输出 employee2 对象
}

```

技术要点

当需要克隆对象时，需要使用 clone() 方法，该方法的声明如下：

protected Object clone() throws CloneNotSupportedException

需要注意的是，该方法是一个受保护的方法，通常需要重写该方法并将访问权限限定符改成 public。该方法将类中各个域进行复制，如果对于引用类型的域，这种操作就会有问题，因此称作浅克隆。提供克隆功能的类需要实现 Cloneable 接口，否则使用 clone() 方法时会抛出 CloneNotSupportedException 异常。

实例 056 Java 对象的深克隆

(实例位置：配套资源\SL\06\056)

实例说明

如果类的成员变量中包括可变引用类型，则在克隆时就需要使用深克隆。本实例将演示如何使用深克隆，其运行效果如图 6.14 所示。

实现过程

(1) 在 Eclipse 中创建项目 056，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 Address。在 Address 类中定义 3 个成员变量，分别表示地址中的国家、省和市，使用构造方法可以对它们进行赋值。此外，可以使用 get() 和 set() 方法来获取和修改成员变量的值，最后重写 toString() 和 clone() 方法。关键代码如下：

```

public class Address implements Cloneable {
    private String state; //表示员工所在的国家
    private String province; //表示员工所在的省
    private String city; //表示员工所在的市
    public Address(String state, String province, String city) { //利用构造方法进行初始化
        this.state = state;
        this.province = province;
        this.city = city;
    }
    //省略 get() 和 set() 方法
    @Override
    public String toString() { //重写 toString() 方法
        StringBuilder sb = new StringBuilder();

```

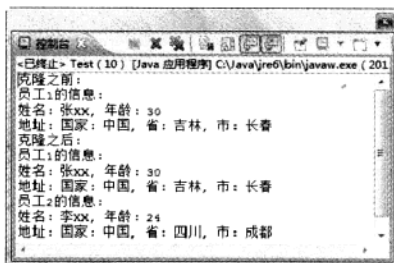


图 6.14 输出克隆前后对象信息



```

        sb.append("国家: " + state + ",");
        sb.append("省: " + province + ",");
        sb.append("市: " + city);
        return sb.toString();
    }

    @Override
    protected Address clone() {                                //实现浅克隆
        Address address = null;
        try {
            address = (Address) super.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return address;
    }
}

```



Note

脚下留神:

Address 类的域不是基本类型就是不可变类型，所以可以直接使用浅克隆。

(3) 在 com.mingrisoft 包中再创建类文件，名称为 Employee。在 Employee 中定义 3 个成员变量，分别表示员工的姓名、年龄和地址，使用构造方法可以对它们进行赋值。此外，可以使用 get()和 set()方法来获取和修改成员变量的值，最后重写 toString()和 clone()方法。关键代码如下：

```

public class Employee implements Cloneable {
    private String name;                                //表示员工的姓名
    private int age;                                    //表示员工的年龄
    private Address address;                            //表示员工的地址
    public Employee(String name, int age, Address address) { //利用构造方法进行初始化
        this.name = name;
        this.age = age;
        this.address = address;
    }
    //省略 get()和 set()方法
    @Override
    public String toString() {                          //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append("姓名: " + name + ",");
        sb.append("年龄: " + age + "\n");
        sb.append("地址: " + address);
        return sb.toString();
    }
    @Override
    public Employee clone() {                            //实现浅克隆
        Employee employee = null;
        try {
            employee = (Employee) super.clone();
            employee.address = address.clone();
        }
    }
}

```




```
} catch (CloneNotSupportedException e) {  
    e.printStackTrace();  
}  
return employee;  
}
```

多学两招:

在 Java 5.0 版中, 支持重写方法时返回协变类型, 因此可以返回 `Employee` 对象。

(4) 在 `com.mingrisoft` 包中再创建类文件, 名称为 `Test`。在该类中, 首先创建 `address` 对象并对其初始化, 然后创建 `employee1` 对象并对其初始化。使用 `employee1` 的克隆方法创建 `employee2` 对象, 修改 `employee2` 的 `address` 属性, 最后将两个对象输出。关键代码如下:

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println("克隆之前: ");  
        Address address = new Address("中国", "吉林", "长春"); //创建 address 对象  
        Employee employee1 = new Employee("张 XX", 30, address); //创建 employee1 对象  
        System.out.println("员工 1 的信息: ");  
        System.out.println(employee1); //输出 employee1 对象  
        System.out.println("克隆之后: ");  
        Employee employee2 = employee1.clone(); //使用克隆创建 employee2 对象  
        employee2.getAddress().setState("中国"); //修改员工地址  
        employee2.getAddress().setProvince("四川"); //修改员工地址  
        employee2.getAddress().setCity("成都"); //修改员工地址  
        employee2.setName("李 XX"); //修改员工名字  
        employee2.setAge(24); //修改员工年龄  
        System.out.println("员工 1 的信息: ");  
        System.out.println(employee1); //输出 employee1 对象  
        System.out.println("员工 2 的信息: ");  
        System.out.println(employee2); //输出 employee2 对象  
    }  
}
```

技术要点

通常情况下, 需要克隆对象时都需要使用深克隆。但需要注意的是, 如果引用类型中还有可变的引用类型成员变量, 则它也需要进行克隆。例如本实例中 `Address` 类如果增加了一个 `Date` 成员变量表示开始在此居住的时间, 则其也需要被克隆。

实例 057 序列化与对象克隆

(实例位置: 配套资源\SL\06\057)

实例说明

如果类的成员变量比较复杂, 例如使用了多个可变引用类型, 使用 `clone()` 方法来克隆是非



常麻烦的。此时,也可以考虑使用序列化的方式完成克隆。本实例将演示其用法,实例的运行效果如图 6.15 所示。

实现过程

(1) 在 Eclipse 中创建项目 057,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 Address。在 Address 类中定义 3 个成员变量,分别表示地址中的国家、省和市,使用构造方法可以对它们进行赋值。此外,可以使用 get() 和 set() 方法来获取和修改成员变量的值,最后重写 toString() 方法。关键代码如下:

```
public class Address implements Serializable {
    private static final long serialVersionUID = 4983187287403615604L;
    private String state;           //表示员工所在的国家
    private String province;       //表示员工所在的省
    private String city;          //表示员工所在的市
    public Address(String state, String province, String city) { //利用构造方法初始化各个域
        this.state = state;
        this.province = province;
        this.city = city;
    }
    //省略 get()和 set()方法
    @Override
    public String toString() { //使用地址属性表示地址对象
        StringBuilder sb = new StringBuilder();
        sb.append("国家: " + state + ",");
        sb.append("省: " + province + ",");
        sb.append("市: " + city);
        return sb.toString();
    }
}
```

(3) 在 com.mingrisoft 包中再创建类文件,名称为 Employee。在 Employee 中定义 3 个成员变量,分别表示员工的姓名、年龄和地址,使用构造方法可以对它们进行赋值。此外,可以使用 get() 和 set() 方法来获取和修改成员变量的值,最后重写 toString() 方法。关键代码如下:

```
public class Employee implements Serializable {
    private static final long serialVersionUID = 3049633059823371192L;
    private String name;           //表示员工的姓名
    private int age;               //表示员工的年龄
    private Address address;       //表示员工的地址
    public Employee(String name, int age, Address address) { //利用构造方法初始化各个域
        this.name = name;
        this.age = age;
        this.address = address;
    }
    //省略 get()和 set()方法
    @Override
    public String toString() { //重写 toString()方法
```

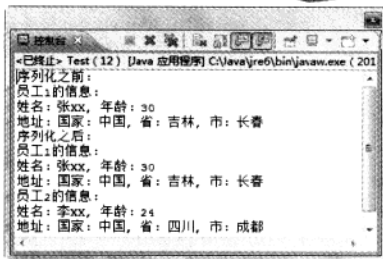


图 6.15 输出序列化前后对象信息



Note



Note

(4) 在 com.mingrisoft 包中再创建类文件, 名称为 Test。在该类中, 首先创建 address 对象并对其初始化, 然后创建 employee1 对象并对其初始化。接着将 employee1 对象写入到本地文件 employee.dat, 然后再读取出来赋值给 employee2。接着修改 employee2 的 address 属性, 最后将两个对象输出。关键代码如下:

```
public class Test {
    public static void main(String[] args) {
        System.out.println("序列化之前: ");
        Address address = new Address("中国", "吉林", "长春");           //创建 address 对象
        Employee employee1 = new Employee("张 XX", 30, address);         //创建 employee1 对象
        System.out.println("员工 1 的信息: ");
        System.out.println(employee1);                                   //输出 employee1 对象
        System.out.println("序列化之后: ");
        ObjectOutputStream out = null;
        ObjectInputStream in = null;
        Employee employee2 = null;
        try {
            out = new ObjectOutputStream(new FileOutputStream("employee.dat"));
            out.writeObject(employee1);                                   //将对象写入到本地文件中
            in = new ObjectInputStream(new FileInputStream("employee.dat"));
            employee2 = (Employee) in.readObject();                     //从本地文件读取对象
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            //省略释放资源代码
        }
        if (employee2 != null) {
            employee2.getAddress().setState("中国");                   //修改员工地址
            employee2.getAddress().setProvince("四川");                 //修改员工地址
            employee2.getAddress().setCity("成都");                     //修改员工地址
            employee2.setName("李 XX");                                 //修改员工名字
            employee2.setAge(24);                                        //修改员工年龄
            System.out.println("员工 1 的信息: ");
            System.out.println(employee1);                               //输出 employee1 对象
            System.out.println("员工 2 的信息: ");
            System.out.println(employee2);                               //输出 employee2 对象
        }
    }
}
```



技术要点

进行序列化的类需要实现 `Serializable` 接口，该接口中并没有定义任何方法，是一个标识接口。如果类中有可变的引用类型成员变量，则该变量也需要实现 `Serializable` 接口，依此类推。本实例中采用将对象写入本地文件的方式完成序列化，读者也可以将其写入到内存中再读取。



Note

实例 058 深克隆效率的比较

(实例位置：配套资源\SL\06\058)

实例说明

前面介绍了两种实现深克隆的方式，本实例将分别使用这两种方式克隆 100000 个对象，并输出花费的时间，这样读者可以对它们的效率差异有个直观的认识。实例的运行效果如图 6.16 所示。

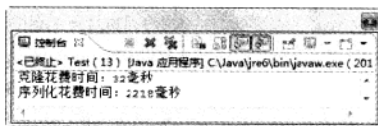


图 6.16 克隆与序列化花费的时间

实现过程

(1) 在 Eclipse 中创建项目 058，并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件，名称为 `Employee`。在 `Employee` 类中定义两个成员变量，分别表示员工的姓名和年龄，并重写 `toString()` 和 `clone()` 方法。关键代码如下：

```
public class Employee implements Cloneable, Serializable {
    private static final long serialVersionUID = 5022956767440380940L;
    private String name;           //表示员工的姓名
    private int age;              //表示员工的年龄
    public Employee(String name, int age) { //利用构造方法初始化各个域
        this.name = name;
        this.age = age;
    }
    @Override
    public String toString() { //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append("姓名: " + name + ", ");
        sb.append("年龄: " + age + "\n");
        return sb.toString();
    }
    @Override
    protected Employee clone() { //使用父类的 clone()方法实现深克隆
        Employee employee = null;
        try {
            employee = (Employee) super.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return employee;
    }
}
```




(3) 在 com.mingrisoft 包中再创建类文件, 名称为 Test。在该类中先创建一个列表保存 Employee 对象, 然后分别使用克隆和序列化的方式依次获得 10000 个对象, 并计算输出花费的时间。关键代码如下:



Note

```
public class Test {  
    public static void main(String[] args) {  
        List<Employee> employees = new ArrayList<Employee>(); //创建列表保存对象  
        Employee employee = new Employee("李 XX", 25); //创建 Employee 类的对象  
        long currentTime = System.currentTimeMillis(); //获得当前系统时间  
        for (int i = 0; i < 100000; i++) {  
            employees.add(employee.clone()); //使用克隆方式获得对象  
        }  
        System.out.println("克隆花费时间: " + (System.currentTimeMillis() - currentTime) + "毫秒");  
        currentTime = System.currentTimeMillis(); //获得当前时间  
        for (int i = 0; i < 100000; i++) {  
            ByteArrayOutputStream baos = new ByteArrayOutputStream(); //创建字节数组输出流  
            ObjectOutputStream out = null;  
            try {  
                out = new ObjectOutputStream(baos); //创建对象输出流  
                out.writeObject(employee); //将对象写入到输出流中  
            } catch (IOException e) {  
                e.printStackTrace();  
            } finally {  
                //省略释放资源代码  
            }  
            //获得字节数组输出流内容  
            ByteArrayInputStream bais = new ByteArrayInputStream(baos.toByteArray());  
            ObjectInputStream in = null;  
            try {  
                in = new ObjectInputStream(bais); //创建对象输入流  
                employees.add((Employee) in.readObject()); //读取对象  
            } catch (IOException e) {  
                e.printStackTrace();  
            } catch (ClassNotFoundException e) {  
                e.printStackTrace();  
            } finally {  
                //省略释放资源代码  
            }  
        }  
        System.out.println("序列化花费时间: " + (System.currentTimeMillis() - currentTime) +  
            "毫秒");  
    }  
}
```

技术要点

使用 ByteArrayOutputStream 和 ByteArrayInputStream 可以将对象保存在内存中, 这样就不必产生一个本地文件来完成序列化的功能。

第7章

面向对象进阶

本章读者可以学到如下实例：

- » 实例 059 经理与员工的差异
- » 实例 060 重写父类中的方法
- » 实例 061 计算几何图形的面积
- » 实例 062 简单的汽车销售商场
- » 实例 063 使用 Comparable 接口自定义排序
- » 实例 064 策略模式的简单应用
- » 实例 065 适配器模式的简单应用
- » 实例 066 普通内部类的简单应用
- » 实例 067 局部内部类的简单应用
- » 实例 068 匿名内部类的简单应用
- » 实例 069 静态内部类的简单应用
- » 实例 070 实例化 Class 类的几种方式
- » 实例 071 查看类的声明
- » 实例 072 查看类的成员
- » 实例 073 查看内部类信息
- » 实例 074 动态设置类的私有域
- » 实例 075 动态调用类中方法
- » 实例 076 动态实例化类
- » 实例 077 创建长度可变的数组
- » 实例 078 利用反射重写 toString() 方法



实例 059 经理与员工的差异

(实例位置: 配套资源\SL\07\059)

实例说明

对于在同一家公司工作的经理和员工而言, 两者是有很多共同点的。例如每个月都要发工资, 但是经理在完成目标任务后, 还会获得奖金。此时, 利用员工类来编写经理类就会少写很多代码, 利用继承技术可以让经理类使用员工类中定义的属性和方法。本实例将通过继承演示经理与员工的差异。实例的运行效果如图 7.1 所示。

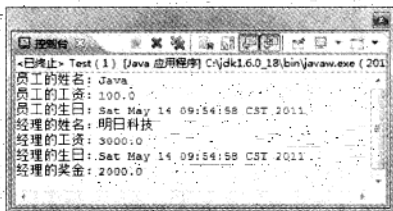


图 7.1 经理与员工的差异

实现过程

(1) 在 Eclipse 中创建项目 059, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Employee。在该类中定义 3 个属性, 分别是 name (表示员工的姓名)、salary (表示员工的工资) 和 birthday (表示员工的生日), 并分别为它们定义 getXXX() 和 setXXX() 方法。关键代码如下:

```
import java.util.Date;
public class Employee {
    private String name;           //员工的姓名
    private double salary;         //员工的工资
    private Date birthday;         //员工的生日

    public String getName() {      //获取员工的姓名
        return name;
    }
    public void setName(String name) { //设置员工的姓名
        this.name = name;
    }

    public double getSalary() {    //获取员工的工资
        return salary;
    }
    public void setSalary(double salary) { //设置员工的工资
        this.salary = salary;
    }

    public Date getBirthday() {    //获取员工的生日
        return birthday;
    }
    public void setBirthday(Date birthday) { //设置员工的生日
        this.birthday = birthday;
    }
}
```

(3) 在 com.mingrisoft 包中再创建一个名称为 Manager 的类, 该类继承自 Employee。在



该类中定义一个 bonus 域，表示经理的奖金，并为其设置 getXXX()和 setXXX()方法。关键代码如下：

```
public class Manager extends Employee {
    private double bonus;           //经理的奖金
    public double getBonus() {       //获得经理的奖金
        return bonus;
    }
    public void setBonus(double bonus) { //设置经理的奖金
        this.bonus = bonus;
    }
}
```

(4) 在 com.mingrisoft 包中再创建一个名称为 Text 的类，用于测试。在该类中分别创建 Employee 和 Manager 对象，并为其赋值，然后输出其属性。关键代码如下：

```
import java.util.Date;           //导入 java.util.Date 类
public class Test {
    public static void main(String[] args) {
        Employee employee = new Employee(); //创建 Employee 对象并为其赋值
        employee.setName("Java");
        employee.setSalary(100);
        employee.setBirthday(new Date());
        Manager manager = new Manager();     //创建 Manager 对象并为其赋值
        manager.setName("明日科技");
        manager.setSalary(3000);
        manager.setBirthday(new Date());
        manager.setBonus(2000);
        //输出经理和员工的属性值
        System.out.println("员工的姓名: " + employee.getName());
        System.out.println("员工的工资: " + employee.getSalary());
        System.out.println("员工的生日: " + employee.getBirthday());
        System.out.println("经理的姓名: " + manager.getName());
        System.out.println("经理的工资: " + manager.getSalary());
        System.out.println("经理的生日: " + manager.getBirthday());
        System.out.println("经理的奖金: " + manager.getBonus());
    }
}
```



Note

指点迷津：

在 Manager 类中并未定义姓名等域，然而却可以使用，这就是继承的好处。

脚下留神：

虽然使用继承能少写很多代码，但是不要滥用继承。在使用继承前，需要考虑一下两者之间是否真的是“is-a”的关系，这是继承的重要特征。本实例中，经理显然是员工，所以可以用继承。另外，子类也可以成为其他类的父类，这样就构成了一棵继承树。

技术要点

在面向对象程序设计中，继承是其基本特性之一。在 Java 中，如果想表明类 A 继承了类 B，



可以使用下面的语法定义类 A:

```
public class A extends B {}
```

类 A 称为子类、派生类或孩子类, 类 B 称为超类、基类或父类。尽管类 B 是一个超类, 但是并不意味着类 B 比类 A 有更多的功能。相反, 类 A 比类 B 拥有的功能更加丰富。

指点迷津:

在继承树中, 从下往上越来越抽象, 从上往下越来越具体。

实例 060 重写父类中的方法

(实例位置: 配套资源\SL\07\060)

实例说明

在继承了一个类之后, 也就可以使用父类中定义的方法。然而, 父类中的方法可能并不完全适用于子类。此时, 如果不想定义新的方法, 则可以重写父类中的方法。本实例将演示如何重写父类中的方法, 实例的运行效果如图 7.2 所示。

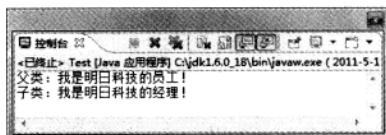


图 7.2 重写父类中的方法

实现过程

(1) 在 Eclipse 中创建项目 060, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 Employee。在该类中添加 getInfo() 方法, 返回值为字符串“父类: 我是明日科技的员工! ”。关键代码如下:

```
public class Employee {  
    public String getInfo() {  
        return "父类: 我是明日科技的员工! ";  
    }  
}
```

//定义测试用的方法

(3) 在 com.mingrisoft 包中再创建一个名称为 Manager 的类, 该类继承自 Employee。在该类中重写 getInfo() 方法。关键代码如下:

```
public class Manager extends Employee {  
    @Override  
    public String getInfo() {  
        return "子类: 我是明日科技的经理! ";  
    }  
}
```

//重写测试用的方法

指点迷津:

在 Java SE 5.0 版中新增了注解功能, @Override 是最常用的注解之一。该注解只能应用在方法上, 可以测试该方法是否重写了父类中的方法, 如果没有则会在编译时报错。使用该注解可以很好地避免重写时发生的各种问题, 因此推荐读者使用。

(4) 在 com.mingrisoft 包中再创建一个名称为 Text 的类, 用于测试。在该类中分别创建



Employee 和 Manager 对象，并分别输出 getInfo()方法的返回值。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        Employee employee = new Employee(); //创建 Employee 对象
        System.out.println(employee.getInfo()); //输出 Employee 对象的 getInfo()方法返回值
        Manager manager = new Manager(); //创建 Manager 对象
        System.out.println(manager.getInfo()); //输出 Manager 对象的 getInfo()方法返回值
    }
}
```



Note

技术要点

本实例主要应用的是方法的重写。方法的重写（Overriding）只能发生在存在继承关系的类中。重写方法需要注意以下几点：

- ☒ 重写方法与原来方法签名要相同，即方法名称和参数（包括顺序）要相同。
- ☒ 重写方法的可见性不能小于原来的方法。
- ☒ 重写方法抛出异常的范围不能大于原来方法抛出异常的范围。

实例 061 计算几何图形的面积

（实例位置：配套资源\SL\07\061）

实例说明

对于每个几何图形而言，都有一些共同的属性，如名字和面积等，而其计算面积的方法却各不相同。为了简化开发，本实例将定义一个超类来实现输出名字的方法，并使用抽象方法来计算面积。实例的运行效果如图 7.3 所示。

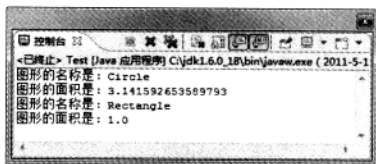


图 7.3 计算几何图形的面积

实现过程

（1）在 Eclipse 中创建项目 061，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个抽象类，名称为 Shape。在该类中定义两个方法，一个是 getName()，用于使用反射机制获得类名称；另一个是抽象方法 getArea()，并未实现。关键代码如下：

```
public abstract class Shape {
    public String getName() { //获得图形的名称
        return this.getClass().getSimpleName();
    }
    public abstract double getArea(); //获得图形的面积
}
```

（3）在 com.mingrisoft 包中再创建一个名称为 Circle 的类，该类继承自 Shape，并实现了抽象方法 getArea()。在该类的构造方法中，获得了圆形的半径，用于在 getArea()中计算面积。关键代码如下：

```
public class Circle extends Shape {
    private double radius;
```



Note

```

public Circle(double radius) {                //获得圆形的半径
    this.radius = radius;
}
@Override
public double getArea() {                      //计算圆形的面积
    return Math.PI * Math.pow(radius, 2);
}
}

```

(4) 在 com.mingrisoft 包中再创建一个名称为 Rectangle 的类, 该类继承自 Shape, 并实现了抽象方法 getArea()。在该类的构造方法中, 获得了矩形的长和宽, 用于在 getArea() 中计算面积。关键代码如下:

```

public class Rectangle extends Shape {
    private double length;
    private double width;
    public Rectangle(double length, double width) {    //获得矩形的长和宽
        this.length = length;
        this.width = width;
    }
    @Override
    public double getArea() {                          //计算矩形的面积
        return length * width;
    }
}

```

(5) 在 com.mingrisoft 包中再创建一个名称为 Test 的类, 用来进行测试, 在该类中创建 Circle 和 Rectangle 对象, 并分别输出图形的名称和面积。关键代码如下:

```

public class Test {
    public static void main(String[] args) {
        Circle circle = new Circle(1);              //创建圆形对象并将半径设置成 1
        System.out.println("图形的名称是: " + circle.getName());
        System.out.println("图形的面积是: " + circle.getArea());
        Rectangle rectangle = new Rectangle(1, 1);    //创建矩形对象并将长和宽设置成 1
        System.out.println("图形的名称是: " + rectangle.getName());
        System.out.println("图形的面积是: " + rectangle.getArea());
    }
}

```

指点迷津:

在抽象类中, 可以定义抽象方法 (使用 abstract 修饰的方法), 也可以定义普通方法。包含抽象方法的类必须是抽象类, 而抽象类不是必须包含抽象方法。对于抽象方法而言, 仅定义一个声明即可, 即抽象方法是没有方法体的。

技术要点

本实例应用的主要技术就是抽象类。下面对抽象类进行介绍。

在设计类的过程中, 通常会将一些类所具有的公共属性和方法移到超类中, 这样就不必重复定义了。然而这些类的超类却经常没有实际意义。通常将它设置成抽象的, 这样可以避免创建该类的对象。声明一个最简单的抽象类的代码如下:

```

public abstract class Shape {}

```




参数说明

Shape: 为抽象类的类名。

脚下留神:

抽象类是不能直接实例化的, 如果要获得该类的实例可以使用静态方法创建其实现类对象。



Note

实例 062 简单的汽车销售商场

(实例位置: 配套资源\SL\07\062)

实例说明

当顾客在商场购物时, 卖家需要根据顾客的需求提取商品。对于汽车销售商场也是如此。用户需要先指定购买的车型, 然后商家去提取该车型的汽车。本实例将实现一个简单的汽车销售商场, 用来演示多态的用法。实例的运行效果如图 7.4 所示。

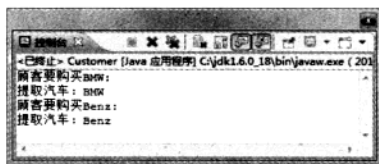


图 7.4 简单的汽车销售商场

实现过程

(1) 在 Eclipse 中创建项目 062, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建一个抽象类, 名称为 Car, 在该类中定义一个抽象方法 getInfo()。关键代码如下:

```
public abstract class Car {
    public abstract String getInfo();           //用来描述汽车的信息
}
```

(3) 在 com.mingrisoft 包中再创建一个名称为 BMW 的类, 该类继承自 Car 并实现其 getInfo() 方法。关键代码如下:

```
public class BMW extends Car {
    @Override
    public String getInfo() {                  //用来描述汽车的信息
        return "BMW";
    }
}
```

(4) 在 com.mingrisoft 包中再创建一个名称为 Benz 的类, 该类继承自 Car 并实现其 getInfo() 方法。关键代码如下:

```
public class Benz extends Car {
    @Override
    public String getInfo() {                  //用来描述汽车的信息
        return "Benz";
    }
}
```

(5) 在 com.mingrisoft 包中再创建一个名称为 CarFactory 的类, 该类定义了一个静态方法 getCar(), 它可以根据用户指定的车型来创建对象。关键代码如下:

```
public class CarFactory {
    public static Car getCar(String name) {
```




Note

```

if (name.equalsIgnoreCase("BMW")) { //如果需要 BMW 则创建 BMW 对象
    return new BMW();
} else if (name.equalsIgnoreCase("Benz")) { //如果需要 Benz 则创建 Benz 对象
    return new Benz();
} else { //暂时不能支持其他车型
    return null;
}
}

```

(6) 在 com.mingrisoft 包中再创建一个名称为 Customer 的类, 用来进行测试。在 main() 方法中, 根据用户的需要提取了不同的汽车。关键代码如下:

```

public class Customer {
    public static void main(String[] args) {
        System.out.println("顾客要购买 BMW:");
        Car bmw = CarFactory.getCar("BMW"); //用户要购买 BMW
        System.out.println("提取汽车: " + bmw.getInfo()); //提取 BMW
        System.out.println("顾客要购买 Benz:");
        Car benz = CarFactory.getCar("Benz"); //用户要购买 Benz
        System.out.println("提取汽车: " + benz.getInfo()); //提取 Benz
    }
}

```

技术要点

本实例应用的主要技术就是面向对象程序设计中的多态。多态是面向对象程序设计的基本特性之一。使用多态的好处就是可以屏蔽对象之间的差异, 从而增强了软件的扩展性和重用性。Java 的多态主要是通过重写父类(或接口)中的方法来实现的。对于香蕉、桔子等水果而言, 人们通常关心其能吃的特性。如果分别说香蕉能吃、桔子能吃, 则当再增加新的水果种类, 如菠萝时还要写个菠萝能吃, 这是非常麻烦的。使用多态的话可以写成水果能吃, 当需要用到具体的水果时, 系统会自动帮忙替换, 从而简化开发。

实例 063 使用 Comparable 接口自定义排序

(实例位置: 配套资源\SL\07\063)

实例说明

默认情况下, 保存在 List 集合中的数组是不进行排序的, 不过可以通过使用 Comparable 接口自定义排序规则并自动排序。本实例将介绍如何使用 Comparable 接口自定义排序规则并自动排序。实例的运行效果如图 7.5 所示。

实现过程

(1) 在 Eclipse 中创建项目 063, 并在该项目中创建 com.mingrisoft 包。

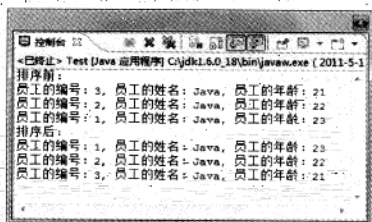


图 7.5 使用 Comparable 接口自定义排序



(2) 在 `com.mingrisoft` 包中新建一个 Java 类, 名称为 `Employee`。在该类中首先定义 3 个属性, 分别是 `id` (表示员工的编号)、`name` (表示员工的姓名) 和 `age` (表示员工的年龄), 然后在构造方法中初始化这 3 个属性, 最后再实现接口中定义的 `compareTo()` 方法, 将对象按编号升序排列。关键代码如下:

```
public class Employee implements Comparable<Employee> {
    private int id;           //员工的编号
    private String name;      //员工的姓名
    private int age;          //员工的年龄
    public Employee(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }
    @Override
    public int compareTo(Employee o) {           //利用编号实现对象间的比较
        if (id > o.id) {
            return 1;
        } else if (id < o.id) {
            return -1;
        }
        return 0;
    }
    @Override
    public String toString() {                   //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append("员工的编号: " + id + ",");
        sb.append("员工的姓名: " + name + ",");
        sb.append("员工的年龄: " + age);
        return sb.toString();
    }
}
```



Note

指点迷津:

重写接口中的方法时, 要将访问权限限定符设为 `public`, 因为接口中的方法默认就是 `public` 的。

(3) 在 `com.mingrisoft` 包中再新建一个名称为 `Test` 的类, 用于进行测试。在该类中首先定义一个 `List` 集合来保存 3 个 `Employee` 对象, 并通过遍历输出集合中的元素, 再通过 `Collections.sort()` 方法执行自动排序, 最后再通过遍历输出排序后的集合中的元素。关键代码如下:

```
public class Test {
    public static void main(String[] args) {
        List<Employee> list = new ArrayList<Employee>();
        list.add(new Employee(3, "Java", 21));
        list.add(new Employee(2, "Java", 22));
        list.add(new Employee(1, "Java", 23));
        System.out.println("排序前:");
        for (Employee employee : list) {
            System.out.println(employee);
        }
    }
}
```



Note

```

System.out.println("排序后: ");
Collections.sort(list);           //执行自动排序
for (Employee employee : list) {
    System.out.println(employee);
}
}
}

```

技术要点

本实例主要应用 `java.lang` 包中的 `Comparable` 接口来实现自定义排序规则。`Comparable` 接口用于强行对实现它的每个类的对象进行整体排序。在实现该接口的类中,必须实现该接口中定义的 `compareTo()` 方法,用于指定排序规则。`Comparable` 接口的定义如下:

```

public interface Comparable<T> {
    int compareTo(T other);
}

```

如果一个类要实现这个接口,可以使用如下语句声明:

```

public class Employee implements Comparable<Employee>{}

```

在 `Employee` 中必须实现接口中定义的 `compareTo()` 方法。实现该方法后,如果将该对象保存到列表中,那么可以通过执行 `Collections.sort()` 方法进行自动排序;如果保存到数组中,那么可以通过执行 `Arrays.sort()` 方法进行自动排序。

指点迷津:

如果不想实现在接口中定义的方法,则可以将类声明为抽象类,将接口中定义的方法声明为抽象方法。

实例 064 策略模式的简单应用

(实例位置: 配套资源\SL\07\064)

实例说明

在使用图像处理软件处理图片后,需要选择一种格式进行保存,然而各种格式在底层实现的算法并不相同,这刚好适合策略模式。本实例将演示如何使用策略模式与简单工厂模式组合进行实例开发。实例的运行效果如图 7.6 所示。

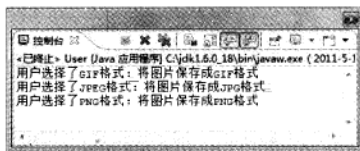


图 7.6 策略模式的简单应用

实现过程

- (1) 在 Eclipse 中创建项目 064,并在该项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中编写接口 `ImageSaver`,在该接口中定义 `save()` 方法。关键代码如下:

```

public interface ImageSaver {
    void save();           //定义 save()方法
}

```



(3) 在 com.mingrisoft 包中再编写类 GIFSaver, 该类实现了 ImageSaver 接口。在实现 save() 方法时将图片保存为 GIF 格式, 关键代码如下:

```
public class GIFSaver implements ImageSaver {
    @Override
    public void save() {
        System.out.println("将图片保存成 GIF 格式");
    }
}
```

//实现 save()方法



Note

指点迷津:

对于将图片保存成其他格式与存储为 GIF 格式类似, 这里就不再赘述。

(4) 在 com.mingrisoft 包中再编写类 TypeChooser, 该类根据用户提供的图片类型来选择合适的图片存储方式。关键代码如下:

```
public class TypeChooser {
    public static ImageSaver getSaver(String type) {
        if (type.equalsIgnoreCase("GIF")) {
            return new GIFSaver();
        } else if (type.equalsIgnoreCase("JPEG")) {
            return new JPEGSaver();
        } else if (type.equalsIgnoreCase("PNG")) {
            return new PNGSaver();
        } else {
            return null;
        }
    }
}
```

//使用 if...else 语句来判断图片的类型

指点迷津:

此处使用了简单工厂模式, 根据描述图片类型的字符串创建相应的图片保存类的对象。

(5) 在 com.mingrisoft 包中再编写类 User, 该类模拟用户的操作, 为类型选择器提供图片的类型。关键代码如下:

```
public class User {
    public static void main(String[] args) {
        System.out.print("用户选择了 GIF 格式: ");
        ImageSaver saver = TypeChooser.getSaver("GIF"); //获得保存图片为 GIF 类型的对象
        saver.save();
        System.out.print("用户选择了 JPEG 格式: "); //获得保存图片为 JPEG 类型的对象
        saver = TypeChooser.getSaver("JPEG");
        saver.save();
        System.out.print("用户选择了 PNG 格式: "); //获得保存图片为 PNG 类型的对象
        saver = TypeChooser.getSaver("PNG");
        saver.save();
    }
}
```



技术要点

本实例应用的最重要的技术就是策略模式。对于策略模式而言，需要定义一个接口或者抽象类来表示各种策略的抽象，这样就可以使用多态来让虚拟机选择不同的实现类。然后让每一种具体的策略来实现这个接口或继承抽象类，并为其中定义的方法提供具体的实现。由于在选择适当的策略上有些不方便，需要不断地判断需要的类型，因此用简单工厂方法来实现判断过程。

实例 065 适配器模式的简单应用

(实例位置: 配套资源\SL\07\065)

实例说明

对于刚从工厂生产出来的商品，有些功能并不能完全满足用户的需要。因此，用户通常会对其进行一定的改装工作。本实例将为普通的汽车增加 GPS 定位功能，借此演示适配器模式的应用。实例的运行效果如图 7.7 所示。

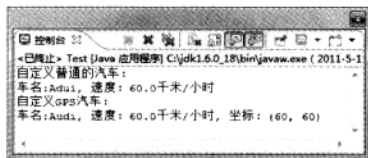


图 7.7 适配器模式的简单应用

实现过程

(1) 在 Eclipse 中创建项目 065，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写类 Car，在该类中，首先定义两个属性，一个是 name，表示汽车的名字；另一个是 speed，表示汽车的速度。并为其提供 getXXX() 和 setXXX() 方法，然后通过重写 toString() 方法来方便输出 Car 对象。关键代码如下：

```
public class Car {  
    private String name;  
    private double speed;  
    //省略 getXXX() 和 setXXX() 方法  
    @Override  
    public String toString() {  
        //表示名称  
        //表示速度  
        //重写 toString() 方法  
        StringBuilder sb = new StringBuilder();  
        sb.append("车名:" + name + ", ");  
        sb.append("速度: " + speed + "千米/小时");  
        return sb.toString();  
    }  
}
```

(3) 在 com.mingrisoft 包中再编写接口 GPS，该接口中定义了 getLocation() 方法，用来确定汽车的位置。关键代码如下：

```
public interface GPS {  
    Point getLocation();  
    //提供定位功能  
}
```

(4) 在 com.mingrisoft 包中再编写类 GPSCar，该类继承 Car 并实现 GPS 接口。在该类中首先实现 getLocation() 方法，用于实现确定汽车位置的功能，然后重写 toString() 方法方便输出



GPSCar 对象。关键代码如下：

```
public class GPSCar extends Car implements GPS {
    @Override
    public Point getLocation() {                //利用汽车的速度来确定汽车的位置
        Point point = new Point();
        point.setLocation(super.getSpeed(), super.getSpeed());
        return point;
    }
    @Override
    public String toString() {                  //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString());
        sb.append(", 坐标: (" + getLocation().x + ", " + getLocation().y + ")");
        return sb.toString();
    }
}
```



Note

指点迷津：

可以使用 super 关键字调用父类中定义的方法。

(5) 在 com.mingrisoft 包中再编写类 Test 进行测试。在该类中，分别创建 Car 和 GPSCar 对象，并对其初始化，然后输出这两个对象。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        System.out.println("自定义普通的汽车: ");
        Car car = new Car();                //创建普通的汽车对象并初始化
        car.setName("Adui");
        car.setSpeed(60);
        System.out.println(car);
        System.out.println("自定义 GPS 汽车: ");
        GPSCar gpsCar = new GPSCar();        //创建带 GPS 功能的汽车对象并初始化
        gpsCar.setName("Audi");
        gpsCar.setSpeed(60);
        System.out.println(gpsCar);
    }
}
```

技术要点

适配器模式可以在符合 OCP 原则（开闭原则）的基础上，为类增加新的功能。该模式涉及的角色主要有以下 3 个。

- ☑ 目标角色：就是期待得到的接口，如本实例的 GPS 接口。
 - ☑ 源角色：需要被增加功能的类或接口，如本实例的 Car 类。
 - ☑ 适配器角色：新创建的类，在源角色的基础上实现了目标角色，如本实例的 GPSCar 类。
- 关于各个类的继承（实现）关系如图 7.8 所示。



Note

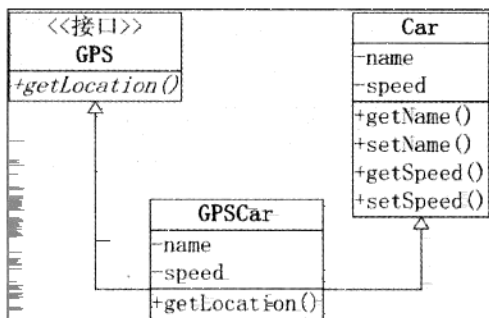


图 7.8 适配器模式 UML 图

实例 066 普通内部类的简单应用

(实例位置: 配套资源\SL\07\066)

实例说明

在使用图形界面程序时,用户总是希望界面是丰富多彩的,这就要求程序员根据不同的情况为界面设置不同的颜色。本实例定义了 3 个按钮,用户通过单击不同的按钮,可以为面板设置不同的颜色。运行本实例,将显示如图 7.9 所示的效果,单击“红色”按钮,即可将背景设置为红色;单击“绿色”按钮,即可将背景设置为绿色;单击“蓝色”按钮,即可将背景设置为蓝色,如图 7.10 所示。

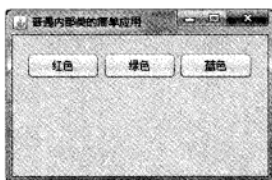


图 7.9 默认的运行效果

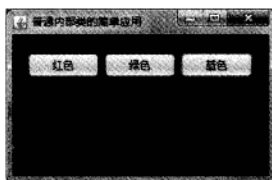


图 7.10 单击“蓝色”按钮的效果

实现过程

- (1) 在 Eclipse 中创建项目 066,并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中编写类 ButtonTest,该类继承自 JFrame。在该窗体中添加 3 个按钮,分别用来为面板设置不同的颜色。
- (3) 在 com.mingrisoft 包中再编写类 ColorAction,该类继承自 ActionListener 接口。在该类的构造方法中,需要为其指定一种颜色,在 actionPerformed()方法中将面板设置成指定的颜色。关键代码如下:

```
private class ColorAction implements ActionListener {  
    private Color background;  
    public ColorAction(Color background) {  
        this.background = background;  
    }  
    @Override
```



```
public void actionPerformed(ActionEvent e) {
    contentPane.setBackground(background);
}
}
```

指点迷津:

panel 是在外部类 ButtonTest 中定义的域,但是在内部类中却可以直接使用。



Note

技术要点

在类中,除了可以定义域、方法和块,还可以定义类,这种类称为内部类。声明一个最简单的内部类的语法如下:

```
public class Outer {
    class Inner {}
}
```

内部类可以使用外部类中定义的属性和方法,即使它们都是私有的。编译器在编译内部类时,将内部类命名为 Outer\$Inner 的形式,虚拟机并不知道有内部类。

实例 067 局部内部类的简单应用

(实例位置: 配套资源\SL\07\067)

实例说明

日常生活中,闹钟的应用非常广泛。使用它可以更好地帮助人们安排时间。本实例将实现一个非常简单的闹钟。运行本实例,控制台会不断输出当前的时间,并且每隔一秒钟会发出提示音。用户可以单击“确定”按钮来退出程序。实例的运行效果如图 7.11 所示。

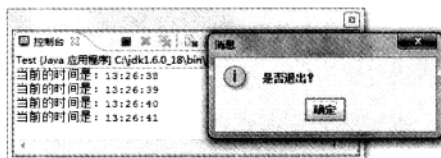


图 7.11 局部内部类的简单应用

实现过程

(1) 在 Eclipse 中创建项目 067,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写 Java 类,名称为 AlarmClock。在该类中,首先定义两个属性,一个是 delay,表示延迟的时间;另一个是 flag,表示是否需要发出提示声音。然后在 start()方法中,使用 Timer 类来安排动作发出事件。关键代码如下:

```
public class AlarmClock {
    private int delay;           //表示延迟时间
    private boolean flag;        //表示是否要发出声音
    public AlarmClock(int delay, boolean flag) {
        this.delay = delay;      //使用构造方法初始化各个域
        this.flag = flag;
    }
    public void start() {
```



Note

```

class Printer implements ActionListener {           //定义内部类实现动作监听接口
    @Override
    public void actionPerformed(ActionEvent e) {
        SimpleDateFormat format = new SimpleDateFormat("k:m:s"); //定义时间的格式
        String result = format.format(new Date());                //获得当前的时间
        System.out.println("当前的时间是: " + result);            //显示当前的时间
        if (flag) {                                                //根据 flag 来决定是否要发出声音
            Toolkit.getDefaultToolkit().beep();
        }
    }
}

new Timer(delay, new Printer()).start();           //创建 Timer 对象并启动
}

```

脚下留神:

如果 Printer 类使用了在 start() 方法内定义的其他变量, 则该变量也必须是 final 的。

(3) 编写类 Test 进行测试, 在该类的 main() 方法中创建 AlarmClock 对象, 并调用其 start() 方法。使用对话框提示用户是否要退出程序。关键代码如下:

```

public class Test {
    public static void main(String[] args) {
        AlarmClock clock = new AlarmClock(1000, true); //创建 AlarmClock 对象
        clock.start();                                 //启动 start() 方法
        JOptionPane.showMessageDialog(null, "是否退出? ");
        System.exit(0);                                //退出程序
    }
}

```

技术要点

本实例的技术要点就是局部内部类。在 Java 中可以将类定义在方法的内部, 称为局部内部类。这种类不能使用 public 和 private 修饰, 它的作用域被限定在声明这个方法中。局部内部类比其他内部类还有一个优点, 就是可以访问方法参数。一个最简单的局部内部类代码如下:

```

public void book () {
    public class MingriSoft {}
}

```

脚下留神:

被局部内部类使用的方法参数必须是 final 的。

实例 068 匿名内部类的简单应用

(实例位置: 配套资源\SL\07\068)

实例说明

在查看数码相片时, 通常会使用一款图片查看软件, 该软件应该能够遍历文件夹下的所有图



片并进行显示。本实例将编写一个非常简单的图片查看软件，它可以支持6张图片。通过单击不同的按钮就可以查看不同的图片。运行本实例，单击不同的按钮将显示不同的图片，例如，单击“图片4”按钮，将显示如图7.12所示的图片；单击“图片5”按钮，将显示如图7.13所示的图片。

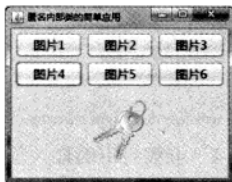


图 7.12 单击“图片4”按钮的运行结果



图 7.13 单击“图片5”按钮的运行结果



Note

实现过程

(1) 在 Eclipse 中创建项目 068，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写类，名称为 ImageViewer，该类继承自 JFrame。在窗体中添加6个按钮和一个标签，单击不同的按钮可以在标签上显示不同的图片。关键代码如下：

```
public ImageViewer() {
    //省略与 button1 无关的代码
    JButton button1 = new JButton("图片 1");           //创建按钮
    button1.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //修改按钮上文本的字体
    button1.addActionListener(new ActionListener() {     //为按钮增加监听器
        @Override
        public void actionPerformed(ActionEvent e) {
            label.setIcon(new ImageIcon("src/images/1.png")); //在标签中显示图片
        }
    });
    //省略与 button1 无关的代码
}
```

技术要点

本实例的技术要点就是匿名内部类。当只需要创建类的一个对象时，可以使用匿名内部类。ActionListener 是 Swing 中动作事件的监听器，如果创建该接口的匿名内部类，可以使用以下代码：

```
ActionListener listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {}
};
```

脚下留神：

不要忘记写最后的“;”分号，这是语句的结束标识，和内部类无关。

实例 069 静态内部类的简单应用

(实例位置：配套资源\SL\07\069)

实例说明

当对元素进行排序时，需要明确各个元素如何比较大小。使用既定的比较方式，就可以求



出一个数组中的最大值和最小值。本实例使用静态内部类来实现使用一次遍历求最大值和最小值。实例的运行效果如图 7.14 所示。

实现过程

(1) 在 Eclipse 中创建项目 069，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写 Java 类，名称为 MaxMin，在该类中，首先定义一个静态内部类 Result，然后在该类中定义两个浮点型属性，一个是 max，表示最大值；另一个是 min，表示最小值。再使用构造方法为其初始化，并提供 getXXX() 方法来获得这两个值。最后定义一个静态方法 getResult()，该方法的返回值是 Result 类型，这样就可以既保存最大值，又保存最小值。关键代码如下：

```
public class MaxMin {
    public static class Result {
        private double max;
        private double min;
        public Result(double max, double min) {
            this.max = max;
            this.min = min;
        }
        public double getMax() {
            return max;
        }
        public double getMin() {
            return min;
        }
    }
    public static Result getResult(double[] array) {
        double max = Double.MIN_VALUE;
        double min = Double.MAX_VALUE;
        for (double i : array) {
            if (i > max) {
                max = i;
            }
            if (i < min) {
                min = i;
            }
        }
        return new Result(max, min);
    }
}
```

//表示最大值
//表示最小值
//使用构造方法进行初始化

//获得最大值

//获得最小值

//遍历数组获得最大值和最小值

//返回 Result 对象

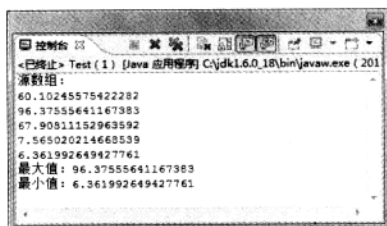


图 7.14 求数组中的最大值和最小值

(3) 在 com.mingrisoft 包中再编写类 Test 进行测试，在该类的 main() 方法中，使用随机数初始化了一个长度为 5 的数组，并求得该数组的最大值和最小值。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        double[] array = new double[5];
```



```

for (int i = 0; i < array.length; i++) {           //初始化数组
    array[i] = 100*Math.random();
}
System.out.println("源数组: ");
for (int i = 0; i < array.length; i++) {           //显示数组中的各个元素
    System.out.println(array[i]);
}
System.out.println("最大值: " + MaxMin.getResult(array).getMax()); //显示最大值
System.out.println("最小值: " + MaxMin.getResult(array).getMin()); //显示最小值
}
}

```



Note

技术要点

本实例的技术要点就是静态内部类。静态内部类是使用 `static` 修饰的内部类，在静态内部类中，可以使用外部类定义的静态域，但是不能使用非静态域。这是静态内部类与非静态内部类的重要区别。定义一个最简单的静态内部类的代码如下：

```

public void book() {
    public static class MingriSoft{}
}

```

脚下留神：

不要将 `MingriSoft` 类声明成 `private` 的，否则不能使用其中定义的方法。

实例 070 实例化 Class 类的几种方式

(实例位置：配套资源\SL\07\070)

实例说明

Java 的数据类型可以分成两类，即引用类型和原始类型。无论哪种类型的对象，Java 虚拟机都会实例化不可变的 `java.lang.Class` 对象。它提供了在运行时检查对象属性的方法，这些属性包括它的成员和类型信息。更重要的是 `Class` 对象是所有反射 API 的入口。本实例将演示如何获得 `Class` 对象。实例的运行效果如图 7.15 所示。



图 7.15 实例化 Class 类的几种方式

多学两招：

`Class` 类是泛型类，可以使用 `@SuppressWarnings("unchecked")` 忽略泛型或者使用 `Class<?>` 类型。

实现过程

(1) 在 Eclipse 中创建项目 070，并在该项目中创建 `com.mingrisoft` 包。



(2) 在 com.mingrisoft 包中编写类 ClassTest, 在该类的主方法中, 演示各种获得 Class 对象的方法。关键代码如下:

```
public class ClassTest {
    @SuppressWarnings("unchecked")
    public static void main(String[] args) throws ClassNotFoundException {
        System.out.println("第 1 种方法: Object.getClass()");
        Class c1 = new Date().getClass();           //使用 getClass()方法获得 Class 对象
        System.out.println(c1.getName());           //输出对象名称
        System.out.println("第 2 种方法: .class 语法");
        Class c2 = boolean.class;                   //使用 .class 语法获得 Class 对象
        System.out.println(c2.getName());           //输出对象名称
        System.out.println("第 3 种方法: Class.forName()");
        Class c3 = Class.forName("java.lang.String"); //使用 Class.forName()方法获得 Class 对象
        System.out.println(c3.getName());           //输出对象名称
        System.out.println("第 4 种方法: 包装类的 TYPE 域");
        Class c4 = Double.TYPE;                     //使用包装类获得 Class 对象
        System.out.println(c4.getName());           //输出对象名称
    }
}
```

技术要点

本实例的技术要点就是如何获得 Class 对象。通常有以下 4 种方式可以获得 Class 对象。

- ☑ Object.getClass(): 如果一个类的对象可用, 则最简单的获得 Class 的方法是使用 Object.getClass()。当然, 这种方式只对引用类型有用。
- ☑ .class 语法: 如果类型可用, 但没有对象则可以在类型后加上 “.class” 来获得 Class 对象。这也是使原始类型获得 Class 对象的最简单的方式。
- ☑ Class.forName(): 如果知道类的全名, 则可以使用静态方法 Class.forName() 来获得 Class 对象。该方法不能用在原始类型上, 但是可以用在原始类型数组上。

脚下留神:

Class.forName()方法会抛出 ClassNotFoundException 异常。

- ☑ 包装类的 TYPE 域: 每个原始类型和 void 都有包装类, 利用其 TYPE 域就可以获得 Class 对象。

实例 071 查看类的声明

(实例位置: 配套资源\SL\07\071)

实例说明

通常类的声明包括常见修饰符 (public、protected、private、abstract、static、final 和 strictfp 等)、类的名称、类的泛型参数、类的继承类 (实现的接口) 和类的注解等。本实例将演示如何用反射获得这些信息。实例的运行效果如图 7.16 所示。



图 7.16 查看类的声明



Note

实现过程

(1) 在 Eclipse 中创建项目 071，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写类 ClassDeclarationViewer，在 main() 方法中输出了与类声明相关的各个项。关键代码如下：

```
public class ClassDeclarationViewer {
    public static void main(String[] args) throws ClassNotFoundException {
        Class<?> clazz = Class.forName("java.util.ArrayList");    //获得 ArrayList 类对象
        System.out.println("类的标准名称: " + clazz.getCanonicalName());
        System.out.println("类的修饰符: " + Modifier.toString(clazz.getModifiers()));
        //输出类的泛型参数
        TypeVariable<?>[] typeVariables = clazz.getTypeParameters();
        System.out.print("类的泛型参数: ");
        if (typeVariables.length != 0) {
            for (TypeVariable<?> typeVariable : typeVariables) {
                System.out.println(typeVariable + "\t");
            }
        } else {
            System.out.println("空");
        }
        //输出类所实现的所有接口
        Type[] interfaces = clazz.getGenericInterfaces();
        System.out.println("类所实现的接口: ");
        if (interfaces.length != 0) {
            for (Type type : interfaces) {
                System.out.println("\t" + type);
            }
        } else {
            System.out.println("\t" + "空");
        }
        //输出类的直接继承类，如果是继承自 Object 则返回空
        Type superClass = clazz.getGenericSuperclass();
        System.out.print("类的直接继承类: ");
        if (superClass != null) {
            System.out.println(superClass);
        } else {
            System.out.println("空");
        }
    }
}
```



Note

```
//输出类的所有注释信息，有些注释信息是不能用反射获得的
Annotation[] annotations = clazz.getAnnotations();
System.out.print("类的注解：");
if (annotations.length != 0) {
    for (Annotation annotation : annotations) {
        System.out.println("\t" + annotation);
    }
} else {
    System.out.println("空");
}
}
```

多学两招：

通常只能通过 API 来查看类的定义，不过 Java 反射还提供了另一种方式来获得类的信息，读者也可以在程序中使用这些信息。另外，使用 `getInterfaces()` 方法也可以获得对象类的所有接口，但是不包含泛型信息。即使是 `getSuperclass()` 方法也不能获得有泛型信息的父类。

技术要点

Class 类的实例表示正在运行的 Java 应用程序中的类和接口。枚举是一种类，注释是一种接口。每个数组属于被映射为 Class 对象的一个类，所有具有相同元素类型和维数的数组都共享该 Class 对象。基本的 Java 类型（boolean、byte、char、short、int、long、float 和 double）和关键字 void 也表示为 Class 对象。它没有公共构造方法。Class 对象是在加载类时由 Java 虚拟机以及通过调用类加载器中的 `defineClass()` 方法自动构造的。本实例使用的方法如表 7.1 所示。

表 7.1 Class 类的常用方法

方 法 名	作 用
<code>forName(String className)</code>	根据给定的名称获得 Class 对象
<code>getAnnotations()</code>	返回此 Class 对象上存在的注释
<code>getCanonicalName()</code>	返回 Java Language Specification 中所定义的底层类的规范化名称
<code>getGenericInterfaces()</code>	返回泛型形式的对象类所实现的接口
<code>getGenericSuperclass()</code>	返回泛型形式的对象类所直接继承的超类
<code>getModifiers()</code>	返回此类或接口以整数编码的 Java 语言修饰符
<code>getTypeParameters()</code>	按声明顺序返回 TypeVariable 对象的一个数组

实例 072 查看类的成员

（实例位置：配套资源\SL\07\072）

实例说明

在一个类的内部，一般包括成员变量、构造方法、普通方法和内部类等。使用反射机制可以在无法查看源代码的情况下查看类的成员。本实例将使用反射机制查看 `ArrayList` 类中定义的成员变量、构造方法和普通方法。实例的运行效果如图 7.17 所示。

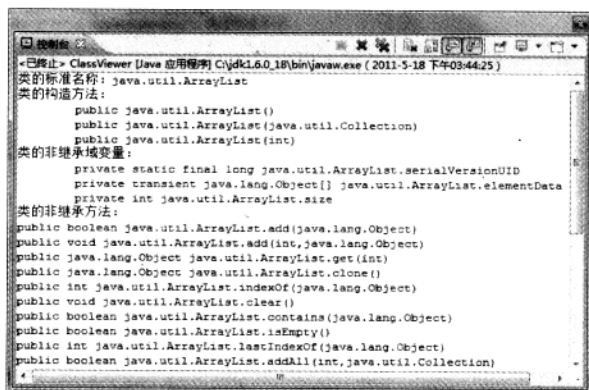


图 7.17 查看类的成员

实现过程

(1) 在 Eclipse 中创建项目 072，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写类 ClassViewer，在 main() 方法中输出类中定义的构造方法、域和普通方法。关键代码如下：

```
public class ClassViewer {
    @SuppressWarnings("unchecked")
    public static void main(String[] args) throws ClassNotFoundException {
        Class<?> clazz = Class.forName("java.util.ArrayList");
        System.out.println("类的标准名称: " + clazz.getCanonicalName());
        Constructor[] constructors = clazz.getConstructors(); //获得该类对象的所有构造方法
        System.out.println("类的构造方法: ");
        if (constructors.length != 0) {
            for (Constructor constructor : constructors) {
                System.out.println("\t" + constructor); //输出构造方法
            }
        } else {
            System.out.println("\t空");
        }
        Field[] fields = clazz.getDeclaredFields(); //获得该类对象的所有非继承域
        System.out.println("类的非继承域变量: ");
        if (fields.length != 0) {
            for (Field field : fields) {
                System.out.println("\t" + field); //输出非继承域
            }
        } else {
            System.out.println("\t空");
        }
        Method[] methods = clazz.getDeclaredMethods(); //获得该类对象的所有非继承方法
        System.out.println("类的非继承方法: ");
        if (methods.length != 0) {
            for (Method method : methods) {
                System.out.println(method); //输出非继承方法
            }
        }
    }
}
```





Note

```

    } else {
        System.out.println("\t 空");
    }
}
}

```

技术要点

Class 类的实例表示正在运行的 Java 应用程序中的类和接口。枚举是一种类，注释是一种接口。每个数组属于被映射为 Class 对象的一个类，所有具有相同元素类型和维数的数组都共享该 Class 对象。基本的 Java 类型（boolean、byte、char、short、int、long、float 和 double）和关键字 void 也表示为 Class 对象。它没有公共构造方法。Class 对象是在加载类时由 Java 虚拟机以及通过调用类加载器中的 `defineClass()` 方法自动构造的。本实例使用的方法如表 7.2 所示。

表 7.2 Class 类的常用方法

方法名	作用
<code>getConstructors()</code>	返回由该类对象的所有构造方法组成的数组
<code>getDeclaredFields()</code>	返回由该类对象的所有非继承域组成的数组
<code>getDeclaredMethods()</code>	返回由该类对象的所有非继承方法组成的数组

实例 073 查看内部类信息

（实例位置：配套资源\SL\07\073）

实例说明

Java 中支持在类的内部定义类，这种类称为内部类。内部类有些像 Java 中的方法，可以使用访问权限限定符修饰，也可以使用 `static` 关键字修饰等。本实例将利用 Java 的反射机制来查看内部类的信息。实例的运行效果如图 7.18 所示。

实现过程

（1）在 Eclipse 中创建项目 073，并在该项目中创建 `com.mingrisoft` 包。

（2）在 `com.mingrisoft` 包中编写类 `NestedClassInformation`，在该类的 `main()` 方法中输出内部类的信息。关键代码如下：

```

public class NestedClassInformation {
    public static void main(String[] args) throws ClassNotFoundException {
        Class<?> cls = Class.forName("java.awt.geom.Point2D");
        Class<?>[] classes = cls.getDeclaredClasses();//获得代表内部类的 Class 对象组成的数组
        for (Class<?> clazz : classes) { //遍历 Class 对象数组
            System.out.println("类的标准名称: " + clazz.getCanonicalName());
            System.out.println("类的修饰符: " + Modifier.toString(clazz.getModifiers()));
            Type[] interfaces = clazz.getGenericInterfaces(); //获得所有泛型接口
        }
    }
}

```

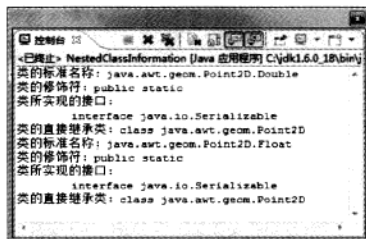


图 7.18 查看内部类信息



```

System.out.println("类所实现的接口: ");
if (interfaces.length != 0) { //如果泛型接口个数不是 0 则输出
    for (Type type : interfaces) {
        System.out.println("\t" + type);
    }
} else {
    System.out.println("\t" + "空");
}
Type superClass = clazz.getGenericSuperclass(); //获得直接父类
System.out.print("类的直接继承类: ");
if (superClass != null) { //如果直接父类不是 Object 就输出
    System.out.println(superClass);
} else {
    System.out.println("空");
}
}
}

```

技术要点

本实例主要应用 Class 类的 `getDeclaredClasses()` 方法获得代表内部类的 Class 对象组成的数组。Class 类的 `getDeclaredClasses()` 方法将返回 Class 对象的一个数组，这些对象包含声明为此 Class 对象所表示的类的成员的所有类和接口。包括该类所声明的公共、保护、默认（包）访问及私有类和接口，但不包括继承的类和接口。如果该类没有将任何类或接口声明为成员，或者此 Class 对象表示基本类型、数组类或 void，则此方法将返回一个长度为 0 的数组。该方法的声明如下：

```
public Class<?>[] getDeclaredClasses() throws SecurityException
```

对于私有的域或方法如果有安全管理器则可能会出现异常。

实例 074 动态设置类的私有域

（实例位置：配套资源\SL\07\074）

实例说明

为了保证面向对象的封装特性，通常会将域设置成私有的，然后提供对应的 `getXXX()` 和 `setXXX()` 方法。对于非内部类而言，只能使用 `getXXX()` 和 `setXXX()` 方法来操作该域。然而利用反射机制，就可以在运行时修改类的私有域。本实例将通过简单的 Student 类来演示反射的这种用法。实例的运行效果如图 7.19 所示。

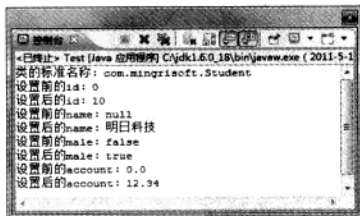


图 7.19 动态设置类的私有域

实现过程

（1）在 Eclipse 中创建项目 074，并在该项目中创建 com.mingrisoft 包。





(2) 在 com.mingrisoft 包中编写类 Student, 在该类中定义 4 个域及其对应的 getXXX() 和 setXXX() 方法。这 4 个域分别是 id (表示学生的序号)、name (表示学生的姓名)、male (表示学生是否为男性) 和 account (表示学生的账户余额)。关键代码如下:

```
public class Student {  
    private int id;                //表示学生的序号  
    private String name;          //表示学生的姓名  
    private boolean male;         //表示学生的性别  
    private double account;       //表示学生的账户余额  
    //省略了各个域的 getXXX()和 setXXX()方法  
}
```

(3) 在 com.mingrisoft 包中再编写类 Test 进行测试。在 main() 方法中, 分别为不同的域设置不同的值, 并输出初始值和新值作为对比。关键代码如下:

```
public class Test {  
    public static void main(String[] args) {  
        Student student = new Student();  
        Class<?> clazz = student.getClass();          //获得代表 student 对象的 Class 对象  
        System.out.println("类的标准名称: " + clazz.getCanonicalName());  
        try {  
            Field id = clazz.getDeclaredField("id");  
            System.out.println("设置前的 id: " + student.getId());  
            id.setAccessible(true);  
            id.setInt(student, 10);                    //设置 id 值为 10  
            System.out.println("设置后的 id: " + student.getId());  
  
            Field name = clazz.getDeclaredField("name");  
            System.out.println("设置前的 name: " + student.getName());  
            name.setAccessible(true);  
            name.set(student, "明日科技");             //设置 name 值为明日科技  
            System.out.println("设置后的 name: " + student.getName());  
  
            Field male = clazz.getDeclaredField("male");  
            System.out.println("设置前的 male: " + student.isMale());  
            male.setAccessible(true);  
            male.setBoolean(student, true);            //设置 male 值为 true  
            System.out.println("设置后的 male: " + student.isMale());  
  
            Field account = clazz.getDeclaredField("account");  
            System.out.println("设置前的 account: " + student.getAccount());  
            account.setAccessible(true);  
            account.setDouble(student, 12.34);         //设置 account 值为 12.34  
            System.out.println("设置后的 account: " + student.getAccount());  
        } catch (SecurityException e) {  
            e.printStackTrace();  
        } catch (NoSuchFieldException e) {  
            e.printStackTrace();  
        } catch (IllegalArgumentException e) {  
            e.printStackTrace();  
        } catch (IllegalAccessException e) {  
            e.printStackTrace();  
        }  
    }  
}
```



Note

```
        e.printStackTrace();
    }
}
```

技术要点

本实例主要应用了 Field 类的相关方法来实现动态设置类的私有域。Field 类提供有关类或接口的单个字段的信息，以及对它的动态访问权限。反射的字段可能是一个类（静态）字段或实例字段。本实例使用的方法如表 7.3 所示。

表 7.3 Field 类的常用方法

方 法 名	作 用
set(Object obj, Object value)	将指定对象变量上 Field 对象表示的字段设置为指定的新值
setBoolean(Object obj, boolean z)	将字段的值设置为指定对象上的一个 boolean 值
setDouble(Object obj, double d)	将字段的值设置为指定对象上的一个 double 值
setInt(Object obj, int i)	将字段的值设置为指定对象上的一个 int 值
setAccessible(boolean flag)	将此对象的 accessible 标志设置为指定的布尔值

脚下留神：

对于私有域，一定要使用 setAccessible()方法将其可见性设置为 true 才能设置新值。

实例 075 动态调用类中方法

（实例位置：配套资源\SL\07\075）

实例说明

在 Java 中，调用类的方法有两种方式：对于静态方法可以直接使用类名调用，对于非静态方法必须使用类的对象调用。反射机制提供了比较另类的调用方式，可以根据需要指定要调用的方法，而不必在编程时确定。调用的方法不仅限于 public 的，还可以是 private 的。本实例将使用反射机制调用 Math 类的静态方法 sin()和 String 类的非静态方法 equals()。实例的运行效果如图 7.20 所示。

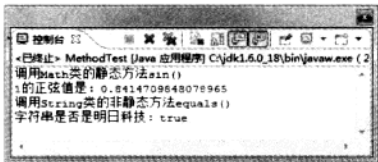


图 7.20 动态调用类中的方法

实现过程

- （1）在 Eclipse 中创建项目 075，并在该项目中创建 com.mingrisoft 包。
- （2）在 com.mingrisoft 包中编写类 MethodTest，在该类的主方法 main()中，分别调用了 Math 类的静态方法 sin()和 String 类的非静态方法 equals()。关键代码如下：

```
public class MethodTest {
    public static void main(String[] args) {
        try {
            System.out.println("调用 Math 类的静态方法 sin()");
        }
    }
}
```



Note

```
Method sin = Math.class.getDeclaredMethod("sin", Double.TYPE);
Double sin1 = (Double) sin.invoke(null, new Integer(1));
System.out.println("1 的正弦值是: " + sin1);
System.out.println("调用 String 类的非静态方法 equals()");
Method equals = String.class.getDeclaredMethod("equals", Object.class);
Boolean mrsoft = (Boolean) equals.invoke(new String("明日科技"), "明日科技");
System.out.println("字符串是否是明日科技: " + mrsoft);
} catch (Exception e) {
    e.printStackTrace();
}
}
```

指点迷津:

由于这几行代码会抛出大量异常, 因此采用捕获 Exception 代替。通常不推荐这种写法。

技术要点

本实例主要通过 Method 类的相关方法实现。Method 类提供类或接口上单独某个方法(以及如何访问该方法)的信息。所反映的方法可能是类方法或实例方法(包括抽象方法)。它允许在匹配要调用的实参与底层方法的形参时进行扩展转换。但是如果要进行收缩转换, 则会抛出异常 IllegalArgumentException。使用 Method 类的 invoke() 方法可以实现动态调用方法, 该方法的声明如下:

```
public Object invoke(Object obj, Object... args) throws IllegalAccessException, IllegalArgumentException,
InvocationTargetException
```

参数说明

- ☒ obj: 从中调用底层方法的对象。
- ☒ args: 用于方法调用的参数。

实例 076 动态实例化类

(实例位置: 配套资源\SL\07\076)

实例说明

在 Java 中, 通常是使用构造方法来创建对象的。构造方法可以分成有参数和无参数两种。如果类中没有定义构造方法, 编译器会自动添加一个无参数的构造方法。使用构造方法创建对象虽然非常常用, 但是并不灵活。本实例将演示如何使用反射创建对象。实例的运行效果如图 7.21 所示。

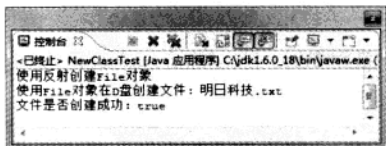


图 7.21 动态实例化类

实现过程

- (1) 在 Eclipse 中创建项目 076, 并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中编写类 NewClassTest, 在该类的 main() 方法中, 创建 File 对象



并使用该对象在 D 盘创建一个文本文件。关键代码如下：

```
public class NewClassTest {
    public static void main(String[] args) {
        try {
            //获得 File 类的 Constructor 对象
            Constructor<File> constructor =
                File.class.getDeclaredConstructor(String.class);
            System.out.println("使用反射创建 File 对象");
            File file = constructor.newInstance("d://明日科技.txt");
            System.out.println("使用 File 对象在 D 盘创建文件: 明日科技.txt");
            file.createNewFile();
            //创建新的文件
            System.out.println("文件是否创建成功: " + file.exists());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



Note

指点迷津：

由于这几行代码会抛出大量异常，因此采用捕获 Exception 代替。通常不推荐这种写法。

技术要点

本实例主要应用 Constructor 类及其相关方法实现。Constructor 类提供类的单个构造方法的信息以及对它的访问权限。它允许在将实参与带有底层构造方法的形参的 newInstance() 匹配时进行扩展转换，但是如果发生收缩转换，则抛出 IllegalArgumentException 异常。newInstance() 方法可以使用指定的参数来创建对象，该方法的声明如下：

```
public T newInstance(Object... initargs) throws InstantiationException, IllegalAccessException,
    IllegalArgumentException, InvocationTargetException
```

参数说明

initargs: 作为变量传递给构造方法调用的对象数组。

实例 077 创建长度可变的数组

(实例位置：配套资源\SL\07\077)

实例说明

在 Java 中，对于数组的支持并不强大。程序员必须时刻注意数组中元素的个数，否则会出现数组下标越界异常。为此，在 Java API 中定义了 ArrayList 帮助开发，但这意味着需要学习新的方法。本实例将使用反射机制实现一个工具方法，每当调用该方法时数组的长度就会增加 5。实例的运行效果如图 7.22 所示。

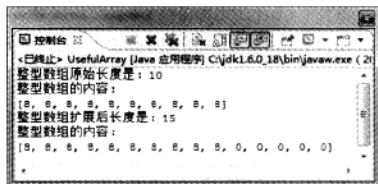


图 7.22 创建长度可变的数组

实现过程

(1) 在 Eclipse 中创建项目 077，并在该项目中创建 com.mingrisoft 包。



(2) 在 com.mingrisoft 包中编写类 UsefulArray。在该类中定义两个方法,一个是 increaseArray() 方法,用于将给定的 array 数组长度加 5;另一个是 main() 方法,用来进行测试。关键代码如下:

```
public class UsefulArray {
    public static Object increaseArray(Object array) {
        Class<?> clazz = array.getClass();           //获得代表数组的 Class 对象
        if (clazz.isArray()) {                        //如果输入是一个数组
            Class<?> componentType = clazz.getComponentType(); //获得数组元素的类型
            int length = Array.getLength(array);        //获得输入的数组的长度
            Object newArray = Array.newInstance(componentType, length + 5); //新建数组
            System.arraycopy(array, 0, newArray, 0, length); //复制原来数组中的所有数据
            return newArray;                            //返回新建数组
        }
        return null;                                   //如果输入的不是数组就返回空
    }
    public static void main(String[] args) {
        int[] intArray = new int[10];
        System.out.println("整型数组原始长度是: " + intArray.length);
        Arrays.fill(intArray, 8);                     //将数组中的元素全部赋值为 8
        System.out.println("整型数组的内容: ");
        System.out.println(Arrays.toString(intArray));
        int[] newIntArray = (int[]) increaseArray(intArray); //增加数组的长度
        System.out.println("整型数组扩展后长度是: " + newIntArray.length);
        System.out.println("整型数组的内容: ");
        System.out.println(Arrays.toString(newIntArray));
    }
}
```

指点迷津:

读者可以在本实例的基础上实现删除数组中的数据等方法。

技术要点

Array 类提供了动态创建和访问 Java 数组的方法。Array 允许在执行 getXXX()或 setXXX()方法操作期间进行扩展转换,但如果发生收缩转换,则抛出 IllegalArgumentException 异常。为了创建新的数组对象,需要使用 newInstance()方法,它可以根据指定的元素类型和长度创建新的数组。该方法的声明如下:

```
public static Object newInstance(Class<?> componentType, int length) throws NegativeArraySizeException
```

参数说明

- ☒ componentType: 表示新数组的组件类型的 Class 对象。
- ☒ length: 新数组的长度。

为了获得给定数组的长度,需要使用 getLength()方法,该方法的声明如下:

```
public static int getLength(Object array) throws IllegalArgumentException
```

参数说明

array: 一个数组。

指点迷津:

对于 Java 的数组,不管几维,都属于 Object 类型。



实例 078 利用反射重写 toString() 方法

(实例位置: 配套资源\SL\07\078)



Note

实例说明

为了输出对象方便, Object 类提供了 toString() 方法。但是该方法的默认值是由类名和哈希码组成的, 实用性并不强。通常需要重写该方法以提供更多的信息。本实例主要实现重写类的 toString() 方法, 用于在调用 toString() 方法时, 使用反射输出类的包、类的名字、类的公共构造方法、类的公共域和类的公共方法。另外, 在重写该方法时, 为其传递一个 Object 型的参数, 这样可以避免多次重写 toString() 方法。实例的运行效果如图 7.23 所示。

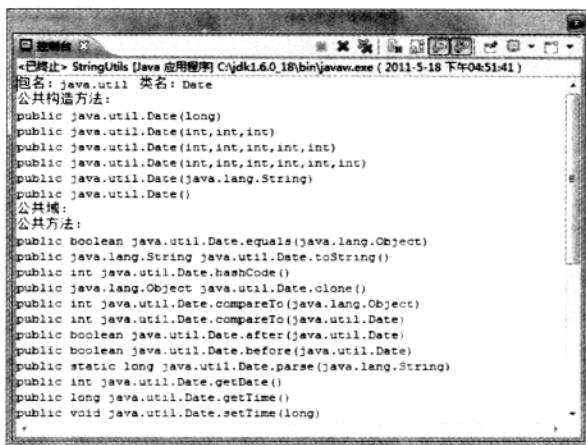


图 7.23 利用反射重写 toString() 方法

实现过程

(1) 在 Eclipse 中创建项目 078, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写类 StringUtils。在该类中定义两个方法, 一个是 toString() 方法, 用于输出类的公共方法和域等信息; 另一个是 main() 方法, 用来进行测试。关键代码如下:

```
public class StringUtils {
    @SuppressWarnings("unchecked")
    public String toString(Object object) {
        Class clazz = object.getClass();           //获得代表该类的 Class 对象
        StringBuilder sb = new StringBuilder();     //利用 StringBuilder 来保存字符串
        Package packageName = clazz.getPackage();  //获得类所在的包
        sb.append("包名: " + packageName.getName() + "\n"); //输出类所在的包
        String className = clazz.getSimpleName();  //获得类的简单名称
        sb.append("类名: " + className + "\n");     //输出类的简单名称
        sb.append("公共构造方法: \n");
        //获得所有代表构造方法的 Constructor 数组
        Constructor[] constructors = clazz.getDeclaredConstructors();
        for (Constructor constructor : constructors) {
```




Note

```
String modifier = Modifier.toString(constructor.getModifiers()); //获得修饰符
if (modifier.contains("public")) { //查看修饰符是否含有 "public"
    sb.append(constructor.toGenericString() + "\n");
}
}
sb.append("公共域: \n");
Field[] fields = clazz.getDeclaredFields(); //获得代表所有域的 Field 数组
for (Field field : fields) {
    String modifier = Modifier.toString(field.getModifiers());
    if (modifier.contains("public")) { //查看修饰符是否含有 "public"
        sb.append(field.toGenericString() + "\n");
    }
}
sb.append("公共方法: \n");
Method[] methods = clazz.getDeclaredMethods(); //获得代表所有方法的 Method 数组
for (Method method : methods) {
    String modifier = Modifier.toString(method.getModifiers());
    if (modifier.contains("public")) { //查看修饰符是否含有 "public"
        sb.append(method.toGenericString() + "\n");
    }
}
return sb.toString();
}
public static void main(String[] args) {
    System.out.println(new StringUtils().toString(new java.util.Date()));
}
}
```

技术要点

本实例应用到的技术与实例 072 介绍的相同, 这里不再赘述。

第 8 章

字符串与包装类

本章读者可以学到如下实例：

- » 实例 079 将数字格式化为货币字符串
- » 实例 080 货币金额大写格式
- » 实例 081 String 类格式化当前日期
- » 实例 082 字符串大小写转换
- » 实例 083 字符与 Unicode 码的转换
- » 实例 084 判断用户名是否正确
- » 实例 085 用户名排序
- » 实例 086 判断网页请求与 FTP 请求
- » 实例 087 判断文件类型
- » 实例 088 判断字符串是否为数字
- » 实例 089 验证 IP 地址的有效性
- » 实例 090 鉴别非法电话号码
- » 实例 091 将字符串转换成整数
- » 实例 092 整数进制转换器
- » 实例 093 获取字符串中汉字的个数
- » 实例 094 批量替换某一类字符串
- » 实例 095 查看数字的取值范围
- » 实例 096 ASCII 编码查看器
- » 实例 097 判断手机号的合法性
- » 实例 098 用字符串构建器追加字符
- » 实例 099 去掉字符串中的所有空格
- » 实例 100 Double 类型的比较



实例 079 将数字格式化为货币字符串

(实例位置: 配套资源\SL\08\079)

实例说明

数字可以标识货币、百分比、积分、电话号码等,就货币而言,在不同的国家会以不同的格式来定义。本实例将接收用户输入的数字,将这个数字转换为不同国家的货币格式,然后在控制台中输出这些货币格式。实例的运行效果如图 8.1 所示。

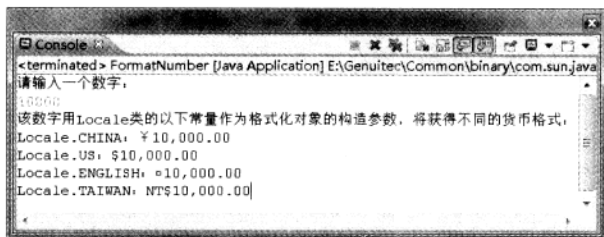


图 8.1 将数字格式化为货币字符串

实现过程

本实例首先创建 `FormatNumber` 类,在该类的主方法中创建标准输入流的扫描器对象,利用该对象从键盘上接收一个需要转换为货币格式的数字,通过 `NumberFormat` 类的 `format()` 方法把接收的数字格式转换为货币字符串。关键代码如下:

```
public class FormatNumber {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);           //创建标注输入流扫描器
        System.out.println("请输入一个数字:");
        double number = scan.nextDouble();                //获取用户输入数字
        System.out.println("该数字用 Locale 类的常量作为格式化对象的构造参数, 将获得不同的货币格式:");
        NumberFormat format = NumberFormat.getCurrencyInstance(Locale.CHINA);
                                                                //创建格式化对象
        System.out.println("Locale.CHINA: " + format.format(number)); //输出格式化货币格式
        format = NumberFormat.getCurrencyInstance(Locale.US);
        System.out.println("Locale.US: " + format.format(number));
        format = NumberFormat.getCurrencyInstance(Locale.ENGLISH);
        System.out.println("Locale.ENGLISH: " + format.format(number));
        format = NumberFormat.getCurrencyInstance(Locale.TAIWAN);
        System.out.println("Locale.TAIWAN: " + format.format(number));
    }
}
```

技术要点

数字格式化是本实例的关键点,实例中应用 `NumberFormat` 类实现了数字格式化,这个类是一个抽象类,但是可以通过其静态方法获取内部实现类的实例对象,本实例获取了货币格式



的格式化对象。

1. 获取货币格式对象

使用 `getCurrencyInstance()` 方法获取 `NumberFormat` 类的货币格式对象。该方法的声明如下：

```
public static NumberFormat getCurrencyInstance(Locale inLocale);
```

参数说明

`inLocale`: 指定语言环境。

返回值: 返回指定语言环境的货币格式。

2. 执行格式化

`format()` 方法是格式化对象中的方法，用于执行针对数字的格式化操作，就本实例使用的货币格式化对象来说，这个方法执行的是将数字格式化为货币字符串。该方法的声明如下：

```
public final String format(double number);
```

参数说明

`number`: 要被格式化的数字。



Note

多学两招:

格式化对象可以指定语言环境，在 Java 中使用 `Local` 类的对象来表示，在该类中包含了各种语言环境。通过它可以获取国际化的字符串信息，如货币、日期时间等。

实例 080 货币金额大写格式

(实例位置: 配套资源\SL\08\080)

实例说明

在处理财务账款时，一般需要使用大写金额。如进行转账时，需要将转账金额写成大写的。也就是说，如果要转账 123456.00 元，则需要写成“壹拾贰万叁仟肆佰伍拾陆元整”。对于这种情况，如果手动填写不仅麻烦，而且容易出错，所以常常需要通过程序控制自动进行转换。本实例实现了小写金额到大写金额的转换，实例的运行效果如图 8.2 所示。

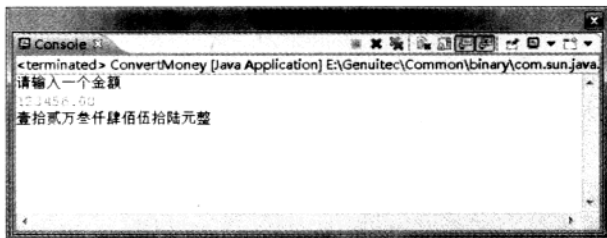


图 8.2 将金额转换为大写格式

实现过程

(1) 在项目中创建 `ConvertMoney` 类，在该类的主方法中接收用户输入的金额，然后通过 `convert()` 方法把这个金额转换成大写金额的字符串格式，并输出到控制台。



Note

```

public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);           //创建扫描器
    System.out.println("请输入一个金额");
    String convert = convert(scan.nextDouble());      //获取金额转换后的字符串
    System.out.println(convert);                      //输出转换结果
}

```

(2) 编写金额转换方法 convert(), 该方法在主方法中被调用, 用于金额大写格式的转换。在该方法中创建 DecimalFormat 类的实例对象, 通过这个格式器对象把金额数字格式化, 值保留 3 位小数。然后分别调用 getInteger() 与 getDecimal() 方法转换整数与小数部分, 并返回转换后的结果。关键代码如下:

```

public static String convert(double d) {
    DecimalFormat df = new DecimalFormat("#0.###"); //实例化 DecimalFormat 对象
    String strNum = df.format(d);                  //格式化 double 数字
    if (strNum.indexOf(".") != -1) {                //判断是否包含小数点
        String num = strNum.substring(0, strNum.indexOf("."));
        if (num.length() > 12) {                    //整数部分大于 12 不能转换
            System.out.println("数字太大, 不能完成转换!");
            return "";
        }
    }
    String point = "";                              //小数点
    if (strNum.indexOf(".") != -1) {
        point = "元";
    } else {
        point = "元整";
    }
    String result = getInteger(strNum) + point + getDecimal(strNum); //转换结果
    if (result.startsWith("元")) {                  //判断字符串是否以“元”结尾
        result = result.substring(1, result.length()); //截取字符串
    }
    return result;                                  //返回新的字符串
}

```

(3) 编写 getInteger() 方法, 用于转换数字整数部分的大写格式。在该方法中判断数字是否包含小数点, 然后把数字转换为字符串并反转字符顺序, 为每个数字添加对应的大写单位。关键代码如下:

```

public static String getInteger(String num) {
    if (num.indexOf(".") != -1) {                  //判断是否包含小数点
        num = num.substring(0, num.indexOf("."));
    }
    num = new StringBuffer(num).reverse().toString(); //反转字符串
    StringBuffer temp = new StringBuffer();        //创建一个 StringBuffer 对象
    for (int i = 0; i < num.length(); i++) {        //加入单位
        temp.append(STR_UNIT[i]);
        temp.append(STR_NUMBER[num.charAt(i) - 48]);
    }
    num = temp.reverse().toString();                //反转字符串
    num = num.replace("零拾", "零");                //替换字符串的字符
    num = num.replace("零佰", "零");                //替换字符串的字符
}

```



```

num = numReplace(num, "零仟", "零");           //替换字符串的字符
num = numReplace(num, "零万", "万");           //替换字符串的字符
num = numReplace(num, "零亿", "亿");           //替换字符串的字符
num = numReplace(num, "零零", "零");           //替换字符串的字符
num = numReplace(num, "亿万", "亿");           //替换字符串的字符
if (num.lastIndexOf("零") == num.length() - 1) { //如果字符串以零结尾将其除去
    num = num.substring(0, num.length() - 1);
}
return num;
}

```



Note

技术要点

实现本实例关键在于以下几点:

- ☑ 将数字格式化, 如果存在小数部分, 将其转换为 3 位小数, 精确到厘。
- ☑ 分别将整数部分与小数部分转换为大写方式, 并插入其单位 (亿、万、仟……)。
- ☑ 组合转换后的整数部分与小数部分。

多学两招:

DecimalFormat 类可以指定格式化模板来格式化浮点数, 如保留几位小数。通过调用该类的 format() 方法可以使用指定模板来格式化任意浮点数字。

实例 081 String 类格式化当前日期

(实例位置: 配套资源\SL\08\081)

实例说明

在输出日期信息时, 经常需要输出不同格式的日期格式, 本实例中介绍了 String 字符串类中的日期格式化方法, 实例使用不同的方式输出 String 类的日期格式参数值, 组合这些值可以实现特殊格式的日期字符串。实例的运行效果如图 8.3 所示。

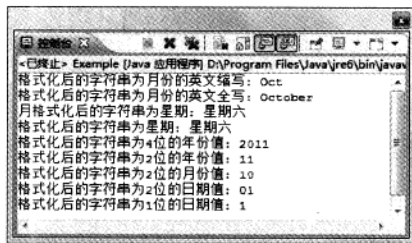


图 8.3 格式化当前日期

实现过程

(1) 在项目中创建 Example 类, 在该类的主方法中创建一个 Date 日期类的实例对象 today。

(2) 利用 format() 方法格式化 today 日期对象, 关键代码如下:

```

public class Example {
    public static void main(String[] args) {
        Date today = new Date();
        String a = String.format(Locale.US, "%tb", today); //格式化后的字符串为月份的英文缩写
        System.out.println("格式化后的字符串为月份的英文缩写: " + a);
        String b = String.format(Locale.US, "%tB", today); //格式化后的字符串为月份的英文全写
        System.out.println("格式化后的字符串为月份的英文全写: " + b);
    }
}

```




Note

```
String c = String.format("%ta", today); //格式化后的字符串为星期（如星期一）
System.out.println("月格式化后的字符串为星期：" + c);
String d = String.format("%tA", today); //格式化后的字符串为星期（如星期二）
System.out.println("格式化后的字符串为星期：" + d);
String e = String.format("%tY", today); //格式化后的字符串为 4 位的年份值
System.out.println("格式化后的字符串为 4 位的年份值：" + e);
String f = String.format("%ty", today); //格式化后的字符串为 2 位的年份值
System.out.println("格式化后的字符串为 2 位的年份值：" + f);
String g = String.format("%tm", today); //格式化后的字符串为 2 位的月份值
System.out.println("格式化后的字符串为 2 位的月份值：" + g);
String h = String.format("%td", today); //格式化后的字符串为 2 位的日期值
System.out.println("格式化后的字符串为 2 位的日期值：" + h);
String i = String.format("%te", today); //格式化后的字符串为 1 位的日期值
System.out.println("格式化后的字符串为 1 位的日期值：" + i);
}
}
```

技术要点

使用 `String` 类的 `format()` 方法不但可完成日期的格式化，也可实现时间的格式化。时间格式化转换符要比日期转换符更多、更精确，它可以将时间格式化为时、分、秒、毫秒。

多学两招：

在深入使用字符串之前，有一个概念一定要理解，字符串是不可变的对象。理解了这一概念，对后面熟练使用字符串有着很大的帮助。字符串的不可变性，意味着每当对字符串进行操作时，都将产生一个新的字符串对象，如果频繁地操作字符串对象，会在托管堆中产生大量无用字符串，增加垃圾收集器的压力，从而造成系统资源的浪费。

实例 082 字符串大小写转换

（实例位置：配套资源\SL\08\082）

实例说明

在程序设计过程中，经常会遇到一种情况，在验证用户登录时，如果用户名不区分大小写，那么在代码中应当使用一种方法排除字母大小写的因素，然后再对比数据库中的用户名与用户输入的用户名是否相等。可以先将数据库中的用户名全部转为大写，再将用户输入的用户名转为大写，最后对比是否相等。本实例中所介绍的技术可以方便地将字符串中的字母全部转换为大写或小写。实例的运行效果如图 8.4 所示。

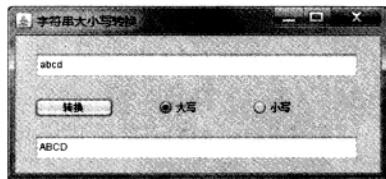


图 8.4 大小写转换

实现过程

- （1）创建窗体，在窗体中放置两个文本框，一个用于输入字符串，另一个用于显示结果，



添加两个单选按钮和一个“转换”按钮控件。

(2) 编写“转换”按钮的事件处理方法,在该方法中获取用户输入的字符串,根据用户的选择进行大小写格式转换并把结果输出到文本框中。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent arg0) {
    String command = buttonGroup.getSelection().getActionCommand();
    //获取“大写”和“小写”单选按钮的选中
    boolean upper = command.equals("大写"); //判断是否选中“大写”单选按钮
    String text = inputTextField.getText(); //获取输入字符串
    if (upper) { //大写转换
        outputTextField.setText(text.toUpperCase());
    } else { //小写转换
        outputTextField.setText(text.toLowerCase());
    }
}
```



Note

指点迷津:

字符串在创建后就成为不可变的对象,当调用字符串对象的方法操作字符串时,会产生新的字符串对象,而不是更改原来的字符串对象。

技术要点

本实例实现时主要用到了字符串对象的 `toUpperCase()` 和 `toLowerCase()` 方法,下面对其进行详细讲解。

使用字符串对象的 `toUpperCase()` 方法可以将字符串中的字母全部转换为大写。格式如图 8.5 所示。



图 8.5 调用字符串对象的 `toUpperCase()` 方法将字母全部转为大写

从图 8.5 中可以看到,字符串对象调用 `toUpperCase()` 方法后,会返回一个将原字符串转换为大写的字符串,并将新字符串的引用交给 `strBook` 变量。

使用字符串对象的 `toLowerCase()` 方法可以将字符串中的字母全部转换为小写。格式如图 8.6 所示。

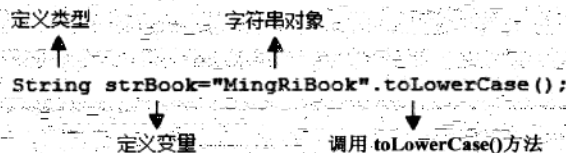


图 8.6 调用字符串对象的 `toLowerCase()` 方法将字母全部转为小写

从图 8.6 中可以看到,字符串对象调用 `toLowerCase()` 方法后,会返回一个将原字符串转换为小写的字符串,并将新字符串的引用交给 `strBook` 变量。

**多学两招：**

单选按钮控件需要分组规划，例如本实例中的两个单选按钮，如果不添加到 `ButtonGroup` 中，就无法显现单选操作，也就是说两个单选按钮可以同时处于选中状态。

**Note**

实例 083 字符与 Unicode 码的转换

(实例位置：配套资源\SL\08\083)

实例说明

Unicode 是一种字符编码，它可以显示各国语言的各种文字、标点、制表符等所有字符，也是现今最通用的字节编码系统。在程序设计中，可以方便地将字符转换为 Unicode 码，也可以将 Unicode 码转换为字符。实例的运行效果如图 8.7 所示。

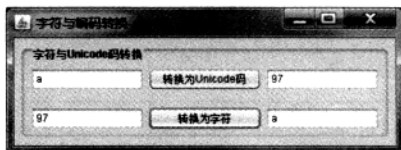


图 8.7 字符与 Unicode 码的转换

实现过程

(1) 在 Eclipse 中新建一个项目，在项目中新建窗体类 `CharacterASCII`，在该窗体中添加 4 个文本框和两个按钮，其中文本框用于接收用户输入的字符和编码以及输出转换结果。两个按钮分别用于控制字符到 Unicode 编码的转换和 Unicode 编码到字符的转换。

(2) 编写“转换为 Unicode 码”按钮的事件处理方法，在该方法中获取用户输入的字符串，然后从该字符串中提取字符数组，在遍历该数组的同时，把每个字符的编码输出到文本框。关键代码如下：

```
protected void do_codeButton_actionPerformed(ActionEvent e) {
    String text = charInputField.getText();           //获取用户输入的字符串
    char[] charArray = text.toCharArray();           //获取字符串的字符数组
    StringBuilder builder = new StringBuilder();       //创建字符串构建器
    for (char c : charArray) {                        //遍历字符数组
        builder.append((int) c + " ");               //连接各字符的编码
    }
    codeOutputField.setText(builder.toString());       //将结果输出到文本框
}
```

(3) 编写“转换为字符”按钮的事件处理方法，在该方法中获取用户输入的 Unicode 编码，然后通过强制类型转换，获取该编码对应的字符并输出到文本框中。关键代码如下：

```
protected void do_charButton_actionPerformed(ActionEvent e) {
    Number value = (Number) codeInputField.getValue(); //获取用户输入 Unicode 编码
    long code = value.longValue();                     //取输入数字的 Long 类型值
    charOutputField.setText(((char) code) + "");        //输出编码到文本框
}
```

指点迷津：

字符串就是由多个字符组成的，灵活地运用字符数组可以实现复杂的字符串操作，通过数组下标的各种算法可以使字符串更加灵活。



技术要点

本实例使用了字符串对象的 `toCharArray()` 方法获取字符数组，数组中的每个元素都是字符串的一部分。其声明语法如下：

```
public char[] toCharArray();
```

返回值：字符串中每个字符组成的字符数组。



Note

多学两招：

现在已经知道 `char` 是值类型，可以将字母强制类型转换为整数数值，从而方便地得到字母的 Unicode 编码。同样地，可以将整数数值强制类型转换为 `char`，从而得到对应编码的字符。

实例 084 判断用户名是否正确

（实例位置：配套资源\SL\08\084）

实例说明

在程序的开发过程中，经常需要判断用户输入的用户名是否正确，可以通过对比用户输入的用户名字符串是否与数据库中或者已经存在集合中的字符串相同来决定用户输入的用户名是否正确。Java 的基本数据类型可以使用“==”判断两个操作数是否相等，但是对于 Java 类创建的对象就不能使用这种方法来判断是否相等了。字符串是基本数据类型之外的，也就是说字符串在 Java 中是对象。本实例将通过字符串相等判断来实现用户名验证。实例的运行效果如图 8.8 所示。

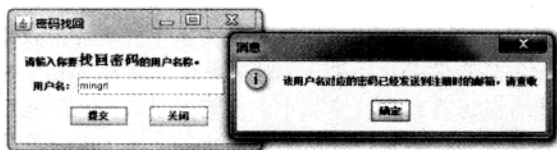


图 8.8 判断用户名是否正确

实现过程

（1）在项目中创建窗体类，在窗体中添加接收用户输入信息的文本框、“提交”和“关闭”按钮。

（2）编写“提交”按钮的事件处理方法，在该方法中接收用户输入的用户名，然后判断输入，如果不是管理员用户名并且输入的用户名是已经注册的则显示正确提示，否则显示错误提示。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String name = usernameField.getText();           //获取用户输入
    if (name.equals("admin")) {                       //判断是否是管理员账号
        JOptionPane.showMessageDialog(null, "对不起，这个用户名是管理员的，不是你的");
    } else if (name.equals("mingri")) {               //判断是否注册用户
        JOptionPane.showMessageDialog(null, "该用户名对应的密码已经发送到注册时的邮箱，
请查收");
    } else {                                           //给错误用户名的提示对话框
        JOptionPane.showMessageDialog(null, "你输入的用户名不存在，留意 Caps Lock 键是否
按下。");
    }
}
```



技术要点

本实例调用了 String 类的 equals() 方法来判断两个字符串内容是否相同, 这个方法是从 Object 类中继承的。在 Java 语言中, 默认所有类都是 Object 类的子类, 也就是说只要是对象, 都会重写或直接使用 Object 类的 equals() 方法, String 类就重写了这个方法实现判断字符串内容是否相同。其声明语法如下:

```
public boolean equals(Object anObject);
```

参数说明

anObject: 与当前字符串进行比较的对象。

返回值: 如果给定对象表示的 String 与此 String 相等, 则返回 true; 否则返回 false。

多学两招:

在 Java 虚拟机中有一个保存字符串的池, 它会记录所有字符串。例如:

```
String str1="abc";
```

```
String str2="abc";
```

```
String str3=new String("abc");
```

```
System.out.println(str1==str2);
```

```
System.out.println(str1==str3);
```

这段代码中 “str1==str2” 的判断将返回 true, 为什么这个等式会成立呢? Java 中基本数据类型使用 “=” 可以判断操作数是否相等, 对于对象使用这个符号判断的是两个对象的内存地址是否相同。而 Java 虚拟机为了提高字符串应用效率, 提供了字符串池来保存字符串常量, str1 创建字符串常量 “abc”, 这时会先检测字符串池中是否包含该字符串, 如果不包含, 则创建字符串常量保存到字符串池中, 然后再返回。str2 也赋值为字符串 “abc”, 这是由于字符串池中已经存在该字符串, 所以不再创建, 直接返回该字符串, 也就是说这两个变量引用同一个字符串, 那么它们的内存地址也是相同的, 所以 str1==str2 成立。但是使用 new 关键字创建的字符串会新开辟内存控件, 所以 str1==str3 不成立。

实例 085 用户名排序

(实例位置: 配套资源\SL\08\085)

实例说明

用户名称也就是登录系统、网站等使用的名称, 也称登录名称。一般情况下, 用户名都要求使用英文、数字与符号组成, 如 li_zhongwei。这些用户名一般是根据用户注册的先后顺序排序的, 这样不利于管理员的查找, 本实例实现对用户名字符串进行排序, 实例的运行效果如图 8.9 所示。

实现过程

(1) 在项目中创建窗体类 UserSort, 在窗体界面中添加一个 JList 列表控件和 “升序”、“降序”、“关闭” 3

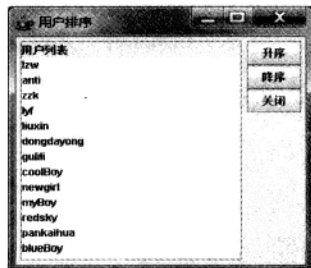


图 8.9 用户名排序



个按钮。

(2) 编写处理升序和降序按钮事件的方法, 该方法将创建线程对象在新的线程中完成排序, 并动态更新界面排序过程。关键代码如下:

```
protected void do_button_actionPerformed(final ActionEvent e) {
    new Thread() {
        int[] indexs = new int[2];
        public void run() {
            for (int i = names.length; --i >= 0;) {           //遍历数组
                indexs[0]=i;
                for (int j = 0; j < i; j++) {                 //遍历并排序所有未排序元素
                    boolean compare = names[j].compareToIgnoreCase(names[j+1]) > 0;
                    if (compare && e.getSource() == ascButton || !compare
                        && e.getSource() == descButton) { //条件判断
                        String temp = names[j];               //数组元素交换
                        names[j] = names[j+1];
                        names[j+1] = temp;
                        sourceList.repaint();
                    }
                }
                try {
                    sleep(100);
                } catch (InterruptedException e1) {
                }
                indexs[1]=j;
                sourceList.setSelectedIndices(indexs);
            }
        }
    }.start();
    sourceList.repaint();                                   //更新列表控件
}
```



Note

技术要点

用户名是以字符串形式保存的, 无论在数据库还是在数据集合中, 所以本实例使用了字符串的 `compareTo()` 和 `compareToIgnoreCase()` 方法实现字符串的对比。下面介绍本实例使用的关键方法。

1. `compareTo()` 方法

该方法将按字典顺序比较两个字符串。该方法比较基于字符串中各个字符的 Unicode 值。如果按字典顺序, 此 String 对象位于参数字符串之前, 则比较结果为一个负整数; 如果按字典顺序, 此 String 对象位于参数字符串之后, 则比较结果为一个正整数。其声明语法如下:

```
public int compareTo(String anotherString);
```

参数说明

anotherString: 要比较的 String 字符串对象。

返回值: 如果参数字符串等于此字符串, 则返回值 0; 如果此字符串按字典顺序小于字符串参数, 则返回一个小于 0 的值; 如果此字符串按字典顺序大于字符串参数, 则返回一个大于 0 的值。

2. `compareToIgnoreCase()` 方法

同上一个方法执行的功能相同, 这个方法也用于对比两个字符串, 但是不再严格区分字母



的大小写。其声明语法如下：

```
public int compareToIgnoreCase(String str);
```

参数说明

str: 要比较的 String 字符串对象。

返回值: 根据指定 String 大于、等于还是小于此 String (不考虑大小写), 分别返回一个负整数、0 或一个正整数。



Note

多学两招:

本实例使用了冒泡排序算法。冒泡排序是最常用的数组排序算法之一, 它排序数组元素的过程总是小数往前放, 大数往后放, 类似水中气泡往上升的动作, 所以称作冒泡排序。冒泡排序的基本思想是对比相邻的元素值, 如果满足条件就交换元素值, 把较小的元素移动到数组前面, 把大的元素移动到数组后面 (也就是交换两个元素的位置), 这样较小的元素就像气泡一样从底部上升到顶部。

实例 086 判断网页请求与 FTP 请求

(实例位置: 配套资源\SL\08\086)

实例说明

大家在访问 Internet 网络时, 经常涉及很多访问协议, 其中最明显、最常用的就是访问网页的 HTTP 协议、访问 FTP 服务器的 FTP 协议等。本实例实现对用户输入进行判断, 根据用户输入的不同请求字符串, 判断请求类型, 如图 8.10 所示。在文本框中输入请求字符串, 并单击“验证”按钮, 程序将提示用户输入的请求类型。



图 8.10 验证地址

实现过程

- (1) 在项目中创建窗体类, 在窗体中添加一个文本框和“验证”、“关闭”两个按钮。
- (2) 编写“验证”按钮的事件处理方法, 在该方法中获取用户输入的请求字符串, 然后通过 String 类的方法判断该字符串以 http 开始还是以 ftp 开始, 并以对话框提示请求类型。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {  
    String request = requestField.getText();           //获取用户输入  
    if (request.startsWith("http")) {                  //判断输入是否以 http 开头  
        JOptionPane.showMessageDialog(null, "您输入的是网页地址, 希望浏览某个网站。");  
    } else if (request.startsWith("ftp")) {             //判断输入是否以 ftp 开头  
        JOptionPane.showMessageDialog(null, "您输入的是 FTP 地址, 希望访问 FTP 服务器。");  
    } else {                                           //其他字符串开头认为信息不完整
```



```
JOptionPane.showMessageDialog(null, "您输入的请求信息不完整。");
```

```
}  
}
```

技术要点

本实例通过调用 `String` 类的 `startsWith()` 方法判断字符串的前缀，根据前缀来辨别请求的类型，该方法将判断字符串是否以指定的前缀开始。其声明语法如下：

```
public boolean startsWith(String prefix);
```

参数说明

`prefix`：字符串前缀。

返回值：如果参数表示的字符序列是此字符串表示的字符序列的前缀，则返回 `true`；否则返回 `false`。

多学两招：

Java 中一句相连的字符串不能分开在两行中写。例如：

```
System.out.println("I like  
Java")
```

修改为：这种写法是错误的，无法通过编译。如果一个字符串太长，为了便于阅读，可以将这个字符串分在两行上书写。此时就可以使用 “+” 将两个字符串连起来，之后在加号处换行。因此上面的语句可以修改为：

```
System.out.println("I like"+  
"Java");
```

实例 087 判断文件类型

（实例位置：配套资源\SL\08\087）

实例说明

在计算机中使用多种类型的文件来保存不同的数据，操作员和系统程序都根据不同的类型查找相应的数据。区分不同文件类型的依据就是文件的扩展名称。本实例利用字符串的判断方法来检测文件结尾的字符串后缀，并提示用户不同文件类型的说明信息，如图 8.11 所示。

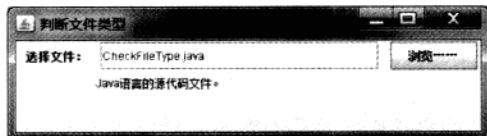


图 8.11 判断文件类型

实现过程

（1）在项目中创建窗体类 `CheckFileType`，在该窗体中添加一个文本框和一个“浏览”按钮，另外在“浏览”按钮和文本框下方还要添加一个 `JTextArea` 文本域控件，该控件用于显示用户选择文件类型的描述信息。





(2) 编写“浏览”按钮的事件处理方法,在该方法中创建说明文件的扫描器,这个说明文件是与程序存放在一起的 extName.inf 文件,笔者在该文件中添加了部分文件类型的描述信息。接着创建文件选择器,用户可以通过该选择器选择文件,然后判断文件扩展名称并从说明文件中提取对应的说明信息显示到文本域控件中。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    Scanner scan = new Scanner(getClass()           //获取说明文件的扫描器
        .getResourceAsStream("extName.inf"));
    JFileChooser chooser = new JFileChooser();        //创建文件选择器
    boolean searched = false;
    int option = chooser.showOpenDialog(this);        //打开文件选择对话框
    if (option == JFileChooser.APPROVE_OPTION) {      //如果正确选择文件
        File file = chooser.getSelectedFile();        //获取用户选择文件
        textField.setText(file.getName());           //把文件名添加到文本框
        String name = file.getName();               //获取文件名
        while (scan.hasNextLine()) {                //遍历说明文件
            String line = scan.nextLine();           //获取一行说明信息
            String[] extInfo = line.split("\t");      //把单行说明信息拆分成数组
            if (name.endsWith(extInfo[0])) {          //数组第一个元素是文件扩展名,与用户选择文件
                name对比
                textArea.setText(extInfo[1]);         //第二个数组元素是文件类型的说明信息,添
                加到文本域控件中
                searched = true;
            }
        }
        scan.close();                               //关闭扫描器
    }
    if (!searched) {                                //如果没找到相关文件类型的说明,则提示用户
        textArea.setText("你选择的文件类型没有相应记录,你可以在 extName.info 文件中添加
        该类型的描述。");
    }
}
```

技术要点

本实例使用了 String 字符串类的 endsWith() 方法来判断字符串结尾的后缀。该方法判断字符串是否以指定的后缀结尾。对于文件来说,结尾的后缀是文件扩展名,通过这个扩展名就可以判断文件类型,所以 endsWith() 方法最适合不过。其声明语法如下:

```
public boolean endsWith(String suffix);
```

参数说明

suffix: 后缀字符串。

返回值: 如果参数表示的字符序列是此对象表示的字符序列的后缀,则返回 true; 否则返回 false。

多学两招:

这个方法接收一个正则表达式字符串作为参数,通过这个表达式指定的用以分隔符来分割字符串为指定长度的字符串数组,但是如果被分割的字符串没有统一的分隔符,可以使用“|”定义多个分隔符。例如“|_|!”分别以“,”、“-”和“!”作为分隔符。



实例 088 判断字符串是否为数字

(实例位置: 配套资源\SL\08\088)



Note

实例说明

软件运行过程中,经常需要用户输入数值、货币值等信息,然后进行处理。由于用户输入只能是字符串类型,如果输入了非法的信息,如在货币值中输入了字母“a”以及其他非数字字符,那么在运行时会抛出异常,可以通过捕获异常来判断输入信息是否合法,但是,这样并不是好的处理方法,本实例将使用 NumberUtils 类中的方法处理此问题,让程序更加方便快捷。实例的运行效果如图 8.12 所示。

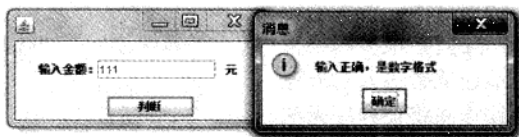


图 8.12 判断输入的是否是数字

实现过程

- (1) 在项目中创建窗体类 CheckNumber, 在窗体中添加一个文本框和一个“判断”按钮。
- (2) 编写“判断”按钮的事件处理方法, 在该方法中获取用户在文本框中输入的数字, 然后利用 NumberUtils 类的 isNumber() 方法判断字符串是不是有效的数字, 然后用对话框输出正确和错误的提示信息。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText();           //获取用户输入的金额字符串
    boolean isnum = NumberUtils.isNumber(text); //判断是不是数字
    if(isnum){                                   //输出正确提示信息
        JOptionPane.showMessageDialog(null, "输入正确, 是数字格式");
    }else{                                       //输出错误提示信息
        JOptionPane.showMessageDialog(null, "输入错误, 请确认格式再输入");
    }
}
```

技术要点

本实例使用了 Apache 提供的 lang 包中的 NumberUtils 类来实现数字判断, 该类的全路径为 “org.apache.commons.lang.math.NumberUtils”, 这个类中的 isNumber() 方法可以接收字符串参数, 然后对字符串进行解析, 如果字符串不能转换为数字格式, 则返回 false。其声明语法如下:

```
public static boolean isNumber(String str);
```

参数说明

str: 字符串。

返回值: 该方法将对该字符串进行判断, 如果是由数字组成则返回 true; 如果无法转换为数字, 则返回 false。

**多学两招:**

本实例还可以通过 Double 类的 parseDouble() 方法把字符串转换为 double 类型, 如果抛出异常说明字符串不是合法数字格式。但是建议不要使用这种方式作判断, 那会降低程序性能, 因为它无法与简单逻辑判断相比, 后者在速度上完全超越前者。

**Note****实例 089 验证 IP 地址的有效性**

(实例位置: 配套资源\SL\08\089)

实例说明

IP 地址是网络上每台计算机的标识, 在浏览器中输入的网址也是要经过 DNS 服务器转换为 IP 地址才能找到服务器的。在很多网络程序中要求设置服务器 IP 地址或者输入对方连接 IP 地址, IP 地址的错误输入将使程序无法运行。本实例实现对 IP 地址的验证功能, 实例的运行效果如图 8.13 所示, 把该功能加载到网络程序中, 可以避免用户输入错误的 IP 地址。



图 8.13 验证一个 IP 地址

实现过程

(1) 在项目中创建窗体类 CheckIPAddress, 在该窗体中添加一个输入 IP 地址的文本框和一个“验证”按钮。

(2) 编写“验证”按钮的事件处理方法, 该方法将获取用户输入, 然后调用 matches() 方法对输入进行判断, 然后在对话框中输出结果。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = ipField.getText();           //获取用户输入
    String info = matches(text);               //对输入文本进行 IP 验证
    JOptionPane.showMessageDialog(null, info); //用对话框输出验证结果
}
```

(3) 编写验证 IP 地址的 matches() 方法, 该方法利用正则表达式对输入字符串进行验证, 并返回验证结果。关键代码如下:

```
public String matches(String text) {
    if(text != null && !text.isEmpty()){
        String regex = "(1\\d{2}|2[0-4]\\d|25[0-5])[1-9]\\d{1-3}\\. +
                        \"(1\\d{2}|2[0-4]\\d|25[0-5])[1-9]\\d{1-3}\\. +
                        \"(1\\d{2}|2[0-4]\\d|25[0-5])[1-9]\\d{1-3}\\. +
                        \"(1\\d{2}|2[0-4]\\d|25[0-5])[1-9]\\d{1-3}$\"; //定义正则表达式
        if(text.matches(regex)){ //判断 IP 地址是否与正则表达式匹配
            return text + "\n 是一个合法的 IP 地址! "; //返回判断信息
        }else{
            return text + "\n 不是一个合法的 IP 地址! "; //返回判断信息
        }
    }
    return "请输入要验证的 IP 地址! "; //返回判断信息
}
```



技术要点

本实例的关键点在于 IP 地址格式与数字范围的验证,用户在输入 IP 地址时,程序可以获取的只有字符串类型,所以本实例利用字符串的灵活性与正则表达式搭配进行 IP 格式与范围的验证。该方法是 String 字符串类的方法,用于判断字符串与指定的正则表达式是否匹配。其声明语法如下:

```
public boolean matches(String regex);
```

参数说明

regex: 用来匹配此字符串的正则表达式。

返回值: 当且仅当此字符串符合给定的正则表达式条件时,返回 true。



Note

多学两招:

在正则表达式中,“.”代表任何一个字符,因此在正则表达式中如果想使用普通意义的点字符“.”,必须使用转义字符“\”。

实例 090 鉴别非法电话号码

(实例位置: 配套资源\SL\08\090)

实例说明

程序经常需要用户录入用户信息或联系方式,其中有一些数组的格式是固定的,程序处理逻辑也是按照这个格式来实现的,但是由于用户输入的是字符串,其灵活性较大,容易输入错误格式的数据。例如,用户联系信息的电话号码就是固定格式的数据,本实例将演示如何利用正则表达式来确定输入的电话号码格式是否匹配,实例的运行效果如图 8.14 所示,在程序中加入该模块可以禁止用户输入错误的电话号码。



图 8.14 验证电话号码

实现过程

(1) 在项目中新建窗体类 CheckPhoneNum, 在该窗体中添加 3 个文本框, 用于输入姓名、年龄与电话号码, 再添加一个“验证”按钮。

(2) 编写“验证”按钮的事件处理方法, 该方法获取用户在文本框中输入的电话号码字符串, 然后调用 check() 方法进行验证, 并将验证结果通过对话框进行输出。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = phoneNumField.getText();           //获取用户输入
    String info = check(text);                       //对输入文本进行 IP 验证
    JOptionPane.showMessageDialog(null, info);       //用对话框输出验证结果
}
```

(3) 编写 check() 方法, 该方法用于验证指定的字符串与正确的电话号码格式是否匹配, 该方法首先判断字符串是否为空, 然后再通过正则表达式对字符串进行验证, 并将验证结果作



为方法的返回值。关键代码如下：

```
public String check(String text){
    if(text == null || text.isEmpty()){
        return "请输入电话号码！";
    }
    String regex = "^\\d{3}-?\\d{8}|\\d{4}-?\\d{8}$";           //定义正则表达式
    if(text.matches(regex)){                                   //判断输入数据是否为电话号码
        return text + "\n 是一个合法的电话号码！";
    }else{
        return text + "\n 不是一个合法的电话号码！";
    }
}
```



Note

技术要点

本实例使用正则表达式对电话号码进行了格式匹配验证。正则表达式通常被用于判断语句中，来检查某一字符串是否满足某一格式。它是含有一些特殊意义字符的字符串，这些特殊字符称为正则表达式的元字符。例如，“\\d”表示字母 0~9 中任何一个。“\\d”就是元字符。正则表达式中的元字符及其意义如表 8.1 所示。

表 8.1 正则表达式中的元字符

元 字 符	正则表达式中的写法	意 义
.	"."	代表任意一个字符
\\d	"\\d"	代表 0~9 的任何一个数字
\\D	"\\D"	代表任何一个非数字字符
\\s	"\\s"	代表空白字符，如 't'、'n'
\\S	"\\S"	代表非空白字符
\\w	"\\w"	代表可用作标识符的字符，但不包括 "\$" 符
\\W	"\\W"	代表不可用于标识符的字符
\\p{Lower}	\\p{Lower}	代表小写字母 {a~z}
\\p{Upper}	\\p{Upper}	代表大写字母 {A~Z}
\\p{ASCII}	\\p{ASCII}	ASCII 字符
\\p{Alpha}	\\p{Alpha}	字母字符
\\p{Digit}	\\p{Digit}	十进制数字，即 [0~9]
\\p{Alnum}	\\p{Alnum}	数字或字母字符
\\p{Punct}	\\p{Punct}	标点符号：!"#\$%&'()*+,-./:;<=>?@[\\]^_`{ }~
\\p{Graph}	\\p{Graph}	可见字符：[\\p{Alnum}\\p{Punct}]
\\p{Print}	\\p{Print}	可打印字符：[\\p{Graph}\\x20]
\\p{Blank}	\\p{Blank}	空格或制表符：[t]
\\p{Cntrl}	\\p{Cntrl}	控制字符：[x00~x1F x7F]

多学两招：

一个 Java 对象（字符串也是 Java 对象）必须先初始化才能使用，否则编译器会报告“使用的变量未初始化”错误。



实例 091 将字符串转换成整数

(实例位置: 配套资源\SL\08\091)

实例说明

在 Swing 程序中, 用户输入的信息通常是使用 `getText()` 方法获得的。该方法的返回值是 `String` 类型。如果用户输入的是一串数字, 而程序又需要使用这些数字进行运算, 则可以将字符串转换成整型或浮点型。本实例将用户的输入转换为整数并计算其平方数。实例的运行效果如图 8.15 所示。

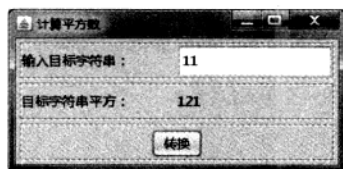


图 8.15 将字符串转换成整数并做运算实现过程

(1) 编写类 `IntegerConversion`, 该类继承了 `JFrame`。在框架中包含了一个文本域用来获得用户的输入、“转换”按钮用于转换用户的输入为整数并计算其平方、显示结果和提示信息的标签。

(2) 编写 `do_button_actionPerformed()` 方法, 用来监听单击“转换”按钮事件。在该方法中, 将用户的输入转换成整数并计算其平方数, 然后将结果显示在标签中。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String input = textField.getText();           //获得用户的输入文本
    int number = Integer.parseInt(input);         //将文本转换成整数
    label3.setText(number * number + "");        //计算平方数并显示结果
}
```

脚下留神:

此处并未对用户的输入进行校验, 如果用户输入的数不能转换成整数则会出现异常。

指点迷津:

该方法是使用 10 为基数来解析字符串的, 因此结果和字符串的内容相同。

技术要点

`Integer` 类是基本类型中 `int` 类型的包装类, 它可以将基本类型转换成引用类型。在 Java 5.0 版增加了自动装箱和拆箱机制后, 该类的这种用法已经不常用。该类还提供了将字符串转换成 `int` 类型的静态方法。其声明语法如下:

```
public static int parseInt(String s);
```

参数说明

s: 要转换的字符串, 如果不能成功转换则会抛出 `NumberFormatException` 异常。

返回值: 用十进制参数表示的整数值。



Note



多学两招:

除了本实例使用的 `parseInt()` 方法外, API 中还提供了其重载方法。该方法可以根据指定的基数来转换字符串。

语法如下:

```
parseInt(String s,int radix);
```

参数说明

- ☒ `s`: 要转换的字符串, 如果不能成功转换则会抛出 `NumberFormatException` 异常。
- ☒ `radix`: 解析 `s` 时使用的基数。



Note

实例 092 整数进制转换器

(实例位置: 配套资源\SL\08\092)

实例说明

由于计算机的特殊结构, 其内部使用二进制数据。为了节约空间, 又定义了八进制和十六进制格式来表示二进制数据。一个八进制数可以表示 3 位二进制数, 一个十六进制数可以表示 4 位二进制数。而对于普通人而言, 使用十进制更加容易阅读。本实例将实现一个简单的进制转换器。实例的运行效果如图 8.16 所示。



图 8.16 进制转换

脚下留神:

需要先在文本域中输入数字才能进行转换, 本实例并未校验空输入的情况。

实现过程

(1) 编写类 `RadixConversion`, 该类继承了 `JFrame`。在框架中包含了一个文本域用来获得用户的输入和显示转换的结果, 一个按钮组用来实现在不同进制之间的转换功能。

(2) 编写 `do_textField_focusLost()` 方法, 用来监听文本域失去焦点事件。在该方法中, 使用一个名为 `number` 的域保存用户输入的字符串。关键代码如下:

```
protected void do_textField_focusLost(FocusEvent e) {  
    number = textField.getText();  
} //获得用户的输出
```

指点迷津:

读者可以使用 `String` 类的 `isEmpty()` 方法来判断用户输入是否为空。

(3) 编写 `do_octalRadioButton_actionPerformed()` 方法, 用来监听选中“二进制”单选按钮事件。在该方法中, 将用户输入的字符串转换成二进制格式并在文本域中显示。关键代码如下:

```
protected void do_binaryRadioButton_actionPerformed(ActionEvent e) {  
    textField.setText(Integer.toString(Integer.parseInt(number))); //显示转换的结果  
}
```



指点迷津:

字符串就是由多个字符组成的,灵活地运用字符数组可以实现复杂的字符串操作,通过数组下标的各种算法可以使字符串更加灵活。

技术要点

Integer 类设计的初衷是为了在基本类型 int 和引用类型之间建立一座桥梁。然而,类库的设计者发现,可以将很多有用的方法也放在该类中。本实例使用其定义的进制转换方法来实现进制转换,使用到的方法如表 8.2 所示。

表 8.2 Integer 类的常用方法

方 法 名	作 用
toBinaryString(int i)	返回指定数字 i 的二进制表示形式
toOctalString(int i)	返回指定数字 i 的八进制表示形式
toHexString(int i)	返回指定数字 i 的十六进制表示形式

脚下留神:

以上方法的返回值都是无符号形式的结果,例如-1 的十六进制表示是 ffffffff。

多学两招:

本实例虽然实现了进制转换功能,但是有个缺点:不能进行连续转换。因为 Integer.parseInt(number) 代码使用 10 为基数来解析字符串,如果用户先将字符串转换成包含字母的十六进制格式,再转换为其他进制就会出现数字格式异常。解决方法是保存当前数字的基数,请读者自行完成。

实例 093 获取字符串中汉字的个数

(实例位置: 配套资源\SL\08\093)

实例说明

字符串中可以包括数字、字母、汉字或者其他字符。使用 Character 类的 isDigit() 方法可以判断字符串中的某个字符是否为数字,使用 Character 类的 isLetter() 方法可以判断字符串中的某个字符是否为字母。实例中将介绍一种方法用来判断字符串中的某个字符是否为汉字,通过此方法可以计算字符串中汉字的数量。实例的运行效果如图 8.17 所示。

实现过程

(1) 在项目中创建窗体类 ChineseAmount, 在窗体中添加接收用户输入的文本域控件、显示汉字数量的文本框控件和计算汉字数量的“计算”按钮。

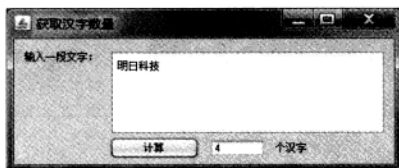


图 8.17 获取字符串中汉字的个数





Note

(2) 编写“计算”按钮的事件处理方法,在该方法中获取用户输入的字符串,然后遍历字符串中的每一个字符,使用正则表达式判断字符是否属于汉字,然后根据判断结果对汉字进行计数,最后把计数结果显示到界面文本框中。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = chineseArea.getText();           //获取用户输入
    int amount = 0;                                //创建汉字数量计数器
    for (int i = 0; i < text.length(); i++) {      //遍历字符串每一个字符
        boolean matches = Pattern.matches("[\u4E00-\u9FA5]{0,}$", text.charAt(i) + "");
                                                //判断字符是否属于汉字编码
        if (matches) {                             //如果是汉字
            amount++;                               //累加计数器
        }
    }
    numField.setText(amount + "");                 //在文本框中显示汉字数量
}
```

指点迷津:

字符串对象的索引是只读的,只可以读取字符串对象中的字符,不可以根据索引更改字符串中的字符。

技术要点

本实例的关键点在于正则表达式的使用。Java 中提供了 `Pattern` 用于正则表达式的编译表示形式,该类提供的静态方法 `matches()` 可以执行正则表达式的匹配。该方法编译给定正则表达式并尝试将给定输入与其匹配。如果要匹配的字符序列与正则表达式匹配则返回 `true`, 否则返回 `false`。其声明语法如下:

```
public static boolean matches(String regex,CharSequence input);
```

参数说明

- ☒ `regex`: 要编译的表达式。
- ☒ `input`: 要匹配的字符序列。

多学两招:

使用正则表达式可以非常方便地操作字符串,经常用来验证用户输入的信息,如可以判断用户输入的手机号码格式是否正确,验证用户输入的身份证号码格式是否正确。在本实例中将使用正则表达式来判断字符串中的字符是否为汉字,如果是汉字则计数器加 1,最后得到字符串中所有汉字的数量。

实例 094 批量替换某一类字符串

(实例位置: 配套资源\SL\08\094)

实例说明

在字符串操作中,可以使用字符串对象的 `split()` 方法拆分字符串,还可以使用字符串对象的



substring()方法截取一部分字符串。字符串对象为开发者提供了很多方便实用的方法,本实例中将会介绍使用字符串对象的replace()方法替换某一类字符串。实例的运行效果如图8.18所示。

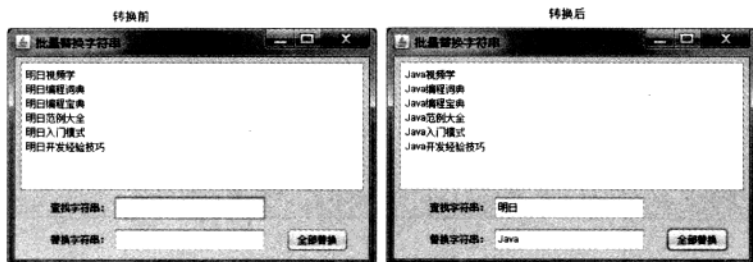


图 8.18 替换字符串前后对比

技术要点

(1) 在项目中创建窗体类 StringReplace。在窗体上创建一个文本框控件,用来显示所有文字;添加两个文本框控件,分别用来接收查找和替换的字符串;再添加一个“全部替换”按钮。

(2) 编写“全部替换”按钮的事件处理方法,在该方法中获取用户输入的搜索字符串、替换字符串和文本域中的文本字符串,然后执行替换操作,最后将替换结果显示在文本域控件中。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String searchStr = searchTextField.getText();           //获取搜索字符串
    String replaceStr = replaceTextField.getText();          //获取替换字符串
    String text = txtArea.getText();                        //获取段落文本
    String newText = text.replace(searchStr, replaceStr);    //执行替换
    txtArea.setText(newText);                              //替换结果显示在文本域控件中
}
```

指点迷津:

字符串就是由多个字符组成的,灵活地运用字符数组可以实现复杂的字符串操作,通过数组下标的各种算法可以使字符串更加灵活。

技术要点

本实例重点在于向读者介绍字符串对象的replace()方法的使用。

使用字符串对象的replace()方法可以方便地替换字符串中指定的内容。由于字符串是不可变的,replace()方法会返回一个新的字符串对象。replace()方法的使用如图8.19所示。

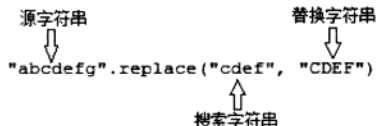


图 8.19 字符串对象 replace()方法的使用

从图8.19中可以看到,字符串对象调用了replace()方法,此方法将会返回一个被替换内容的新字符串。新字符串的值为“abCDEFg”。



脚下留神:

由于字符串是不可变的, 字符串对象调用了 `replace()` 方法后返回的是一个新的字符串而不是原来的字符串。

多学两招:

用字符串对象的 `replace()` 方法可以方便地替换字符串中指定的内容。有一点要注意, `replace()` 方法并不是只替换掉一个匹配的字符串, 而是一次性替换掉所有匹配的字符串, 由于字符串是不可变的, `replace()` 方法会返回一个新的字符串对象。

实例 095 查看数字的取值范围

(实例位置: 配套资源\SL\08\095)

实例说明

Java 是一种强类型语言, 每当定义变量时, 都需要先指明其类型。对于数字基本类型而言, 其取值范围是有限制的。例如, `byte` 类型的范围是从 `-128~127`, 如果将 `128` 赋值给一个 `byte` 类型的数就会报错。本实例将用来显示各种数字基本类型的最大值和最小值。实例的运行效果如图 8.20 所示。



图 8.20 查看取值范围

实现过程

(1) 编写类 `NumberLimitation`, 该类继承了 `JFrame`。在框架中包含 4 个标签和一个单选按钮组。两个标签被用来显示最大值和最小值, 单选按钮组用来供用户选择要显示的基本类型。

(2) 编写方法 `do_byteRadioButton_actionPerformed()`, 用来监听选中“`byte` 类型”单选按钮事件。在该方法中, 更新了 4 个标签的文本信息, 关键代码如下:

```
protected void do_byteRadioButton_actionPerformed(ActionEvent e) {  
    maxLabel.setText("byte 类型的最大值: ");           //更新最大值标签  
    minLabel.setText("byte 类型的最小值: ");           //更新最小值标签  
    maxResult.setText(Byte.MAX_VALUE + "");             //显示最大值  
    minResult.setText(Byte.MIN_VALUE + "");             //显示最小值  
}
```

指点迷津:

其他单选按钮的代码与此类似, 在此就不做讲解。

技术要点

为了方便基本类型和引用类型之间的转换, Java 为每种基本类型都提供了对应的包装类。现说明如下: `byte` 的包装类是 `Byte`、`short` 的包装类是 `Short`、`int` 的包装类是 `Integer`、`long` 的包装类是 `Long`、`float` 的包装类是 `Float`、`double` 的包装类是 `Double`、`boolean` 的包装类是 `Boolean`、`char` 的包装类是 `Character`。在各个包装类中, 定义了一些常用的域和方法。对于数字基本类型



的包装类而言，其 MAX_VALUE 域表示该类型所能取得的最大值、MIN_VALUE 域表示该类型所能取得的最小值。以 Byte 为例，下面的代码可以获得 byte 类型的最大值：

```
public static final byte MAX_VALUE;
```

对于其他类型而言，代码是类似的。

多学两招：

在 8 种基本类型中，最常用的是 boolean、int 和 double。byte 类型用于输入流和输出流的操作。short、float 通常用于需要节约空间的情况。如果不是与字符编码有关的操作，最好不要使用 char 类型。使用 long 和 float 类型时，需要在数字后面增加 L (l) 和 F (f) 标识。



Note

实例 096 ASCII 编码查看器

(实例位置：配套资源\SL\08\096)

实例说明

ASCII 是 American Standard Code Information Interchange 的缩写。它是基于拉丁字母的一套电脑编码系统，主要用于显示英语字符，是目前世界上最通用的单字节编码。基本的 ASCII 编码包括了 128 个字符。本实例将编写一个 ASCII 编码查看器，可以将字符转换成数字，也可以反向转换。实例的运行效果如图 8.21 所示。

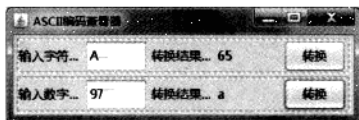


图 8.21 ASCII 编码查看器

实现过程

(1) 编写类 ASCIIViewer，该类继承了 JFrame。在框架中主要包含了两个文本域和两个“转换”按钮，文本域用来获得用户的输入，“转换”按钮用来完成转换功能，并在标签上显示。

(2) 编写 do_toNumberButton_actionPerformed() 方法，用来监听单击第一个“转换”按钮事件。在该方法中，将用户输入的字符转换成数字，关键代码如下：

```
protected void do_toNumberButton_actionPerformed(ActionEvent e) {
    String ascii = asciiTextField.getText();           //获得用户输入的字符串
    int i = Character.codePointAt(ascii, 0);           //求字符串的第一个字符的代码点
    label3.setText("" + i); //更新标签
}
```

(3) 编写 do_toASCIIButton_actionPerformed() 方法，用来监听单击第二个“转换”按钮事件。在该方法中，将用户输入的数字转换成字符，关键代码如下：

```
protected void do_toASCIIButton_actionPerformed(ActionEvent e) {
    String number = numberTextField.getText();          //获得用户输入的字符串
    char[] a = Character.toChars(Integer.parseInt(number)); //求数字所对应的字符数组
    label6.setText(new String(a));                      //更新标签
}
```

指点迷津：

如果输入了多个字符，则在转换时只取第一个字符转换成数字。本实例未对数字范围进行校验。



技术要点

Character 类是 char 类型的包装类，该类除了能将 char 类型转换成引用类型外，还包括了大量处理字符编码的方法。本实例使用 codePointAt() 方法获得字符的代码点。其声明语法如下：

```
public static int codePointAt(char[] a, int index);
```

参数说明

- ☑ a: char 数组。
 - ☑ index: 要转换的 char 数组中的 char 值 (Unicode 代码单元) 的索引。
- 返回值: 给定索引上的 Unicode 代码点。

多学两招:

Character 类的方法和数据是通过 UnicodeData 文件中的信息定义的，该文件是 Unicode Consortium 维护的 Unicode Character Database 的一部分。此文件指定了各种属性，其中包括每个已定义 Unicode 代码点或字符范围的名称和常规类别。此文件及其描述可从 Unicode Consortium 获得。

实例 097 判断手机号的合法性

(实例位置: 配套资源\SL\08\097)

实例说明

程序开发中经常需要用户录入某些标准格式的数据，如 E-mail、身份证、手机号码等。由于这些数据有固定的标准格式，所以程序开发时经常习惯性地不加判断或进行简单判断方便处理数据，但是对于有针对性的程序需要对标准格式中的数据进行详细解析，确认无误后才进行处理。本实例以手机号码为例，使用正则表达式进行匹配判断。实例的运行效果如图 8.22 所示。



图 8.22 判断是否是合法的手机号

实现过程

(1) 在项目中创建窗体类 CheckPhoneNum。在窗体中添加一个接收用户输入的文本框和一个“提交”按钮。

(2) 编写“提交”按钮的事件处理方法，在该方法中获取用户输入的手机号码，然后定义正则表达式，如果用户输入的手机号码能够与这个正则表达式匹配则手机号码格式正确，否则认为是错误的手机号码。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText();           //获取用户输入号码
    String regex = "^13\\d{9}|15\\d{9}|18\\d{9}$"; //定义正则表达式
    if (text.matches(regex)) {                  //测试匹配结果
        showMessageDialog(null, text + " 是合法的手机号"); //提示合法手机号码
    } else {
```



```
showMessageDialog(null, text + " 不是合法的手机号"); //提示非法手机号码
```

```
}
```

技术要点

本实例的关键点在于手机号的长度与手机前两位数字范围的验证，用户在输入手机号时，程序可以获取的只有字符串类型，所以本实例利用字符串的灵活性与正则表达式搭配进行验证。该方法是 `String` 字符串类的方法，用于判断字符串与指定的正则表达式是否匹配。其声明语法如下：

```
public boolean matches(String regex);
```

参数说明

`regex`：用来匹配此字符串的正则表达式。

返回值：当且仅当此字符串符合给定的正则表达式条件时，返回 `true`。

多学两招：

由于正则表达式的存在，验证与匹配各种规律的字符串数据更加方便、快捷，所以尽量使用正则表达式控制与限制用户操作初期输入正确的数据，这样可以提高程序数据的价值，使程序更容易控制数据。

实例 098 用字符串构建器追加字符

（实例位置：配套资源\SL\08\098）

实例说明

字符串是程序开发中使用最频繁的数据，在 Java 中字符串是 `String` 类的对象，它是不可变数据，当执行字符串连接操作时将生成新的字符串，而不是修改原有字符串，所以大量字符串操作非常耗时。本实例分别演示使用 `String` 类与 `StringBuilder` 类进行 3 万个字符串追加的操作，并输出其运行时间，结果如图 8.23 所示。从本实例的运行结果中可以看出，普通的字符串连接操作将耗时 2100 多毫秒，而使用 `StringBuilder` 字符串构建器却在 0.3 毫秒左右的时间完成了 3 万个字符的追加操作。所以对于大量字符串操作，应该使用 `StringBuilder` 字符串构建器来完成。

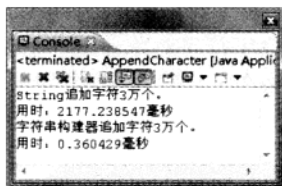


图 8.23 显示追加时间

实现过程

在项目中新建 `AppendCharacter` 类。在该类的主方法中创建字符串对象 `appendStr`，并通过循环为该字符串连接 3 万个字符，计算并输出其用时。再创建 `StringBuilder` 字符串构建器，同样为其追加 3 万个字符，计算并输出用时。关键代码如下：

```
public static void main(String[] args) {
    String appendStr="";
    long startTime = System.nanoTime();
    for(int i=20000;i<50000;i++){
```

```
//创建字符串变量
```

```
//开始计时
```

```
//遍历 3 万个字符
```





```

        appendStr+=(char)i; //字符串与每个字符执行连接操作
    }
    long endTime = System.nanoTime(); //结束计时
    System.out.println("String 追加字符 3 万个。");

    System.out.println("用时: "+(endTime-startTime)/1000000d+"毫秒"); //输出用时
    //////////////////////////////////////
    StringBuilder strBuilder=new StringBuilder(); //创建字符串构建器
    startTime = System.nanoTime(); //开始计时
    for(int i=20000;i<50000;i++){ //遍历 3 万个字符
        strBuilder.append((char)i); //把每个字符追加到构建器
    }
    endTime = System.nanoTime(); //结束计时
    System.out.println("字符串构建器追加字符 3 万个。");
    System.out.print("用时: "+(endTime-startTime)/1000000d+"毫秒"); //输出用时
}

```

多学两招:

StringBuilder 对于线程来说,是不安全的,它适用于单任务的字符串操作,如果把它应用于多线程中将会涉及异步访问的安全性。Java 早期版本提供的 StringBuffer 类可以作为多线程应用的考虑,它是线程安全的,也正因为考虑到线程安全问题,它会比 StringBuilder 稍微慢一些,但是差距很小,读者在开发程序时应灵活运用。

实例 099 去掉字符串中的所有空格

(实例位置: 配套资源\SL\08\099)

实例说明

在字符串操作中,可以使用字符串对象的 trim()方法去除字符串对象前端和后端的所有空格,但是,如果空格在字符串的中间位置出现,使用 trim()方法是没效果的,那么怎样才能有效地去除空格呢?本实例将通过字符串操作实现这个功能。实例的运行效果如图 8.24 所示。

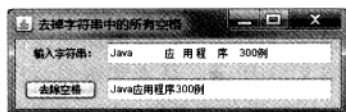


图 8.24 去掉字符串中的所有空格

实现过程

- (1) 在项目中创建窗体类 DeleteBlank。在窗体中添加两个文本框和一个“去除空格”按钮。
- (2) 编写“去除空格”按钮的事件处理方法,在该方法中获取用户输入的带有空格的字符串,创建一个字符串构建器用于提取非空格的字符,遍历字符串的每个字符,过滤所有空格,并将非空格字符追加到字符串构建器中,最后将构建器中的字符串显示到文本框中。关键代码如下:

```

protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText(); //获取用户输入文本
    StringBuilder strBuilder=new StringBuilder(); //创建字符串构建器
}

```



```

for(int i=0;i<text.length();i++){           //遍历字符串
    char charAt = text.charAt(i);           //获取每个字符
    if(charAt==' ')                          //过滤空格字符
        continue;
    strBuilder.append(charAt);               //追加非空格字符到字符构建器
}
resultField.setText(strBuilder.toString());  //把构建器中的字符串显示到文本框
}

```

技术要点

本实例重点在于向读者介绍怎样使用 `StringBuilder` 便捷、高效地操作字符串，下面介绍本实例对 `StringBuilder` 构建器的应用。构建器的 `append()` 方法可以向其尾部追加新的字符串。其声明语法如下：

```
append(String str);
```

参数说明

`str`: 要向构建器尾部追加的字符串。

返回值: 此对象的一个引用。

多学两招:

有些读者认为只要把 Java 类放到某个文件夹（目录）下，这个文件夹就是类的包名，这是一种误解。有文件夹结构不等于有了包名，必须在类的首行代码中通过 `package` 语句指定包的名称而不是靠文件结构类指定。

实例 100 Double 类型的比较

（实例位置：配套资源\SL\08\100）

实例说明

对于 `double` 类型（基本类型）的数据，可以直接使用普通的运算符来进行比较，如“`=`”。然而，对于 `Double` 类型（引用类型）却不行。引用类型如果使用“`=`”来进行比较的话是判断内存地址是否相同，答案通常是否定的。本实例演示如何使用 `Double` 类中定义的方法来进行对象间比较，实例的运行效果如图 8.25 所示。

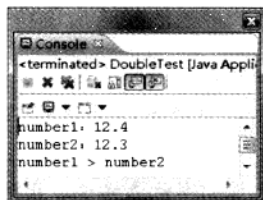


图 8.25 Double 类型的比较

实现过程

编写类 `DoubleTest`，在该类的 `main()` 方法中定义了两个 `Double` 类型的数据，并使用 `compareTo()` 方法对其进行比较。关键代码如下：

```

public class DoubleTest {
    public static void main(String[] args) {
        Double number1 = 12.4;           //定义 Double 类型的数据
        Double number2 = 12.3;           //定义 Double 类型的数据
        System.out.println("number1: " + number1); //输出定义的数据
    }
}

```





Note

```
System.out.println("number2: " + number2); //输出定义的数据
switch (number1.compareTo(number2)) { //判断两个 Double 类型数据之间的关系
    case -1:
        System.out.println("number1 < number2"); //输出比较的结果
        break;
    case 0:
        System.out.println("number1 == number2"); //输出比较的结果
        break;
    case 1:
        System.out.println("number1 > number2"); //输出比较的结果
        break;
}
```

脚下留神:

不要忘记在每个 case 子句的末尾增加 break 来跳出比较。

技术要点

Double 类是基本类型 double 的包装类, 该类提供了比较两个 Double 类型对象的 compareTo() 方法。其声明语法如下:

```
public int compareTo(Double anotherDouble);
```

参数说明

anotherDouble: 要比较的 Double 值。

返回值: 如果 anotherDouble 在数字上等于此 Double, 则返回 0; 如果此 Double 在数字上小于 anotherDouble, 则返回小于 0 的值; 如果此 Double 在数字上大于 anotherDouble, 则返回大于 0 的值。

指点迷津:

如果仅需要判断是否相等, 可以使用 equals() 方法, 该方法已经被重写了。

多学两招:

所有包装类的对象都不能使用 “==” 来进行比较, 还好每个包装类都定义了 compareTo() 方法, 可以使用该方法比较相同类型的对象, 然后根据返回值的不同来确定比较的结果。负数代表小于, 零代表等于, 正数代表大于。另外, 包装类重写了从 Object 类继承的 equals() 方法, 因此可以使用该方法来比较两个引用类型是否相等。

第9章

Java 集合类框架

本章读者可以学到如下实例：

- » 实例 101 用动态数组保存学生姓名
- » 实例 102 用 List 集合传递学生信息
- » 实例 103 Map 集合二级联动
- » 实例 104 不重复随机数组排序
- » 实例 105 for 循环遍历 ArrayList
- » 实例 106 Iterator 遍历 ArrayList
- » 实例 107 ListIterator 逆序遍历 ArrayList
- » 实例 108 制作电子词典
- » 实例 109 制作手机电话簿



实例 101 用动态数组保存学生姓名

(实例位置: 配套资源\SL\09\101)



Note

实例说明

Java 中提供了各种数据集合类, 这些类主要用于保存复杂结构的数据, 其中 ArrayList 集合可以看作动态数组。它突破了普通数组固定长度的限制, 可以随时向数组中添加和移除元素, 这将使数组更加灵活。如果要获取普通数组, 还可以通过该类的 toArray() 方法获得。本实例通过 ArrayList 集合类实现了向程序动态添加与删除学生姓名的功能, 其中所有数据都保存在 ArrayList 集合的实例对象中。实例的运行效果如图 9.1 所示。

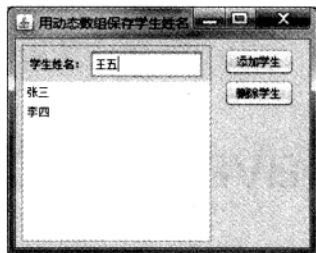


图 9.1 添加删除学生姓名

实现过程

(1) 在项目中新建窗体类 DynamicArray。在窗体中添加文本框控件、列表控件、“添加学生”按钮和“删除学生”按钮。

(2) 编写“添加学生”按钮的事件处理方法, 在该方法中获取用户在文本框中输入的字符串, 并将这个字符串添加到 ArrayList 集合中, 然后调用 replaceModel() 方法把集合中的数据显式到窗体的列表控件中。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {  
    textField.requestFocusInWindow();  
    textField.selectAll();           //选择文本框文本准备下次输入  
    String text = textField.getText(); //获取用户输入姓名  
    if (text.isEmpty())              //过滤为输入姓名的情况  
        return;  
    arraylist.add(text);              //把姓名添加到数组集合中  
    replaceModel();                  //把数组集合中的内容显示到界面列表控件中  
}
```

(3) 编写“删除学生”按钮的事件处理方法, 在该方法中获取列表控件的当前选择项, 然后从 ArrayList 集合中移除这个选择项的值, 最后调用 replaceModel() 方法把集合中的数据显式到窗体的列表控件中。关键代码如下:

```
protected void do_button_1_actionPerformed(ActionEvent e) {  
    Object value = list.getSelectedValue(); //获取列表控件的选择项  
    arraylist.remove(value);              //从数组集合中移除用户的选择项  
    replaceModel();                      //把数组集合中的内容显示到界面列表控件中  
}
```

(4) 编写 replaceModel() 方法, 在该方法中重新设置列表控件的模型, 模型要读取 ArrayList 集合的元素并显示到列表控件中。关键代码如下:

```
private void replaceModel() {  
    list.setModel(new AbstractListModel() { //为列表控件设置数据模型显示数组集合中的数据  
        @Override  
        public int getSize() {              //获取数组大小
```



```
        return arraylist.size();
    }
    @Override
    public Object getElementAt(int index) {           //获取指定索引元素
        return arraylist.get(index);
    }
}
});
```

技术要点

本实例使用了 ArrayList 集合类的相关操作方法。下面分别介绍程序中对 ArrayList 类 API 的引用。

1. 添加元素

add()方法可以为数组集合添加任意类型的元素。其声明语法如下：

```
public boolean add(E element);
```

参数说明

element: 要添加到集合中的任意类型的元素值或对象。

返回值: true。

2. 移除元素

remove()方法可以移除集合中的指定元素，其中只包含 Object 类型参数的此方法的重载格式可以从集合中移除首次出现指定值的元素。其声明语法如下：

```
public boolean remove(Object object);
```

参数说明

object: 要从集合中移除的对象。

返回值: 如果此列表包含指定的元素，则返回 true。

多学两招:

ArrayList 集合可以看作是一个动态的数组，它比普通数组更加灵活，更适合保存未知数量的数据。例如从数据库中读取指定条件的数据，并且在以后可能会不断地添加新数据，这种情况如果使用普通数组不但会受到长度限制，而且还会受到类型限制，如果采用 ArrayList 集合，这些问题就会迎刃而解。

实例 102 用 List 集合传递学生信息


(实例位置: 配套资源\SL\09\102)

实例说明

集合在程序开发中经常用到，如在业务方法中将学生信息、商品信息等存储到集合中然后作为方法的返回值返回给调用者，以此传递大量有序数据。本实例将使用 List 集合在方法之间传递学生信息。实例的运行效果如图 9.2 所示。



Note



姓名	性别	出生日期
李哥	男	1981-1-1
小陈	女	1981-1-1
小刘	男	1981-1-1
小张	男	1981-1-1
小董	男	1981-1-1
小吕	男	1981-1-1

图 9.2 将学生信息添加到集合中

实现过程

- (1) 在项目中新建窗体类 ClassInfo。在窗体中添加滚动面板，这个面板将放置表格控件。
- (2) 编写 getTable() 方法。在该方法中创建表格对象设置表格的数据模型，然后调用 getStudents() 方法获取保存学生信息的集合对象，在遍历该集合对象的同时把每个元素添加到表格模型的行，并显示到表格控件中。关键代码如下：

```
private JTable getTable() {  
    if (table == null) {  
        table = new JTable();  
        table.setRowHeight(23);  
        String[] columns = {"姓名", "性别", "出生日期"};  
        DefaultTableModel model = new DefaultTableModel(columns, 0);  
        table.setModel(model);  
        List<String> students = getStudents();  
        for (String info : students) {  
            String[] args = info.split(",");  
            model.addRow(args);  
        }  
    }  
    return table;  
}
```

- (3) 编写 getStudents() 方法，该方法将向调用者传递 List 集合对象，方法中为集合对象添加了多个元素，每个元素值都是一个学生信息，其中包括姓名、性别、出生日期。关键代码如下：

```
private List<String> getStudents() {  
    List<String> list = new ArrayList<String>();  
    list.add("李哥,男,1981-1-1");  
    list.add("小陈,女,1981-1-1");  
    list.add("小刘,男,1981-1-1");  
    list.add("小张,男,1981-1-1");  
    list.add("小董,男,1981-1-1");  
    list.add("小吕,男,1981-1-1");  
    return list;  
}
```

技术要点

实现本实例的关键在于以下几点：

- ☑ 将数字格式化，如果存在小数部分，将其转换为 3 位小数到单位厘。
- ☑ 分别将整数部分与小数部分转换为大写方式，并插入其单位（亿、万、仟……）。



☑ 组合转换后的整数部分与小数部分。

多学两招:

List<T>泛型集合表示可通过索引访问对象的强类型列表,它提供用于对列表进行搜索、排序和操作的方法,相对于 ArrayList 类来说, List<T>泛型集合在大多数情况下执行得更好并且是类型安全的。



Note

实例 103 Map 集合二级联动

(实例位置: 配套资源\SL\09\103)

实例说明

Map 集合可以保存键值映射关系,这非常适合本实例所需要的数据结构,所有省份信息可以保存为 Map 集合的键,而每个键可以保存对应的城市信息,本实例就利用这个 Map 集合实现了省市级联选择框,当选择省份信息时,将改变城市下拉列表框对应的内容。实例的运行效果如图 9.3 所示。

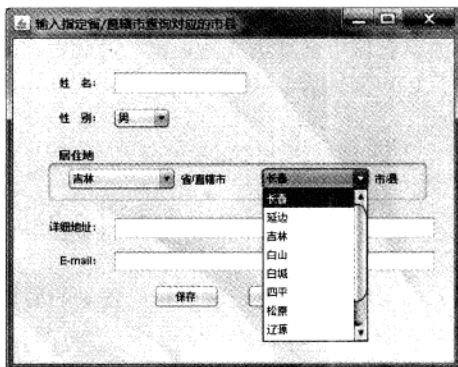


图 9.3 省市级联选择框

实现过程

(1) 在项目中新建窗体类 CityMap。在该类中创建并初始化 Map 集合对象,在该集合对象中保存各省市的关联信息。关键代码如下:

```
public class CityMap {
    public static Map<String,String[]> model=new LinkedHashMap();
    static{
        model.put("北京", new String[]{"北京"});
        model.put("上海", new String[]{"上海"});
        model.put("天津", new String[]{"天津"});
        model.put("重庆", new String[]{"重庆"});
        model.put("黑龙江", new String[]{"哈尔滨","齐齐哈尔","牡丹江","大庆","伊春","双鸭山",
"鹤岗","鸡西","佳木斯","七台河","黑河","绥化","大兴安岭"});
        model.put("吉林", new String[]{"长春","延边","吉林","白山","白城","四平","松原","辽源",
"大安","通化"});
```




```
model.put("辽宁", new String[]{"沈阳", "大连", "葫芦岛", "旅顺", "本溪", "抚顺", "铁岭", "辽
阳", "营口", "阜新", "朝阳", "锦州", "丹东", "鞍山"});
//省略类似代码
```

```
}
}
```

(2) 创建 `MainFrame` 主窗体类, 在该类中添加 3 个文本框用于输入姓名、详细地址和 E-mail 信息, 添加选择性别的下拉列表框, 添加“保存”和“重置”按钮, 最后添加两个核心控件, 也就是选择省份与选择城市的下拉列表框。

(3) 编写 `getProvince()` 方法, 在该方法中获取 `Map` 集合的键映射, 也就是省份信息的 `Set` 集合, 然后将该集合转换为数组, 并作为方法的返回值, 这个方法将在省份下拉列表框的初始化代码时被调用。关键代码如下:

```
public Object[] getProvince() {
    Map<String, String[]> map = CityMap.model; //获取省份信息保存到 Map 中
    Set<String> set = map.keySet(); //获取 Map 集合中的键, 并以 Set 集合返回
    Object[] province = set.toArray(); //转换为数组
    return province; //返回获取的省份信息
}
```

(4) 编写选择省份的下拉列表框的事件处理方法, 该方法在省份下拉列表框改变选项时被调用, 方法首先获取下拉列表框的选项值, 然后把该值作为键到 `Map` 集合中查找对应该键的值, 返回结果是对应省份的所有城市名称组成的数组, 最后用这个数组创建一个数据模型添加到城市下拉列表框控件中, 以更新内容。关键代码如下:

```
private void itemChange() {
    String selectProvince = (String) comboBox.getSelectedItemAt();
    cityComboBox.removeAllItems(); //清空市/县列表
    String[] arrCity = getCity(selectProvince); //获取市/县
    cityComboBox.setModel(new DefaultComboBoxModel(arrCity)); //重新添加市/县列表的值
}
```

(5) 编写 `getCity()` 方法, 该方法主要负责获取对应省份的城市数组, 它将在省份下拉列表框控件的事件处理方法中被调用。关键代码如下:

```
public String[] getCity(String selectProvince) {
    Map<String, String[]> map = CityMap.model; //获取省份信息保存到 Map 中
    String[] arrCity = map.get(selectProvince); //获取指定键的值
    return arrCity; //返回获取的市/县
}
```

技术要点

本实例的关键技术是 `Map` 集合的运用。`Map` 集合可以保存键值对数据, 这样可以根据指定的键名称来获取值数据。

1. 添加映射键值对

本实例通过 `Map` 集合来保存省市信息, 其中省份作为映射的键, 而城市数组作为键对应的值, `Map` 映射提供了 `put()` 方法来为集合添加数据。其声明语法如下:

```
put(K key, V value);
```

参数说明

☒ **key**: 与指定值关联的键。



☑ value: 与指定键关联的值。

2. 获取键对应的值

Map 集合的 get() 方法返回指定键所映射的值, 如果此映射不包含该键的映射关系, 则返回 null 值。其声明语法如下:

```
V get(Object key);
```

参数说明

key: 要返回其关联值的键。

返回值: 指定键所映射的值; 如果此映射不包含该键的映射关系, 则返回 null。

3. 获取键的 Set 集合

Map 集合可以获取所有键的 Set 集合, 这个集合中包含 Map 中的所有键, 本实例通过 keySet() 方法获取所有键信息来为下拉列表框添加内容。其声明语法如下:

```
Set<K> keySet();
```

多学两招:

Map 集合的具体实现有很多, 应该根据需要来选择, 其中 HashMap 是最常用的映射集合, 它只允许一条记录的键为 null, 但是却不限制集合中值为 null 的数量。HashTable 实现一个映射, 它不允许任何键值为空。TreeMap 集合将对集合中的键值排序, 默认排序方式为升序。

实例 104 不重复随机数组排序

(实例位置: 配套资源\SL\09\104)

实例说明

随机数组就是在指定长度的数组中用随机数字为每个元素赋值, 这常用于需要不确定数值的环境, 如拼图游戏需要随机数组来打乱图片排序。可是随机数的重复问题也同时存在, 这个问题也常常被忽略, 本实例将利用 TreeSet 集合实现不重复的数列, 并自动完成元素的排序然后生成数组。实例的运行效果如图 9.4 所示。

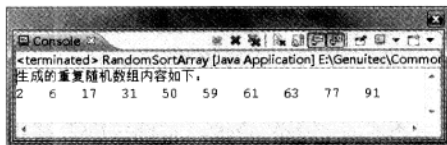


图 9.4 不重复随机数组排序

实现过程

(1) 在项目中新建类 RandomSortArray。

(2) 在类的主方法中创建 TreeSet 集合对象, 再创建 Random 随机数对象, 然后通过计数器控制循环生成随机数并添加到集合对象中, 最后通过集合对象提取数组并显示在控制台中。关键代码如下:

```
public static void main(String[] args) {
    TreeSet<Integer> set = new TreeSet<Integer>(); //创建 TreeSet 集合对象
    Random ran = new Random(); //创建随机数对象
    int count = 0; //定义随机数计数器
    while (count < 10) { //循环生成随机数
```



```
boolean succeed = set.add(ran.nextInt(100)); //为集合添加数字
if (succeed)                                //累加成功添加到集合中数字的数量
    count++;
}
int size = set.size();                       //获取集合大小
Integer[] array = new Integer[size];        //创建同等大小的数组
set.toArray(array);                         //获取集合中的数组
System.out.println("生成的重复随机数组内容如下: ");
for (int value : array) {                   //遍历输出数组内容
    System.out.print(value + " ");
}
}
```

技术要点

本实例使用了 TreeSet 集合对象的 API 实现元素的添加以及数组的提取。

1. 添加元素

TreeSet 类的 add() 方法可以为集合添加元素，TreeSet 集合属于 Set 集合的子类，Set 集合不允许有重复的元素存在，所以重复数据是不允许添加到 Set 集合中的，而 add() 方法的返回值可以确定添加操作是否成功完成。其声明语法如下：

```
public boolean add(E e);
```

参数说明

e: 要添加到集合中的任意类型的数据。

返回值: 如果此 set 尚未包含指定元素，则返回 true。

2. 提取集合中的数组

Java 的集合对象可以调用 toArray() 方法将集合中的所有数据提取到一个新的数组中，本实例就调用了该方法。其声明语法如下：

```
public <T> T[] toArray(T[] array);
```

参数说明

array: 保存集合数据的数组。

返回值: 如果参数 array 指定的数组长度小于 Set 集合元素的数量，则返回新的可以容纳 Set 集合所有元素的数组；否则返回参数指定的数组对象。

脚下留神:

如果方法指定的数组参数可以容纳下 Set 集合中的所有元素，那么方法的返回值就是这个数组参数，它们用 “==” 判断结果为 true，因为同一个数组对象的内存地址相等。

如果数组参数长度大于 Set 集合，那么剩余数组元素都赋值为 null，这时要注意可能发生的空指针异常。

多学两招:

本实例首先调用了 size() 方法确定 Set 集合的大小，然后创建同等大小的数组，再通过 toArray() 方法将集合的所有元素提取到该数组中。更简便的方法是创建一个最小的数组作为方法的参数，这时 toArray() 方法会返回与参数数组类型相同的能容纳 Set 集合所有元素的新数组。



实例 105 for 循环遍历 ArrayList

(实例位置: 配套资源\SL\09\105)

实例说明

在使用集合类时, 不仅关心容器是如何保存元素的, 而且关心如何取出元素。本实例将使用普通 for 循环遍历 ArrayList, 从中取出所有序号为奇数的元素。实例的运行效果如图 9.5 所示。

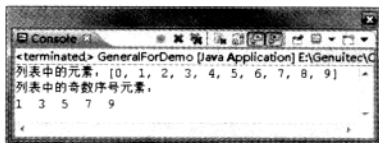


图 9.5 输出序号为奇数的元素

实现过程

(1) 在 Eclipse 中新建一个项目, 在项目中新建类 GeneralForDemo。

(2) 在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型, 并为其添加 10 个元素。

关键代码如下:

```
List<Integer> list = new ArrayList<Integer>();           //创建集合
for (int i = 0; i < 10; i++) {                          //向列表中增加 10 个元素
    list.add(i);
}
```

(3) 利用 for 循环遍历 ArrayList 集合, 输出列表中序号为奇数的元素。关键代码如下:

```
for (int i = 1; i < list.size(); i += 2) {               //输出列表中序号为奇数的元素
    System.out.print(list.get(i) + " ");
}
```

技术要点

本实例主要应用到了 List 集合中的 get() 方法获取列表指定位置的元素。其声明语法如下:

```
E get(int index);
```

参数说明

index: 要返回元素的索引。

返回值: 列表中指定位置的元素。

多学两招:

for 循环中的迭代表达式是用于改变循环条件的语句, 根据自己的需要为继续执行循环体语句作准备, 如自增、自减等运算。

实例 106 Iterator 遍历 ArrayList

(实例位置: 配套资源\SL\09\106)

实例说明

ArrayList 在以后的开发中会经常用到, 本实例将练习使用 Iterator 和 for 循环组合遍历集合。实例的运行效果如图 9.6 所示。



Note

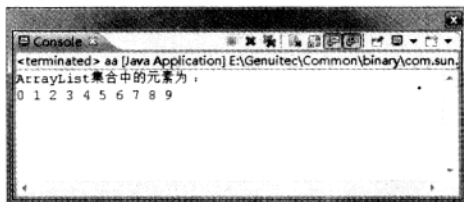


图 9.6 使用 Iterator 遍历 ArrayList

实现过程

(1) 在 Eclipse 中新建一个项目，在项目中新建类 IteratorDemo。

(2) 在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型，并为其添加 10 个元素。

关键代码如下：

```
List<Integer> list = new ArrayList<Integer>();           //创建集合
for (int i = 0; i < 10; i++) {                          //向列表中增加 10 个元素
    list.add(i);
}
```

(3) 利用迭代器遍历 ArrayList 集合，其循环条件为如果迭代器中仍有元素可以迭代则继续循环，如果没有则跳出循环。关键代码如下：

```
for(Iterator<Integer> it = list.iterator();it.hasNext();) { //利用迭代器遍历 ArrayList 集合
    System.out.print(it.next()+" ");
}
```

技术要点

本实例调用了 Iterator 接口中的 next()方法，返回迭代的下一个元素。其声明语法如下：

```
E next();
```

返回值：迭代的下一个元素。

实例 107 ListIterator 逆序遍历 ArrayList

(实例位置：配套资源\SL\09\107)

实例说明

对于列表而言，除了 Iterator，还提供了一个功能更加强大的 ListIterator，它可以实现逆序遍历列表中的元素。本实例将使用其逆序遍历 ArrayList。实例的运行效果如图 9.7 所示。

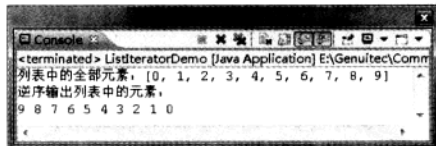


图 9.7 逆序遍历 ArrayList

实现过程

(1) 在 Eclipse 中新建一个项目，在项目中新建类 ListIteratorDemo。



(2) 在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型, 并为其添加 10 个元素。关键代码如下:

```
List<Integer> list = new ArrayList<Integer>();           //创建集合
for (int i = 0; i < 10; i++) {                          //向列表中增加 10 个元素
    list.add(i);
}
```

(3) 获得迭代器对象, 然后利用 hasPrevious() 方法逆序输出 ArrayList 集合中的元素。关键代码如下:

```
ListIterator<Integer> li = list.listIterator();          //获得 ListIterator 对象
for (li = list.listIterator(); li.hasNext(); ) {         //将游标定位到列表结尾
    li.next();
}
for (; li.hasPrevious(); ) {                             //逆序输出列表中的元素
    System.out.print(li.previous() + " ");
}
```

技术要点

本实例主要应用了 ListIterator 中的 hasPrevious() 方法, 该方法如果以逆向遍历列表, 列表迭代器有多个元素, 则返回 true。(换句话说, 如果 previous 返回一个元素而不是抛出异常, 则返回 true)。其声明语法如下:

```
boolean hasPrevious();
```

返回值: 如果以逆向遍历列表, 列表迭代器有多个元素, 则返回 true。

实例 108 制作电子词典

(实例位置: 配套资源\SL\09\108)

实例说明

词典通常用于解释一个词的含义, 这是一种映射关系, 因此可以使用 Map 来实现。本实例将制作一个电子词典, 实例的运行效果如图 9.8 所示。

实现过程

(1) 在项目中新建窗体类 DictionaryDemo。在窗体中添加标签等控件。

(2) 编写 do_this_windowActivated() 方法完成窗体激活事件监听。在该方法中向 Map 中增加数据, 关键代码如下:

```
protected void do_this_windowActivated(WindowEvent e) {
    words = new HashMap<String, String>();
    words.put("apple", "苹果");
    words.put("banana", "香蕉");
    words.put("water", "水");
}
```

(3) 编写 do_button_actionPerformed() 方法, 用来响应按钮单击事件。在该方法中完成了



图 9.8 电子词典





对单词的查询操作，并根据不同的情况进行提示。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText();           //获得用户输入的单词
    if (text.isEmpty()) {                       //如果用户输入为空则提示用户
        JOptionPane.showMessageDialog(this, "请输入要查询的单词！", null, JOptionPane.
WARNING_MESSAGE);
        return;
    }
    String meaning = words.get(text);           //查询单词的含义
    if (meaning == null) {                      //如果没有这个单词则提示用户
        JOptionPane.showMessageDialog(this, "要查询的单词不存在！", null, JOptionPane.
WARNING_MESSAGE);
        return;
    } else {
        textArea.setText(meaning);              //显示单词的含义
    }
}
```

技术要点

Map 集合中的元素是通过 key、value 进行存储的。要获取集合中指定的 key 或 value 值，需要先通过相应的方法获取 key 或 value 集合，再遍历 key 或 value 集合获取指定值。

实例 109 制作手机电话簿

(实例位置：配套资源\SL\09\109)

实例说明

为了便于保存联系方式，手机都提供电话簿功能。它也是一种映射关系，因此可以使用 Map 来实现。本实例将制作一个手机电话簿，实例的运行效果如图 9.9 所示。

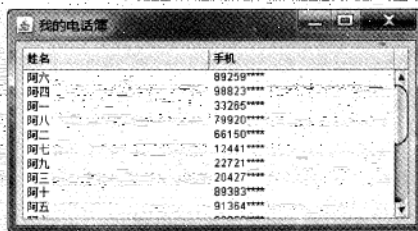


图 9.9 手机电话簿

实现过程

- (1) 在项目中新建窗体类 Phonebook。在窗体中添加表格控件。
- (2) 编写 do_this_windowActivated() 方法完成窗体激活事件监听。在该方法中向表格模型中增加数据，最后更新表格模型。关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    Map<String, String> directory = new HashMap<String, String>(); //创建集合
```



Note

```

directory.put("阿一", "33265****"); //向集合中添加元素
directory.put("阿二", "66150****"); //向集合中添加元素
directory.put("阿三", "20427****"); //向集合中添加元素
directory.put("阿四", "98823****"); //向集合中添加元素
directory.put("阿五", "91364****"); //向集合中添加元素
directory.put("阿六", "89259****"); //向集合中添加元素
directory.put("阿七", "12441****"); //向集合中添加元素
directory.put("阿八", "79920****"); //向集合中添加元素
directory.put("阿九", "22721****"); //向集合中添加元素
directory.put("阿十", "89383****"); //向集合中添加元素
DefaultTableModel model = (DefaultTableModel) table.getModel(); //获得表格模型
model.setColumnIdentifiers(new Object[] { "姓名", "手机" }); //设置表头
Set<String> names = directory.keySet(); //获得键集合
for (Iterator<String> it = names.iterator(); it.hasNext(); ) {
    String name = it.next(); //获得键
    model.addRow(new Object[] { name, directory.get(name) }); //向表格中添加元素
}
table.setModel(model); //更新表格模型
}

```

技术要点

Map 集合中的 put() 方法用来向集合中添加指定的 key 与 value 的映射关系, Map 集合中同样提供了集合的常用方法, 如 clear()、isEmpty()、size() 等, 除此之外还包括如表 9.1 所示的常用方法。

表 9.1 Map 集合的常用方法

方 法	返 回 值	功 能 描 述
put(key k, value v)	Object	向集合中添加指定的 key 与 value 的映射关系
containsKey(Object key)	boolean	如果此映射包含指定键的映射关系, 则返回 true
containsValue(Object value)	boolean	如果此映射将一个或多个键映射到指定值, 则返回 true
get(Object key)	Object	如果存在指定的键对象, 则返回该对象对应的值, 否则返回 null
keySet()	Set	返回该集合中的所有键对象组成的 Set 集合
values()	Collection	返回该集合中所有值对象形成的 Collection 集合

第10章

常用数学工具类

本章读者可以学到如下实例：

- » 实例 110 角度和弧度的转换
- » 实例 111 三角函数的使用
- » 实例 112 反三角函数的使用
- » 实例 113 双曲函数的使用
- » 实例 114 指数与对数运算
- » 实例 115 高精度整数运算
- » 实例 116 高精度浮点运算
- » 实例 117 七星彩号码生成器
- » 实例 118 大乐透号码生成器



实例 110 角度和弧度的转换

(实例位置: 配套资源\SL\10\110)

实例说明

有两种单位可以用来度量角的大小, 即角度和弧度。通常在三角运算中使用弧度比较方便, 在表示角的大小时使用角度比较方便, 本实例将实现角度和弧度之间的转换。实例的运行效果如图 10.1 所示。

实现过程

(1) 在 Eclipse 中新建项目 110, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 RadianTest 类, 在该类的主方法中, 使用 Math 类的方法输出 30° 、 45° 角对应的弧度值和 $\pi/6$ 、 $\pi/4$ 弧度对应的角度值。关键代码如下:

```
public class RadianTest {
    public static void main(String[] args) {
        System.out.println("30° 对应的弧度是: " + Math.toRadians(30));
        System.out.println("π/6 对应的角度是: " + Math.toDegrees(Math.PI / 6));
        System.out.println("45° 对应的弧度是: " + Math.toRadians(45));
        System.out.println("π/4 对应的角度是: " + Math.toDegrees(Math.PI / 4));
    }
}
```

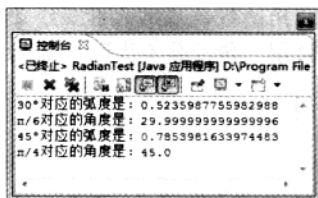


图 10.1 角度和弧度的转换

技术要点

本实例主要使用了 Math 类的 toRadians() 和 toDegrees() 方法实现了角度与弧度的转换。

1. toRadians() 方法

Math 类的 toRadians() 方法用于将角度转换为弧度, 该方法是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double toRadians(double angdeg)
```

参数说明

angdeg: 用角度表示的角。

返回值: 角 angdeg 用弧度表示的值。

2. toDegrees() 方法

Math 类的 toDegrees() 方法用于将弧度转换为角度, 该方法也是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double toDegrees(double angrad)
```

参数说明

angrad: 用弧度表示的角。

返回值: 角 angrad 用角度表示的值。



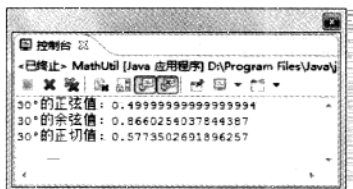


实例 111 三角函数的使用

(实例位置: 配套资源\SL\10\111)

实例说明

三角函数是数学的重要分支之一,很多问题使用三角函数来解决能容易很多。`Math` 类提供了常用三角函数的实现,本实例将演示它们的用法。实例的运行效果如图 10.2 所示。

图 10.2 输出 30° 的正弦值、余弦值和正切值

指点迷津:

由于虚拟机原因,浮点运算结果并不精确,所以使用三角函数得到的结果也是不精确的。

实现过程

(1) 在 Eclipse 中新建项目 111, 在项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建 `MathUtil` 类, 在该类的 `main()` 方法中输出了 30° 的正弦值、余弦值和正切值。关键代码如下:

```
public class MathUtil {  
    public static void main(String[] args) {  
        System.out.println("30° 的正弦值: " + Math.sin(Math.PI / 6)); //计算 30° 的正弦值  
        System.out.println("30° 的余弦值: " + Math.cos(Math.PI / 6)); //计算 30° 的余弦值  
        System.out.println("30° 的正切值: " + Math.tan(Math.PI / 6)); //计算 30° 的正切值  
    }  
}
```

技术要点

本实例主要使用了 `Math` 类的 `sin()`、`cos()` 和 `tan()` 方法实现了求指定角度的正弦值、余弦值和正切值。

1. `sin()` 方法

`Math` 类的 `sin()` 方法用于求指定角的正弦值, 该方法是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double sin(double a)
```

参数说明

a: 用弧度表示的角。

返回值: 角 a 的正弦值。



2. cos()方法

Math 类的 cos()方法用于求指定角的余弦值,该方法是一个静态方法,可以通过类名直接调用。其语法如下:

```
public static double cos(double a)
```

参数说明

a: 用弧度表示的角。

返回值: 角 a 的余弦值。

3. tan()方法

Math 类的 tan()方法用于求指定角的正切值,该方法是一个静态方法,可以通过类名直接调用。其语法如下:

```
public static double tan(double a)
```

参数说明

a: 用弧度表示的角。

返回值: 角 a 的正切值。



Note

实例 112 反三角函数的使用

(实例位置: 配套资源\SL\10\112)

实例说明

反三角函数通常用于获得某些三角函数值所对应的弧度,进而转换为角度,以满足生产和生活中不同角度的应用。Math 类提供了常用反三角函数的实现,本实例将演示反三角函数的用法。实例的运行效果如图 10.3 所示。

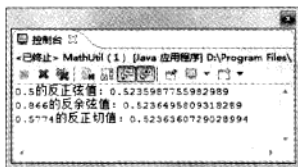


图 10.3 输出指定值的反正弦值、反余弦值和反正切值

指点迷津:

反三角函数返回的值是以弧度为单位的角,该值是一个近似值,图 10.3 中的 0.5、0.866 和 0.5774 分别是 30° 对应弧度值 (30° 对应的弧度近似值为 0.524) 的正弦值、余弦值和正切值的近似值。

实现过程

(1) 在 Eclipse 中新建项目 112,在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 MathUtil 类,在该类的主方法中,分别输出 0.5、0.866 和 0.5774 的反正弦值、反余弦值和反正切值。关键代码如下:

```
public class MathUtil {
    public static void main(String[] args) {
        System.out.println("0.5 的反正弦值: " + Math.asin(0.5));    //计算 0.5 的反正弦值
```




```
System.out.println("0.866 的反余弦值: " + Math.acos(0.866)); //计算 0.866 的反余弦值
System.out.println("0.5774 的反正切值: " + Math.atan(0.5774)); //计算 0.5774 的反正切值
}
}
```



Note

技术要点

本实例主要使用了 Math 类的 asin()、acos()和 atan()方法实现了求指定值的反正弦值、反余弦值和反正切值。

1. asin()方法

Math 类的 asin()方法用于求指定值的反正弦值,该方法是一个静态方法,可以通过类名直接调用。其语法如下:

```
public static double asin(double a)
```

参数说明

a: 要返回其反正弦的值。

返回值: 值 a 的反正弦值。

2. acos()方法

Math 类的 acos()方法用于求指定值的反余弦值,该方法是一个静态方法,可以通过类名直接调用。其语法如下:

```
public static double acos(double a)
```

参数说明

a: 要返回其反余弦的值。

返回值: 值 a 的反余弦值。

3. atan()方法

Math 类的 atan()方法用于求指定值的反正切值,该方法是一个静态方法,可以通过类名直接调用。其语法如下:

```
public static double atan(double a)
```

参数说明

a: 要返回其反正切的值。

返回值: 值 a 的反正切值。

实例 113 双曲函数的使用

(实例位置: 配套资源\SL\10\113)

实例说明

双曲函数在物理学中有重要应用,如阻尼落体、导电容等。Math 类提供了常用双曲函数的实现,本实例将演示它们的用法。实例的运行效果如图 10.4 所示。

实现过程

(1) 在 Eclipse 中新建项目 113,在项目中创建 com.mingrisoft 包。

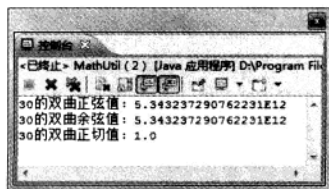


图 10.4 输出 30 的指定双曲函数值



(2) 在 com.mingrisoft 包中创建 MathUtil 类, 在该类的主方法 main() 方法中输出 30 的双曲正弦值、双曲余弦值和双曲正切值。代码如下:

```
public class MathUtil {
    public static void main(String[] args) {
        System.out.println("30 的双曲正弦值: " + Math.sinh(30)); //计算 30 的双曲正弦值
        System.out.println("30 的双曲余弦值: " + Math.cosh(30)); //计算 30 的双曲余弦值
        System.out.println("30 的双曲正切值: " + Math.tanh(30)); //计算 30 的双曲正切值
    }
}
```



Note

技术要点

本实例主要使用了 Math 类的 sinh()、cosh() 和 tanh() 方法实现了求指定值的双曲正弦值、双曲余弦值和双曲正切值。

1. sinh() 方法

Math 类的 sinh() 方法用于求指定值的双曲正弦值, 该方法是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double sinh(double a)
```

参数说明

a: 要返回其双曲正弦的数值。

返回值: 值 a 的双曲正弦值。

2. cosh() 方法

Math 类的 cosh() 方法用于求指定值的双曲余弦值, 该方法是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double cosh(double a)
```

参数说明

a: 要返回其双曲余弦的数值。

返回值: 值 a 的双曲余弦值。

3. tanh() 方法

Math 类的 tanh() 方法用于求指定值的双曲正切值, 该方法是一个静态方法, 可以通过类名直接调用。其语法如下:

```
public static double tanh(double a)
```

参数说明

a: 要返回其双曲正切的数值。

返回值: 值 a 的双曲正切值。

实例 114 指数与对数运算

(实例位置: 配套资源\SL\10\114)

实例说明

指数与对数运算是初等函数的重点, 它们在数学分析中有重要的应用。Math 类提供了常用指数与对数运算的实现, 本实例将演示它们的用法。实例的运行效果如图 10.5 所示。



图 10.5 指数与对数运算的结果

实现过程

- (1) 在 Eclipse 中新建项目 114，在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建 MathUtil 类，在该类的 main() 方法中，实现求指数、对数和方根的运算。关键代码如下：

```
public class MathUtil {  
    public static void main(String[] args) {  
        System.out.println("8 的立方根是: " + Math.cbrt(8));  
        System.out.println("e 的 8 次方是: " + Math.exp(8));  
        System.out.println("e 的 9 次方是: " + Math.expml(8));  
        System.out.println("8 的自然对数是: " + Math.log(8));  
        System.out.println("8 的 10 为底的对数是: " + Math.log10(8));  
        System.out.println("9 的自然对数是: " + Math.log1p(8));  
        System.out.println("2 的 3 次方是: " + Math.pow(2, 3));  
        System.out.println("8 的平方根是: " + Math.sqrt(8));  
    }  
}
```

技术要点

本实例主要使用了 Math 类的 cbrt()、exp()、expml()、log()、log10()、log1p()、pow() 和 sqrt() 方法实现了求指定值的方根、对数和平方根等运算。

实例 115 高精度整数运算

(实例位置: 配套资源\SL\10\115)

实例说明

为了弥补虚拟机在高精度计算方面的不足，Java 推出了 BigInteger 类，它可以用来完成任意精度的整数运算。本实例将演示其基本的四则运算。实例的运行效果如图 10.6 所示。

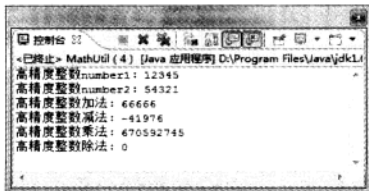


图 10.6 高精度整数运算



指点迷津:

使用该类虽然能大幅度提高运算的精度,但是牺牲的却是性能,因此对于普通运算不推荐使用。

实现过程

(1) 在 Eclipse 中新建项目 115, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建 MathUtil 类, 在 main() 方法中创建两个 BigInteger 对象, 并演示基本的四则运算。关键代码如下:

```
import java.math.BigInteger;
public class MathUtil {
    public static void main(String[] args) {
        BigInteger number1 = new BigInteger("12345");           //声明高精度整数 number1
        BigInteger number2 = new BigInteger("54321");           //声明高精度整数 number2
        BigInteger addition = number1.add(number2);             //计算 number1 加 number2
        BigInteger subtraction = number1.subtract(number2);     //计算 number1 减 number2
        BigInteger multiplication = number1.multiply(number2);  //计算 number1 乘 number2
        BigInteger division = number1.divide(number2);          //计算 number1 除 number2
        System.out.println("高精度整数 number1: " + number1);
        System.out.println("高精度整数 number2: " + number2);
        System.out.println("高精度整数加法: " + addition);
        System.out.println("高精度整数减法: " + subtraction);
        System.out.println("高精度整数乘法: " + multiplication);
        System.out.println("高精度整数除法: " + division);
    }
}
```

指点迷津:

BigInteger 没有 int、long 等类型的构造方法, 所以本实例使用字符串来构造高精度整数。

技术要点

BigInteger 类可以表示不可变的、任意精度的整数。所有操作中, 都以二进制补码形式表示 BigInteger (如 Java 的基本整数类型)。BigInteger 提供了所有 Java 的基本整数操作符的对应物, 并提供了 java.lang.Math 的所有相关方法。另外, BigInteger 还提供了以下运算: 模算术、GCD 计算、质数测试、素数生成、位操作以及一些其他操作。本实例使用 BigInteger 类的 add()、subtract()、multiply() 和 divide() 方法实现了加、减、乘、除四则运算。

实例 116 高精度浮点运算

(实例位置: 配套资源\SL\10\116)

实例说明

为了弥补虚拟机在高精度计算方面的不足, Java 还提供了 BigDecimal 类, 它可以用来完



Note



Note

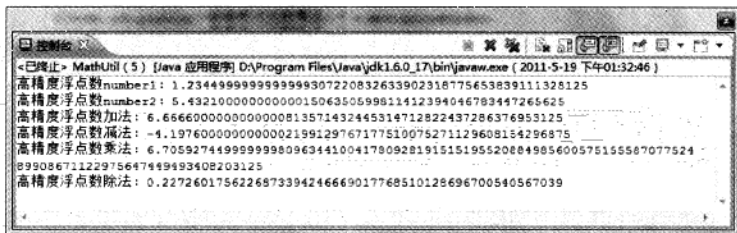


图 10.7 高精度浮点数运算

使用 `BigDecimal` 类虽然可以大幅度提高运算的精度，但是牺牲的却是系统的性能，因此对于普通运算也不推荐使用该类。

实现过程

- (1) 在 Eclipse 中新建项目 116, 在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建 MathUtil 类, 在 main() 方法中创建两个 BigDecimal 对象, 并演示基本的四则运算。关键代码如下:

```
import java.math.BigDecimal;
import java.math.RoundingMode;

public class MathUtil {
    public static void main(String[] args) {
        BigDecimal number1 = new BigDecimal(1.2345);           //声明高精度浮点数 number1
        BigDecimal number2 = new BigDecimal(5.4321);           //声明高精度浮点数 number2
        BigDecimal addition = number1.add(number2);             //计算 number1 加 number2
        BigDecimal subtraction = number1.subtract(number2);     //计算 number1 减 number2
        BigDecimal multiplication = number1.multiply(number2);  //计算 number1 乘 number2
        //以四舍五入的方式获得高精度除法运算的结果
        BigDecimal division = number1.divide(number2, RoundingMode.HALF_UP);
        System.out.println("高精度浮点数 number1: " + number1);
        System.out.println("高精度浮点数 number2: " + number2);
        System.out.println("高精度浮点数加法: " + addition);
        System.out.println("高精度浮点数减法: " + subtraction);
        System.out.println("高精度浮点数乘法: " + multiplication);
        System.out.println("高精度浮点数除法: " + division);
    }
}
```

技术要点

BigDecimal 表示不可变的、任意精度的有符号十进制数。BigDecimal 由任意精度的整数非标度值和 32 位的整数标度 (scale) 组成。如果为零或正数, 则标度是小数点后的位数; 如果为负数, 则将该数的非标度值乘以 10 的负 scale 次幂。因此, BigDecimal 表示的数值是 (unscaledValue \times 10^{scale})。本实例使用 BigInteger 类的 add()、subtract()、multiply() 和 divide() 方法实现了加、减、乘、除四则运算。



实例 117 七星彩号码生成器

(实例位置: 配套资源\SL\10\117)



Note

实例说明

七星彩是中国体彩推出的一种彩票,其基本玩法是:从 0~9 十个数字中随机选择一个,一共选择 7 次,组成一个 7 位数。如果完全和中奖号码相同则中一等奖。本实例将实现一个七星彩号码生成器,实例的运行效果如图 10.8 所示。

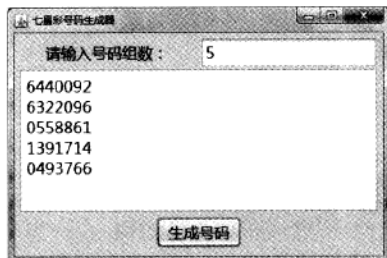


图 10.8 七星彩号码生成器

实现过程

(1) 在 Eclipse 中新建项目 117, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建继承 JFrame 的窗体类 SevenStar, 在该窗体中添加相应控件, 其中获得用户输入号码组数的 JTextField 的名称为 textField, 显示生成号码的 JTextArea 的名称为 textArea, “生成号码”按钮的名称为 button。

(3) 在“生成号码”按钮的事件中, 根据用户输入的组数, 实现随机生成七星彩号码的功能。关键代码如下:

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int times = Integer.parseInt(textField.getText()); // 获得用户输入的需要生成的中奖号码个数
        // 省略提示购买数量太多的代码
        StringBuilder sb = new StringBuilder(); // 利用 StringBuilder 类保存彩票中奖号码
        for (int i = 0; i < times; i++) {
            int number = new Random().nextInt((int) Math.pow(10, 7)); // 生成随机数
            String luckNumber = "" + number;
            while (luckNumber.length() < 7) {
                luckNumber = "0" + luckNumber; // 如果随机数长度不够 7 位用 0 补齐
            }
            sb.append(luckNumber + "\n");
        }
        textArea.setText(sb.toString()); // 显示生成的中奖号码
    }
});
```




技术要点

通过 `Random` 类的实例生成伪随机数流。该类提供了常用的伪随机数生成方法，类型包括 `boolean`、`int`、`long`、`double` 等。本实例使用 `setSeed()` 方法设置随机数种子值，使用 `nextInt()` 方法获得一个小于参数值的随机整数。

1. `setSeed()` 方法

`Random` 类的 `setSeed()` 方法使用单个 `long` 种子，设置此随机数生成器的种子值。其语法如下：

```
public void setSeed(long seed)
```

参数说明

`seed`：为随机数生成器设置的种子值。

2. `nextInt()` 方法

`Random` 类的 `nextInt()` 方法用于返回一个伪随机数，它是取自此随机数生成器序列在 0（包括）和指定值（不包括）之间均匀分布的 `int` 值。其语法如下：

```
public int nextInt(int n)
```

参数说明

`n`：要返回的随机数的范围。必须为正数。

返回值：下一个伪随机数，它是取自此随机数生成器序列中 0（包括）和 `n`（不包括）之间均匀分布的 `int` 值。

实例 118 大乐透号码生成器

（实例位置：配套资源\SL\10\118）

实例说明

大乐透是中国体彩推出的一种彩票，其基本玩法是：从 1~35 随机选取不重复的 5 个数字，从 1~12 随机选取不重复的两个数字组成一个七位数。如果完全和中奖号码相同则中一等奖。本实例将实现一个大乐透号码生成器，实例的运行效果如图 10.9 所示。



图 10.9 大乐透号码生成器

实现过程

（1）在 Eclipse 中新建项目 118，在项目中创建 `com.mingrisoft` 包。



(2) 在 com.mingrisoft 包中创建继承 JFrame 的窗体类 SuperFun, 在该窗体中添加相应控件, 其中获得用户输入号码组数的 JTextField 的名称为 textField, 显示生成号码的 JTextArea 的名称为 textArea, “生成号码”按钮的名称为 button。

(3) 在窗体类 SuperFun 中, 创建生成前段号码的 getStartNumber()方法。关键代码如下:

```
public List<String> getStartNumber() {
    List<String> list = new ArrayList<String>();           //创建前段号码集合
    String luckyNumber = "";
    for (int i = 1; i < 36; i++) {                          //初始化前段号码集合
        if (i < 10) {
            list.add("0" + i + " ");                      //添加 0~9 的号码
        } else {
            list.add("" + i + " ");                        //添加大于 9 的号码
        }
    }
    int roundIndex = 0;
    List<String> luckylist = new ArrayList<String>();       //保存前段号码的 List 集合
    for (int j = 0; j < 5; j++) {
        int amount = list.size();                          //获取前段号码的个数
        Random r = new Random();                          //创建并实例化 Random 的对象
        roundIndex = r.nextInt(amount);                    //获取一个 0~amount-1 的随机数
        luckyNumber = list.get(roundIndex);                 //获取幸运数字
        luckylist.add(luckyNumber);                       //添加到 luckylist 中
        list.remove(roundIndex);                           //移除刚刚产生的号码
    }
    Collections.sort(luckylist);                          //对前段号码进行排序
    return luckylist;
}
```

(4) 在窗体类 SuperFun 中, 创建生成后段号码的 getEndNumber()方法。关键代码如下:

```
public List<String> getEndNumber() {
    List<String> list = new ArrayList<String>();           //创建后段号码集合
    String luckyNumber = "";
    for (int i = 1; i < 13; i++) {                          //初始化后段号码集合
        if (i < 10) {
            list.add("0" + i + " ");                      //添加 0~9 的号码
        } else {
            list.add("" + i + " ");                        //添加大于 9 的号码
        }
    }
    int roundIndex = 0;
    List<String> luckylist = new ArrayList<String>();       //保存后段号码的 List 集合
    for (int j = 0; j < 2; j++) {
        int amount = list.size();                          //获取后段号码的个数
        Random r = new Random();                          //创建并实例化 Random 的对象
        roundIndex = r.nextInt(amount);                    //获取一个 0~amount-1 的随机数
        luckyNumber = list.get(roundIndex);                 //获取幸运数字
        luckylist.add(luckyNumber);                       //添加到 luckylist 中
        list.remove(roundIndex);                           //移除刚刚产生的号码
    }
}
```





```
Collections.sort(luckylist);
```

```
//对后段号码进行排序
```

```
return luckylist;
```

```
}
```

(5) 在“生成号码”按钮的事件中, 根据用户输入的组数, 实现随机生成大乐透号码的功能。关键代码如下:

```
button.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        int times = Integer.parseInt(textField.getText()); //获得用户输入的需要生成的中奖号码个数  
        //省略提示购买数量太多的代码  
        StringBuilder sb = new StringBuilder(); //创建字符串生成器对象  
        for (int i = 0; i < times; i++) {  
            List<String> startList = getStartNumber(); //获得前段号码的集合  
            List<String> endList = getEndNumber(); //获得后段号码的集合  
            for (int m = 0; m < startList.size(); m++) {  
                sb.append(startList.get(m)); //在字符串生成器中添加前段号码  
            }  
            sb.append(" ");  
            for (int n = 0; n < endList.size(); n++) {  
                sb.append(endList.get(n)); //在字符串生成器中添加后段号码  
            }  
            sb.append("\n");  
        }  
        textArea.setText(sb.toString()); //在文本域中显示号码  
    }  
});
```

技术要点

本实例使用 List 集合存储号码, 然后从该 List 集合中随机获得号码, 由于大乐透的每一组号码中的前段号码或后段号码是不允许重复的, 因此, 从 List 集合中随机获得号码后, 就将所获得的号码从该 List 集合中移除, 这样可以避免获得的号码重复。

第 11 章

错误处理

本章读者可以学到如下实例：

- » 实例 119 算数异常
- » 实例 120 数组下标越界异常
- » 实例 121 空指针异常
- » 实例 122 类未发现异常
- » 实例 123 非法访问异常
- » 实例 124 文件未发现异常
- » 实例 125 数据库操作异常
- » 实例 126 方法中抛出异常
- » 实例 127 方法上抛出异常
- » 实例 128 自定义异常类
- » 实例 129 捕获单个异常
- » 实例 130 捕获多个异常



实例 119 算数异常

(实例位置: 配套资源\SL\F1\F119)

实例说明

算数异常即 `ArithmeticException`, 是指整数被 0 除产生的异常。在 Java 中, 如果一个整数被 0 除, 那么将抛出 `ArithmeticException` 异常, 但是浮点数被 0 除, 将不引发算数异常, 这与数学中不同。本实例将演示出现算数异常的情况, 并进行处理。实例的运行效果如图 11.1 所示。

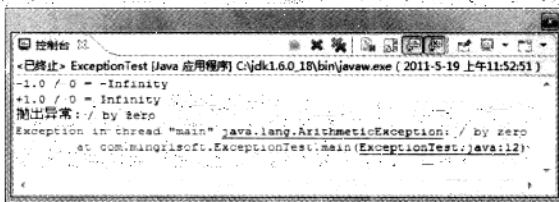


图 11.1 算数异常

实现过程

- (1) 在 Eclipse 中创建项目 119, 并在该项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中创建类文件, 名称为 `ExceptionTest`。在该类的主方法中, 演示出现算数异常的情况。这里将第一条可能出现异常的语句应用 `try...catch` 语句捕获, 并输出异常信息; 第二条可能出现异常的语句不做处理。关键代码如下:

```
public class ExceptionTest {  
    public static void main(String[] args) {  
        System.out.println("-1.0 / 0 = " + (-1.0 / 0)); //演示负浮点数除 0  
        System.out.println("1.0 / 0 = " + (1.0 / 0)); //演示正浮点数除 0  
        try { //捕获异常  
            System.out.println("-1 / 0 = " + (-1 / 0)); //演示负整数除 0  
        } catch (Exception e) {  
            System.out.println("抛出异常: " + e.getMessage());  
        }  
        System.out.println("1 / 0 = " + (1 / 0)); //演示正整数除 0  
        System.out.println("输出结束。");  
    }  
}
```

指点迷津:

由于第二条可能出现异常的语句没有捕获, 在发生异常时, 程序终止了, 所以最后一条语句并没有被输出。

技术要点

本实例的技术要点就是何时抛出算数异常, 以及如何处理。在 Java 程序运行, 系统检查到整数被 0 除的情况下, 它将构造一个新的异常对象, 然后引发该异常。这时将导致出现异常语



句后面的语句不会被执行,也就是终止程序的执行。为了让后面的语句继续执行,可以应用 try...catch 语句捕获该异常,并进行处理,这样将不影响后面语句的执行。

指点迷津:

在 Java 的异常处理机制中,有一个默认处理异常的程序。当程序出现异常时,默认处理程序将显示一个描述异常的字符串,打印异常发生处的堆栈轨迹,并终止程序。

**Note**

实例 120 数组下标越界异常

(实例位置: 配套资源\SL\11\120)

实例说明

数组下标越界异常即 `ArrayIndexOutOfBoundsException`, 当访问的数组元素的下标值大于数组的最大下标值时发生,也就是数组元素的下标值大于等于数组的长度时发生。本实例将演示出现数组下标越界异常 (`ArrayIndexOutOfBoundsException`) 的情况。实例的运行效果如图 11.2 所示。



图 11.2 数组下标越界异常

实现过程

(1) 在 Eclipse 中创建项目 120, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 `ArrayExceptionTest`。在该类的 `main()` 方法中, 首先声明一个长度为 5 的整型数组, 并应用 `Arrays` 对象的 `fill()` 方法, 将数组中所有元素赋值为 8, 然后应用 `for` 循环遍历输出所有数组元素。关键代码如下:

```
public class ArrayExceptionTest {
    public static void main(String[] args) {
        int[] array = new int[5];           //声明一个长度为 5 的整型数组
        Arrays.fill(array, 8);              //将新声明数组的所有元素赋值为 8
        for (int i = 0; i < 6; i++) {        //遍历输出所有数组元素
            System.out.println("array[" + i + "] = " + array[i]);
        }
    }
}
```

指点迷津:

上面代码运行时, 当数组元素的下标值大于 4 时, 将抛出 `ArrayIndexOutOfBoundsException` 异常, 并终止程序的执行。



技术要点

本实例应用的主要技术就是数组下标越界异常。数组下标越界异常属于运行时错误，在程序编译阶段并不会产生错误，只有在程序运行时才有可能出现。例如，一个包含 6 个元素的数组，它的最大下标值应该是 5，如果在程序中访问 `array[6]`，将抛出数组下标越界异常。

指点迷津：

如果要遍历数组中的全部元素，则推荐使用 `foreach` 循环，它可以避免数组的下标越界。如果要使用数组的下标，则需要记住数组的下标是从 0 开始计算的。如果需要使用数组的长度，则推荐使用 `length` 属性。另外使用 `ArrayList` 类也可以避免这些问题。

实例 121 空指针异常

(实例位置：配套资源\SL\11\121)

实例说明

空指针异常即 `NullPointerException`，当应用程序试图在需要对象的地方使用 `null` 时，将抛出该异常。本实例将演示出现空指针异常的情况。实例的运行效果如图 11.3 所示。

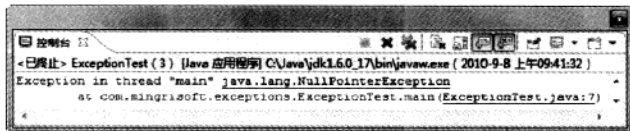


图 11.3 空指针异常

实现过程

- (1) 在 Eclipse 中创建项目 121，并在该项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中创建一个 Java 类，名称为 `NullPointerException`。在该类的 `main()` 方法中定义一个字符串，并将其设置为 `null`，然后调用 `toLowerCase()` 方法将字符串转换成小写并输出。关键代码如下：

```
public class NullPointerException {  
    @SuppressWarnings("null")  
    public static void main(String[] args) {  
        String string = null; //将字符串设置为 null  
        System.out.println(string.toLowerCase()); //将字符串转换成小写  
    }  
}
```

指点迷津：

上面的代码在执行时，由于使用了 `null` 值，所以会抛出 `NullPointerException` 异常。

技术要点

本实例应用的主要技术点就是空指针异常 (`NullPointerException`)。当应用程序试图在需要



对象的地方使用 null 时, 抛出 NullPointerException 异常。这种情况包括:

- ☒ 调用 null 对象的实例方法。
- ☒ 访问或修改 null 对象的字段。
- ☒ 将 null 作为一个数组, 获得其长度。
- ☒ 将 null 作为一个数组, 访问或修改其元素值。
- ☒ 将 null 作为 Throwable 值抛出。

实例 122 类未发现异常

(实例位置: 配套资源\SL\11\122)

实例说明

类未发现异常即 ClassNotFoundException, 表示类没有找到。通常情况下, 在进行数据连接时, 需要借助第三方的数据库驱动包才能完成。如果代码中指定了所需的 Jar 包, 而在程序中并没有提供所需的 Jar 包, 在运行时将产生类未发现异常。本实例将演示抛出类未发现异常的情况。实例的运行效果如图 11.4 所示。

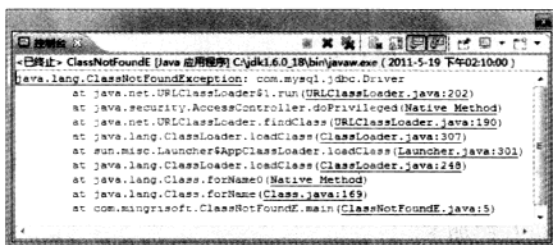


图 11.4 产生的类未发现异常

实现过程

(1) 在 Eclipse 中创建项目 122, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中新建一个 Java 类, 名称为 ClassNotFountE, 在该类的 main() 方法中加载 MySQL 数据库的驱动, 并应用 try...catch 语句捕获异常。关键代码如下:

```
public class ClassNotFountE {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver"); //加载 MySQL 驱动程序
        } catch (ClassNotFoundException e) { //捕获异常
            e.printStackTrace(); //打印堆栈信息
        }
    }
}
```

技术要点

本实例应用的主要技术点就是类未发现异常。当应用程序试图使用以下方法通过字符串名加载类时, 将抛出 ClassNotFoundException 异常。例如:

- ☒ Class 类中的 forName() 方法。



- ☑ ClassLoader 类中的 findSystemClass()方法。
- ☑ ClassLoader 类中的 loadClass()方法。



Note

实例 123 非法访问异常

(实例位置: 配套资源\SL\11\123)

实例说明

非法访问异常即 `IllegalAccessException`, 当不允许访问某类时发生。本实例将演示出现非法访问异常的情况。实例的运行效果如图 11.5 所示。

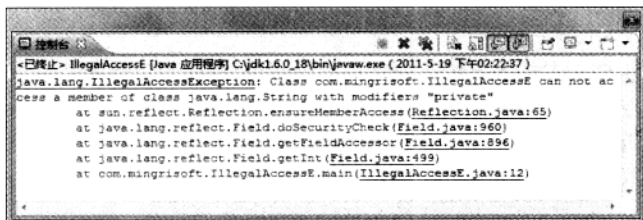


图 11.5 非法访问异常

实现过程

- (1) 在 Eclipse 中创建项目 123, 并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中编写类 `IllegalAccessE`, 在该类的 `main()`方法中, 输出了可能发生异常的运行结果。关键代码如下:

```
import java.lang.reflect.Field;
public class IllegalAccessE {
    public static void main(String[] args) {
        Class<?> clazz = String.class; //获得代表 String 类的类对象
        Field[] fields = clazz.getDeclaredFields(); //获得 String 类的所有域
        for (Field field : fields) { //遍历所有域
            if (field.getName().equals("hash")) { //如果域的名字是 hash
                try {
                    System.out.println(field.getInt("hash")); //输出 hash 的值
                } catch (IllegalArgumentException e) { //捕获 IllegalArgumentException 异常
                    e.printStackTrace();
                } catch (IllegalAccessException e) { //捕获 IllegalAccessException 异常
                    e.printStackTrace();
                }
            }
        }
    }
}
```

技术要点

本实例应用的主要技术点就是非法访问异常, 即 `IllegalAccessException`。当应用程序试图



反射性地创建一个实例（而不是数组）、设置或获取一个字段，或者调用一个方法时，并且当前正在执行的方法无法访问指定类、字段、方法或构造方法的定义时抛出的异常。

实例 124 文件未发现异常

（实例位置：配套资源\SL\11\124）



Note

实例说明

文件未发现异常即 `FileNotFoundException`，当要访问的文件找不到时发生。本实例将演示出现文件未发现异常的情况。实例的运行效果如图 11.6 所示。

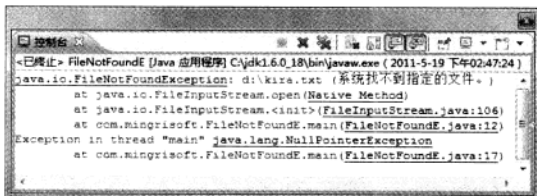


图 11.6 文件未发现异常

实现过程

（1）在 Eclipse 中创建项目 124，并在该项目中创建 `com.mingrisoft` 包。

（2）在 `com.mingrisoft` 包中编写名称为 `FileNotFoundE` 的类，在该类的 `main()` 方法中，捕获可能发生的异常并输出。关键代码如下：

```
public class FileNotFoundE {
    public static void main(String[] args) {
        FileInputStream fis = null;           //创建一个文件输入流对象
        try {
            File file = new File("d:\\kira.txt"); //创建一个文件对象
            fis = new FileInputStream(file);      //初始化文件输入流对象
        } catch (FileNotFoundException e) {    //捕获异常
            e.printStackTrace();
        } finally {
            try {
                fis.close();                    //释放资源
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

指点迷津：

上面的代码在执行时，当 D 盘根目录下不存在 `kira.txt` 文件时，将会抛出 `FileNotFoundException` 异常。



技术要点

本实例应用的主要技术点就是文件未发现异常 (FileNotFoundException)。该异常当程序试图打开指定路径名表示的文件失败时抛出。在不存在具有指定路径名的文件时, 此异常将由 FileInputStream、FileOutputStream 和 RandomAccessFile 构造方法抛出。如果该文件存在, 但是由于某些原因不可访问, 例如试图打开一个只读文件进行写入, 则此时这些构造方法仍然会抛出该异常。

**Note**

实例 125 数据库操作异常

(实例位置: 配套资源\SL\11\125)

实例说明

数据库操作异常即 SQLException, 通常发生在出现数据库访问错误时。本实例将演示出现数据库操作异常的情况。实例的运行效果如图 11.7 所示。

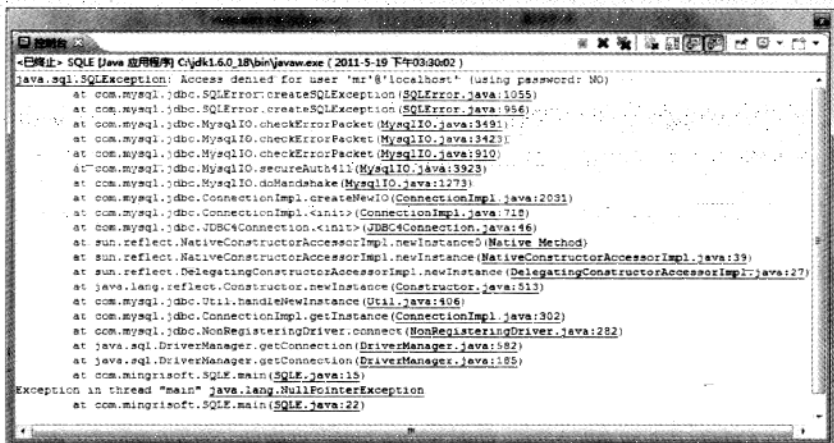


图 11.7 数据库操作异常

指点迷津:

在运行本实例时, 需要将 MySQL 数据库的驱动包配置到构建路径中, 否则将不能抛出 SQLException 异常, 而是抛出 ClassNotFoundException 异常。

实现过程

(1) 在 Eclipse 中创建项目 125, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写类 ExceptionTest, 在该类的主方法中, 编写数据库连接的代码, 并且捕获可能抛出的异常。关键代码如下:

```
public class ExceptionTest {  
    public static void main(String[] args) {  
        String URL = "jdbc:mysql://localhost:3306/db_database"; //MySQL 数据库的 URL
```



```
String DRIVER = "com.mysql.jdbc.Driver";           //MySQL 数据库的驱动
String USERNAME = "mr";                           //数据库的用户名
Connection connection = null;
try {
    Class.forName(DRIVER);                         //加载驱动
    connection = DriverManager.getConnection(URL, USERNAME, ""); //建立连接
} catch (SQLException e) {                         //捕获 SQLException 异常
    e.printStackTrace();
} catch (ClassNotFoundException e) {               //捕获 ClassNotFoundException 异常
    e.printStackTrace();
} finally {
    try {
        connection.close();                       //释放资源
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```



Note

技术要点

本实例应用的主要技术点就是数据库操作异常（SQLException）。该异常用于提供关于数据库访问错误或其他错误信息。提供的具体信息如下：

- ☒ 描述错误的字符串。
- ☒ “SQLstate” 字符串，该字符串遵守 XOPEN SQLstate 约定或 SQL:2003 约定。
- ☒ 特定于每个供应商的整数错误代码。
- ☒ 到下一个 Exception 的链接。
- ☒ 因果关系，如果存在任何导致此 SQLException 的原因。

实例 126 方法中抛出异常

（实例位置：配套资源\SL\11\126）

实例说明

在项目开发中，通常是自顶向下进行的。在完成项目的整体设计后，需要对每个接口和类进行编写。如果一个类使用了其他类还没有实现的方法，则可以在实现其他类方法时让其抛出 UnsupportedOperationException 异常，以便在以后进行修改完成。实例的运行效果如图 11.8 所示。

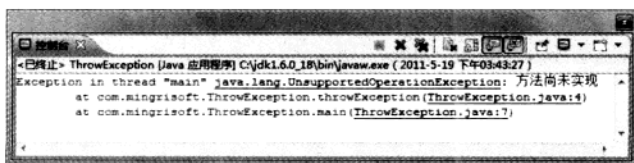


图 11.8 方法中抛出异常



实现过程

(1) 在 Eclipse 中创建项目 126, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写类 ThrowException, 在该类中定义两个方法, 一个是 throwException()方法, 用于抛出异常; 另一个是 main()方法, 用于进行测试。关键代码如下:

```
public class ThrowException {
    public static void throwException() {
        throw new UnsupportedOperationException("方法尚未实现");    //抛出异常
    }
    public static void main(String[] args) {
        ThrowException.throwException();                            //调用抛出异常的方法
    }
}
```



Note

技术要点

本实例应用的主要技术点就是使用 throw 关键字在方法中抛出异常。使用 throw 关键字可以在方法体中抛出异常。该异常既可以是系统预定义异常, 又可以是用户自定义异常。其格式如下:

throw 异常对象;

throw 关键字可以抛出一个异常对象, 并且仅可以应用在方法体中。

脚下留神:

请读者不要和 throws 关键字混淆。

实例 127 方法上抛出异常

(实例位置: 配套资源\SL\11\127)

实例说明

在方法的执行过程中, 如果存在可能遇到引发问题的因素, 则应该在定义方法时加以说明。例如读取文件的方法可能遇到文件不存在的情况, 此时需要在方法声明时抛出文件不存在异常。本实例将演示如何在方法上抛出异常。实例的运行效果如图 11.9 所示。

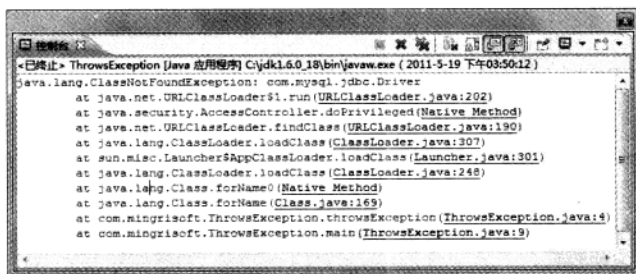


图 11.9 方法上抛出异常

实现过程

(1) 在 Eclipse 中创建项目 127, 并在该项目中创建 com.mingrisoft 包。



(2) 在 com.mingrisoft 包中编写类 ThrowsException, 并且在该类中定义两个方法, 一个是 throwException() 方法, 用于抛出异常; 另一个是 main() 方法, 用于进行测试。关键代码如下:

```
public class ThrowsException {
    public static void throwsException() throws ClassNotFoundException { //抛出异常
        Class.forName("com.mysql.jdbc.Driver");
    }

    public static void main(String[] args) {
        try {
            ThrowsException.throwsException();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

//捕获异常
//调用抛出异常的方法



Note

技术要点

本实例应用的主要技术点是使用 throws 关键字抛出异常。使用 throws 关键字可以在方法体外抛出异常。该异常既可以是系统预定义异常, 又可以是用户自定义异常。下面以 FileInputStream 类的构造方法为例进行讲解, 其声明如下:

```
public FileInputStream(String name) throws FileNotFoundException
```

该方法在定义时就抛出了 FileNotFoundException 异常, 因此如果其他的类使用该方法, 则必须捕获或者继续抛出该异常。throws 关键字可以声明抛出多个异常, 并且仅可以应用在方法体外。

脚下留神:

请读者不要和 throw 关键字混淆。

实例 128 自定义异常类

(实例位置: 配套资源\SL\11\128)

实例说明

在 Java SE API 中, 已经定义了几十种异常类, 它们基本包括了常用的异常类型。然而, 实际开发中有可能遇到 Java SE API 中没有提供的情况, 这时就需要自定义异常类了。本实例将演示如何自定义一个除零异常类。实例的运行效果如图 11.10 所示。

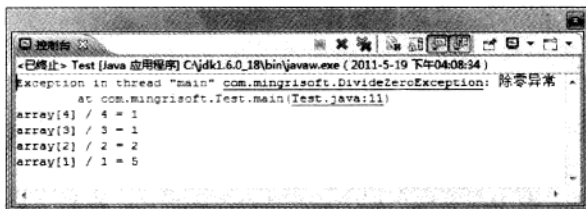


图 11.10 自定义除零异常类



实现过程



Note

- (1) 在 Eclipse 中创建项目 128，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中编写类 DivideZeroException，该类继承自 ArithmeticException 并提供了两个构造方法。关键代码如下：

```
public class DivideZeroException extends ArithmeticException {    //自定义异常类
    public DivideZeroException() {
    }                                //实现默认构造方法
    public DivideZeroException(String msg) {
        super(msg);
    }                                //实现有输出信息的构造方法
}
```

- (3) 在 com.mingrisoft 包中再编写类 Test 进行测试，在 main() 方法中，抛出自定义的异常。关键代码如下：

```
public class Test {
    public static void main(String[] args) {
        int[] array = new int[5];                                //定义长度为 5 的数组
        Arrays.fill(array, 5);                                    //将数组中的元素赋值为 5
        for (int i = 4; i > -1; i--) {                            //遍历整个数组
            if (i == 0) {                                         //如果除 0
                throw new DivideZeroException("除零异常");        //如果除零就抛出有异常信息
            }                                                       //如果不是除零就输出结果
            System.out.println("array[" + i + "] / " + i + " = " + array[i] / i);
        }
    }
}
```

技术要点

编写一个自定义异常类非常简单，只需要继承 Exception 或者 Exception 的子类。一个最简单的自定义异常类代码如下：

```
public class MRSoft extends Exception {}
```

在自定义类时，推荐提供两个构造方法：一个无参数构造方法和一个字符串参数构造方法。它可以为调试提供更加详细的信息。

实例 129 捕获单个异常

(实例位置：配套资源\SL\11\129)

实例说明

当遇到异常时，除了可以将异常抛出，还可以将其捕获。抛出异常虽然简单，但是有时却不得不使用捕获来处理异常。如果程序遇到异常而没有捕获，则程序会直接退出。这在大多数情况下是不能被接受的，至少需要保存程序当前状态才能退出。本实例将演示如何捕获单个异常。实例的运行效果如图 11.11 所示。

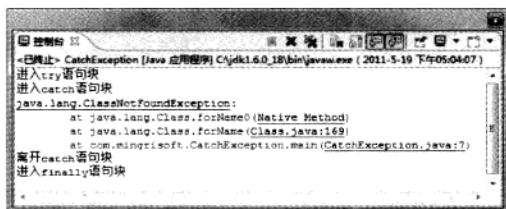


图 11.11 捕获单个异常



Note

实现过程

- (1) 在 Eclipse 中创建项目 129，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中编写类 CatchException，在该类的 main() 方法中应用 try...catch...finally 语句捕获单个异常。关键代码如下：

```
public class CatchException {
    public static void main(String[] args) {
        try {                                     //定义 try 语句块
            System.out.println("进入 try 语句块");
            @SuppressWarnings("unused")
            Class<?> clazz = Class.forName("");    //得到一个空的 Class 对象
            System.out.println("离开 try 语句块");
        } catch (ClassNotFoundException e) {      //定义 catch 语句块
            System.out.println("进入 catch 语句块");
            e.printStackTrace();
            System.out.println("离开 catch 语句块");
        } finally {                               //定义 finally 语句块
            System.out.println("进入 finally 语句块");
        }
    }
}
```

技术要点

本实例主要的技术点是捕获单个异常。Java 中捕获异常是通过 try...catch...finally 语句来完成的。其中 try 语句块是必需的，catch 和 finally 语句块可以选择一个或者两个。try 语句块用来放置可能出现问题的语句，catch 语句块用来放置异常发生后执行的代码，finally 语句块用来放置无论是否发生异常都需要执行的代码。

指点迷津：

捕获异常是一个高开销的操作，因此 try 语句块中的语句应该尽量少。

实例 130 捕获多个异常

(实例位置：配套资源\SL\11\130)

实例说明

在程序中，有时可能会遇到出现多个异常的情况，这时就需要分别捕获这些异常。本实例



将演示如何捕获多个异常。实例的运行效果如图 11.12 所示。

实现过程

(1) 在 Eclipse 中创建项目 130，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中编写类 CatchExceptions，在该类的主方法中，应用 try...catch...finally 语句捕获多个异常。关键代码如下：

```
public class CatchExceptions {
    private static String URL = "jdbc:mysql://localhost:3306/db_database"; //数据库 URL
    private static String DRIVER = "com.mysql.jdbc.Driver"; //数据库驱动
    private static String USERNAME = "mr"; //用户名
    private static String PASSWORD = "mingri"; //密码
    private static Connection conn;
    public static Connection getConnection() {
        try {
            Class.forName(DRIVER); //加载驱动程序
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD); //建立连接
            return conn;
        } catch (ClassNotFoundException e) { //捕获类未发现异常
            e.printStackTrace();
        } catch (SQLException e) { //捕获 SQL 异常
            e.printStackTrace();
        }
        return null;
    }
    public static void main(String[] args) {
        CatchExceptions.getConnection();
    }
}
```

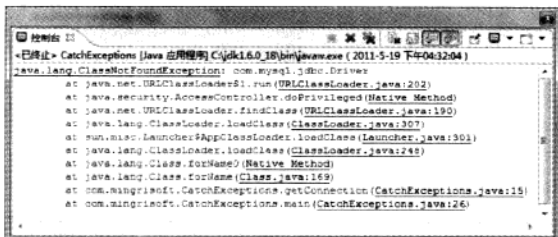


图 11.12 捕获多个异常

指点迷津：

在上面的代码中首先捕获 ClassNotFoundException 异常，然后是 SQLException 异常。

技术要点

本实例主要的技术点是捕获多个异常。在 Java 中捕获异常是通过 try...catch...finally 语句来完成的。其中 try 语句块是必需的，catch 和 finally 语句块可以选择一个或者两个。try 语句块用来放置可能出现问题的语句，如果在 try 语句块中可能出现多个异常，则最好提供多个 catch 语句块来进行捕获，这样可以针对不同的异常提供不同的处理方案。如果 try 语句块中出现的异常和第一个 catch 语句块捕获的异常不匹配，JVM 将比较第二个 catch 语句块，依此类推，直到出现匹配的为止。如果没有找到匹配的，异常对象将抛给调用该方法的方法。

第12章

输入/输出

本章读者可以学到如下实例：

- » 实例 131 显示指定类型的文件
- » 实例 132 以树结构显示文件路径
- » 实例 133 查找替换文本文件内容
- » 实例 134 设置 Windows 系统的文件属性
- » 实例 135 文件批量重命名
- » 实例 136 快速批量移动文件
- » 实例 137 删除文件夹中的.tmp 文件
- » 实例 138 将图片文件保存到数据库
- » 实例 139 从数据库读取图片文件
- » 实例 140 窗体动态加载磁盘文件
- » 实例 141 删除文件夹中所有文件
- » 实例 142 创建磁盘索引文件
- » 实例 143 控制台记录器
- » 实例 144 防止创建多个字符串对象
- » 实例 145 合并多个文本文件
- » 实例 146 对大文件实现分割处理
- » 实例 147 将分割后的文件重新合并
- » 实例 148 读取属性文件单个属性值
- » 实例 149 向属性文件中添加信息
- » 实例 150 在复制文件时使用进度条
- » 实例 151 从 XML 文件中读取数据
- » 实例 152 读取 Jar 文件属性
- » 实例 153 电子通讯录
- » 实例 154 批量复制指定扩展名文件
- » 实例 155 分类保存文件
- » 实例 156 搜索指定文件夹中的文件
- » 实例 157 实现文件锁定功能
- » 实例 158 简单的投票软件
- » 实例 159 压缩所有文本文件
- » 实例 160 将压缩包解压到指定文件夹
- » 实例 161 压缩所有子文件夹
- » 实例 162 深层文件夹压缩包的释放
- » 实例 163 解决压缩包中文乱码
- » 实例 164 Apache 实现文件解压缩
- » 实例 165 把窗体压缩成 ZIP 文件
- » 实例 166 解压缩 Java 对象
- » 实例 167 文件压缩为 RAR 文档
- » 实例 168 解压缩 RAR 压缩包
- » 实例 169 为 RAR 压缩包添加注释
- » 实例 170 获取压缩包详细文件列表
- » 实例 171 从 RAR 压缩包中删除文件
- » 实例 172 在压缩文件中查找字符串
- » 实例 173 重命名 RAR 压缩包中文件
- » 实例 174 创建自解压 RAR 压缩包



实例 131 显示指定类型的文件

(实例位置: 配套资源\SL\12\131)

实例说明

在 Windows 系统中, 通过文件的扩展名来区别不同类型的文件。本实例将根据用户输入的扩展名, 列出指定文件夹内该类型文件的文件名、文件大小和修改时间。实例的运行效果如图 12.1 所示。

实现过程

(1) 在 Eclipse 中创建项目 131, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 FilesList。该类继承了 JFrame 并增加了标签、文本框等控件。单击“选择文件夹”按钮, 将判断用户输入的扩展名是否为空。如果不为空则根据该字符串过滤用户选择的文件夹中的文件, 将符合条件的文件信息增加到表格模型中, 最后更新表格。关键代码如下:



文件名	文件大小	修改时间
第16章 网络通信.doc	2143744	2011-05-11 14:29:42
第1章 Java语言概述.d...	466432	2011-05-11 14:26:52
第2章 Eclipse开发工...	1376768	2011-05-10 16:39:45
第3章 Java语言基础.d...	513024	2011-05-11 14:39:23
第4章 流程控制.doc	834560	2011-05-11 13:34:21
第6章 面向对象入门.d...	929792	2011-05-12 14:44:46
第8章 字符串与包装类	618432	2011-05-12 14:32:33

图 12.1 查看文件夹中指定类型文件的信息

```
protected void do_button_actionPerformed(ActionEvent e) {
    final String fileType = textField.getText();           //获得用户输入的文件类型
    if (fileType.isEmpty()) {                             //提示用户输入文件类型
        JOptionPane.showMessageDialog(this, "请输入文件类型!", "", JOptionPane.WARNING_
MESSAGE);
        return;
    }
    JFileChooser chooser = new JFileChooser();             //创建文件选择器
    chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY); //设置仅能选择文件夹
    chooser.setMultiSelectionEnabled(false);              //禁止选择多个文件夹
    int result = chooser.showOpenDialog(this);             //打开文件选择器
    if (result == JFileChooser.APPROVE_OPTION) {
        File[] listFiles = chooser.getSelectedFile().listFiles(new java.io.FileFilter() {
            @Override
            public boolean accept(File pathname) {
                if (pathname.getName().endsWith(fileType)) {
                    return true;
                } else {
                    return false;
                }
            }
        }); //获得符合条件的文件
        DefaultTableModel model = (DefaultTableModel) table.getModel(); //获得默认表格模型
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"); //指定日期格式

        for (File file : listFiles) {
            String name = file.getName();                 //获得文件名
```



```

        long size = file.length();                //获得文件大小
        String modifyDate = format.format(new Date(file.lastModified())); //获得文件修改日期
        model.addRow(new String[] { name, "" + size, modifyDate }); //向表格中增加数据
    }
    table.setModel(model);                        //更新表格模型
}

```



Note

技术要点

本实例使用 File 类的 listFiles() 方法来获得符合条件的文件，该方法的声明如下：

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

参数说明

filter: 指定的文件过滤器。

实例 132 以树结构显示文件路径

(实例位置: 配套资源\SL\12\132)

实例说明

目前主流的操作系统都是以树结构来管理文件。本实例将使用 Java 中的树控件来显示 Windows 系统中的文件结构，其运行效果如图 12.2 所示。

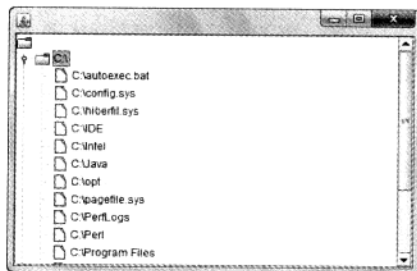


图 12.2 查看 C 盘中的非隐藏文件

实现过程

(1) 在 Eclipse 中创建项目 132，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 FilesTree。该类继承了 JFrame 并增加了一个 JTree 控件。在创建 JTree 对象时，为其增加了可用的系统文件根（如 Windows 系统的 C 盘、D 盘等）。关键代码如下：

```

File[] disks = File.listRoots();                //获得所有可用的文件系统根
DefaultMutableTreeNode root = new DefaultMutableTreeNode(); //创建节点
for (File disk : disks) {                        //将 File 数组中的元素增加到节点上
    root.add(new DefaultMutableTreeNode(disk));
}
tree = new JTree(root);                        //使用节点创建树控件

```

(3) 监听树节点选择事件，当用户选择一个节点后，向该节点中增加子节点。关键代码如下：

```

protected void do_tree_valueChanged(TreeSelectionEvent e) {
    //获得用户选择的节点
    DefaultMutableTreeNode selectNode = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
    File selectFile = (File) selectNode.getUserObject(); //获得节点代表的 File 类型对象
}

```



Note

```

if (selectFile.isDirectory()) { //如果 File 类型对象是文件夹
    File[] files = selectFile.listFiles(new FileFilter() {
        @Override
        public boolean accept(File pathname) { //过滤掉隐藏类型文件
            if (pathname.isHidden()) {
                return false;
            } else {
                return true;
            }
        }
    });
    for (File file : files) { //将符合条件的 File 类型对象增加到用户选择的节点中
        selectNode.add(new DefaultMutableTreeNode(file));
    }
} else {
    return;
}
}

```

技术要点

在 Windows 操作系统中，可以使用 File 类的静态方法 listRoots() 来查看可用系统文件根。该方法的声明如下：

```
public static File[] listRoots()
```

返回值：由可用系统文件根组成的数组。

实例 133 查找替换文本文件内容

(实例位置：配套资源\SL\12\133)

实例说明

目前主流的文本编辑工具都提供了替换功能，但是需要先打开文本文件。本实例可以直接替换用户选择的文本文件内容，其运行效果如图 12.3 所示。

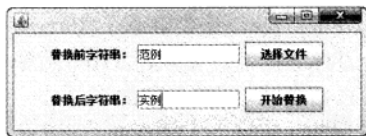


图 12.3 将文件中的“范例”替换成“实例”

实现过程

(1) 在 Eclipse 中创建项目 133，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 ReplaceTool。该类继承了 JFrame 并增加了标签、文本框和按钮控件。单击“开始替换”按钮，可以完成文本替换功能，其事件监听器的关键代码如下：

```

protected void do_replaceButton_actionPerformed(ActionEvent e) {
    String before = beforeTextField.getText(); //获得替换前字符串
    //省略校验代码
    String after = afterTextField.getText(); //获得替换后字符串
    //省略校验代码
    FileReader reader = null; //创建文件读流
}

```



```

FileWriter writer = null;           //创建文件写流
StringBuilder sb = new StringBuilder(); //使用 StringBuilder 对象保存文件内容
int flag = 0;                       //声明文件读入标识
char[] temp = new char[1024];       //使用字符数组读入文件
try {
    reader = new FileReader(textFile); //使用选择的文件创建读流
    while ((flag = reader.read(temp)) != -1) { //读入文件中的内容
        sb.append(temp);
    }
    String content = sb.toString().replace(before, after); //替换字符串
    writer = new FileWriter(textFile); //创建文件写流
    writer.write(content); //将替换后的字符串写入到文件
} catch (FileNotFoundException e1) {
    e1.printStackTrace();
} catch (IOException e1) {
    e1.printStackTrace();
} finally {
    //省略释放资源代码
}
OptionPane.showMessageDialog(this, "字符串替换成功!");
return;
}

```

技术要点

String 类是不可变类型，如果修改该类对象则会产生一个新的对象，因此不适合用于做大量修改。如果遇到这个需求，可以使用 StringBuilder 类替代 String 类。

实例 134 设置 Windows 系统的文件属性

(实例位置：配套资源\SL\12\134)

实例说明

在 Windows 操作系统中，可以将文件设置成不同的属性以此来保护文件。例如，只读属性可以限制修改文件的内容，隐藏属性可以限制用户使用该文件。本实例将开发一个工具来修改 Windows 系统的文件属性，其运行效果如图 12.4 所示。

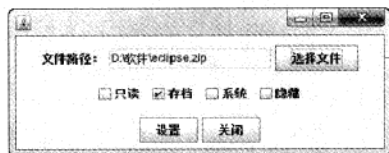


图 12.4 Eclipse 文件的属性

实现过程

(1) 在 Eclipse 中创建项目 134，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 FileAttributeModification。该类继承了 JFrame 并增加了标签、文本框、按钮和复选框控件。单击“设置”按钮，可以设置选中的属性。其事件监听器的关键代码如下：

```

protected void do_setButton_actionPerformed(ActionEvent e) {
    StringBuilder command = new StringBuilder("attrib "); //创建命令

```





Note

```
if (readCheckBox.isSelected()) { //如果需要增加只读属性
    command.append(" +R "); //在命令行增加参数
}
if (archiveCheckBox.isSelected()) { //如果需要增加存档属性
    command.append(" +A "); //在命令行增加参数
}
if (systemCheckBox.isSelected()) { //如果需要增加系统属性
    command.append(" +S "); //在命令行增加参数
}
if (hiddenCheckBox.isSelected()) { //如果需要增加隐藏属性
    command.append(" +H "); //在命令行增加参数
}
command.append(selectFile.getAbsolutePath()); //增加文件路径
try {
    Runtime.getRuntime().exec(command.toString()); //执行命令
} catch (IOException e1) {
    e1.printStackTrace();
}
```

技术要点

Java 程序也可以调用操作系统的命令来完成特定的功能，可以使用 `Runtime` 类的 `exec()` 方法实现。该方法的声明如下：

```
public Process exec(String command)throws IOException
```

参数说明

command: 操作系统的命令。

实例 135 文件批量重命名

(实例位置: 配套资源\SL\12\135)

实例说明

在 Windows 操作系统中，支持文件重命名但是不支持批量重命名。对于多个文件，如果要命名为希望的格式显得非常麻烦。本实例将开发一个文件批量重命名工具，其运行效果如图 12.5 和图 12.6 所示。



图 12.5 替换前的预览效果



图 12.6 替换后的实际效果



实现过程

(1) 在 Eclipse 中创建项目 135，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 RenameTool。该类继承了 JFrame 并增加了标签、文本框、按钮和复选框等控件。选中复选框控件，单击“重命名”按钮，可以完成文件的重命名操作。其事件监听器的关键代码如下：

```
protected void do_renameButton_actionPerformed(ActionEvent e) {
    //省略获得用户输入以及校验是否选择了非空文件夹代码
    textArea.setText(""); //清空文本域中的数据
    for (File selectFile : selectFiles) {
        String fileName = selectFile.getName(); //获得文件名
        //省略处理文件名字符串代码
        fileName = selectFile.getParent() + File.separator + fileName; //获得修改后的文件名
        if (choiceCheckBox.isSelected()) {
            textArea.append(fileName + "\n\r"); //在文本区中显示重命名的结果
            selectFile.renameTo(new File(fileName)); //重命名文件
        } else {
            textArea.append(fileName + "\n\r"); //重命名文件
        }
    }
}
```



Note

技术要点

File 类的 renameTo() 方法可以用来对操作系统中的文件重新命名，该方法的声明如下：

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

参数说明

dest: 使用新文件名表示的 File 对象。

实例 136 快速批量移动文件

(实例位置：配套资源\SL\12\136)

实例说明

在 Windows 操作系统中，移动大量文件花费的时间非常可观。使用 Java 语言提供的文件重命名方式也可以实现文件移动功能，而且能够节约大量的时间。本实例将完成这个功能，其运行效果如图 12.7 所示。

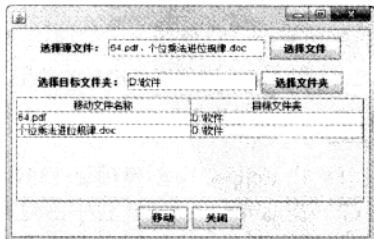


图 12.7 移动文件后的效果

实现过程

(1) 在 Eclipse 中创建项目 136，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 FileMoveTool。该类继承了 JFrame 并增加了标签、文本框、按钮和复选框等控件。在选择源文件、目标文件夹后，单击“移动”按



钮，可以完成文件的移动功能。其事件监听器关键代码如下：

```
protected void do_moveButton_actionPerformed(ActionEvent e) {
    //省略选择文件和文件夹的校验代码
    DefaultTableModel model = (DefaultTableModel) table.getModel();           //获得表格模型
    for (File selectFile : selectFiles) {
        String fileName = targetDirectory.getAbsolutePath() + File.separator + selectFile.getName(); //获
        //移动文件
        selectFile.renameTo(new File(fileName));
        //向表格模型中增加数据
        model.addRow(new String[] { selectFile.getName(), targetDirectory.getAbsolutePath() });
    }
    table.setModel(model);           //设置表格模型
}
```

技术要点

File 类的 renameTo() 方法除了可以用于文件重命名，还可以同时修改文件所在的绝对路径，这样就相当于完成了移动文件功能。

实例 137 删除文件夹中的.tmp 文件

(实例位置：配套资源\SL\12\137)

实例说明

操作系统在运行一段时间后，会生成大量的.tmp 临时文件，它们会影响系统的性能。本实例将删除指定文件夹中的.tmp 类型文件，其运行效果如图 12.8 和图 12.9 所示。读者可以在此实例基础上实现整个系统内的文件删除功能。



图 12.8 选择文件夹后的效果



图 12.9 清理后的效果

实现过程

(1) 在 Eclipse 中创建项目 137，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 TempDeletionTool。该类继承了 JFrame 并增加了标签、文本框、按钮和表格等控件。在选择完文件夹后，单击“清理”按钮将删除文件夹内的.tmp 文件。其事件监听器关键代码如下：

```
protected void do_clearButton_actionPerformed(ActionEvent e) {
    //省略校验代码
    DefaultTableModel model = (DefaultTableModel) table.getModel();           //获得表格模型
}
```



```

for (int i = 0; i < tempFiles.length; i++) {
    if (tempFiles[i].delete()) {
        model.setValueAt("已处理", i, 3);
    }
}
}

```

//删除文件
//修改表格内容

技术要点

File 类的 delete()方法可以删除操作系统中指定路径的文件，该方法的声明如下：

```
public boolean delete()
```

返回值：如果删除文件成功则返回 true，否则返回 false。



Note

实例 138 将图片文件保存到数据库

(实例位置：配套资源\SL\12\138)

实例说明

为了提高数据的安全性，可以将其写入到数据库中。本实例将在 MySQL 数据库中保存图片，其运行效果如图 12.10 所示。

实现过程

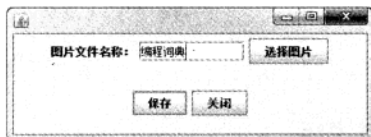


图 12.10 输入文件名称后的效果

(1) 在 Eclipse 中创建项目 138，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 DBHelper。其中定义了一个静态方法 savePicture()，用于完成保存图片的功能。关键代码如下：

```

public static boolean savePicture(Picture picture) {
    try {
        Class.forName(DRIVER);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    FileInputStream in = null;
    String sql = "insert into tb_picture (picturename, picturefile) values (?, ?)"; //定义 SQL 语句
    Connection conn = null;
    PreparedStatement ps = null;
    try {
        in = new FileInputStream(picture.getPictureFile());
        conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        ps = conn.prepareStatement(sql);
        ps.setString(1, picture.getPictureName());
        ps.setBlob(2, in);
        ps.execute();
        return true;
    } catch (SQLException e) {

```

//加载数据库驱动
//获得文件输入流
//获得数据库连接
//获得预处理对象
//设置文件名
//设置文件输入流
//保存数据



Note

```

        c.printStackTrace();
    } catch (FileNotFoundException e) {
        c.printStackTrace();
    } finally {
        //省略释放资源代码
    }
    return false;
}

```

指点迷津:

由于篇幅限制, 关于窗体以及 JDBC 的信息请参考源代码文件。

技术要点

在向 MySQL 中保存图片时, 数据表中对应字段的类型应该是 TINYBLOB、BLOB、MEDIUMBLOB 或者 LONGBLOB 之一, 然后使用 PreparedStatement 接口中定义的 setBlob() 方法将文件流写入数据库。

实例 139 从数据库读取图片文件

(实例位置: 配套资源\SL\12\139)

实例说明

在向数据库中写入图片之后, 还可以根据需要将其读取。本实例将完成在 MySQL 中读取图片的功能, 其运行效果如图 12.11 所示。

实现过程

(1) 在 Eclipse 中创建项目 139, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 DBHelper。其中定义了一个静态方法 retrievePicture, 用于完成读取图片的功能。关键代码如下:

```

public static ImageIcon retrievePicture(Picture picture) {
    try {
        Class.forName(DRIVER); //加载数据库驱动
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    //定义 SQL 语句
    String sql = "select picturefile from tb_picture where id = " + picture.getId() + " and picturename = " + picture.getPictureName() + "";
    Connection conn = null;
    Statement stat = null;
    ResultSet rs = null;
    try {

```



图 12.11 成功查询数据库中图片的效果



```

conn = DriverManager.getConnection(URL, USERNAME, PASSWORD); //获得数据库连接
stat = conn.createStatement(); //获得语句对象
rs = stat.executeQuery(sql); //获得查询结果
if (rs.next()) {
    Blob pictureFile = rs.getBlob("picturefile"); //获得 Blob 对象
    return new ImageIcon(pictureFile.getBytes(1, (int) pictureFile.length())); //创建图表
} else {
    return null;
}
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    //省略释放资源代码
}
return null;
}

```



Note

指点迷津:

由于篇幅限制, 关于窗体以及 JDBC 的信息请参考源代码文件。

技术要点

使用 ResultSet 接口中定义的 getBlob() 方法可以从数据库中读取 Blob 及其相关类型的数据, 然后使用 ImageIcon 类的构造方法可以将读取的数据转换成图标。

实例 140 窗体动态加载磁盘文件

(实例位置: 配套资源\SL\12\140)

实例说明

使用图形化操作系统时, 打开一个文件夹会显示当前文件夹中的文件及子文件夹。本实例将完成类似的功能。用户在选择一个文件夹后, 可以在表格中动态加载该文件夹的内容, 其运行效果如图 12.12 所示。

实现过程

(1) 在 Eclipse 中创建项目 140, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 FileListFrame。该类继承了 JFrame 并增加了标签、文本框、按钮和表格等控件。在选择完文件夹后, 将在表格中动态加载文件夹中的文件。其事件监听器的关键代码如下:

```

protected void do_chooseButton_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser(); //创建文件选择器
    chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY); //限制仅能选择文件夹
    chooser.setMultiSelectionEnabled(false); //禁止多重选择
}

```

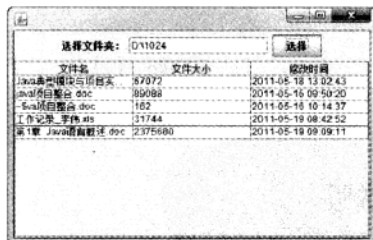


图 12.12 显示文件夹中的文件



间格式化方式

```

int result = chooser.showOpenDialog(this); //打开文件选择器
if (result == JFileChooser.APPROVE_OPTION) {
    File selectDirectory = chooser.getSelectedFile(); //获得选择的文件夹
    chooseTextField.setText(selectDirectory.getAbsolutePath()); //显示文件夹位置
    final File[] files = selectDirectory.listFiles(); //获得文件夹中的文件及其子目录
    final DefaultTableModel model = (DefaultTableModel) table.getModel(); //获得表格模型
    final SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"); //创建时间
    new Thread() {
        public void run() {
            for (File file : files) {
                if (file.isFile()) { //如果是文件则向表格模型中增加数据
                    model.addRow(new Object[] { file.getName(), file.length(), format.format(
(new Date(file.lastModified()))));
                }
            }
        }
    }.start();
}
}

```

技术要点

File 类提供了很多方法可以获得指定文件的属性，如文件大小、修改时间等。由于 Java 中将文件和文件夹统一使用 File 进行管理，因此提供了 isFile() 和 isDirectory() 方法来判断是否为文件或者文件夹。

实例 141 删除文件夹中所有文件

(实例位置：配套资源\SL\12\141)

实例说明

删除文件是操作系统提供的基本功能之一，本实例将使用 Java 语言实现这个功能，同时记录删除的文件路径。实例的运行效果如图 12.13 所示。

实现过程

- (1) 在 Eclipse 中创建项目 141，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建类文件，名称



图 12.13 删除指定文件夹中的文件



为 FileDeletionFrame。该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择完文件夹后,单击“删除”按钮,将删除该文件夹。这里使用 deleteFile()方法完成删除任务,关键代码如下:

```
private void deleteFile(File root, JTextArea textArea) {
    if (root.isFile()) { //如果是文件则直接删除
        root.delete();
        textArea.append(root.getAbsolutePath() + "\n\r");//将删除的文件记录到文本区中
    } else {
        File[] files = root.listFiles(); //获得文件夹中的文件及其子文件夹
        for (File file : files) {
            if (file.isFile()) { //如果是文件则直接删除
                file.delete();
                textArea.append(file.getAbsolutePath() + "\n\r");//将删除的文件记录到文本区中
            } else { //如果是文件夹则进行迭代
                deleteFile(file, textArea);
            }
        }
        root.delete(); //删除空文件夹
    }
}
```



Note

技术要点

File 类中的 delete()方法可以删除文件和空文件夹,对于非空文件夹需要先删除该文件夹中的文件。该方法的声明如下:

```
public boolean delete()
```

返回值: 如果删除成功返回 true, 否则返回 false。

实例 142 创建磁盘索引文件

(实例位置: 配套资源\SL\12\142)

实例说明

如果磁盘中存在大量的文件,则在查找时非常耗时,因此通常建立一个磁盘索引文件来节约查找时间。本实例将实现这个功能,其运行效果如图 12.14 所示。



图 12.14 为 D 盘建立磁盘索引文件



实现过程

(1) 在 Eclipse 中创建项目 142，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 IndexFileFrame。该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择完文件夹后，单击“创建索引”按钮，将根据用户在组合框中选择的磁盘创建文件索引，并在文本区中显示。该按钮事件监听器的代码如下：

```
protected void do_createButton_actionPerformed(ActionEvent arg0) {
    //省略校验代码
    String disc = comboBox.getSelectedItem().toString(); //获得用户选择的磁盘
    final List<String> llist = new ArrayList<String>();    //用 list 保存索引
    final File rootFile = new File(disc);                //利用用户选择的磁盘创建 File 对象
    final StringBuilder sb = new StringBuilder();        //利用 StringBuilder 对象保存写入的索引
    progressBar.setIndeterminate(true);                 //设置滚动条开始滚动
    new Thread() {                                       //在一个新的线程中处理创建索引和写入索引的操作
        @Override
        public void run() {
            getFilePath(list, rootFile);                //获得磁盘上所有文件的路径
            Iterator<String> iterator = list.iterator(); //创建迭代器
            while (iterator.hasNext()) {                 //遍历 list
                sb.append(iterator.next());
                sb.append("\r\n");
                try {
                    Thread.sleep(100);                 //线程休眠 0.1 秒
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                textArea.setText(sb.toString());        //在文本域中显示文件路径
            }
            FileWriter fileWriter = null;
            try {
                fileWriter = new FileWriter(chooseFile);
                fileWriter.write(textArea.getText());   //向用户选择的文本文件写入数据
                fileWriter.flush();
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                //省略释放资源代码
            }
            progressBar.setIndeterminate(false);        //停止进度条的滚动
            JOptionPane.showMessageDialog(null, "索引创建成功");//提示用户索引创建成功
        }
    }.start();
}
```

技术要点

在将文件名写入到索引文件时，可以使用 FileWriter 类的 write()方法，它可以直接将字符串写入到文件。该方法的声明如下：

```
public void write(String str) throws IOException
```



Note

参数说明

str: 写入的字符串。

实例 143 控制台记录器

(实例位置: 配套资源\SL\12\143)

实例说明

使用 Java 提供的 IO 工具类可以方便地读写文件。本实例将演示如何保存用户在控制台上输入的内容, 其运行效果如图 12.15 所示。

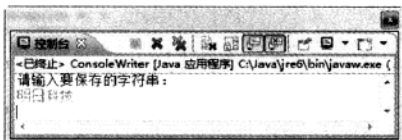


图 12.15 控制台输入内容

实现过程

(1) 在 Eclipse 中创建项目 143, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 ConsoleWriter。在该类的 main() 方法中完成了控制台输入内容的获取和在磁盘上输出。关键代码如下:

```
public static void main(String[] args) {
    System.out.println("请输入要保存的字符串: ");
    Scanner scanner = new Scanner(System.in);
    String text = scanner.nextLine();
    FileWriter writer = null;
    try {
        writer = new FileWriter("d://test.txt");
        writer.write(text);
        writer.flush();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (writer != null) {
            try {
                writer.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

//提示用户输入字符串
//获得控制台输入流
//获得用户输入
//获得文件写入流
//写入字符串
//刷新缓存

技术要点

在 System 类中, 提供了标准输入 (in)、标准输出 (out) 和标准错误 (error)。如果需要



获取控制台上输入的内容，可以使用 `System.in`。

实例 144 防止创建多个字符串对象

(实例位置：配套资源\SL\12\144)



Note

实例说明

`String` 类是不可变的，这意味着每次修改都需要创建多个对象，这势必增加系统开销。如果使用 `StringBuilder` 类就能很好地回避这个问题。本实例使用 `StringBuilder` 类将用户选择的爱好连接成一个字符串进行保存，收集用户爱好的界面运行效果如图 12.16 所示。

实现过程

(1) 在 Eclipse 中创建项目 144，并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件，名称为 `HobbyFrame`。该类继承了 `JFrame` 并增加了标签、复选框和按钮控件。在选择完爱好后，单击“写入文件”按钮，可以将选择的爱好写入到本地文件。其事件监听器的关键代码如下：

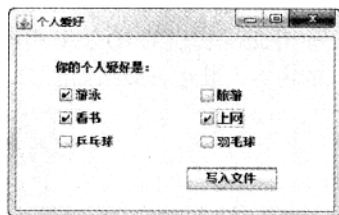


图 12.16 用户选择爱好后的效果

```
protected void do_button_actionPerformed(ActionEvent arg0) {
    StringBuffer buffer = new StringBuffer();
    if (checkBox1.isSelected()) { //判断指定的复选框 checkBox 是否被选中
        buffer.append(checkBox1.getText() + " "); //将可变的字符序列进行追加信息
    }
    //省略添加其他选项的代码
    FileWriter writer = null;
    try {
        writer = new FileWriter("d://hobby.txt"); //创建文件输出流对象
        writer.write(buffer.toString()); //写入用户选择的爱好
        writer.flush(); //清除缓存
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        //释放资源
    }
}
```

技术要点

`StringBuffer` 类也是可变字符串类，它的 `append()` 方法可以在不产生新对象的情况下增加字符串。该方法的声明如下：

```
public StringBuffer append(String str)
```

参数说明

`str`: 增加的字符串。

此外这个类也是线程安全的。如果不是在多线程环境下，也可以使用 `StringBuilder` 类。



实例 145 合并多个文本文件

(实例位置: 配套资源\SL\12\145)

实例说明

文本文件是操作系统常用的文件类型, 由于没有各种复杂的格式, 它可以方便地进行读写。本实例将合并多个文本文件, 实例的运行效果如图 12.17 所示。

实现过程

(1) 在 Eclipse 中创建项目 145, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 TextFileConcatenation。该类继承了 JFrame 并增加了标签、文本框、按钮控件。在选择完文件夹后, 单击“合并文件”按钮, 可以将选择的文件夹中所有文本文件内容写入到 D 盘的 concatenation.txt 文件中。该按钮事件监听器的关键代码如下:



图 12.17 选择文件夹后的效果

```
protected void do_concatButton_actionPerformed(ActionEvent e) {
    //省略校验代码
    BufferedReader reader = null;
    FileWriter writer = null;
    try {
        writer = new FileWriter("d://concatenation.txt");           //创建文件输出流
        for (File textFile : textFiles) {                          //遍历用户选择的文本文件
            reader = new BufferedReader(new FileReader(textFile)); //创建缓冲输入流
            String line;
            while ((line = reader.readLine()) != null) {           //将读入的数据写入到文件中
                writer.write(line);
            }
        }
        JOptionPane.showMessageDialog(this, "文件合并成功!", "提示信息", JOptionPane.
INFORMATION_MESSAGE);
        return;
    } catch (IOException e1) {
        e1.printStackTrace();
    } finally {
        //省略释放资源代码
    }
}
```

技术要点

在读取文本文件时, 可以使用 `BufferedReader` 类。它的 `readLine()` 方法可以方便地从文件



Note



中读入一行数据，该方法的声明如下：

```
public String readLine() throws IOException
```

返回值：读入的字符串。



Note

实例 146 对大文件实现分割处理

(实例位置：配套资源\SL\12\146)

实例说明

为了方便携带和传输（如邮箱的附件都有大小限制），可以将文件进行分割。本实例将完成这个功能，其运行效果如图 12.18 所示。

实现过程

(1) 在 Eclipse 中创建项目 146，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 ComminuteFrame。该类继承了 JFrame 并增加了标签、文本框和按钮控件。在选择完文件后，单击“分割”按钮，可以将选择的文件进行分割。完成分割的代码位于 ComminuteUtil 工具类中。关键代码如下：

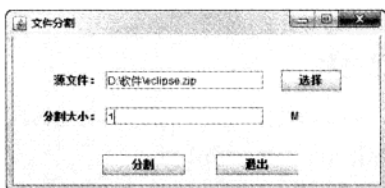


图 12.18 选择文件夹后的效果

```
public void fenGe(File commFile, File untieFile, int filesize) {
    FileInputStream fis = null;
    int size = 1024 * 1024; //用来指定分割文件以 MB 为单位
    try {
        if (!untieFile.isDirectory()) { //如果保存分割文件地址不是路径
            untieFile.mkdirs(); //创建该路径
        }
        size = size * filesize;
        int length = (int) commFile.length(); //获取文件大小
        int num = length / size; //获取文件大小除以 MB 的得数
        int yu = length % size; //获取文件大小与 MB 相除的余数
        String newfengeFile = commFile.getAbsolutePath(); //获取保存文件的完成路径信息
        int fileNew = newfengeFile.lastIndexOf(".");
        String strNew = newfengeFile.substring(fileNew, newfengeFile.length()); //截取字符串
        fis = new FileInputStream(commFile); //创建 FileInputStream 类对象
        File[] fl = new File[num + 1]; //创建文件数组
        int begin = 0;
        for (int i = 0; i < num; i++) { //循环遍历数组
            //指定分割后小文件的文件名
            fl[i] = new File(untieFile.getAbsolutePath() + "\\\" + (i + 1) + strNew + ".tem");
            if (!fl[i].isFile()) {
                fl[i].createNewFile(); //创建该文件
            }
        }
        FileOutputStream fos = new FileOutputStream(fl[i]);
        byte[] bl = new byte[size];
        fis.read(bl); //读取分割后的小文件
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (fis != null) {
            fis.close();
        }
    }
}
```




```

        fos.write(bl);                                //写文件
        begin = begin + size * 1024 * 1024;
        fos.close();                                //关闭流
    }
    if (yu != 0) {                                    //文件大小与指定文件分割大小相除的余数不为 0
        //指定文件分割后数组中最后一个文件名
        fl[num] = new File(untieFile.getAbsolutePath() + "\\" + (num + 1) + strNew + ".tem");
        if (!fl[num].isFile()) {
            fl[num].createNewFile();                //新建文件
        }
        FileOutputStream fyu = new FileOutputStream(fl[num]);
        byte[] byt = new byte[yu];
        fis.read(byt);
        fyu.write(byt);
        fyu.close();
    }
} catch (Exception e) {
    e.printStackTrace();
}
}

```

技术要点

实现本实例的关键是通过输入流读取要分割的文件，再分别从流中读取相应的字节数，将其写入到以.tem 为后缀的文件中。通过 FileInputStream 类的 read()方法可实现读取文件。

实例 147 将分割后的文件重新合并

(实例位置：配套资源\SL\12\147)

实例说明

实例 146 完成了文件的分割任务，如果需要正常使用文件，还需要将其合并。本实例将完成这个功能，其运行效果如图 12.19 所示。

实现过程

(1) 在 Eclipse 中创建项目 147，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 UniteFrame。该类继承了 JFrame 并增加了标签、文本区、按钮等控件。在选择完文件后，单击“合并”按钮，可以将选择的文件进行合并。完成合并的代码位于 UniteUtil 工具类中。关键代码如下：



图 12.19 选择分割文件后的效果

```

public void heBing(File[] file, File cunDir, String hz) {
    try {

```

```

        File heBingFile = new File(cunDir.getAbsolutePath() + "\\UNTIE" + hz); //指定分割后文件
        的文件名

```




```
if (!heBingFile.isFile()) {
    heBingFile.createNewFile();
}
FileOutputStream fos = new FileOutputStream(heBingFile); //创建 FileOutputStream 对象
for (int i = 0; i < file.length; i++) { //循环遍历要进行合并的文件数组对象
    FileInputStream fis = new FileInputStream(file[i]);
    int len = (int) file[i].length(); //获取文件长度
    byte[] bRead = new byte[len];
    fis.read(bRead); //读取文件
    fos.write(bRead); //写入文件
    fis.close(); //将流关闭
}
fos.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

技术要点

本实例实现文件合并，仍然是通过文件字节输入/输出流。在进行文件合并时，需要将要进行合并的所有文件全部读取之后，写入到新文件中。

实例 148 读取属性文件单个属性值

(实例位置：配套资源\SL\12\148)

实例说明

在编程中，有些信息经常需要修改，如 JDBC 属性，此时可以考虑将它们写入到属性文件中。本实例将完成属性文件的读入功能，其运行效果如图 12.20 所示。

实现过程

(1) 在 Eclipse 中创建项目 148，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 PropertiesFileTool。该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择完文件后，会在下面的表格中显示文件中保持的键和值。该按钮事件监听器的关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser(); //创建文件选择器
    chooser.setFileFilter(new FileNameExtensionFilter("属性文件", "properties")); //设置文件过滤器
    chooser.setSelectionMode(JFileChooser.FILES_ONLY); //设置仅能选择文件
    chooser.setMultiSelectionEnabled(false); //禁止多个选择
    int result = chooser.showOpenDialog(this);
```

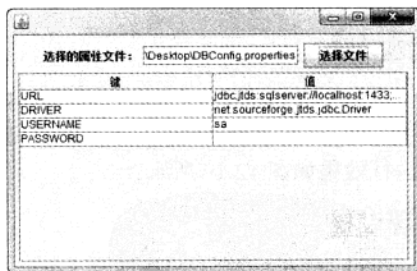


图 12.20 选择 DBConfig 属性文件



```

if (result == JFileChooser.APPROVE_OPTION) {
    File selectFile = chooser.getSelectedFile();           //获得选择的文件
    textField.setText(selectFile.getAbsolutePath());      //显示选择的文件路径
    Properties properties = new Properties();
    FileReader reader = null;
    DefaultTableModel model = (DefaultTableModel) table.getModel();//获得表格模型
    try {
        reader = new FileReader(selectFile);
        properties.load(reader);                          //加载属性文件
        Enumeration<?> keys = properties.propertyNames(); //获得属性文件的键枚举
        while (keys.hasMoreElements()) {
            String key = (String) keys.nextElement();    //获得键
            String value = properties.getProperty(key);   //获得值
            model.addRow(new String[] { key, value });    //向表格中增加记录
        }
    } catch (FileNotFoundException e1) {
        e1.printStackTrace();
    } catch (IOException e1) {
        e1.printStackTrace();
    } finally {
        //省略释放资源代码
    }
}
}

```



Note

技术要点

Properties 类提供了读取属性文件的方法。在使用该类时需要先使用 load()方法加载属性文件。该方法的声明如下:

```
public void load(Reader reader) throws IOException
```

参数说明

reader: 文件输入流。

实例 149 向属性文件中添加信息

(实例位置: 配套资源\SL\12\149)

实例说明

属性文件使用键值对的形式保存数据, 键和值之间需要使用等号分割。本实例实现一个小工具, 通过在窗体中输入内容, 可实现向属性文件中写数据。实例的运行效果如图 12.21 所示。

实现过程

(1) 在 Eclipse 中创建项目 149, 并在该项目中创建 com.mingrisoft 包。



图 12.21 填写增加的属性文件值



(2) 在 com.mingrisoft 包中创建类文件, 名称为 SavePropertiesFrame。该类继承了 JFrame 并增加了标签、文本框和按钮控件。在输入完键和值后, 单击“写入”按钮, 可以将用户输入的内容写入到 D 盘的 message.properties 文件中。写入功能的代码位于 SaveProperties 类, 关键代码如下:

```
public void saveProperties(String key, String value) {
    Properties properties = new Properties();           //定义 Properties 对象
    properties.setProperty(key, value);                 //设置属性文件值
    try {
        FileOutputStream out = new FileOutputStream("D://message.properties");//创建输出流对象
        properties.store(out, "test");                 //将信息通过流写入到属性文件
        out.close();                                   //关闭流
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

技术要点

Properties 类的 setProperty() 方法可以将键和值增加到 Properties 对象中, 该方法的声明如下:

```
public Object setProperty(String key,String value)
```

参数说明

- ☒ key: 属性文件中的键。
- ☒ value: 属性文件中的值。

实例 150 在复制文件时使用进度条

(实例位置: 配套资源\SL\12\150)

实例说明

在图形界面操作系统中, 如果复制大量的文件会显示进度条提示用户正在复制中。本实例将实现类似的功能, 其运行效果如图 12.22 所示。

实现过程

(1) 在 Eclipse 中创建项目 150, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 UserMonitorFrame。该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择完文件和复制后的地址之后, 单击“确定复制”按钮可以完成复制。复制过程中显示的进度窗体是在 ProgressMonitorTest 类中实现的, 关键代码如下:

```
public void useProgressMonitor(JFrame frame, String copyPath, String newPath) {
    try {
        File file = new File(copyPath);           //根据要复制的文件创建 File 对象
        File newFile = new File(newPath);         //根据复制后文件的保存地址创建 File 对象
    }
}
```

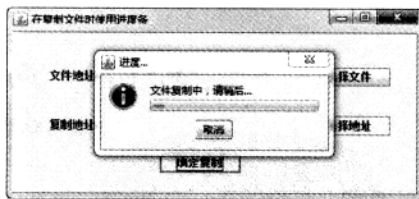


图 12.22 复制文件过程



```

FileOutputStream fop = new FileOutputStream(newFile); //创建 FileOutputStream 对象
InputStream in = new FileInputStream(file);
//读取文件, 如果总耗时超过 2 秒, 将会自动弹出一个进度监视窗口
ProgressMonitorInputStream pm = new ProgressMonitorInputStream(
    frame, "文件复制中, 请稍后...", in);
int c = 0;
byte[] bytes = new byte[1024]; //定义 byte 数组
while ((c = pm.read(bytes)) != -1) { //循环读取文件
    fop.write(bytes, 0, c); //通过流写数据
}
fop.close(); //关闭输出流
pm.close(); //关闭输入流
} catch (Exception ex) {
    ex.printStackTrace();
}
}

```



Note

技术要点

ProgressMonitorInputStream 类提供了自动地弹出进度窗口和事件机制。该类的构造方法如下:

```
public ProgressMonitorInputStream(Component parentComponent, Object message, InputStream in)
```

参数说明

- ☒ parentComponent: 触发被监视操作的组件。
- ☒ message: 要在对话框中放置的描述性文本。
- ☒ in: 要监视的输入流。

实例151 从 XML 文件中读取数据

(实例位置: 配套资源\SE\12\151)

实例说明

XML 文件是以节点的形式保存信息, 以树形分层结构排列。本实例将实现从 XML 文件中读取数据并在窗体中显示的功能, 其运行效果如图 12.23 所示。

实现过程

(1) 在 Eclipse 中创建项目 151, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 ReadXmlFrame。该类继承了 JFrame 并增加了标签和文本框控件。在程序启动中会从 src 文件夹读入 user.xml 文件。使用 ReadXMLDataBasc 类完成读入功能, 关键代码如下:

```

public String readXml(String passWord) {
    File xml_file = new File("src/users.xml");
}

```

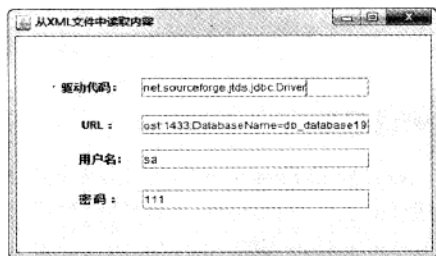


图12.23 读取 XML 文件中的信息



Note

```
//定义从 XMI 文档获取生成 DOM 对象的解析器
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    document = builder.parse(xml_file);           //根据 XML 获取 DOM 文档实例
} catch (Exception e) {
    e.printStackTrace();
}
String subNodeTag = document.getElementsByTagName(password).item(0)
    .getFirstChild().getNodeValue();           //获取指定节点保存的值
return subNodeTag;                             //返回读取的信息
}
```

技术要点

使用 DOM、SAX 技术都可以操作 XML 文件。但是都需要下载相关的文件，并将其添加到项目中。为了简便操作本实例使用 JDK 内置类来实现从 XML 文件中读取数据。实现本实例功能，涉及以下重要的类与方法。

☒ DocumentBuilderFactory 类

该类表示工厂 API，可以使应用程序能够从 XML 文档获取生成 DOM 对象树的解析器。

☒ DocumentBuilder 类

使用此类，可以从 XML 文件读取一个 Document 对象。

☒ Document 接口

该接口表示整个 HTML 或 XML 文档。从概念上讲，该接口表示文档树的根。

通过 Document 接口的 `getElementsByTagName()` 方法，从 XML 文档中读取具有指定标记名称的所有程序元素的有序集合 `NodeList` 对象。具体语法如下：

```
getElementsByTagName(String tagname)
```

参数说明

tagname: 要匹配的标记名称。对于 XML 文件，该参数值是区分大小写的。

实例 152 读取 Jar 文件属性

(实例位置: 配套资源\SL\12\152)

实例说明

在开发完 Java 项目后，可以将其打包成 Jar 文件直接运行。本实例完成了读取 Jar 文件内容的功能，其运行效果如图 12.24 所示。

实现过程

(1) 在 Eclipse 中创建项目 152，并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类



图 12.24 读取 JDK 中 `htmlconverter.jar` 文件信息



文件, 名称为 ReaderJarFrame。该类继承了 JFrame 并增加了标签、文本框、按钮和表格等控件。在选择完 Jar 文件后, 会在表格中显示其中的内容。在 ReadJar 类中完成了读取功能, 关键代码如下:

```
static List<FileName> process(String fileName) {
    List<FileName> list = new ArrayList<FileName>();           //创建 List 集合对象
    try {
        JarFile jarFile = new JarFile(fileName);              //创建 JarFile 对象
        Enumeration en = jarFile.entries();
        while (en.hasMoreElements()) {                         //测试枚举中是否包含更多的元素
            FileName file = new FileName();                    //定义 JavaBean 对象
            JarEntry entry = (JarEntry) en.nextElement();      //获取集合中的元素
            String name = entry.getName();                      //获取文件名称
            long size = entry.getSize();                        //获取文件大小
            file.setName(name);
            file.setSize(size + "");
            list.add(file);                                     //将对象添加到集合中
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return list;
}
```



Note

技术要点

本实例读取 Jar 文件主要应用了 JarFile 类与 Enumeration 接口。JarFile 是一个 Jar 文件自身的引用。Enumeration 对象可生成一系列的元素。

JarFile 类用于从任何可以使用 java.io.RandomAccessFile 打开的文件中读取 Jar 文件的内容。该类的常用构造方法如下。

- ☑ public JarFile(File file): 创建一个要从指定的 File 对象读取的新的 JarFile。
- ☑ JarFile(File file, boolean verify, int mode): 创建一个要从指定的 File 对象中以指定模式读取的新的 JarFile。

Enumeration 接口对象可生成一系列元素, 一次生成一个。该类有以下两个重要的方法。

- ☑ hasMoreElements(): 测试枚举是否包含更多的元素。
- ☑ nextElement(): 如果此枚举对象至少还有一个可提供的元素, 则返回此枚举的下一个元素。

实例 153 电子通讯录

(实例位置: 配套资源\SL\12\153)

实例说明

在日常生活中, 使用通讯录来保存姓名、电话、邮箱等信息。本实例将开发一个类似的程序, 其运行效果如图 12.25 所示。



Note

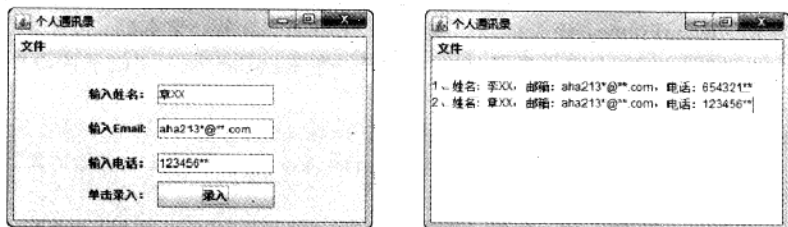


图 12.25 录入信息并显示其信息

实现过程

(1) 在 Eclipse 中创建项目 153，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 AddressList。该类继承了 JFrame 并增加了标签、文本框、按钮控件。在输入完信息之后，单击“录入”按钮会将其保存到 D 盘中，选择菜单栏中的“文件”/“显示”命令，会显示已经录入的信息。其中录入信息的相关代码如下：

```
private void kinbuttonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        if (nametextField.getText().equals("") || (emailtextField.getText().equals("")) || (phonetextField.
            getText().equals(""))) { //如果用户没有将信息输入完整
            JOptionPane.showMessageDialog(this, "请输入完整内容", "信息提示框", JOptionPane.
                WARNING_MESSAGE); //给出提示信息
            return; //退出程序
        }
        if (!file.exists()) //如果文件不存在
            file.createNewFile(); //新建文件
        //创建 BufferedWriter 对象
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream
            (file, true)));
        out.write("姓名: " + nametextField.getText() + ", "); //向文件中写内容
        out.write("邮箱: " + emailtextField.getText() + ", ");
        out.write("电话: " + phonetextField.getText());
        out.newLine(); //新建一行
        out.close(); //关闭流
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
```

技术要点

本实例实现向文件中写数据使用的是 `BufferedWriter` 类，该类将文本写入字符输出流，缓冲各个字符，从而提供单个字符、数组和字符串的高效写入。该类提供了 `newLine()` 方法，它可向文件中写入一个空白行。该类的构造方法介绍如下。

- ☑ `BufferedWriter(Writer out)`: 创建一个使用默认大小输出缓冲区的缓冲字符输出流。
- ☑ `BufferedWriter(Writer out, int sz)`: 创建一个使用给定大小输出缓冲区的新缓冲字符输出流。



在从文本中读取数据时,使用的是 `BufferedReader` 类,该类从字符输入流中读取文本,缓冲各个字符,从而实现字符、数组和行的高效读取。该类常用的构造方法介绍如下。

- ☑ `BufferedReader(Reader in)`: 创建一个使用默认大小输入缓冲区的缓冲字符输入流。
- ☑ `BufferedReader(Reader in, int sz)`: 创建一个使用指定大小输入缓冲区的缓冲字符输入流。



Note

实例 154 批量复制指定扩展名文件

(实例位置: 配套资源\SL\12\154)

实例说明

Windows 系统可以方便地完成批量复制,但是仅复制指定类型的文件就不那么方便了。本实例将实现这个功能,用户可以复制指定文件夹中的指定扩展名文件,其运行效果如图 12.26 所示。

实现过程

(1) 在 Eclipse 中创建项目 154,并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件,名称为 `CopyFileFrame`。该类继承了 `JFrame` 并增加了标签、文本框、按钮等控件。在输入文件地址、保存地址并选择文件类型后,单击“复制”按钮可以完成复制功能。复制功能是在 `CopyUtil` 类中实现的,关键代码如下:

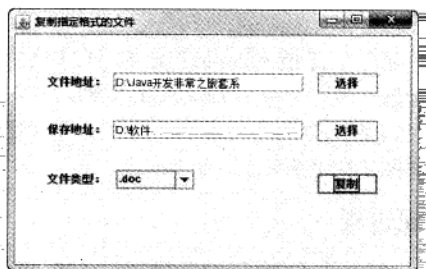


图 12.26 复制指定文件夹中的 Word 文件

```
public void copyFile(String oldPath, String newPath) {
    try {
        int bytesum = 0;
        int byteread = 0;
        File oldfile = new File(oldPath);
        if (oldfile.exists()) { //文件存在时
            InputStream inStream = new FileInputStream(oldPath); //读入源文件
            FileOutputStream fs = new FileOutputStream(newPath);
            byte[] buffer = new byte[1444];
            while ((byteread = inStream.read(buffer)) != -1) { //循环读取文件
                bytesum += byteread; //获取文件大小
                fs.write(buffer, 0, byteread); //向文件中写数据
            }
            inStream.close();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

技术要点

实现本实例需要遍历指定文件夹下的文件,之后将满足条件的文件复制到相应的地址下。



通过 File 类的 listFiles() 方法可以获取指定路径下的文件集合。该方法的语法如下：

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

该方法返回 File 数组。要获取数组中每个 File 文件的绝对路径，可以使用 File 类的 getAbsolutePath() 方法，该方法以 String 形式返回文件的绝对路径。具体代码如下：

```
String getAbsolutePath()
```



Note

实例 155 分类保存文件

(实例位置：配套资源\SL\12\155)

实例说明

随着信息技术的高速发展，计算机使用越来越频繁。为了方便管理其中的文件，本实例完成了根据文件类型进行分类保存的功能，其运行效果如图 12.27 所示。

实现过程

(1) 在 Eclipse 中创建项目 155，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 SortFrame。该类继承了 JFrame 并增加了标签、文本框和按钮控件。选择文件夹后，单击“确定分类”按钮，可以完成文件的分类保存。该按钮事件监听器的关键代码如下：

```
protected void do_sortButton_actionPerformed(ActionEvent arg0) {
    SortUtil sortUtil = new SortUtil();
    List list = sortUtil.getList(pathTextField.getText()); //获取用户选择文件夹中所有文件集合
    for (int i = 0; i < list.size(); i++) {                //循环遍历该文件集合
        String strFile = list.get(i).toString();
        int index = strFile.lastIndexOf(".");
        if (index != -1) {
            String strN = strFile.substring(index + 1, strFile.length()); //对文件夹进行截取，获
            取文件扩展名

            int ind = strFile.lastIndexOf("\\");
            String strFileName = strFile.substring(ind, index);
            sortUtil.createFolder(pathTextField.getText() + "\\\" + "分类"); //调用创建文件夹方法，
            新建文件夹

            sortUtil.createFolder(pathTextField.getText() + "\\\" + "分类" + "\\\" + strN);
            if (strFile.endsWith(strN)) {
                //将文件集中与文件夹名称相同的文件复制到相应的文件夹中
                sortUtil.copyFile(strFile,
                    pathTextField.getText() + "\\\" + "分类" + "\\\" + strN + "\\\" +
                    strFileName + strFile.substring(index, strFile.length()));
            }
        }
    }
    JOptionPane.showMessageDialog(getContentPane(), //给出用户分类完成提示框
```

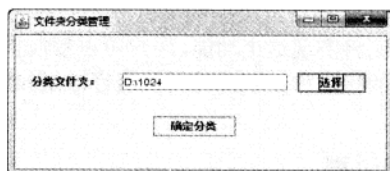


图 12.27 选择文件夹后的效果



```
"文件分类成功!", "信息提示框", JOptionPane.WARNING_MESSAGE);
```

技术要点

本实例实现文件分类存储的关键是获取某文件夹下的文件, 并通过字符串截取的方式提取文件的格式, 且根据获取的文件格式创建文件夹。提取文件格式使用的是 String 类的 substring() 方法。该方法可在指定的字符串中截取子字符串。语法格式如下:

```
public String substring(int beginIndex,int endIndex)
```

参数说明

- ☑ beginIndex: 要截取字符串的开始索引位置, 包括该索引位置处的字符。
- ☑ endIndex: 要截取字符串的借宿索引, 不包括该索引位置处的字符。



Note

实例 156 搜索指定文件夹中的文件

(实例位置: 配套资源\SL\12\156)

实例说明

Windows 操作系统提供了文件搜索功能。本实例将完成类似功能, 同时也支持模糊搜索, 使用星号“*”表示任意多个字符, 使用“?”表示任意一个字符。实例的运行效果如图 12.28 所示。

实现过程

(1) 在 Eclipse 中创建项目 156, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 SearchFrame。该类继承了 JFrame 并增加了标签、文本框、按钮等控件。选择文件夹后, 单击“搜索”按钮, 可以完成文件的搜索任务, 并在文本区中显示搜索结果。搜索任务是在工具类 FileSearch 中完成的, 关键代码如下:

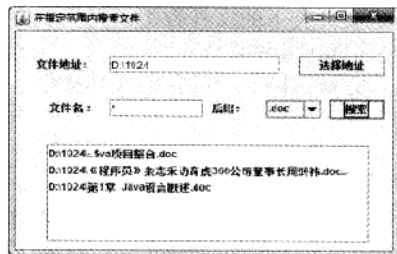


图 12.28 搜索指定文件夹中全部.doc 文件

```
public static boolean findName(String pattern, String str) {
    int patternLength = pattern.length();           //获取参数字符串的长度
    int strLength = str.length();
    int strIndex = 0;
    char eachCh;
    for (int i = 0; i < patternLength; i++) {         //循环字符串参数字符串中的每个字符
        eachCh = pattern.charAt(i);                  //获取字符串中每个索引位置的字符
        if (eachCh == "*") {                          //如果这个字符是一个星号
            while (strIndex < strLength) {
                if (findName(pattern.substring(i + 1), str
                    .substring(strIndex))) {           //如果文件名与搜索模型匹配
                    return true;
                }
            }
        }
    }
}
```



Note

的文件

```

        strIndex++;
    }
    } else if (eachCh == "?") { //如果包含问号
        strIndex++;
        if (strIndex > strLength) { //如果 str 中没有字符可以匹配 “?” 号
            return false;
        }
    } else { //如果要寻找的是普通的文件
        if ((strIndex >= strLength) || (eachCh != str.charAt(strIndex))) { //如果没有查找到匹配
            return false;
        }
        strIndex++;
    }
}
return (strIndex == strLength);
}

```

技术要点

本实例的实现首先要获取指定文件夹下的文件数组。再从数组中查询满足条件的文件。获取指定文件夹下的文件数组，可用 File 类的 listFiles() 方法，具体语法如下：

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

该方法返回一个抽象路径名数组，表示此抽象路径名表示的文件夹中的文件。

实例 157 实现文件锁定功能

(实例位置：配套资源\SL\12\157)

实例说明

在操作文件时，可能会遇到这样的问题。当打开一个文件时遇到“该文件已经被另一个程序占用，打开失败”的问题。这是因为另一个程序正在编辑该文件，在这个过程中，不允许其他程序修改这个程序。这就是文件的锁定。本实例将实现通过 Java 程序将 D 盘的 count.txt 文件锁定 1 分钟，当对该文件进行编辑保存时，会出现如图 12.29 所示的结果。

实现过程

- (1) 在 Eclipse 中创建项目 157，并在该项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建类文件，名称为 EncryptInput。在该类中定义 fileLock() 方法完成文件的锁定。在 main() 方法中运行 fileLock() 方法。关键代码如下：

```

public class EncryptInput {
    @SuppressWarnings("unused")
    public static void fileLock(String file) {

```

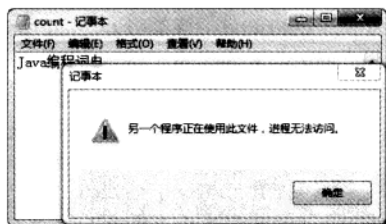


图 12.29 编辑处于锁定状态的文件



```

FileOutputStream fous = null;           //创建 FileOutputStream 对象
FileLock lock = null;                   //创建 FileLock 对象
try {
    fous = new FileOutputStream(file);    //实例化 FileOutputStream 对象
    lock = fous.getChannel().tryLock();  //获取文件锁定
    Thread.sleep(60 * 1000);             //线程锁定 1 分钟
} catch (Exception e) {
    e.printStackTrace();
}
}

public static void main(String[] args) {
    String file = "D://count.txt";        //创建文件对象
    fileLock(file);                       //调用文件锁定方法
}
}

```



Note

技术要点

本实例通过使用 FileChannel 类的 tryLock() 方法获得文件锁。该方法的声明如下：

```
public final FileLock tryLock() throws IOException
```

返回值：新获得的文件锁，如果获取失败则返回 null。

实例 158 简单的投票软件

（实例位置：配套资源\SL\12\158）

实例说明

使用 IO 技术能够将信息保存到本地文件中，这样就可以持久保存。本实例将完成一个简单的投票软件，其运行效果如图 12.30 所示。

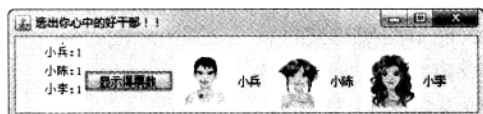


图 12.30 显示投票结果

实现过程

（1）在 Eclipse 中创建项目 158，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建类文件，名称为 Ballot。在该文件中定义 MyMin 类完成窗体上控件的增加。编写 Candidate 类完成票数的保存和获得。其中获得投票的关键代码如下：

```

public int getBallot(String name) {
    File file = new File("D://count.txt");           //创建文件对象
    FileReader fis;
    try {
        if (!file.exists())                          //如果该文件不存在
            file.createNewFile();                     //新建文件
        fis = new FileReader(file);
    }
}

```




Note

```
BufferedReader bis = new BufferedReader(fis);           //创建 BufferedReader 对象
String str[] = new String[3];
String size;
int i = 0;
while ((size = bis.readLine()) != null) {               //循环读取文件内容
    str[i] = size.trim();                               //去除字符串中的空格
    if (str[i].startsWith(name)) {
        int length = str[i].indexOf(".");
        String sub = str[i].substring(length + 1, str[i].length()); //对字符串进行截取
        len = Integer.parseInt(sub);
        continue;
    }
    i++;
}
} catch (Exception e) {
    e.printStackTrace();
}
return len;
}
```

技术要点

本实例通过 `BufferedReader` 从文件中读取上次投票的结果，当用户单击按钮后将增加投票并写入到文件中，这样就可以及时保存数据。

实例 159 压缩所有文本文件

(实例位置：配套资源\SL\12\159)

实例说明

使用文件压缩技术不仅能节约磁盘空间，还便于管理。本实例将完成指定文件夹中文本文件的压缩功能，其运行效果如图 12.31 所示。

实现过程

(1) 在 Eclipse 中创建项目 159，并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件，名称为 `ZipTextFileFrame`。该类继承了 `JFrame` 并增加了标签、文本框、按钮等控件。选择文件夹后，将在表格中显示文件夹中的文本文件。单击“开始压缩”按钮，可以完成对它们的压缩。实现压缩功能的 `zipFile` 方法的代码如下：

```
private static void zipFile(File[] files, File targetZipFile) throws IOException {
    //利用给定的 targetZipFile 对象创建文件输出流对象
    FileOutputStream fos = new FileOutputStream(targetZipFile);
}
```

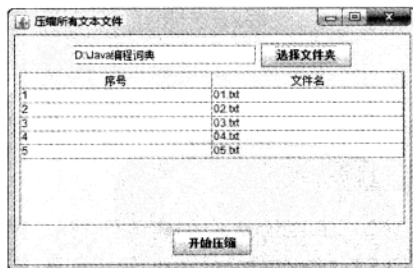


图 12.31 选择需要压缩的文件



Note

```

ZipOutputStream zos = new ZipOutputStream(fos); //利用文件输出流创建压缩输出流
byte[] buffer = new byte[1024]; //创建写入压缩文件的数组
for (File file : files) { //遍历全部文件
    ZipEntry entry = new ZipEntry(file.getName()); //利用每个文件的名字创建 ZipEntry 对象
    FileInputStream fis = new FileInputStream(file); //利用每个文件创建文件输入流对象
    zos.putNextEntry(entry); //在压缩文件中添加一个 ZipEntry 对象
    int read = 0;
    while ((read = fis.read(buffer)) != -1) {
        zos.write(buffer, 0, read); //将输入写入到压缩文件
    }
    zos.closeEntry(); //关闭 ZipEntry
    fis.close(); //释放资源
}
zos.close();
fos.close();
}

```

技术要点

压缩文件和复制文件类似，只是用另外一种格式来保存输入流。本实例使用到了 ZipOutputStream，它以 ZIP 文件格式写入文件实现输出流过滤器。每一个文件在压缩过程中都被存到 ZipEntry 中。使用 putNextEntry() 方法增加 ZipEntry，该方法的声明如下：

```
public void putNextEntry(ZipEntry e) throws IOException
```

参数说明

e: 压缩的文件实体。

实例 160 将压缩包解压到指定文件夹

(实例位置：配套资源\SE\12\160)

实例说明

在获得一个以 ZIP 格式压缩的文件之后，需要将其进行解压缩，还原成压缩前的文件。本实例使用 Java 自带的压缩工具包来实现解压缩文件到指定文件夹的功能。实例的运行效果如图 12.32 所示。



图 12.32 完成解压缩



实现过程

(1) 在 Eclipse 中创建项目 160，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 UnZipTextFileFrame。该类继承了 JFrame 并增加了标签、文本框、按钮等控件。选择 ZIP 文件和解压缩位置后，单击“开始解压缩”按钮完成解压缩操作。同时在表格中显示解压缩的文件。该按钮事件监听器的代码如下：

```
protected void do_unzipButton_actionPerformed(ActionEvent arg0) {
    DefaultTableModel model = (DefaultTableModel) table.getModel(); //获得表格模型
    model.setColumnIdentifiers(new Object[] { "序号", "文件名" }); //设置表头
    int id = 1; //声明序号变量
    ZipFile zf = null;
    try {
        zf = new ZipFile(zipFile); //利用用户选择的 ZIP 文件创建 ZipFile 对象
        Enumeration e = zf.entries(); //创建枚举变量
        while (e.hasMoreElements()) { //遍历枚举变量
            ZipEntry entry = (ZipEntry) e.nextElement(); //获得 ZipEntry 对象
            if (!entry.getName().endsWith(".txt")) { //如果不是文本文件就不进行解压缩
                continue;
            }
            //利用用户选择的文件夹和 ZipEntry 对象名称创建解压后的文件
            File currentFile = new File(targetFile + File.separator + entry.getName());
            FileOutputStream out = new FileOutputStream(currentFile);
            InputStream in = zf.getInputStream(entry); //利用获得的 ZipEntry 对象的输入流
            int buffer = 0;
            while ((buffer = in.read()) != -1) { //将输入流写入到本地文件
                out.write(buffer);
            }
            model.addRow(new Object[] { id++, currentFile.getName() }); //增加一行表格数据
            in.close(); //释放资源
            out.close();
        }
        table.setModel(model); //更新表格
        JOptionPane.showMessageDialog(this, "解压缩完成"); //提示用户解压缩完成
    } catch (ZipException e) { //捕获异常
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        //省略释放资源代码
    }
}
```

技术要点

ZipFile 是用来从 ZIP 文件中读取 ZipEntry 的类。使用该类的 getInputStream() 方法可以从 ZipEntry 中读取压缩的数据，该方法的声明如下：

```
public InputStream getInputStream(ZipEntry entry) throws IOException
```

参数说明

entry: 压缩的 Zip 实体。



实例 161 压缩所有子文件夹

(实例位置: 配套资源\SL\12\161)

实例说明

在压缩文件时, 通常情况下一个文件夹都会有若干个子文件夹。此时该怎样处理呢? 本实例将展示如何压缩包含子文件夹的文件夹。实例的运行效果如图 12.33 所示。

实现过程

(1) 在 Eclipse 中创建项目 161, 并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件, 名称为 ZipDirectoryFrame。该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择需要压缩的文件夹之后, 单击“开始压缩”按钮完成压缩任务。压缩任务的代码位于 zipFile 方法中, 关键代码如下:

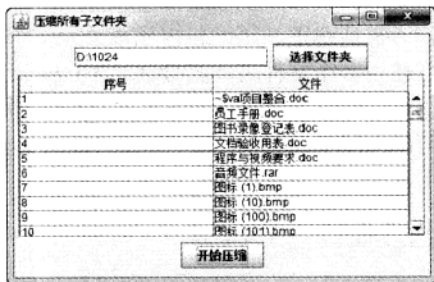


图 12.33 压缩完用户选择的文件夹

```
private void zipFile(List<String> path, File targetZipFile, String base) throws IOException {
    //根据给定的 targetZipFile 创建文件输出流对象
    FileOutputStream fos = new FileOutputStream(targetZipFile);
    ZipOutputStream zos = new ZipOutputStream(fos); //利用文件输出流对象创建 Zip 输出流对象
    byte[] buffer = new byte[1024];
    for (String string : path) { //遍历所有要压缩文件的路径
        File currentFile = new File(string);
        //利用要压缩文件的相对路径创建 ZipEntry 对象
        ZipEntry entry = new ZipEntry(string.substring(base.length() + 1, string.length()));
        FileInputStream fis = new FileInputStream(currentFile);
        zos.putNextEntry(entry);
        int read = 0;
        while ((read = fis.read(buffer)) != -1) { //将数据写入到 Zip 输出流中
            zos.write(buffer, 0, read);
        }
        zos.closeEntry(); //关闭 ZipEntry 对象
        fis.close();
    }
    zos.close(); //释放资源
    fos.close();
}
```

技术要点

压缩包含子文件夹的文件夹的方法和压缩全是文件的文件夹类似, 区别在于如何找出包含子文件夹的文件夹的所有文件, 并且构造 ZipEntry 时不会出现重名现象。本实例采用获得要压



缩文件夹中所有文件的相对路径来创新地解决这个问题。

实例 162 深层文件夹压缩包的释放

(实例位置: 配套资源\SL\12\162)



Note

实例说明

通常情况下,压缩包内应该会有多个文件,并且使用多个文件夹将其分类。为了从压缩包中获得这些文件,需要将其解压缩。本实例将演示如何解压缩复杂的压缩文件,并且还还原出文件夹的层次关系。实例的运行效果如图 12.34 所示。

实现过程

(1) 在 Eclipse 中创建项目 162,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 UnZipDirectoryFrame。该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择需要解压缩的文件之后,单击“开始解压缩”按钮完成解压缩任务。这里主要用到了自定义的 unzip() 方法。关键代码如下:



图 12.34 解压缩用户选择的文件

```
private static void unzip(File zipFile, List<String> list) throws IOException {  
    //利用用户选择的 ZIP 文件创建 ZipInputStream 对象  
    ZipInputStream in = new ZipInputStream(new FileInputStream(zipFile));  
    ZipEntry entry;  
    while ((entry = in.getNextEntry()) != null) {  
        //遍历所有 ZipEntry 对象  
        if (!entry.isDirectory()) {  
            //如果是文件则创建并写入  
            File tempFile = new File(zipFile.getParent() + File.separator + entry.getName());  
            list.add(tempFile.getName());  
            new File(tempFile.getParent()).mkdirs();  
            tempFile.createNewFile();  
            //增加文件名  
            //创建文件夹  
            //创建新文件  
            FileOutputStream out = new FileOutputStream(tempFile);  
            int b;  
            while ((b = in.read()) != -1) {  
                //写入数据  
                out.write(b);  
            }  
            out.close();  
            //释放资源  
        }  
    }  
    in.close();  
}
```

技术要点

解压缩包含子文件夹的文件夹的想法和解压缩全是文件的文件夹类似,区别在于如何找出包含子文件夹的文件夹的所有文件,并且构造 ZipEntry 时不会出现重名现象。



实例 163 解决压缩包中文乱码

(实例位置: 配套资源\SL\12\163)

实例说明

在使用 Java 自带的 ZIP 工具类时,会出现中文乱码的问题。为了完善工具箱,本实例使用 Apache 的 Ant 包来解决压缩包中文乱码的问题。实例的运行效果如图 12.35 所示。对比压缩包中的乱码效果如图 12.36 所示。



图 12.35 压缩用户选择的文件夹

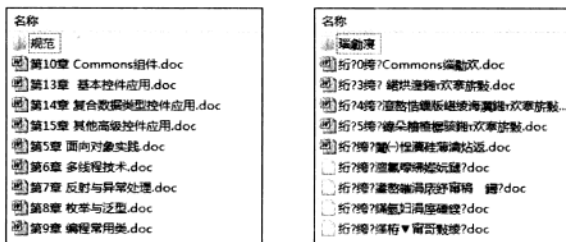


图 12.36 没有乱码的压缩包(左侧)和有乱码的压缩包(右侧)

实现过程

(1) 在 Eclipse 中创建项目 163,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 ZipByAntFrame。该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择完需要压缩的文件夹后,单击“开始压缩”按钮完成压缩。zipFile 方法实现了压缩功能,关键代码如下:

```
private void zipFile(List<String> path, File targetZipFile, String base) throws IOException {
    //根据给定的 targetZipFile 创建文件输出流对象
    FileOutputStream fos = new FileOutputStream(targetZipFile);
    ZipOutputStream zos = new ZipOutputStream(fos); //利用文件输出流对象创建 Zip 输出流对象
    byte[] buffer = new byte[1024];
    for (String string : path) {
        //遍历所有要压缩文件的路径
        File currentFile = new File(string);
        //利用要压缩文件的相对路径创建 ZipEntry 对象
        ZipEntry entry = new ZipEntry(string.substring(base.length() + 1, string.length()));
        FileInputStream fis = new FileInputStream(currentFile);
```





```

zos.putNextEntry(entry);
int read = 0;
while ((read = fis.read(buffer)) != -1) {           //将数据写入到 Zip 输出流中
    zos.write(buffer, 0, read);
}
zos.closeEntry();                                  //关闭 ZipEntry 对象
fis.close();
}
zos.close();                                       //释放资源
fos.close();
}

```

技术要点

Apache 的 Ant 包提供了对压缩文件功能的支持。它的 `org.apache.tools.zip.ZipOutputStream` 实现了 `java.util.ZipFile` 的功能，并且还对它进行了扩展。

实例 164 Apache 实现文件解压缩

(实例位置：配套资源\SL\12\164)

实例说明

在获得一个以 ZIP 格式压缩的文件之后，需要将其进行解压缩，还原成压缩前的文件。然而当被压缩的文件名中有中文时，Java 自带的压缩工具类会出现 `java.lang.IllegalArgumentException` 异常。因此本实例使用 ANT 实现文件解压缩功能。实例的运行效果如图 12.37 所示。

实现过程

(1) 在 Eclipse 中创建项目 164，并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件，名称为 `UnZipByAntFrame`。该类继承了 `JFrame` 并增加了标签、文本框、按钮等控件。在选择完压缩文件后，单击“开始解压缩”按钮，完成解压缩任务。自定义的 `unzip` 完成了核心任务，关键代码如下：

```

private void unzip(File zipFile, File targetFile, List<String> list) throws IOException {
    String zipFilePath = zipFile.getAbsolutePath();
    int lastDot = zipFilePath.lastIndexOf(".");
    String parentPath = zipFilePath.substring(0, lastDot);           //去掉 ZIP 文件的后缀名
    ZipFile zf = new ZipFile(zipFile);
    Enumeration<ZipEntry> e = zf.getEntries();                       //获得所有 ZipEntry 对象
    while (e.hasMoreElements()) {
        ZipEntry entry = e.nextElement();
        if (!entry.isDirectory()) {
            File newFile = new File(parentPath + File.separator + entry.getName());
            list.add(newFile.getName());
        }
    }
}

```

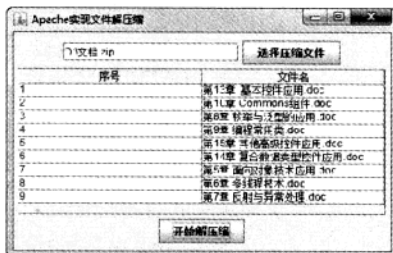


图 12.37 解压缩用户选择的 ZIP 文件



Note

```

new File(newFile.getParent()).mkdirs();
newFile.createNewFile();
FileOutputStream out = new FileOutputStream(newFile);
InputStream in = zf.getInputStream(entry);
int b;
while ((b = in.read()) != -1) {           //写入数据
    out.write(b);
}
out.close();                             //释放资源
}
}

```

技术要点

Apache 的 Ant 包提供了对压缩文件功能的支持。它的 `org.apache.tools.zip.ZipFile` 类可以作为 `java.util.ZipFile` 的替代品。除了 UTF-8, 这个类对于文件名编码方式还提供了其他支持, 这样就可以避免中文乱码的问题。

实例 165 把窗体压缩成 ZIP 文件

(实例位置: 配套资源\SL\12\165)

实例说明

Java 中使用 `new` 运算符创建的对象是存储在内存中的, 当虚拟机关闭或重启时这个对象就消失了。如果我们想在以后还能使用这个对象该怎么办呢? 使用 Java 的序列化功能就能实现对象的持久化存储。本实例演示了如何将一个窗体序列化并压缩成为一个 ZIP 文件。实例的运行效果如图 12.38 所示。

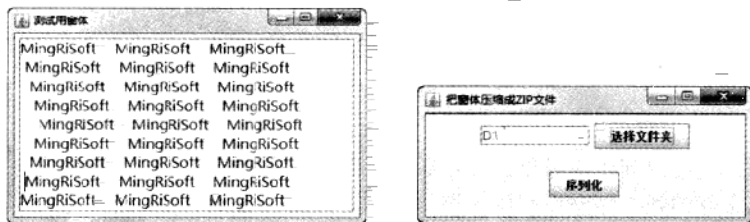


图 12.38 测试用窗体如左图, 主窗体如右图

实现过程

(1) 在 Eclipse 中创建项目 165, 并在该项目中创建 `com.mingrisoft` 包。

(2) 在 `com.mingrisoft` 包中创建类文件, 名称为 `TestFrame`。该类继承了 `JFrame` 并增加了一个文本区。创建类文件, 名称为 `SerializationFrame`, 并增加文本框和按钮控件。选择完序列化文件保存位置后, 单击“序列化”按钮将完成窗体序列化功能。`zipSerializationObject()`方法完成了核心功能, 关键代码如下:

```

private void zipSerializationObject(Object object, File path) throws IOException {
    File serializeFile = new File(path + "serialization.dat"); //根据用户选择的路径创建文件
}

```



Note

```
FileOutputStream fos = new FileOutputStream(serializableFile);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(object); //将对象写入到创建的 DAT 文件
oos.close(); //释放资源
fos.close();
File zipFile = new File(path + "serialization.zip"); //创建压缩文件
fos = new FileOutputStream(zipFile);
ZipOutputStream zos = new ZipOutputStream(fos);
byte[] buffer = new byte[1024];
ZipEntry entry = new ZipEntry(serializableFile.getName());
FileInputStream fis = new FileInputStream(serializableFile);
zos.putNextEntry(entry);
int read = 0;
while ((read = fis.read(buffer)) != -1) {
    zos.write(buffer, 0, read); //写入压缩文件
}
zos.closeEntry();
fis.close(); //释放资源
zos.close();
fos.close();
serializableFile.delete(); //删除创建的 DAT 文件
}
```

技术要点

ObjectOutputStream 将 Java 对象的基本数据类型和图形写入 OutputStream，可以使用 ObjectInputStream 读取（重构）对象。通过在流中使用文件可以实现对象的持久存储。如果流是网络套接字流，则可以在另一台主机上或另一个进程中重构对象。

实例 166 解压缩 Java 对象

（实例位置：配套资源\SL\12\166）

实例说明

对于一个已经被序列化的对象，如果要将其还原该怎么办呢？本实例在实例 165 的基础上实现对序列化文件的解压缩操作和反序列化操作。实例的运行效果如图 12.39 所示。



图 12.39 主窗体如左图，反序列化生成窗体如右图

实现过程

（1）在 Eclipse 中创建项目 166，并在该项目中创建 com.mingrisoft 包。



(2) 在 com.mingrisoft 包中创建类文件, 名称为 TestFrame。该类继承了 JFrame 并增加了一个文本区。创建类文件, 名称为 UnSerializationFrame, 并增加文本框和按钮控件。选择完序列化文件并保存位置后, 单击“反序列化”按钮将完成窗体反序列化功能。unzipSerializationObject() 方法完成了核心功能, 关键代码如下:

```
private void unzipSerializationObject(File file) throws IOException, ClassNotFoundException {
    ZipFile zipFile = new ZipFile(file);           //创建 ZipFile 对象
    File currentFile = null;
    Enumeration e = zipFile.entries();
    while (e.hasMoreElements()) {
        ZipEntry entry = (ZipEntry) e.nextElement();
        if (!entry.getName().endsWith(".dat")) {    //遇到后缀名是.dat 的文件就进行解压缩
            continue;
        }
        currentFile = new File(file.getParent() + entry.getName());
        FileOutputStream out = new FileOutputStream(currentFile);
        InputStream in = zipFile.getInputStream(entry);
        int buffer = 0;
        while ((buffer = in.read()) != -1) {         //写入文件
            out.write(buffer);
        }
        in.close();                                 //释放资源
        out.close();
    }
    FileInputStream in = new FileInputStream(currentFile);
    ObjectInputStream ois = new ObjectInputStream(in); //读入解压缩后的文件
    TestFrame frame = (TestFrame) ois.readObject();  //还原被序列化的对象
    frame.setVisible(true);                          //显示被序列化的对象
    currentFile.delete();                             //删除解压缩产生的文件
}
```



Note

技术要点

当使用序列化来保存对象后, 如果需要再次将序列化文件还原成原来的对象, 就要进行反序列化。反序列化就是打开字节流并且重构对象, 此时新的对象和序列化时的对象是一样的。如果读者在前面的实例中改变了对象的状态, 那么在本实例中反序列化生成的对象也保存了前面的操作。

实例 167 文件压缩为 RAR 文档

(实例位置: 配套资源\SL\12\167)

实例说明

文件压缩是对数据的一种紧凑存储格式, 通过压缩能够使文件更小, 占用更少的磁盘空间, 同时也可以减少网络传输的时间。本实例将实现文件到 RAR 文档的压缩, 实例的运行效果如图 12.40 所示。



图 12.40 解压缩用户选择的 ZIP 文件

实现过程

(1) 在 Eclipse 中创建项目 167，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 CompressTxtToRAR。该类继承了 JFrame 并增加了标签、文本框、按钮、表格、进度条等控件。单击“增加”按钮增加要压缩的文件。单击“浏览”按钮选择用于保存的 WinRAR 文件。单击“压缩”按钮开始压缩。压缩功能在 CompressThread 内部类实现，关键代码如下：

```
private final class CompressThread extends Thread {
    public void run() {
        try {
            //获取表格控件的数据模型
            DefaultTableModel model = (DefaultTableModel) table.getModel();
            int rowCount = model.getRowCount(); //获取数据模型中表格行数
            StringBuilder fileList = new StringBuilder();
            for (int i = 0; i < rowCount; i++) { //遍历数据表格模型中的文件对象
                File file = (File) model.getValueAt(i, 2);
                fileList.append(file.getPath() + "\n"); //把文件路径存到字符串构建器中
            }
            //创建临时文件，用于保存压缩文件列表
            File listFile = File.createTempFile("fileList", ".tmp");
            FileOutputStream fout = new FileOutputStream(listFile);
            fout.write(fileList.toString().getBytes()); //保存字符串构建器数据到临时文件
            fout.close();
            //创建压缩命令字符串
            final String command = "rar a " + rarFile.getPath() + " @" + listFile.getPath();
            Runtime runtime = Runtime.getRuntime(); //获取 Runtime 对象
            progress = runtime.exec(command.toString() + "\n"); //执行压缩命令
            progress.getOutputStream().close(); //关闭进程输出流
            progressBar.setString(null); //初始化进度条控件
            progressBar.setValue(0);
            Scanner scan = new Scanner(progress.getInputStream()); //获取进程输入流
            while (scan.hasNext()) {
                String line = scan.nextLine(); //获取进程提示单行信息
                int index = line.lastIndexOf("%") - 3; //获取提示信息的进度百分比的索引位置
                if (index <= 0)
                    continue;
                String substring = line.substring(index, index + 3); //获取进度百分比字符串
                int percent = Integer.parseInt(substring.trim()); //获取整数的百分比数值
                progressBar.setValue(percent); //在进度条控件显示百分比
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```




```

    }
    progressBar.setString("完成");
    scan.close();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```



Note

脚下留神:

在运行该实例前,需要将 WinRAR 安装目录中的 Rar.exe 命令所在位置增加到环境变量中。

技术要点

Runtime 类是每个 Java 程序都内置的一个运行时对象。通过这个对象可以执行外部命令,这样就可以执行 RAR 的压缩、解压缩添加注释等各种命令。但是这个类不能直接创建对象,需要使用静态方法来获取实例对象并且调用对象的方法来执行外部命令。

实例 168 解压缩 RAR 压缩包

(实例位置: 配套资源\SL\12\168)

实例说明

文件解压缩是最常用的数据操作,目前大多数资料 and 软件都采用 RAR 格式进行压缩并在网站上提供下载,经过压缩的资源体积更小,在网络中的传输速度更快。用户从网站下载该文件之后,需要使用 RAR 软件进行解压缩才能获取自己想要的资源。本实例将实现对 RAR 压缩包的解压缩功能,可以针对指定的压缩包文件定制解压的目标文件夹。实例的运行效果如图 12.41 所示。

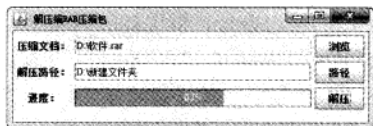


图 12.41 解压缩用户选择的 RAR 文件

实现过程

(1) 在 Eclipse 中创建项目 168,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 DeCompressRAR。该类继承了 JFrame 并增加了标签、文本框、按钮和进度条控件。选择了压缩文档和解压路径后,单击“解压”按钮完成解压缩。在进度条中显示解压缩的进度。解压缩是在 DeCompressThread 类中完成的,关键代码如下:

```

private final class DeCompressThread extends Thread {
    private final String command;
    private DeCompressThread(String command) {
        this.command = command;
    }
    public void run() {
        try {

```




Note

```
final Process process = Runtime.getRuntime().exec(command);
process.getOutputStream().close();
final Scanner scan = new Scanner(process.getInputStream());
progressBar.setString(null); //初始化进度条控件
progressBar.setValue(0);
while (scan.hasNext()) {
    String line = scan.nextLine(); //获取进程提示单行信息
    int index = line.lastIndexOf("%") - 3; //获取提示信息的进度百分比的索引位置
    if (index <= 0)
        continue;
    String substring = line.substring(index, index + 3); //获取进度百分比字符串
    int percent = Integer.parseInt(substring.trim()); //获取整数的百分比数值
    progressBar.setValue(percent + 1); //在进度条控件显示百分比
}
progressBar.setString("完成");
process.getInputStream().close();
} catch (IOException e1) {
    e1.printStackTrace();
}
}
```

脚下留神:

在运行该实例前,需要将 WinRAR 安装目录中的 Rar.exe 命令所在位置增加到环境变量中。

技术要点

解压缩动作根据压缩包的大小、包含的文件数量来决定,如果压缩包内容较多、体积较大,解压缩就会变成一个非常耗时的操作,这样的业务是不允许在 GUI 线程中处理的,所以应该采用新的线程来接收文件的解压缩操作,从而把 GUI 线程解放,避免程序界面死锁。

实例 169 为 RAR 压缩包添加注释

(实例位置: 配套资源\SL\12\169)

实例说明

通过压缩文件的名称,可以简单地了解其用途。此外,还可以为其增加注释以便做更详尽的说明。本实例将使用户可以通过图形界面完成为 RAR 文件添加注释的功能,其运行效果如图 12.42 所示。

实现过程

(1) 在 Eclipse 中创建项目 169,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名

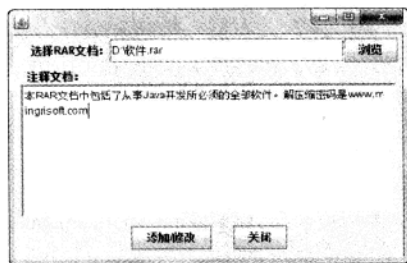


图 12.42 为指定的文档增加注释



称为 RarAnnotate。该类继承了 JFrame 并增加了标签、文本框、按钮和文本区控件。在选择了 RAR 文档后,可以在文本区中写入需要添加的注释信息,单击“添加修改”按钮完成写入注释的功能,该按钮事件监听器的关键代码如下:

```
protected void do_annotateButton_actionPerformed(ActionEvent e) {
    String annotateStr = annotateArea.getText();           //获取注释文本
    int length = annotateStr.getBytes().length;           //获取注释文本长度
    if (length > 32767) {                                  //限制文本长度
        JOptionPane.showMessageDialog(null, "注释长度不能大于 32767");
        return;
    }
    try {
        Process process = getRuntime().exec(               //执行添加注释命令
            "rar c \"" + rarFile + "\"");
        //把注释文本传递给注释命令
        process.getOutputStream().write(annotateStr.getBytes());
        process.getOutputStream().close();                //关闭输出流
        process.getInputStream().close();                   //关闭输入流
    } catch (IOException e1) {
        e1.printStackTrace();
    }
}
```



Note

脚下留神:

在运行该实例前,需要将 WinRAR 安装目录中的 Rar.exe 命令所在位置增加到环境变量中。

技术要点

RAR 命令列表中包含一个“c”命令,该命令可以为 RAR 压缩文档添加长度小于 32767 的注释文本。当压缩文件被处理时注释被显示。

实例 170 获取压缩包详细文件列表

(实例位置: 配套资源\SL\12\170)

实例说明

使用 RAR 压缩文件之后,并不能直接在文件资源管理器中浏览压缩包的内容,这给资源查找带来了不便。本实例实现将 RAR 文档列表解析并将其显示在一个表格中,其运行效果如图 12.43 所示。

实现过程

(1) 在 Eclipse 中创建项目 170,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,

文件名称	大小	压缩后大小	压缩率	时间
个人集速通	51712	8553	16%	15-05-11
Java电子书\	10749208	3984550	37%	14-03-11
Java电子书\	6173317	2088459	33%	11-04-11
Java电子书\	1526426	1504908	98%	01-04-11
Java电子书\	187842735	129405700	68%	15-03-11
Java电子书\	29019161	15288191	52%	19-05-11
Java电子书\	34687495	21636642	62%	29-03-11
Java电子书\	108521666	102687163	96%	18-05-11

图 12.43 查看“软件.rar”中的文件



名称为 FileCompressList。该类继承了 JFrame 并增加了标签、文本框、按钮和表格控件。在选择了 RAR 文档后,将在表格中显示 RAR 文档中文件的信息。关键代码如下:

```
protected void do_browseButton_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser(); //创建文件选择器
    chooser.setFileFilter(new FileNameExtensionFilter("RAR 文档", "rar"));
    chooser.setAcceptAllFileFilterUsed(false);
    int option = chooser.showOpenDialog(this); //显示文件打开对话框
    if (option != JFileChooser.APPROVE_OPTION)
        return;
    rarFile = chooser.getSelectedFile(); //获取选择的 rar 文件
    rarTextField.setText(rarFile.toString()); //显示 rar 文件到文本框
    try {
        //执行提取注释命令,把注释信息保存在临时文件中
        Process process = getRuntime().exec("rar v -c- \"\" + rarFile + \"\"");
        process.getOutputStream().close(); //关闭进程输出流
        Scanner sc = new Scanner(process.getInputStream());
        int count = 0; //创建行索引
        DefaultTableModel model = (DefaultTableModel) table.getModel(); //获取表格控件模型
        Vector<String> row = new Vector<String>(); //创建行数据向量
        do {
            String line = sc.nextLine(); //获取文件列表信息的一行
            if (line.contains("-----")) { //标记起始结束索引
                count = (count == 0 ? count + 1 : -1);
                continue;
            }
            if (count == 0) //跳过起始标记
                continue;
            if (count == -1) //在结束标记终止循环
                break;
            if (++count % 2 == 0) { //获取文件名称
                row.add(line);
            } else { //获取文件详细信息
                String[] split = line.trim().split("\\s+"); //把文件详细信息分割为数组
                for (String string : split) { //遍历详细信息数组
                    row.add(string); //把每个详细属性添加为表格单元数据
                }
                model.addRow(row.toArray()); //把行数据添加到表格数据模型
                row.clear(); //清除行数据向量对象,为下一行解析作准备
            }
        } while (sc.hasNext());
        process.getInputStream().close(); //关闭输入流
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
```

脚下留神:

在运行该实例前,需要将 WinRAR 安装目录中的 Rar.exe 命令所在位置增加到环境变量中。



技术要点

本实例实现 RAR 压缩文件列表的读取与解析,并把详细信息显示在表格控件中,这一切都需要利用 RAR 命令来实现。RAR 有一个命令参数“v”用于显示指定 RAR 文件的列表,而“-c-”开关参数不显示 RAR 文档的注释信息,方便了程序对文件列表的解析。



Note

实例 171 从 RAR 压缩包中删除文件

(实例位置: 配套资源\SL\12\171)

实例说明

如果需要删除压缩包内的文件,通常需要先解压缩,删除文件再重新压缩。如果是比较大的文件,这样的操作非常耗时。本实例将使用 RAR 命令直接从压缩文档中删除文件,其运行效果如图 12.44 所示。

实现过程

(1) 在 Eclipse 中创建项目 171,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 DeleteFileFromRAR。该类继承了 JFrame 并增加了标签、文本框、按钮和表格控件。在选择了 RAR 文档后,将在表格中显示 RAR 文档中文件的信息。选择一个文件,单击“删除”按钮,完成删除功能,其事件监听器的代码如下:

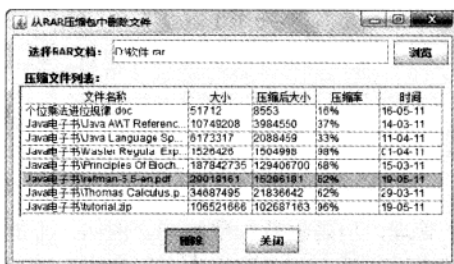


图 12.44 从“软件.rar”文件中删除文件

```
protected void do_delButton_actionPerformed(ActionEvent e) {
    //获取表格数据模型
    DefaultTableModel model = (DefaultTableModel) table.getModel();
    int selectedRow = table.getSelectedRow();           //获取表格当前选择行
    if (selectedRow < 0)
        return;
    String path = model.getValueAt(selectedRow, 0).toString(); //获取选择行中的文件名
    try {
        Process exec = getRuntime().exec("rar d -c-\"" + rarFile + "\" " + path); //执行 RAR 删除命令
        Scanner scan = new Scanner(exec.getInputStream()); //创建进程输入流
        while (scan.hasNext()) { //变量输入流内容
            scan.nextLine(); //清空输入流数据
        }
        scan.close(); //关闭输入流
    } catch (IOException e1) {
        e1.printStackTrace();
    }
    resolveFileList(); //重载表格中文件列表数据
}
```

**脚下留神:**

在运行该实例前,需要将 WinRAR 安装目录中的 Rar.exe 命令所在位置增加到环境变量中。

技术要点

本实例使用 RAR 的命令实现从 RAR 压缩包中删除指定名称的文件。本实例中用到的 RAR 完整命令格式如下:

```
rar d -c- "rarfile" deleteFile
```

参数说明

- ☒ rarfile: 是一个 RAR 压缩文档文件。
- ☒ deleteFile: 是将要从 RAR 压缩文件中删除的文件。

实例 172 在压缩文件中查找字符串

(实例位置: 配套资源\SL\12\172)

实例说明

文件被压缩以后,虽然能够节省空间、便于管理,但是也有一些弊端。例如,不能直接查看文件中的内容。本实例完成在指定压缩文件中查找字符串的功能,如果文本文件中包含指定的字符串,将在文本区中显示,其运行效果如图 12.45 所示。

实现过程

(1) 在 Eclipse 中创建项目 172,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 FindRARString。该类继承了 JFrame 并增加了标签、文本框、按钮和文本区等控件。在选择 RAR 文档后,输入搜索的文本以及文件类型,单击“搜索”按钮完成字符串的搜索,并将结果显示在文本区中。“搜索”按钮事件监听器的关键代码如下:

```
protected void do_searchButton_actionPerformed(ActionEvent e) {
    String searchText = searchStringField.getText();
    if (searchText.isEmpty() || rarFile == null) {
        getToolkit().beep(); //发出提示声音
        return;
    }
    String arg = group.getSelection().getActionCommand(); //获取区分大小写的标记
    int count = 0;
    try {
        //执行 RAR 命令
        Process process = getRuntime().exec("rar i" + arg + "=\"" + searchText + "\" -c- \"" + rarFile
        + "\" " + extNameField.getText());
```



图 12.45 查找压缩文件中的字符串



```

Scanner scan = new Scanner(process.getInputStream()); //获取进程的输入流扫描器
infoArea.setText("");
while (scan.hasNext()) {                               //遍历进程执行结果
    String line = scan.nextLine();                     //获取单行信息
    if (line.isEmpty())
        count++;
    if (count < 2)                                     //过滤非查询结果
        continue;
    infoArea.append(line + "\n");                       //将查询结果添加到文本域控件
}
} catch (IOException e1) {
    e1.printStackTrace();
}
}

```



Note

脚下留神:

在运行该实例前,需要将 WinRAR 安装目录中的 Rar.exe 命令所在位置增加到环境变量中。

技术要点

本实例使用 RAR 的命令实现从 RAR 压缩包中搜索指定的字符串,并确定该字符串位于某个文件中。本实例中用到的 RAR 完整命令格式如下:

```
rar i[i]c="sText" -c- "rarFile" extName
```

参数说明

- ☒ sText: 是要搜索的文本字符串。
- ☒ rarfile: 是一个 RAR 压缩文档文件。
- ☒ extName: 是要搜索的文件类型,也就是文件的扩展名,可以使用通配符。

实例 173 重命名 RAR 压缩包中文件

(实例位置: 配套资源\SL\12\173)

实例说明

当文件被压缩之后,并不能很方便地修改压缩包内文件的名字。本实例将完成修改压缩包内文件名字的功能,其运行效果如图 12.46 所示。

实现过程

(1) 在 Eclipse 中创建项目 173,并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件,名称为 RenameFileFromRAR。该类继承了 JFrame 并增加了标签、文本框、按钮和表格

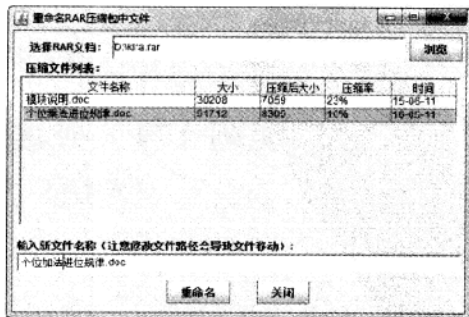


图 12.46 修改指定压缩包内的文件名称



控件。在选择了 RAR 文档后，会在下方的表格中显示压缩包内的文件。选择一个文件，然后输入新的名字，单击“重命名”按钮，完成文件的重命名功能，该按钮事件监听器的代码如下：

```
protected void do_renameButton_actionPerformed(ActionEvent e) {
    DefaultTableModel model = (DefaultTableModel) table.getModel(); //获取表格数据模型
    int selectedRow = table.getSelectedRow(); //获取表格当前选择行
    if (selectedRow < 0)
        return;

    String path = model.getValueAt(selectedRow, 0).toString(); //获取选择行中的文件名
    String newFile = newFileField.getText(); //获取新文件名称
    try {
        //执行 RAR 改名命令
        Process exec = getRuntime().exec("rar rn -c- \"" + rarFile + "\" " + path + " " + newFile);

        Scanner scan = new Scanner(exec.getInputStream()); //创建进程输入流
        while (scan.hasNext()) { //变量输入流内容
            scan.nextLine(); //清空输入流数据
        }
        scan.close(); //关闭输入流
    } catch (IOException e1) {
        e1.printStackTrace();
    }
    resolveFileList(); //重载表格中文件列表数据
}
```

脚下留神：

在运行该实例前，需要将 WinRAR 安装目录中的 Rar.exe 命令所在位置添加到环境变量中。

技术要点

本实例使用 RAR 命令实现从 RAR 压缩包中搜索指定的字符串，并确定该字符串位于某个文件中。本实例中用到的 RAR 完整命令格式如下：

```
rar rn <rarFile> <sourceFile1> <target1>
```

参数说明

- ☒ rarfile：是一个 RAR 压缩文档文件。
- ☒ sourceFile1：是要修改的位于压缩包中的文件名称。
- ☒ target1：是新的文件名称，源文件将使用这个新名称命名。

实例 174 创建自解压 RAR 压缩包

（实例位置：配套资源\SL\12\174）

实例说明

在使用 WinRAR 压缩软件后，通常使用者的计算机需要安装解压缩软件。为了避免这个限制，可以制作自解压 RAR 包。本实例将完成这个功能，其运行效果如图 12.47 所示。

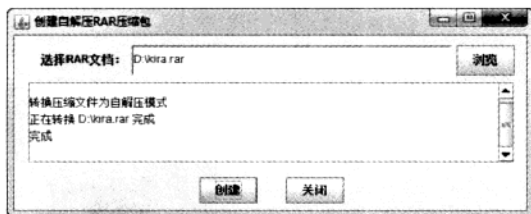


图 12.47 制作自解压 RAR 文件



Note

实现过程

(1) 在 Eclipse 中创建项目 174，并在该项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建类文件，名称为 SFXRAR。该类继承了 JFrame 并增加了标签、文本框、按钮和文本区控件。在选择了 RAR 文档后，单击“创建”按钮会创建自解压文件，并在文本区显示提示信息。该按钮事件监听器的代码如下：

```
protected void do_createButton_actionPerformed(ActionEvent e) {
    if (rarFile == null) //验证用户是否选择了 RAR 文件
        return;
    try {
        Process process = getRuntime().exec("rar s -y -c- " + rarFile); //执行 RAR 命令
        Scanner scan = new Scanner(process.getInputStream());
        infoArea.setText(""); //清空文本域控件的内容
        int count = 0;
        while (scan.hasNext()) { //遍历进程执行结果
            String line = scan.nextLine(); //获取单行信息
            if (line.isEmpty())
                count++;
            if (count < 2) //过滤非查询结果
                continue;
            infoArea.append(line + "\n"); //将查询结果添加到文本域控件
        }
    } catch (IOException e1) {
        e1.printStackTrace();
    }
}
```

脚下留神：

在运行该实例前，需要将 WinRAR 安装目录中的 Rar.exe 命令所在位置添加到环境变量中。

技术要点

本实例使用 RAR 命令实现为指定 RAR 压缩包添加自解压模块的功能，这将使目标 RAR 压缩文件拥有自行解压缩的能力，而不需要其他软件来实现解压缩。本实例中用到的 RAR 完整命令格式如下：

```
rar s <rarFile>
```

参数说明

rarFile: 是一个 RAR 压缩文档文件。

第 13 章

枚举类型与泛型

本章读者可以学到如下实例：

- » 实例 175 查看枚举类型的定义
- » 实例 176 枚举类型的基本特性
- » 实例 177 增加枚举元素的信息
- » 实例 178 选择合适的枚举元素
- » 实例 179 高效的枚举元素集合
- » 实例 180 高效的枚举元素映射
- » 实例 181 使用枚举接口遍历元素
- » 实例 182 使用泛型实现栈结构
- » 实例 183 自定义泛型化数组类
- » 实例 184 泛型方法与数据查询
- » 实例 185 使用通配符增强泛型
- » 实例 186 泛型化的折半查找法



实例 175 查看枚举类型的定义

(实例位置: 配套资源\SL\13\175)

实例说明

Java SE 5.0 版中新增了一个重要类型: 枚举。它可以用来表示一组取值范围固定的变量。使用 `enum` 关键字可以定义枚举类型。在深入学习枚举类型之前, 使用反射查看一下其定义是很有益处的。本实例将查看枚举类型的修饰符、父类和自定义方法, 实例的运行效果如图 13.1 所示。

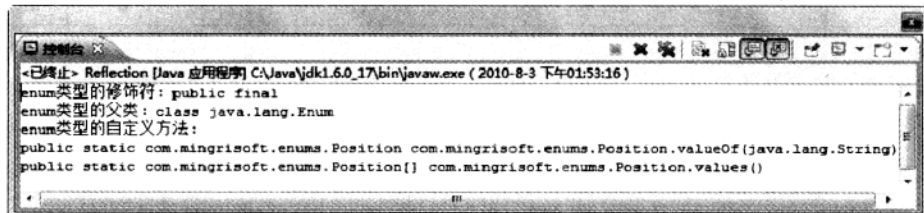


图 13.1 查看枚举类型的定义

实现过程

(1) 定义一个简单的枚举类型 `Position`, 它有两个元素 `HERE` 和 `THERE`, 用来表示方位。关键代码如下:

```
public enum Position {                //定义含有两个元素的简单枚举类型
    HERE, THERE
}
```

指点迷津:

对于枚举类型的元素, 其命名方式与常量相同, 即全部使用大写字母。

(2) 编写 `Reflection` 类, 在该类的 `main()` 方法中输出了枚举类型的修饰符、父类和自定义方法。关键代码如下:

```
public class Reflection {
    public static void main(String[] args) {
        Class<Position> enumClass = Position.class;        //获得表示枚举类型的 Class 对象
        String modifiers = Modifier.toString(enumClass.getModifiers()); //获得枚举类型修饰符
        System.out.println("enum 类型的修饰符: " + modifiers);
        System.out.println("enum 类型的父类: " + enumClass.getSuperclass());
        System.out.println("enum 类型的自定义方法: ");
        Method[] methods = enumClass.getDeclaredMethods(); //获得枚举类型的自定义方法
        for (Method method : methods) {
            System.out.println(method);                    //输出方法的完整名称
        }
    }
}
```





技术要点

利用 Java 的反射机制，可以在运行时分析类。在使用反射之前需要获得 Class 类的对象，该类是反射的入口。反射机制非常强大，如果读者对此感兴趣可以参考第 7 章中的反射实例。本实例使用的方法如表 13.1 所示。



Note

表 13.1 Class 类的常用方法

方 法 名	作 用
getModifiers()	返回一个整数值来表示该类的修饰符
getSuperclass()	返回该类的超类
getDeclaredMethods()	返回该类声明的方法

多学两招：

Java 中并不是所有方法都在 API 中有说明，如由编译器增加的方法。如果使用 IDE 的提示功能，则可能遇到 API 中没有的方法。此时正是反射机制大显身手的时候，读者可以参考本实例自己编写一个工具类来分析“隐藏”方法。

实例 176 枚举类型的基本特性

(实例位置：配套资源\SL\13\176)

实例说明

在枚举类型出现以前，要定义一组取值范围固定的变量通常的做法是定义一个接口，将不同的变量使用不同的整数进行赋值。这样做的缺点是明显的：首先不能确保数字的合法性，其次使用也很不方便，不能根据数字知道它所代表的含义。本实例将演示枚举类型是如何解决这些问题的。实例的运行效果如图 13.2 所示。

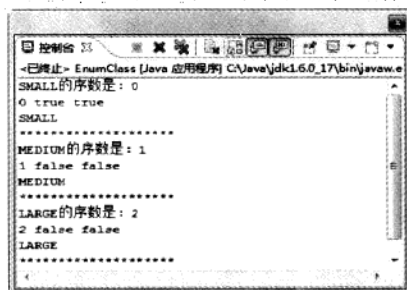


图 13.2 枚举类型的基本特性

实现过程

(1) 定义一个简单的枚举类型 Size，并定义 3 个简单的枚举类型变量。关键代码如下：

```
public enum Size {                                     //定义含有 3 个元素的简单枚举类型
    SMALL, MEDIUM, LARGE
}
```

**指点迷津:**

对于枚举类型的元素,其命名方式与常量相同,即全部使用大写字母。

(2) 编写 EnumClass 类,在该类的 main()方法中输出了枚举元素的序数、与 SMALL 元素比较的 3 种方式得到的结果、枚举元素的名称等。关键代码如下:

```
public class EnumClass {
    public static void main(String[] args) {
        for (Size size : Size.values()) {
            System.out.println(size + "的序数是: " + size.ordinal()); //查看枚举元素的顺序
            System.out.print(size.compareTo(Size.SMALL) + " "); //将枚举元素与 Size.SMALL 比较
            System.out.print(size.equals(Size.SMALL) + " "); //将枚举元素与 Size.SMALL 比较
            System.out.println(size == Size.SMALL); //将枚举元素与 Size.SMALL 比较
            System.out.println(size.name()); //获得枚举元素的名称
            System.out.println("*****"); //分隔线
        }
    }
}
```



Note

指点迷津:

可以使用加强版 for 循环输出 values()方法返回的数组。

技术要点

Enum 类是所有枚举类型的父类,它是一个没有抽象方法的抽象类。该类定义了枚举类型的常用方法,如枚举元素间的比较、获得枚举元素定义的次序、枚举元素定义的名称等。本实例使用的方法如表 13.2 所示。

表 13.2 Enum 类的常用方法

方 法 名	作 用
compareTo(E o)	比较枚举元素的顺序
equals(Object other)	判断枚举元素是否相同
name()	获得枚举元素在定义时的名称
ordinal()	获得枚举元素在定义时的顺序,从 0 开始计数

指点迷津:

可以使用“==”来比较两个枚举元素,不需要重写 equals()和 hashCode()方法,它们已经自动生成了。

实例 177 增加枚举元素的信息

(实例位置: 配套资源\SL\13\177)

实例说明

除了不能继承外,枚举类型可以看作是普通类。这意味着可以在枚举类型中增加方法,甚



Note

至是 `main()` 方法。由于 `toString()` 方法只是简单地返回定义枚举变量时指定的名称, 提供的信息非常有限。可以为枚举类型提供一个构造方法来增加额外的信息, 并提供相应的方法来获得这些信息。本实例将演示如何实现。实例的运行效果如图 13.3 所示。

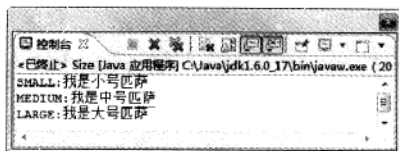


图 13.3 增加枚举元素信息

实现过程

定义一个简单的枚举类型 `Size`, 它有 3 个元素, 分别用来表示匹萨饼的大小。关键代码如下:

```
public enum Size {  
    SMALL("我是小号匹萨"), MEDIUM("我是中号匹萨"), LARGE("我是大号匹萨");  
    private String description;           //定义一个字符串保存描述信息  
    private Size(String description) {    //定义一个私有的构造方法来使枚举元素具有指定描述  
        this.description = description;  
    }  
    public String getDescription() {      //获得枚举元素指定的描述信息  
        return description;  
    }  
    public static void main(String[] args) {  
        for (Size size : Size.values()) {  
            System.out.println(size + ":" + size.getDescription()); //输出所有枚举元素的信息  
        }  
    }  
}
```

脚下留神:

必须先定义枚举类型才能定义方法, 而且两种之间要使用分号分隔。

技术要点

在枚举类型中定义的方法和在类中定义的方法是一样的, 对于普通方法由修饰符、返回值、方法名称和方法参数组成; 对于构造方法由修饰符、方法名称和方法参数组成。需要特别注意的是, 构造方法能使用 `public` 和 `protected` 修饰符。枚举类型的构造方法只能用来创建枚举元素, 而不能用来创建枚举类型的实例。

实例 178 选择合适的枚举元素

(实例位置: 配套资源\SL\13\178)

实例说明

在使用枚举类型时, 会遇到根据不同的枚举元素完成不同操作的情况, 这涉及如何选择枚举元素。本实例将使用枚举类型保存 JDBC 参数, 并自定义方法以根据不同的枚举元素获得相应的参数值。实例的运行效果如图 13.4 所示。

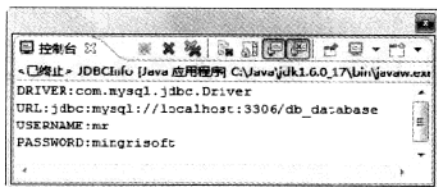


图 13.4 选择合适的枚举元素



Note

实现过程

定义一个简单的枚举类型 JDBCInfo, 其中定义了 4 个枚举元素: DRIVER 代表 JDBC 的驱动, URL 代表 JDBC 的 URL, USERNAME 代表数据库的用户名, PASSWORD 代表数据库的密码。利用 getJDBCInfo() 可以获得枚举元素所对应的实际值。关键代码如下:

```
public enum JDBCInfo {
    DRIVER, URL, USERNAME, PASSWORD;    //定义 4 个枚举元素
    public String getJDBCInfo(JDBCInfo info) { //定义方法来根据不同的枚举元素返回不同的字符串
        switch (info) {
            case DRIVER:    //如果枚举元素是 DRIVER, 则返回数据库驱动
                return "com.mysql.jdbc.Driver";
            case URL:    //如果枚举元素是 URL, 则返回数据库 URL
                return "jdbc:mysql://localhost:3306/db_database";
            case USERNAME:    //如果枚举元素是 USERNAME, 则返回数据库用户名
                return "mr";
            case PASSWORD:    //如果枚举元素是 PASSWORD, 则返回数据库密码
                return "mingrisoft";
            default:
                return null;
        }
    }
    public static void main(String[] args) {
        for (JDBCInfo info : JDBCInfo.values()) { //遍历输出枚举元素的名称和对应的字符串
            System.out.println(info + ":" + info.getJDBCInfo(info));
        }
    }
}
```

多学两招:

可以使用静态导入来避免使用枚举类型引用枚举元素。

技术要点

枚举类型的一种方便用法是它可以用在 switch 语句中。通常情况下, switch 语句只能用于整数值, 如 byte、short 和 int。由于枚举元素在定义时编译器会自动为其生成整数序号, 这些序号可以通过 ordinal() 方法查看, 所以也可在 switch 语句中使用枚举。

多学两招:

在 case 子句中, 可以直接使用枚举元素而不需要使用枚举类型来引用该元素。



实例 179 高效的枚举元素集合

(实例位置: 配套资源\SL\13\179)



Note

实例说明

Set 是 Java 集合类的重要组成部分, 它用来存储不能重复的对象。枚举类型也要求其枚举元素各不相同。看起来枚举类型和集合是很相似的, 然而枚举类型中的元素不能随意地增加、删除, 作为集合而言, 枚举类型非常不实用。EnumSet 是专门为 enum 实现的集合类, 本实例将演示其用法。实例的运行效果如图 13.5 所示。

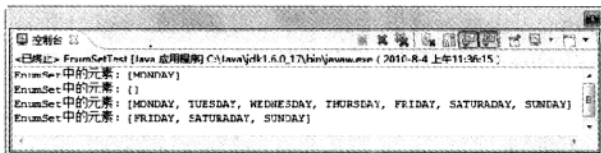


图 13.5 枚举元素集合

实现过程

(1) 定义一个简单的枚举类型 Weeks, 它有 7 个元素, 分别代表一周的 7 天。关键代码如下:

```
public enum Weeks { //定义含有 7 个元素的简单枚举类型
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURADAY, SUNDAY
}
```

(2) 定义 EnumSetTest 类, 在其 main() 方法中对 EnumSet 进行一些基本操作, 如增加元素、删除元素等。关键代码如下:

```
public class EnumSetTest {
    public static void main(String[] args) {
        EnumSet<Weeks> week = EnumSet.noneOf(Weeks.class); //创建一个 Weeks 类型的 EnumSet
        week.add(MONDAY); //向 EnumSet 中增加元素 MONDAY
        System.out.println("EnumSet 中的元素: " + week);
        week.remove(MONDAY); //删除 Enumset 中的元素 MONDAY
        System.out.println("EnumSet 中的元素: " + week);
        week.addAll(EnumSet.complementOf(week)); //向 EnumSet 中增加 week 中元素的补集
        System.out.println("EnumSet 中的元素: " + week);
        week.removeAll(EnumSet.range(MONDAY, THURSDAY)); //删除 week 中 MONDAY 到 THURSDAY 元素
        System.out.println("EnumSet 中的元素: " + week);
    }
}
```

多学两招:

可以使用静态导入来避免使用枚举类型引用枚举元素。

技术要点

当创建 EnumSet 对象时, 需要显式或隐式指明其中元素的枚举类型。该对象中的元素仅能



取自同一种枚举类型。EnumSet 在内部用比特向量表示。这种结构特别紧凑和高效。该类的时间、空间性能十分优越，可以高质量替代传统的“位标志”。本实例使用的方法如表 13.3 所示。

表 13.3 EnumSet 类的常用方法

方法名	作用
allOf(Class<E> elementType)	创建一个 EnumSet，它包含了 elementType 中的所有枚举元素
complementOf(EnumSet<E> s)	创建一个 EnumSet，其中的元素是 s 的补集
noneOf(Class<E> elementType)	创建一个 EnumSet，其中元素的类型是 elementType，但是没有元素
range(E from, E to)	创建一个 EnumSet，其中的元素在 from 和 to 之间，包括端点
add(E e)	向 EnumSet 对象中增加元素 e
remove(Object o)	从 EnumSet 对象中删除元素 o
addAll(Collection<? extends E> c)	向 EnumSet 对象中增加集合元素 c
removeAll(Collection<?> c)	从 EnumSet 对象中删除集合元素 c

脚下留神：

不能在 EnumSet 中增加 null 元素，否则会出现空指针异常。

实例 180 高效的枚举元素映射

(实例位置：配套资源\SL\13\180)

实例说明

Map 是 Java 集合类的重要组成部分，其用途是利用键值对来保存对象。当需要使用值时，可以根据键获得。这要求 Map 的键必须唯一，而枚举类型的元素都是唯一的，因此可以用来做 Map 的键。EnumMap 类就是 Java 专门为枚举类型提供的 Map 实现类。本实例将演示其用法，实例的运行效果如图 13.6 所示。

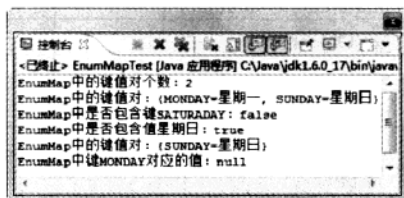


图 13.6 枚举元素映射

实现过程

(1) 定义一个简单的枚举类型 Weeks，它有 7 个元素，分别代表一周的 7 天。关键代码如下：

```
public enum Weeks { //定义含有 7 个元素的简单枚举类型
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURADAY, SUNDAY
}
```

(2) 定义 EnumMapTest 类，在其 main() 方法中对 EnumMap 进行一些基本操作，如增加键值对、删除键值对等。关键代码如下：

```
public class EnumMapTest {
    public static void main(String[] args) {
        EnumMap<Weeks, String> weeks = new EnumMap<Weeks, String>(Weeks.class);
        weeks.put(MONDAY, "星期一"); //增加键 MONDAY，值为“星期一”
        weeks.put(SUNDAY, "星期日"); //增加键 SUNDAY，值为“星期日”
    }
}
```



Note



Note

```
System.out.println("EnumMap 中的键值对个数: " + weeks.size()); //查看键值对个数
System.out.println("EnumMap 中的键值对: " + weeks); //查看键值对内容
System.out.println("EnumMap 中是否包含键 SATURADAY: " + weeks.containsKey
(SATURADAY));
System.out.println("EnumMap 中是否包含值星期日: " + weeks.containsValue("星期日"));
weeks.remove(MONDAY); //删除键为 MONDAY 的键值对
System.out.println("EnumMap 中的键值对: " + weeks);
System.out.println("EnumMap 中键 MONDAY 对应的值: " + weeks.get(MONDAY));
}
}
```

多学两招:

可以使用静态导入来避免使用枚举类型引用枚举元素。

技术要点

当创建 EnumMap 对象时,需要显式或隐式指明其中元素的枚举类型。该对象中元素的键仅能取自同一种枚举类型。EnumMap 在内部用数组表示,这种结构特别紧凑和高效。本实例使用的方法如表 13.4 所示。

表 13.4 EnumMap 类的常用方法

方 法 名	作 用
clear()	删除该 map 中的所有映射关系
containsKey(Object key)	如果包含值为 key 的键则返回 true, 否则返回 false
containsValue(Object value)	如果包含值为 value 的值则返回 true, 否则返回 false
put(K key, V value)	在 EnumMap 中存入键为 key、值为 value 的键值对
get(Object key)	获得键 key 所对应的值
size()	查看 EnumMap 中键值对的个数
remove(Object key)	从 EnumMap 中删除键为 key 的键值对

脚下留神:

不能在 EnumMap 中增加 null 元素,否则会出现空指针异常。

实例 181 使用枚举接口遍历元素

(实例位置: 配套资源\SL\13\181)

实例说明

早在 Java SE 1.0 版,就存在集合类。集合类可以用来管理一组相关的对象。当需要查看、使用集合中的所有对象时可以使用枚举接口对其进行遍历。枚举接口中定义了两个方法,它通常和向量一起使用。本实例将演示其用法,实例的运行效果如图 13.7 所示。

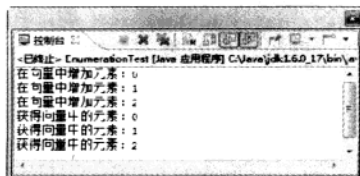


图 13.7 使用枚举接口遍历元素



实现过程

编写 EnumerationTest 类, 在该类的 main() 方法中, 首先在向量中增加 3 个元素, 然后利用枚举接口将其取出。关键代码如下:

```
public class EnumerationTest {
    public static void main(String[] args) {
        Vector<Integer> vector = new Vector<Integer>(); //定义一个向量保存测试用的数据
        for (int i = 0; i < 3; i++) {
            vector.add(i); //在向量中存入数据
            System.out.println("在向量中增加元素: " + i);
        }
        Enumeration<Integer> e = vector.elements(); //将向量转换成枚举接口类型
        while (e.hasMoreElements()) { //输出枚举接口中的全部元素
            System.out.println("获得向量中的元素: " + e.nextElement());
        }
    }
}
```



Note

技术要点

实现了 Enumeration 接口的对象可以生成一系列元素, 每次生成一个。通过连续调用 nextElement() 方法可以连续获得枚举接口中的元素。但是如果枚举接口中已经没有元素, 调用该方法会抛出异常, 因此应该先用 hasMoreElements() 方法判断枚举中是否还有可用元素。该接口定义了两个方法, 其声明如下:

```
boolean hasMoreElements()
```

测试枚举接口中是否还含有可用元素, 因为其返回值是 boolean, 所以适合放在 while 循环中。

```
E nextElement()
```

如果枚举接口中还有可用元素则返回下一个元素, 否则会出现 NoSuchElementException 异常。

多学两招:

Collections 类的静态方法 enumeration() 可以用来将任意集合转换成枚举接口类型。

实例 182 使用泛型实现栈结构

(实例位置: 配套资源\SL\13\182)

实例说明

泛型是 Java SE 5.0 版的重要特性, 使用泛型编程可以使代码获得最大的重用。由于在使用泛型时要指明泛型的具体类型, 这样就避免了类型转换。本实例将使用泛型来实现一个栈结构, 并对其进行测试。实例的运行效果如图 13.8 所示。

实现过程

(1) 编写泛型类 Stack, 该类中定义了 3 个方

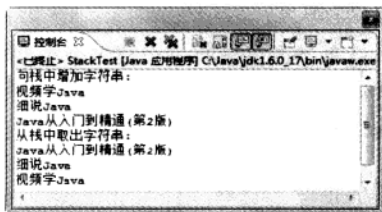


图 13.8 使用泛型实现栈结构



法，分别是用来入栈的 `push()` 方法、用来出栈的 `pop()` 方法和用来判断栈是否为空的 `empty()` 方法。在底层上，本类使用 `LinkedList` 作为容器，它是 Java 集合类的一员，可以用来简化开发。关键代码如下：

```
public class Stack<T> {                                //定义参数类型为 T 的类
    private LinkedList<T> container = new LinkedList<T>(); //使用 T 类型的链表保存入栈的元素
    public void push(T t) {                               //实现了向栈中增加元素的功能
        container.addFirst(t);
    }
    public T pop() {                                       //实现了从栈中删除元素的功能
        return container.removeFirst();
    }
    public boolean empty() {                               //判断链表中是否有可用元素
        return container.isEmpty();
    }
}
```

多学两招：

泛型参数的命名一般使用单个的大写字母，如对于任意类型可以使用字母 `T` 等。

(2) 编写测试类 `StackTest`，在该类的 `main()` 方法中向栈中增加了 3 个字符串，又从栈中删除 3 个字符串并进行输出。关键代码如下：

```
public class StackTest {
    public static void main(String[] args) {
        Stack<String> stack = new Stack<String>(); //在创建栈对象时就指明该栈中只能保存字符串
        System.out.println("向栈中增加字符串：");
        System.out.println("视频学 Java");
        System.out.println("细说 Java");
        System.out.println("Java 从入门到精通(第 2 版)");
        stack.push("视频学 Java");                //向栈中增加字符串
        stack.push("细说 Java");                  //向栈中增加字符串
        stack.push("Java 从入门到精通(第 2 版)"); //向栈中增加字符串
        System.out.println("从栈中取出字符串：");
        while (!stack.empty()) {
            System.out.println((String) stack.pop()); //删除栈中全部元素并进行输出
        }
    }
}
```

技术要点

泛型类就是含有一个或多个类型参数的类。定义泛型类很简单，只需要在类的名称后面加上 “<” 和 “>”，并在其中指明类型参数，如本例中的 `T`。也可以在其中指明多个参数，如 `K` 和 `V`。多个参数之间使用逗号分隔。在定义完类后，就可以在类中的域和方法中使用泛型参数。

多学两招：

泛型类型的参数只能使用类类型，而不能使用基本数据类型。



实例 183 自定义泛型化数组类

(实例位置: 配套资源\SL\13\183)



Note

实例说明

Java 虚拟机中并没有泛型类型的对象, 所有有关泛型的信息都被擦除了。这虽然可以避免 C++ 语言的模板代码膨胀问题, 但是也引起了其他问题, 如不能直接创建泛型数组。为了弥补这个不足, 本实例将利用反射机制创建一个泛型化数组。实例的运行效果如图 13.9 所示。

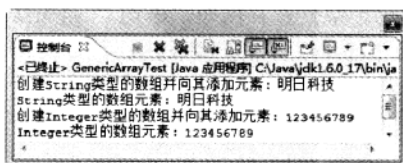


图 13.9 自定义泛型数组

实现过程

(1) 自定义泛型类 `GenericArray`, 该类实现了泛型数组的功能。它定义了两个方法, `put()` 方法用于在指定位置插入元素, `get()` 方法用于获得指定位置的元素。关键代码如下:

```
public class GenericArray<T> {
    private T[] array;           //声明一个类型为 T 的数组 array
    private int size;           //声明一个整型变量保存数组的长度
    @SuppressWarnings("unchecked")
    public GenericArray(Class<T> type, int size) {
        this.size = size;
        array = (T[]) Array.newInstance(type, size); //利用反射根据指定的类型和长度创建泛型数组
    }
    public void put(int index, T item) {           //向数组中增加元素的方法
        if (size > index) {
            array[index] = item;
        }
    }
    public T get(int index) {                       //获得数组中元素的方法
        if (size > index) {
            return array[index];
        } else {
            return null;
        }
    }
}
```

(2) 编写 `GenericArrayTest` 类, 在该类的 `main()` 方法中定义了两个数组用来进行测试。关键代码如下:

```
public class GenericArrayTest {
    public static void main(String[] args) {
        System.out.println("创建 String 类型的数组并向其添加元素: 明日科技");
    }
}
```



Note

```
GenericArray<String> stringArray = new GenericArray<String>(String.class, 10);
stringArray.put(0, "明日科技");
System.out.println("String 类型的数组元素: " + stringArray.get(0));
System.out.println("创建 Integer 类型的数组并向其添加元素: 123456789");
GenericArray<Integer> integerArray = new GenericArray<Integer>(Integer.class, 10);
integerArray.put(0, 123456789);
System.out.println("Integer 类型的数组元素: " + integerArray.get(0));
}
```

技术要点

Java 中的泛型不支持实例化类型变量。例如，语句“`T[] array = new T[10];`”在 Java 语言中是非法的。但是在自定义数据结构时，如果需要使用泛型数组该怎么办呢？答案是反射机制。Array 类中的 `newInstance()` 方法可以根据指定的类型和长度创建一个数组，该方法的声明如下：

```
newInstance(Class<?> componentType, int length)
```

参数说明

- ☒ `componentType`: 数组元素的类型。
- ☒ `length`: 数组的长度。

脚下留神:

Java 泛型使用起来有很多的局限性，如不能使用基本类型作为其类型参数、不能抛出或捕获泛型类型的实例、不能直接使用泛型数组、不能实例化类型变量等。希望读者在使用泛型时多加注意。对于其中的某些不足，可以使用 Java 的反射机制进行弥补。虽然反射的效率不高，但是也只能忍受了。

实例 184 泛型方法与数据查询

(实例位置: 配套资源\SL\13\184)

实例说明

在使用 JDBC 查询数据库中的数据时，返回的结果是 `ResultSet` 对象，使用十分不方便。`Commons DbUtils` 组件提供了将 `ResultSet` 转换为 `Bean` 列表的方法，但是该方法在使用时需要根据不同的 `Bean` 对象创建不同的查询方法。本实例将在该方法的基础上使用泛型进行包装，使其通用性更强。实例的运行效果如图 13.10 所示。

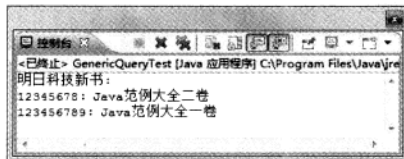


图 13.10 泛型方法与数据查询

实现过程

(1) 编写 `GenericQuery` 类，该类实现了两个方法，即 `getConnection()` 和 `query()`。



getConnection()方法用于获得数据库的连接, query()方法用于根据用户指定的 SQL 语句进行查询, 并将查询的结果转换成 Bean 列表。关键代码如下:

```
public class GenericQuery {                                //定义 JDBC 参数
    private static String URL = "jdbc:mysql://localhost:3306/db_database";
    private static String DRIVER = "com.mysql.jdbc.Driver";
    private static String USERNAME = "mr";
    private static String PASSWORD = "mingrisoft";
    private static Connection conn;
    public static Connection getConnection() {
        DbUtils.loadDriver(DRIVER);                        //加载数据库驱动
        try {
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD); //获得数据库连接
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return conn;
    }
    public static <T> List<T> query(String sql, Class<T> type) {
        QueryRunner qr = new QueryRunner();
        List<T> list = null;                                //定义泛型参数类型的列表
        try {                                                //将 ResultSet 转换成类型为 T 的参数类型的列表
            list = qr.query(getConnection(), sql, new BeanListHandler<T>(type));
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DbUtils.closeQuietly(conn);                      //释放连接
        }
        return list;
    }
}
```



Note

(2) 编写 Books 类代表数据库中的 books 表, 它有两个域, 即 id 和 name, 分别对应 books 表的 id 和 name。关键代码如下:

```
public class Books {
    private int id;                                         //代表图书的编号
    private String name;                                   //代表图书的名称
    ...//省略了 get()和 set()方法
    @Override
    public String toString() {                             //重写 toString()方法方便输出 Books 类的对象
        return id + ": " + name;
    }
}
```

(3) 编写 GenericQueryTest 类来对 GenericQuery 类进行测试, 在 GenericQueryTest 类的主方法 main()中进行简单的查询, 结果被转换成 Bean 列表, 然后遍历该列表进行输出。关键代码如下:

```
public class GenericQueryTest {
    public static void main(String[] args) {
        String sql = "select * from books;";              //简单的查询语句
        List<Books> list = GenericQuery.query(sql, Books.class); //获得 Bean 列表
    }
}
```



```
System.out.println("明日科技新书: ");
for (Books books : list) {
    System.out.println(books);
}
//输出 Bean 列表中的全部对象
```

技术要点

在 Java 中,不仅可以声明泛型类,而且还可以在普通类中声明泛型方法。声明泛型方法需要注意如下几点:

- ☑ 使用<T>格式来表示泛型类型参数,参数的个数可以不是一个。
- ☑ 类型参数列表要放在访问权限修饰符、static 和 final 之后。
- ☑ 类型参数列表要放在返回值类型、方法名称、方法参数之前。

实例 185 使用通配符增强泛型

(实例位置: 配套资源\SL\13\185)

实例说明

利用泛型类型参数<T>,可以将类、方法或接口的类型限制为 T 类型,但是这种方式显然不够灵活。例如,<T extends Number>可以将类型限制为 Number 的一种子类型,一旦指定了该类型,就不能再修改了,而如果使用通配符就会让代码更加灵活。本实例将演示如何在泛型方法中使用通配符,实例的运行效果如图 13.11 所示。

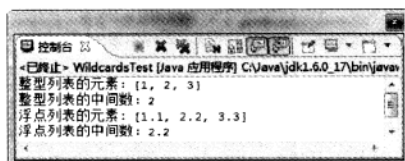


图 13.11 使用通配符增强泛型

实现过程

编写 WildcardsTest 类,该类包含两个方法,getMiddle()方法用于获得给定列表的中间值,其参数中要求列表参数的类型是任何 Number 类型的子集;main()方法用来进行测试。关键代码如下:

```
public class WildcardsTest {
    public static Object getMiddle(List<? extends Number> list) {
        return list.get(list.size() / 2);
    }
    public static void main(String[] args) {
        List<Integer> ints = new ArrayList<Integer>(); //创建一个整型参数的列表进行测试
        ints.add(1); //在列表中增加元素
        ints.add(2);
        ints.add(3);
        System.out.print("整型列表的元素: ");
        System.out.println(Arrays.toString(ints.toArray()));
    }
    //返回列表的中间值
    //输出列表中的全部元素
```




```
System.out.println("整型列表的中间数: " + getMiddle(ints));
List<Double> doubles = new ArrayList<Double>();//创建一个浮点参数的列表进行测试
doubles.add(1.1);                               //在列表中增加元素
doubles.add(2.2);
doubles.add(3.3);
System.out.print("浮点列表的元素: ");
System.out.println(Arrays.toString(doubles.toArray())); //输出列表中的全部元素
System.out.println("浮点列表的中间数: " + getMiddle(doubles));
}
}
```



Note

技术要点

泛型中, 使用“?”作为通配符。通配符的使用与普通的类型参数类似, 如通配符也可以利用 extends 关键字来设置取值上限。“<? extends Number>”表示 Byte、Double、Float、Integer 等都适合这个类型参数。此外, 通配符还可以设置取值下限, 语法如下:

```
<? super Number>
```

其含义是类型参数是 Number 类的父类, 如 Object。

实例 186 泛型化的折半查找法

(实例位置: 配套资源\SL\13\186)

实例说明

查找就是在一组给定的数据集合中找出满足条件的数据。在数据结构中, 查找有很多类型, 如顺序查找、折半查找、散列查找等。作为泛型的一个简单应用, 本实例使用泛型实现折半查找法。实例的运行效果如图 13.12 所示。

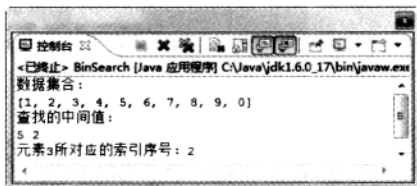


图 13.12 泛型化的折半查找法

实现过程

编写 BinSearch 类, 它有两个方法, search()方法用来在给定的数组 array 中查找 key 的索引位置, main()方法用来进行测试。关键代码如下:

```
public class BinSearch {
    public static <T extends Comparable<? super T>> int search(T[] array, T key) {
        int low = 0; //利用整型变量 low 保存数组的最小索引
        int mid = 0; //利用整型变量 mid 保存数组的中间索引
        int high = array.length; //利用整型变量 high 保存数组的最大索引
        System.out.println("查找的中间值: ");
        while (low <= high) {
```




Note

```
mid = (low + high) / 2; //获得中间索引
System.out.print(mid+" ");
if (key.compareTo(array[mid]) > 0) { //如果 key 大于中间元素，则比较右边
    low = mid + 1;
} else if (key.compareTo(array[mid]) < 0) { //如果 key 小于中间元素，则比较左边
    high = mid - 1;
} else {
    System.out.println();
    return mid; //获得对应元素的索引
}
}
return -1; //如果没有找到则返回-1
}

public static void main(String[] args) {
    Integer[] ints = {1,2,3,4,5,6,7,8,9,0}; //测试数组
    System.out.println("数据集: ");
    System.out.println(Arrays.toString(ints));
    System.out.println("元素 3 所对应的索引序号: "+search(ints, 3));
}
```

技术要点

折半查找要求数据集中的元素必须可比较，并且各元素按升序或降序排列。

取集合的中间元素作为比较对象，则：

- (1) 如果给定的值与比较对象相等，则查找成功，返回中间元素的序号。
- (2) 如果给定的值大于比较对象，则在中间元素的右半段进行查找。
- (3) 如果给定的值小于比较对象，则在中间元素的左半段进行查找。

重复上述过程，直至查找成功。折半算法的平均时间复杂度是 $\log_2 n$ 。

第14章

Swing 入门

本章读者可以学到如下实例：

- » 实例 187 从上次关闭位置启动窗体
- » 实例 188 始终在桌面最顶层显示的窗体
- » 实例 189 设置窗体大小
- » 实例 190 根据桌面大小调整窗体大小
- » 实例 191 自定义最大化、最小化和关闭按钮
- » 实例 192 禁止改变窗体的大小
- » 实例 193 指定窗体标题栏图标
- » 实例 194 设置闪烁的标题栏
- » 实例 195 实现带背景图片的窗体
- » 实例 196 背景为渐变色的主界面
- » 实例 197 随机更换窗体背景
- » 实例 198 椭圆形窗体界面
- » 实例 199 钻石形窗体
- » 实例 200 创建透明窗体
- » 实例 201 信息提示对话框
- » 实例 202 设置信息提示对话框的图标
- » 实例 203 指定打开对话框的文件类型
- » 实例 204 为保存对话框设置默认文件名
- » 实例 205 支持图片预览的文件选择对话框
- » 实例 206 颜色选择对话框
- » 实例 207 信息输入对话框
- » 实例 208 定制信息对话框
- » 实例 209 拦截事件的玻璃窗格
- » 实例 210 简单的每日提示信息
- » 实例 211 震动效果的提示信息
- » 实例 212 制作圆形布局管理器
- » 实例 213 制作阶梯布局管理器
- » 实例 214 密码域控件简单应用
- » 实例 215 文本域设置背景图片
- » 实例 216 文本区设置背景图片
- » 实例 217 简单的字符统计工具
- » 实例 218 能预览图片的复选框
- » 实例 219 简单的投票计数软件
- » 实例 220 单选按钮的简单应用
- » 实例 221 能显示图片的组合框
- » 实例 222 使用滑块来选择日期
- » 实例 223 模仿记事本的菜单栏
- » 实例 224 自定义纵向的菜单栏
- » 实例 225 复选框与单选按钮菜单项
- » 实例 226 包含图片的弹出菜单
- » 实例 227 工具栏的实现与应用
- » 实例 228 修改列表项显示方式
- » 实例 229 列表项与提示信息
- » 实例 230 表头与列的高度设置
- » 实例 231 调整表格各列的宽度
- » 实例 232 设置表格的选择模式
- » 实例 233 为表头增添提示信息
- » 实例 234 单元格的粗粒度排序
- » 实例 235 实现表格的查找功能
- » 实例 236 应用网格布局设计计算器窗体



实例 187 从上次关闭位置启动窗体

(实例位置: 配套资源\SL\14\187)

实例说明

实际开发中, 有很多软件都有一个通用的功能: 从上次关闭位置启动窗体, 那么可不可以用 Java 语言实现这样的功能? 答案是肯定的, 本实例将使用 Java 语言的 Preferences 首选项类实现从上次关闭位置启动窗体的功能, 实例的运行效果如图 14.1 所示。

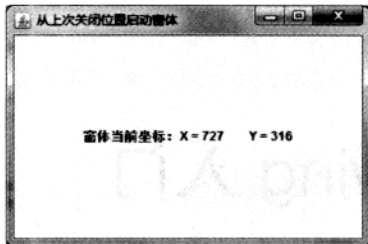


图 14.1 从上次关闭位置启动窗体

实现过程

(1) 在项目中创建窗体类 StartFormByLClosePosition, 在窗体中添加一个标签控件用于显示当前窗体坐标。

(2) 编写窗体移动的事件处理方法, 在该方法中控制标签控件来显示当前窗体的位置, 只要窗体移动就会立刻更新标签控件的信息。关键代码如下:

```
protected void do_this_componentMoved(ComponentEvent e) {  
    Point location = getLocation();           //获取窗体坐标  
    int x = location.x;  
    int y = location.y;  
    //把窗体当前坐标显示在标签控件中  
    label.setText("窗体当前坐标: X = " + x + " Y = " + y);  
}
```

(3) 编写窗体关闭的事件处理方法, 在进行关闭窗体的过程中, 该方法会读取当前窗体的坐标信息并保存到首选项对象中。关键代码如下:

```
protected void do_this_windowClosing(WindowEvent e) {  
    Preferences root = Preferences.userRoot();           //获取用户首选项  
    Point location = getLocation();           //获取窗体位置  
    root.putInt("locationX", location.x);           //保存窗体 X 坐标  
    root.putInt("locationY", location.y);           //保存窗体 Y 坐标  
}
```

(4) 编写窗体打开的事件处理方法, 该方法在窗体打开时被调用, 方法中首先获取首选项对象中的坐标信息, 然后利用该坐标重新为窗体定位。关键代码如下:

```
protected void do_this_windowOpened(WindowEvent e) {  
    Preferences root = Preferences.userRoot();           //获取用户首选项  
    int x = root.getInt("locationX", 100);           //提取窗体 X 坐标  
    int y = root.getInt("locationY", 100);           //提取窗体 Y 坐标  
    setLocation(x, y);           //恢复窗体坐标  
}
```

技术要点

本实例在实现时, 主要是 Preferences 首选项类的应用, 该类的实例对象可以保存程序的各



种参数与设置。下面分别介绍该类对象的相关操作。

(1) 获取用户的根首选项节点，语法如下：

```
public static Preferences userRoot()
```

(2) 向首选项保存整型数值，语法如下：

```
public abstract void putInt(String key, int value)
```

参数说明

- ☑ key: 要与字符串形式的 value 相关联的键。
- ☑ value: 要与 key 相关联的字符串形式的值。

(3) 获取首选项中的整数数值，语法如下：

```
public abstract int getInt(String key, int def)
```

参数说明

- ☑ key: 要作为 int 返回其关联值的键。
- ☑ def: 此首选项节点不具有与 key 相关联的值或者无法将该关联值解释为 int 或者内部存储不可访问时要返回的默认值。



Note

实例 188 始终在桌面最顶层显示窗体

(实例位置: 配套资源\SL\14\188)

实例说明

Windows 桌面上允许多个窗体同时显示，但是只有一个窗体能够得到焦点，当一个窗体得到焦点后在其上面的窗体会被得到焦点的窗体遮挡，得到焦点的窗体会显示在最上层，这样被覆盖的窗体就不能完全地显示给用户，也有某些窗体中具有实时性和比较重要的信息需要随时置顶的特殊情况。本实例将实现此功能，运行本实例后，主窗体会始终显示在桌面的最上面。实例的运行效果如图 14.2 所示。

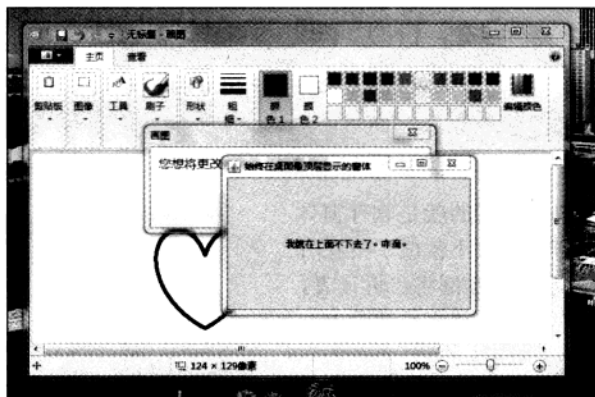


图 14.2 始终在桌面最顶层显示的窗体

指点迷津:

在图 14.2 中，画图程序与其弹出的保存窗口都位于本程序窗口之下，而且画图程序拥有系统与鼠标焦点。



实现过程

(1) 在项目中新建窗口类 AlwaysActiveWindows, 在窗体上添加标签控件。

(2) 编写窗体界面设计代码, 设置窗体标题即添加内容面板与标签控件。最主要的代码在于 setAlwaysOnTop() 方法设置窗体置顶。关键代码如下:

```
public AlwaysActiveWindows() {  
    setTitle("始终在桌面最顶层显示的窗体");           //设置窗体标题  
    setAlwaysOnTop(true);                             //设置窗体显示在最顶端。本实例的核心代码  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    setBounds(100, 100, 319, 206);                   //设置窗体位置  
    contentPane = new JPanel();                       //创建内容面板  
    contentPane.setLayout(new BorderLayout(0, 0));  
    setContentPane(contentPane);                     //设置内容面板  
    JLabel label = new JLabel("我就在上面不下去了。咋滴。");  
    label.setHorizontalAlignment(SwingConstants.CENTER);  
    contentPane.add(label, BorderLayout.CENTER);       //添加标签控件  
}
```

技术要点

在其他开发语言中实现窗体始终在最顶层比较复杂, 但在 Java 中实现非常简单, 只要调用窗体的 setAlwaysOnTop() 方法即可。

setAlwaysOnTop() 方法可以设置窗体是否置顶显示, 也就是说是否使窗体始终显示在其他窗体之上。其语法格式如下:

```
public final void setAlwaysOnTop(boolean alwaysOnTop) throws SecurityException
```

参数说明:

alwaysOnTop: 如果该属性为 true, 则窗体保持置顶显示。

实例 189 设置窗体大小

(实例位置: 配套资源\SL\14\189)

实例说明

用户打开软件后, 首先看到的就是软件窗体的大小, 那么如何设置窗体的大小就成了摆在开发者面前的一个首要问题。本实例将告诉读者, 如何使用 Java 语言实现限制窗体大小的功能, 实例的运行效果如图 14.3 所示。

实现过程

(1) 在项目中新建窗体类 ControlFormSize, 在窗体中添加一个标签控件。

(2) 编写程序代码, 设置窗体的标题、默认关闭方式与窗体大小, 然后在标签控件中显示

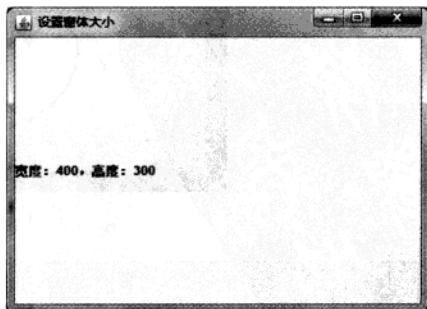


图 14.3 限制窗体大小



对窗体设置的大小。关键代码如下：

```
public ControlFormSize() {
    setTitle("设置窗体大小");           //设置窗体标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //默认关闭方式
    setSize(400, 300);                   //设置窗体大小
    contentPane = new JPanel();           //创建内容面板
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);         //设置内容面板
    JLabel label = new JLabel("宽度：400，高度：300"); //创建标签控件
    contentPane.add(label, BorderLayout.CENTER); //添加标签控件到窗体
}
```



Note

技术要点

本实例在实现设置窗体大小的功能时，主要用到了窗体的 `setSize()` 方法。设置窗体大小的方法有以下两种重载格式。

(1) Dimension 参数

第一个重载方法以 `Dimension` 类的实例对象作为参数，其语法格式如下：

```
public void setSize(Dimension size)
```

参数说明

`size`：封装单个控件中宽度与高度的对象。

(2) int 参数

第二个重载方法以 `int` 类型常量作为参数，其语法格式如下：

```
public void setSize(int width, int height)
```

参数说明

☒ `width`：窗体的宽度（以像素为单位）。

☒ `height`：窗体的高度（以像素为单位）。

`Dimension` 对象用于封装单个控件的宽度与高度，创建其对象的语法格式如下：

```
public Dimension(int width, int height)
```

参数说明

☒ `width`：控件的宽度（以像素为单位）。

☒ `height`：控件的高度（以像素为单位）。

实例 190 根据桌面大小调整窗体大小

（实例位置：配套资源\SL\14\190）

实例说明

窗体与桌面的大小比例是软件运行时用户经常会注意到的一个问题，如在 1024×768 的桌面上，如果放置一个很大（如 1280×1024 ）或者很小（如 10×10 ）的正方形窗体，会显得非常不协调。正是基于以上这种情况，所以大部分软件的窗体都是根据桌面的大小进行自动调整的，本实例就实现这样的功能。实例的运行效果如图 14.4 所示。



Note

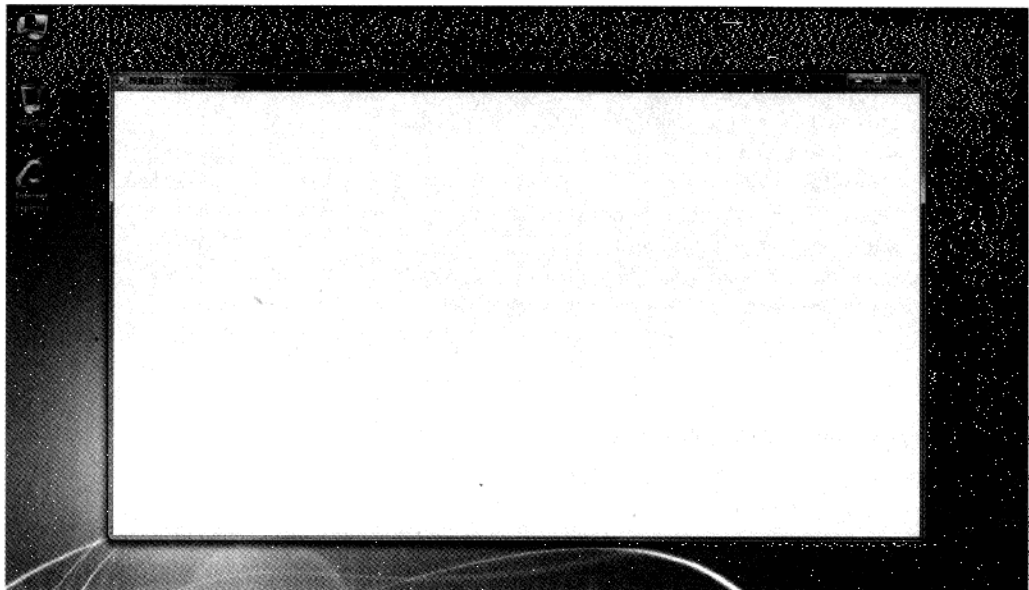


图 14.4 根据桌面大小调整窗体大小

实现过程

(1) 在项目中创建窗体类 `SetFormSizeByDeskSize`。

(2) 编写窗体的打开事件处理方法，该方法在窗体打开时被执行，在方法中，首先获取窗体工具包对象，然后通过工具包对象的 `getScreenSize()` 方法获取屏幕的大小，最后把窗体设置为屏幕大小的 80%。关键代码如下：

```
protected void do_this_windowOpened(WindowEvent e) {  
    Toolkit toolkit = getToolkit();           //获得窗体工具包  
    Dimension screenSize = toolkit.getScreenSize(); //获取屏幕大小  
    int width=(int) (screenSize.width*0.8);      //计算窗体新宽度  
    int height=(int) (screenSize.height*0.8);    //计算窗体新高度  
    setSize(width,height);                     //设置窗体大小  
}
```

技术要点

本实例实现的重点是如何获取桌面的大小，而获取桌面大小时，主要用到窗体的工具包 `Toolkit` 类。下面对本实例中用到的关键技术进行详细介绍。

(1) 获取窗体工具包

每个窗体类都提供了 `getToolkit()` 方法来获取窗体的工具包对象。在窗体内部已经封装了这个工具包，随时可以获取。该方法的声明如下：

```
public Toolkit getToolkit()
```

(2) 获取桌面屏幕大小

窗体的工具包提供了 `getScreenSize()` 方法来获取当前屏幕的大小，该方法的声明如下：

```
public abstract Dimension getScreenSize() throws HeadlessException
```



实例 191 自定义最大化、最小化和关闭按钮

(实例位置: 配套资源\SL\14\191)

实例说明

用户在制作应用程序时, 为了使用户界面更加美观, 一般都自己设计窗体的外观, 以及窗体的最大化、最小化和关闭按钮。本实例实现设计窗体的外观, 及最大化、最小化和关闭按钮, 再通过鼠标来实现窗体移动效果。实例的运行效果如图 14.5 所示。

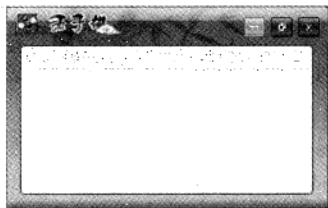


图 14.5 自定义最大化、最小化和关闭按钮

实现过程

(1) 在项目中新建窗体类 `ControlFormStatus`。在窗体中添加背景图片, 在窗体右上角放置 3 个按钮, 分别是最小化、最大化和关闭按钮, 然后设置窗体的 `Undecorated` 属性为 `true` 来阻止窗体采用本机系统的修饰, 这样窗体就没有标题栏和边框了。

(2) 编写最小化按钮的事件处理方法, 在该方法中改变窗体的状态值为 `ICONIFIED` 最小化常量。关键代码如下:

```
protected void do_button_itemStateChanged(ActionEvent e) {
    setExtendedState(JFrame.ICONIFIED);           //窗体最小化
}
```

(3) 编写关闭按钮的事件处理方法, 在该方法中调用销毁窗体的方法, 如果窗体是当前仅剩的唯一窗体, 那么程序就会自动退出; 如果存在执行业务处理的线程, 那么会等待线程结束而关闭虚拟机。关键代码如下:

```
protected void do_button_2_actionPerformed(ActionEvent e) {
    dispose();                                     //销毁窗体
}
```

(4) 编写最大化按钮的事件处理方法, 该按钮是 `JToggleButton` 按钮类的实例对象, 所以它有选择与取消选择两种状态, 在按钮处于选择状态时, 应设置窗体最大化; 而当按钮被取消选择时, 恢复窗体原有大小。关键代码如下:

```
protected void do_button_1_itemStateChanged(ItemEvent e) {
    if (e.getStateChange() == ItemEvent.SELECTED) {
        setExtendedState(JFrame.MAXIMIZED_BOTH);           //最大化窗体
    } else {
        setExtendedState(JFrame.NORMAL);                   //恢复普通窗体状态
    }
}
```

(5) 编写自定义窗体标题栏面板的鼠标事件处理方法, 当用户拖动自定义窗体标题栏时, 应该实现窗体移动的效果。关键代码如下:

```
protected void do_topPanel_mousePressed(MouseEvent e) {
    pressedPoint = e.getPoint();                          //记录鼠标坐标
}
```



Note



Note

```
protected void do_topPanel_mouseDragged(MouseEvent e) {  
    Point point = e.getPoint();           //获取当前坐标  
    Point locationPoint = getLocation();   //获取窗体坐标  
    int x = locationPoint.x + point.x - pressedPoint.x; //计算移动后的新坐标  
    int y = locationPoint.y + point.y - pressedPoint.y;  
    setLocation(x, y);                   //改变窗体位置  
}
```

技术要点

本实例使用的关键技术较多，其中包括取消窗体修饰、按钮外观设置、改变窗体状态等。

(1) 取消窗体修饰

JFrame 窗体默认采用本地系统的窗体修饰，这样会使窗体有标题栏以及标题栏上的所有按钮。但是有些情况需要开发人员根据需求自己定义窗体外观，这时就要禁止 JFrame 继承本地系统的窗体外观修饰，这可以通过 setUndecorated() 方法来实现。该方法的声明如下：

```
public void setUndecorated(boolean undecorated)
```

参数说明

undecorated: 该参数用于指定是否禁止采用本地系统对窗体的修饰，默认值为 false。如果该参数为 true，窗体将没有任何标题栏内容及窗体边框，它看上去像一块灰色的布贴在屏幕上。

(2) 设置按钮外观

按钮的外观一般需要设置其图标属性，这包括按钮按下与抬起的图标、鼠标经过的图标等。但设置图标无法达到预期效果，因为按钮原有外观与边框会显得不自然，所以要对按钮进行特殊设置。下面介绍有关按钮的关键技术。

☒ 设置鼠标经过图标

除了 setIcon() 方法可以为鼠标设置普通状态图标之外，还可以设置按钮的其他状态图标，例如设置鼠标经过按钮时显示的图标。这需要调用按钮的 setRolloverIcon() 方法，其方法声明如下：

```
public void setRolloverIcon(Icon rolloverIcon)
```

参数说明

rolloverIcon: 鼠标经过按钮时显示的图标对象。

☒ 取消鼠标外观

要定义鼠标新的外观就必须取消原有外观的绘制，主要有以下 3 个关键方法：

```
button.setFocusPainted(false); //取消焦点绘制  
button.setBorderPainted(false); //取消边框绘制  
button.setContentAreaFilled(false); //取消内容绘制
```

— 这 3 个方法分别取消按钮的焦点绘制、边框绘制及内容绘制，这样按钮就没有外观和任何效果了，就像窗体取消修饰效果一样。

(3) 改变窗体状态

实例中自定义的最大化、最小化按钮都需要控制窗体的状态，这需要通过 JFrame 类的 setExtendedState() 方法来实现，其方法声明如下：

```
public void setExtendedState(int state)
```

参数说明

state: 该参数是位于 JFrame 类中的窗体状态常量，其可选值如表 14.1 所示。



表 14.1 窗体状态常量说明

枚举值	描述
ICONIFIED	最小化的窗口
NORMAL	默认大小的窗口
MAXIMIZED_HORIZ	水平方向最大化窗口
MAXIMIZED_VERT	垂直方向最大化窗口
MAXIMIZED_BOTH	水平与垂直方向都最大化的窗口



Note

实例 192 禁止改变窗体的大小

(实例位置: 配套资源\SL\14\192)

实例说明

本实例主要实现禁止改变窗体大小的功能。运行本实例, 默认可以通过鼠标拖曳的方式改变窗体大小, 但是当用户单击“禁止改变窗体大小”按钮后, 窗体将会以一种对话框的方式进行显示, 这时就不可以再用鼠标拖曳的方式改变窗体的大小了。实例的运行效果如图 14.6 所示。

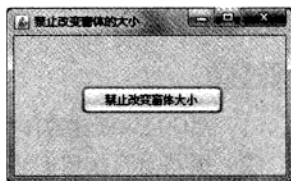


图 14.6 禁止改变窗体的大小

实现过程

(1) 在项目中新建窗体类 LimitChangeFormSize。在该窗体中添加一个按钮控件, 用来执行禁止改变窗体大小功能。

(2) 编写“禁止改变窗体大小”按钮的事件处理方法, 在该方法中设置窗体的 Resizable 属性为 false。关键代码如下:

```
protected do_button_actionPerformed(ActionEvent e) {
    setResizable(false); //禁止改变窗体大小
}void
```

技术要点

本实例在实现禁止改变窗体的大小功能时, 主要是通过将窗体的 Resizable 属性设置为 false 值实现的。下面介绍设置该属性的方法:

```
public void setResizable(boolean resizable)
```

参数说明

resizable: 如果此窗体是可调整大小的, 则为 true; 否则为 false。

实例 193 指定窗体标题栏图标

(实例位置: 配套资源\SL\14\193)

实例说明

窗体的标题栏图标也称为窗体的图标, 当窗体显示时在左上角标题栏位置会显示这个图标



与窗体标题信息,这用于区分不动窗体或标注窗体。本实例通过 Java 语言实现窗体图标 的设置,其效果如图 14.7 所示,单击任意一个按钮,就会把窗体图标更换为按钮的图标。另外,本实例在 Windows 7 系统中开发,程序运行后,除窗体之外,在任务栏也会显示窗体的图标,效果如图 14.8 所示。



Note



图 14.7 设置窗体标题栏图标



图 14.8 Windows 7 任务栏图标

实现过程

- (1) 在项目中新建窗体类 `FrameIcon`。在窗体中设置背景,并添加 4 个更改窗体图标的按钮。
- (2) 编写所有按钮的事件处理方法,在用户单击不同按钮时,该方法可以判断事件源并获取相对图标文件的 URL 路径,然后通过该路径创建图片对象设置窗体的 `iconImage` 属性。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {  
    String resource = "";  
    if (e.getSource() == button1) {  
        resource = "icon1.png";  
    }  
    if (e.getSource() == button2) {  
        resource = "icon2.png";  
    }  
    if (e.getSource() == button3) {  
        resource = "icon3.png";  
    }  
    if (e.getSource() == button4) {  
        resource = "icon4.png";  
    }  
    URL url = getClass().getResource(resource);  
    setIconImage(Toolkit.getDefaultToolkit().getImage(url));  
}
```

//定义图标文件名称变量
//确定用户单击的按钮
//确定按钮对应的图标文件
//获取图标文件路径
//设置窗体的图标

技术要点

本实例主要通过设置 `JFrame` 窗体类的 `iconImage` 属性来实现窗体图标的设定。下面将介绍如何改变该属性的值。

`setIconImage()`方法是 `JFrame` 类提供的,用于设置窗体的标题栏中的图标图片。该方法的声明格式如下:

```
public void setIconImage(Image image)
```

参数说明

image: 要设置为窗体标题栏图标的图片对象。



实例 194 设置闪烁的标题栏

(实例位置: 配套资源\SL\14\194)



Note

实例说明

在大型项目中常出现多个窗口同时处理并显示业务数据的情况,每个窗口和窗口中的数据分类与重要性都不相同,有的窗口用于显示实时信息,必须时刻保持醒目位置,但是有的信息重要性非常高,必须第一时间让用户注意到。本实例就实现窗体标题栏闪烁效果,这将以动态的、明显的方式突出某窗体信息的重要性。实例闪烁过程如图 14.9 所示。

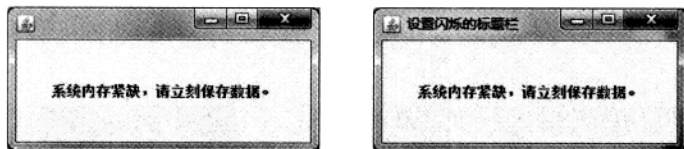


图 14.9 闪烁中的窗体标题栏

实现过程

(1) 在新建窗体类 FlashTitleBar。在窗体中添加标签控件并显示重要信息。

(2) 编写窗体打开的事件处理方法,在该方法中创建 Timer 控件,并在控件内部实现窗体闪烁效果,而且该效果一直循环,每秒闪烁一次。关键代码如下:

```
protected void do_this_windowOpened(WindowEvent e) {
    Timer timer = new Timer(500, new ActionListener() {           //创建 timer 控件
        String title = getTitle();                                //获取窗体标题
        @Override
        public void actionPerformed(ActionEvent e) {              //实现窗体闪烁
            if (getTitle().isEmpty()) {                             //如果标题为空
                setTitle(title);                                     //恢复窗体标题
            } else {
                setTitle("");                                       //如果窗体标题不为空,则清空窗体标题
            }
        }
    });
    timer.start();                                                 //启动 Timer 控件
}
```

技术要点

本实例的关键技术在于 Timer 控件的使用,窗体标题闪烁效果就是依靠该控件不断地产生 ActionEvent 事件,并在事件处理中实现的。

(1) 创建 Timer 对象

本实例所使用的是 Java.swing 包中的 Timer 对象,而非 Java.util 包的。创建这个控件的构造方法如下:

```
public Timer(int delay, ActionListener listener)
```




参数说明

- ☒ delay: 触发事件的时间间隔, 单位为毫秒。
- ☒ listener: 初始事件监听器, 用于获取控件的 action 事件。

(2) 启动 Timer

对象 Timer 控件的 start() 方法将启动 Timer, 使它开始向其侦听器发送动作事件。该方法的声明如下:

```
public void start()
```

实例 195 实现带背景图片的窗体

(实例位置: 配套资源\SL\14\195)

实例说明

开发桌面窗体应用程序时, 界面的美观是程序的一个重要组成部分, 一般的应用程序界面背景都是非常漂亮或者代表实际意义的图片, 那么如何为窗体设置背景图片呢? 本实例将通过 Java 代码重写 JPanel 面板来实现窗体背景图片的设置, 实例的运行效果如图 14.10 所示。

实现过程

(1) 在项目中新建窗体类 SetFormBackImage。在窗体中添加自定义的 BackgroundPanel 面板。

(2) 在窗体类的构造方法中设置窗体标题, 并为添加的 BackgroundPanel 自定义面板设置背景图片。关键代码如下:

```
public SetFormBackImage() {
    setTitle("实现带背景图片的窗体");           //设置窗体标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);               //设置窗体位置
    contentPane = new JPanel();                    //创建内容面板
    setContentPane(contentPane);                   //设置窗体内容面板
    contentPane.setLayout(new BorderLayout(0, 0));
    BackgroundPanel backgroundPanel = new BackgroundPanel(); //创建背景面板
    backgroundPanel.setImage(getToolkit().getImage(
        getClass().getResource("Penguins.jpg"))); //设置面板背景图片
    contentPane.add(backgroundPanel);               //将背景面板添加到窗体内容面板
}
```



图 14.10 设置窗体背景为指定图片

指点迷津:

上面代码中的 Penguins.jpg 图片需要放置在 SetFormBackImage 类同级文件夹中, 在编译 Java 代码时将自动发布一份到 bin 文件夹中, 这个文件夹在 Eclipse 中是隐藏的, 读者只要把文件放置在指定位置即可。



(3) 继承 JPanel 类编写自己的面板。自定义面板类名定义为 BackgroundPanel, 重写 JPanel 类的 paintComponent() 方法, 在该方法中实现绘制面板背景图片的代码。关键代码如下:

```
protected void paintComponent(Graphics g) {           //重写绘制控件外观
    if (image != null) {
        g.drawImage(image, 0, 0, this);               //绘制图片与控件大小相同
    }
    super.paintComponent(g);                           //执行超类方法
}
```

技术要点

本实例在设置窗体的背景图片时, 继承 JPanel 自定义了自己的面板控件, 并重写了面板绘制方法, 为自己绘制了背景图片。面板绘制方法的声明格式如下:

```
protected void paintComponent(Graphics graphics)
```

参数说明

graphics: 控件中的绘图对象。

实例 196 背景为渐变色的主界面

(实例位置: 配套资源\SL\14\196)

实例说明

窗体背景颜色可以通过属性进行设置, 但是通过属性设置的窗体背景颜色都是单一的颜色。在以往程序安装界面中, 背景色都是上下渐变的蓝色背景, 看上去不伤眼睛, 而且没有强烈的疲劳感, 一时成为安装界面的流行背景。本实例实现这个背景颜色渐变窗体的效果, 使窗体更加美观。实例的运行效果如图 14.11 所示。

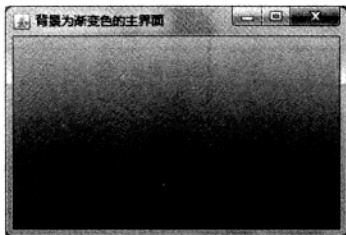


图 14.11 设置渐变的背景色

实现过程

(1) 在项目中新建窗体类 ImageInFormCenter。设置窗体类的标题, 在窗体中添加自定义的渐变背景面板。关键代码如下:

```
public ShadeBackgroundImage() {
    setTitle("背景为渐变色的主界面");           //设置窗体标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();                   //创建内容面板
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    ShadePanel shadePanel = new ShadePanel();     //创建渐变背景面板
    contentPane.add(shadePanel, BorderLayout.CENTER); //添加面板到窗体内容面板
}
```

(2) 继承 JPanel 类编写自己的渐变面板控件, 重写 paintComponent() 方法, 在该方法中创建 GradientPaint 填充类的实例对象, 然后把它设置为当前绘图对象的填充模式, 再使用新的填





充模式绘制一个与控件相同大小的矩形。关键代码如下：

```
protected void paintComponent(Graphics g1) { //重写绘制控件外观
    Graphics2D g = (Graphics2D) g1;
    super.paintComponent(g); //执行超类方法
    int width = getWidth(); //获取控件大小
    int height = getHeight();
    //创建填充模式对象
    GradientPaint paint = new GradientPaint(0, 0, Color.CYAN, 0, height,
        Color.MAGENTA);
    g.setPaint(paint); //设置绘图对象的填充模式
    g.fillRect(0, 0, width, height); //绘制矩形填充控件界面
}
```

技术要点

本实例在实现背景色渐变时涉及设置渐变填充方式和绘制矩形两个关键技术。

(1) 设置渐变填充模式

☒ 创建渐变填充模式对象

设置填充模式首先要创建填充模式对象，本实例要实现渐变效果，所以创建的是 `GradientPaint` 类的实例对象，创建该对象的构造方法的参数包括填充起点的坐标与颜色、填充终点的坐标与颜色。其方法声明如下：

```
public GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2)
```

该方法中的参数说明如表 14.2 所示。

表 14.2 参数说明

参 数 名	说 明
x1	起始位置的 X 坐标
y1	起始位置的 Y 坐标
color1	起始渐变点的颜色
x2	终止位置的 X 坐标
y2	终止位置的 Y 坐标
color2	终止渐变点的颜色

☒ 设置绘图对象填充模式

创建渐变填充模式对象以后需要设置为 `Graphics2D` 绘图上下文对象的填充属性，然后由此绘图对象绘制的所有图形，都使用这个新的填充模式。设置绘图上下文填充模式的方法的声明如下：

```
public abstract void setPaint(Paint paint)
```

参数说明

paint: 填充模式对象。

(2) 绘制矩形图形

设置绘图上下文对象使用渐变填充模式以后，还要以自定义控件相同的大小来绘制控件界面，绘制内容是一个矩形图形，这样可以均匀地遮盖整个控件界面。绘制矩形图形的方法的声明如下：

```
public abstract void fillRect(int x, int y, int width, int height)
```



该方法中的参数说明如表 14.3 所示。

表 14.3 参数说明

参 数 名	说 明
x	绘制矩形的起始位置的 X 坐标
y	绘制矩形的起始位置的 Y 坐标
width	指定绘制矩形的宽度
height	指定绘制矩形的高度



Note

实例 197 随机更换窗体背景

(实例位置: 配套资源\SL\14\197)

实例说明

在一些管理软件中实现随机更换窗体背景可以增加软件的人性化程度, 使用户心情愉悦。本实例将实现随机更换窗体的功能, 每次窗体恢复显示时窗体背景图片都会随机更换。实例的运行效果如图 14.12 所示。



图 14.12 随机更换窗体背景

实现过程

(1) 在项目中新建窗体类 RandomBackgroundImage。在窗体中添加自定义的支持背景的面板控件。

(2) 编写初始化背景图片数组的 initPhotoArray() 方法, 在该方法中创建图片数组, 然后通过 for 循环初始化数组中的元素, 每个元素赋值为一个图片对象。关键代码如下:

```
private void initPhotoArray() {
    images = new Image[6]; //初始化背景图片数组
```



Note

```
String photoPath = "";
for (int i = 0; i < images.length; i++) {           //遍历数组并初始化所有元素
    photoPath = "/com/img/photo" + (i + 1) + ".jpg"; //生成文件名
    images[i] = getToolkit()
        .getImage(getClass().getResource(photoPath)); //初始化数组元素
}
}
```

(3) 编写窗体激活事件的处理方法, 当窗体刚刚被显示, 或者从最小化恢复到显示状态时, 都会触发这个窗体激活事件。在事件处理方法中, 首先生成随机数, 然后用这个随机数作为图片数组的下标索引获取一个图片设置给支持背景的面板控件, 最后重新绘制窗体界面。关键代码如下:

```
protected void do_this_windowActivated(WindowEvent arg0) {
    Random random = new Random();           //创建随机数对象
    int num = random.nextInt(6);           //生成随机数
    panel.setImage(images[num]);           //设置面板背景图片
    repaint();                             //重绘窗体界面
}
```

技术要点

Java util 包中的 Random 类提供了常用的伪随机数生成方法, 类型包括 boolean、int、long、double 等。下面介绍本实例如何使用该类的对象生成整型随机数字。

(1) 创建随机数对象

Random 的对象可以生成各种类型的随机数字, 但在此之前必须先创建它。本实例使用了默认构造方法直接传递的实例对象, 这非常简单无须介绍, 所以这里介绍一个接收参数的构造方法。其方法声明如下:

```
public Random(long seed)
```

参数说明

seed: 创建随机数对象的种子。

(2) 生成指定范围的随机整数

Random 的对象可以调用多个方法来生成不同数据类型的随机数, 本实例需要随机指定数组下标索引来获取数组中的某个背景图片对象, 而数组下标只能是整数, 并且不能超出数组范围, 所以介绍一下如何生成指定范围的随机整数。方法声明如下:

```
public int nextInt(int num)
```

参数说明

num: 要返回的随机数的范围, 必须为正数。

多学两招:

Java 开发的桌面程序需要很多本机资源, 如美化界面的图片、相应动作的音频等。这些资源在程序开发时涉及资源路径的确定问题, 如果使用绝对路径当然没问题, 但是会把资源固定在一个磁盘上, 影响到软件的发布与传播。所以使用相对路径才是首选, Java 中任何类都有 Class 实例, 通过这个实例的 getResource() 方法便可以获取指定资源的 URL 对象, 从而准确地定义资源文件位置。



实例 198 椭圆形窗体界面

(实例位置: 配套资源\SL\14\198)



Note

实例说明

个性的窗体形状可以增加程序的趣味性, 可以使程序更具吸引力。见惯了方方正正的矩形窗体, 椭圆形的窗体更会使用户眼前一亮。本实例设计一个椭圆形的窗体, 运行程序, 窗体为椭圆形, 如图 14.13 所示。单击窗体, 即可退出程序。



图 14.13 椭圆形窗体

实现过程

(1) 在项目中创建窗体类 `EllipseFrame`, 去掉窗体修饰。关键代码如下:

```
public EllipseFrame() {
    setUndecorated(true);           //去掉窗体修饰
    //省略其他代码
}
```

(2) 编写窗体打开时的事件处理方法, 在该方法中创建椭圆对象, 并把这个椭圆设置为窗体的形状。关键代码如下:

```
protected void do_this_windowOpened(WindowEvent e) {
    //创建椭圆对象
    Ellipse2D.Float ellipse = new Ellipse2D.Float(0f, 10f, 400f, 130f);
    AWTUtilities.setWindowShape(this, ellipse); //设置窗体椭圆形状
}
```

(3) 编写窗体的鼠标单击事件处理方法, 在该方法中销毁窗体对象, 由于这是程序唯一的一个窗体与线程, 所以销毁窗体后, 程序会自动退出。关键代码如下:

```
protected void do_this_mouseClicked(MouseEvent e) {
    dispose(); //销毁窗体
}
```

技术要点

本实例的关键技术在于椭圆类 `Ellipse2D` 的应用与设置窗体形状的 API。

(1) 创建椭圆对象

Java 的椭圆对象由 `Ellipse2D` 类定义, 该类是一个抽象类不能实例化, 但是在其内部包含了两个静态内部实现类, 分别为 `Double` 与 `Float`, 它们接收 `double` 与 `float` 类型的参数定义椭圆大小, 本实例使用的是 `Float` 实现类。下面是该类构造方法的声明:

```
public Ellipse2D.Float(float x, float y, float w, float h)
```

该方法中的参数说明如表 14.4 所示。



表 14.4 参数说明

参 数 名	说 明
x	椭圆对应矩形左上角的 X 坐标
y	椭圆对应矩形左上角的 Y 坐标
w	椭圆对应矩形的宽度
h	椭圆对应矩形的高度

(2) 设置窗体形状

在 JDK 1.6 中提供了设置窗体形状的 API，通过这个 API 可以设置窗体为指定图形。该方法的声明如下：

```
public static void setWindowShape(Window window, Shape shape)
```

参数说明

- ☑ window：窗体对象。
- ☑ shape：图形接口的实现。

多学两招：

有些 API 在 Eclipse 开发工具中是被限制的，例如本实例中设置窗体形状的 API 在 Eclipse 中就无法编译，并报错。这需要设置 Eclipse 编译器信息来解除这个限制，具体步骤为：选择“窗口”/“首选项”命令，在弹出的“首选项”对话框中展开 Java/“编译器”/“错误/警告”节点，然后在对话框右侧展开“建议不要使用和限制使用的 API”节点，将其所有下拉列表框中的“错误”修改为“警告”或者“忽略”。

实例 199 钻石形窗体

(实例位置：配套资源\SL\14\199)

实例说明

设置个性形状的窗体可以增加程序的趣味性，例如定义奇特形状的登录窗体，可以让用户在进入系统之前对程序有好奇并急于了解的情绪。本实例实现自定义钻石图形的窗体，运行程序后显示窗体界面。实例的运行效果如图 14.14 所示。

实现过程

(1) 在项目中新建窗体类 DiamondFrame，去掉窗体修饰。关键代码如下：

```
public DiamondFrame() {  
    setUndecorated(true);  
    //省略其他代码  
}
```

//去掉窗体修饰



图 14.14 钻石形窗体

(2) 编写窗体打开时的事件处理方法，在该方法中创建多边形组成的钻石图形对象，并把这个图形设置为窗体的形状。关键代码如下：



```
protected void do_this_windowOpened(WindowEvent e) {
    int[] xPoints={0,50,350,400,200,0};
    int[] yPoints={200,100,100,200,400,200};
    Polygon polygon=new Polygon(xPoints,yPoints,6);
    AWTUtilities.setWindowShape(this, polygon);
}
```

//定义各顶点的 X 坐标
//定义各顶点的 Y 坐标
//创建多边形
//设置窗体形状

技术要点

Java 创建钻石图形需要依靠 Polygon 类创建多边形来实现，所以 Polygon 类是本实例的关键技术。多边形可以实现任意图形，只要有合理的顶点位置。下面来看一下如何通过 Polygon 类创建多边形实现钻石图形。

Polygon 类有一个接收坐标点参数的构造方法，通过这个构造方法可以直接创建想要的图形。其方法声明如下：

```
public Polygon(int[] xpoints, int[] ypoints, int npoints)
```

参数说明

- ☒ xpoints: 顶点 X 坐标的数组。
- ☒ ypoints: 顶点 Y 坐标的数组。
- ☒ npoints: 多边形中顶点的数量。

实例 200 创建透明窗体

(实例位置: 配套资源\SL\14\200)

实例说明

有些实时信息、事物提醒和各类助手程序要保持窗体置顶状态，即始终显示在所有窗体之上，这样可以保持信息的实时显示、提高助手类程序操作的方便性等。但是程序保持置顶就会遮盖窗体下方的其他窗口或者桌面上的图标，被遮盖的位置也许只包含部分信息，如 Eclipse 的代码编辑窗口，如果为程序提供窗体透明功能，窗体就不会成为屏幕上的补丁似的障碍物，反而会更受欢迎。本实例通过 Java 技术实现窗体透明效果，并可以控制透明度。实例的运行效果如图 14.15 所示，透过窗体可以看见底部的 Eclipse 代码编辑器中的代码。

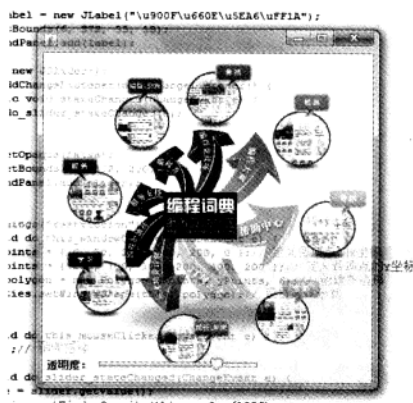


图 14.15 透明窗体

实现过程

(1) 在项目中创建窗体类 TransparencyFrame。设置窗体置顶，在窗体中添加一个滑块控件。

(2) 编写滑块控件的事件处理方法，在改变滑块值时通过该方法获取滑块当前值，并把它转换为百分比来改变窗体的透明度。关键代码如下：

```
protected void do_slider_stateChanged(ChangeEvent e) {
    int value = slider.getValue();
    //获取滑块当前值
```





```
AWTUtilities.setWindowOpacity(this, value/100f);
```

```
//使用滑块值改变窗体透明度
```

技术要点

实例的关键技术是使用了 AWTUtilities 类的 setWindowOpacity() 方法, 用于设置窗体的透明度。该方法的声明如下:

```
public static void setWindowOpacity (Window window, float alpha)
```

参数说明

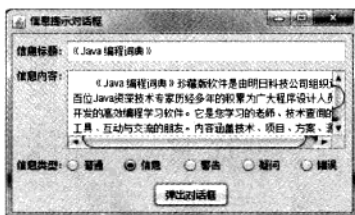
- ☒ window: 窗体对象。
- ☒ alpha: 窗体透明度, 取值范围是在 0~1 之间的小数。

实例 201 信息提示对话框

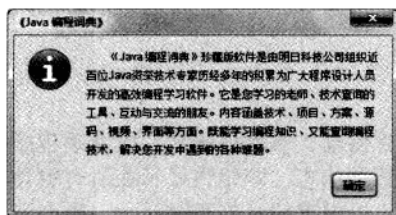
(实例位置: 配套资源\SL\14\201)

实例说明

信息提示对话框是程序开发中经常用到的功能, 它可以为用户显示警告、错误、提示等信息内容, 也可以根据不同的信息级别选择不同类型的对话框。本实例实现了每个类型对话框的信息提示效果, 实例的运行效果如图 14.16 (a) 所示, 显示一个信息提示对话框的效果如图 14.16 (b) 所示。



(a) 运行界面



(b) 信息提示对话框

图 14.16 实例运行效果

实现过程

(1) 在项目中创建窗体类 MessageDialog。在窗体中添加接收对话框标题和内容文本的文本框和文本域控件, 添加选择信息类别的 5 个单选按钮和 1 个“弹出对话框”按钮。

(2) 编写“弹出对话框”按钮的事件处理方法, 在该方法中接收用户输入的对话框标题与内容文本, 根据用户选择的对话框类型来显示对话框。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String title = titleField.getText();           //获取标题文本
    String message = messageArea.getText();       //获取内容文本
    String command = bg.getSelection().getActionCommand(); //获取选中的单选按钮
    int messageType = JOptionPane.INFORMATION_MESSAGE; //创建信息类型
    if (command.equals("普通"))                   //根据用户选择确定对话框类型
        messageType = JOptionPane.PLAIN_MESSAGE;
    if (command.equals("疑问"))
        messageType = JOptionPane.QUESTION_MESSAGE;
```



```

if (command.equals("警告"))
    messageType = JOptionPane.WARNING_MESSAGE;
if (command.equals("错误"))
    messageType = JOptionPane.ERROR_MESSAGE;
//显示对话框
JOptionPane.showMessageDialog(this, message, title, messageType);
}

```



Note

技术要点

Java 语言中提供了 `JDialog` 类实现对话框效果, 程序开发人员可以继承该类编写自定义的对话框。但是这相对比较复杂, 所以 Java 提供了一个工具类 `JOptionPane` 来负责常见对话框的创建与使用。本实例调用了 `JOptionPane` 类的静态方法 `showMessageDialog()` 来显示信息提示对话框, 该方法有多种重载格式, 下面介绍本实例使用的重载格式的方法声明。

```

public static void showMessageDialog(Component parentComponent, Object message,
    String title, int messageType)
    throws HeadlessException

```

该方法中的参数说明如表 14.5 所示。

表 14.5 参数说明

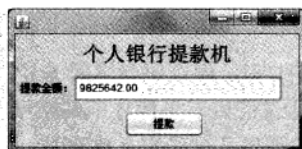
参 数 名	说 明
parentComponent	对话框的父窗体
message	对话框显示的信息字符串
title	对话框的标题字符串
messageType	对话框类型, 这个类型值确定对话框的信息图标样式

实例 202 设置信息提示对话框的图标

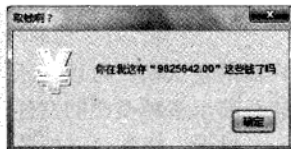
(实例位置: 配套资源\SL\14\202)

实例说明

信息提示对话框可以作为向用户说明程序当前情况与问题的工具。虽然 `JOptionPane` 类已经提供了多种类型的信息提示, 但是这些类型仅限于普通、信息、警告、疑问、错误等。在实际项目应用中, 涉及的信息提示类型会很多, 而且需要有针对性。本实例实现自定义对话框提示图标, 以满足不同程序的不同类型的信息提示需求。程序主界面如图 14.17 (a) 所示, 当单击“提款”按钮时将弹出带有自定义图标的信息提示对话框, 如图 14.17 (b) 所示。



(a) 程序主界面



(b) 信息提示对话框

图 14.17 实例运行效果



实现过程

(1) 在项目中创建窗体类 `MessageDialogIcon`。在窗体中添加文本框和“提款”按钮。

(2) 编写“提款”按钮的事件处理方法，在该方法中获取用户输入，并将输入作为信息提示内容的一部分，然后获取图片资源文件的路径，并用该路径创建图标对象，最后显示对话框。关键代码如下：



Note

```
protected void do_button_actionPerformed(ActionEvent arg0) {  
    String text = textField.getText();  
    URL resource = getClass().getResource("money.png");  
    ImageIcon icon = new ImageIcon(resource);  
    //显示带有自定义图标的信息提示对话框  
    JOptionPane.showMessageDialog(this, "你在我这存 " + text + " 这些钱了吗", "取钱啊?",  
        JOptionPane.QUESTION_MESSAGE, icon);  
}
```

技术要点

本实例也使用到了 `JOptionPane` 类的 `showMessageDialog()` 方法，但是使用的是该方法的最终重载格式，也就是参数最多的，并且支持自定义对话框图标的方法。下面介绍本实例使用的重载格式的方法声明：

```
public static void showMessageDialog(Component parentComponent, Object message,  
    String title, int messageType, Icon icon)  
    throws HeadlessException
```

该方法中的参数说明如表 14.6 示。

表 14.6 参数说明

参 数 名	说 明
parentComponent	对话框的父窗体
message	对话框显示的信息字符串
title	对话框的标题字符串
messageType	对话框类型，这个类型值确定对话框的信息图标样式
icon	对话框的图标对象

实例 203 指定打开对话框的文件类型

(实例位置：配套资源\SL\14\203)

实例说明

Java 中文件选择器的默认格式可以显示文件的打开、保存以及各种自定义的文件选择对话框，而且对话框默认是显示多种类型的文件，如果浏览某个文件夹中的必要类型的文件过多，用户就不得不从中一一挑选，找到自己想要的文件类型，再确定文件名称，这样会给用户带来不便。本实例将介绍如何给文件选择器添加一个过滤器，使文件选择对话框只显示需要的文件类型。实例的运行效果如图 14.18 所示，当单击“打开图片文件”按钮时，将弹出只能选择 JPG、PNG、GIF 格式的 3 种图片文件，如图 14.19 所示。

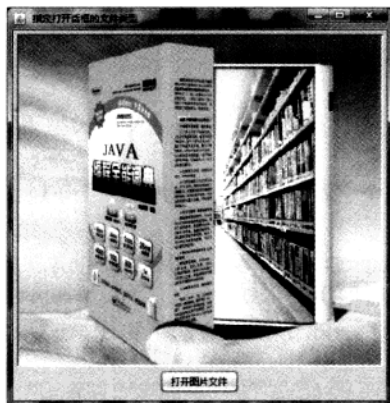


图 14.18 程序主窗体



图 14.19 文件选择对话框



Note

实现过程

(1) 在项目中创建窗体类 CustomSelectFileType。在窗体中添加背景面板与“打开图片文件”按钮。

(2) 编写“打开图片文件”按钮的事件处理方法，在该方法中创建文件选择器，创建 FileNameExtensionFilter 过滤器的实例对象，这个过滤器只接收.JPG、.GIF 和.PNG 类型的图片，然后调用文件选择器的方法显示文件打开对话框，并获取用户选择的文件，最后把图片文件内容显示在界面中。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser();           //创建文件选择器
    FileNameExtensionFilter filter = new FileNameExtensionFilter("图片文件",
        "jpg", "gif", "png", "jpeg");                 //创建文件类型过滤器
    chooser.setFileFilter(filter);                      //设置选择器的过滤器
    int option = chooser.showOpenDialog(this);          //显示“打开”对话框
    if(option==JFileChooser.APPROVE_OPTION){
        File file = chooser.getSelectedFile();          //获取用户选择文件
        try {
            //加载图片文件
            ImageIcon image=new ImageIcon(file.toURI().toURL());
            backgroundPanel.setImage(image.getImage()); //显示图片文件
        } catch (MalformedURLException e1) {
            e1.printStackTrace();
        }
    }
}
```

技术要点

本实例为文件选择器设置了一个文件类型过滤器，这个过滤器是 FileFilter 抽象类的一个子类 FileNameExtensionFilter。下面介绍本实例中如何创建这个过滤器以及如何为文件选择器设置这个过滤器。

(1) 创建文件类型过滤器

FileNameExtensionFilter 类的构造方法中可以指定文件类型的描述与支持的图片扩展名称，



其中扩展名称可以是多个。下面是该类的构造方法的声明：

```
public FileNameExtensionFilter(String description, String extensions)
```

参数说明

- ☒ description: 文件类型的描述信息, 它将显示在文件选择对话框的文件类型下拉列表中。
- ☒ extensions: 文件支持的扩展名称, 可以是多个字符串参数, 并使用逗号分割。

(2) 设置文件选择器的过滤器

JFileChooser 文件选择器可以设置文件过滤器, 这个过滤器可以是 FileFilter 抽象类的各种实现类。本实例使用了一个文件类型过滤器, 这是 Java 提供的实现类。

设置文件选择器的过滤器要通过 JFileChooser 类的 setFileFilter() 方法来实现, 该方法声明如下:

```
public void setFileFilter(FileFilter filter)
```

参数说明

filter: FileFilter 抽象类的各种实现类的对象。

实例 204 为保存对话框设置默认文件名

(实例位置: 配套资源\SL\14\204)

实例说明

文件保存对话框常用于各类编辑器, 在一些编辑软件中常进行电子文档的编写, 要把这些文档保存下来就要使用文件。随着计算机的普及, 计算机编程的日益成熟, 人们对应用程序的要求也会越来越高, 不可能要求用户自己去查询文件夹路径, 再根据这个路径和要保存的文件名定义出资源文件的字符串。这在早期的 DOS 操作系统时代是有可能的, 但目前 Windows 操作系统以及 GUI 应用程序能够为用户提供更遍历的操作。文件保存对话框可以让用户在窗体中浏览文件夹的同时就确认保存文件的路径。而本实例将在程序中动态指定保存的文件名称, 这样用户输入文件名的步骤都可以省略。运行实例, 选择“文件”/“新建”命令, 将弹出输入新建文档名称的对话框, 如图 14.20 所示。输入文档名称单击“确定”按钮, 对文档内容进行编辑后, 单击“保存”按钮, 将弹出如图 14.21 所示的“保存”对话框, 在该对话框中已经指定了文件名称, 用户可以修改或直接使用这个名称。



图 14.20 输入新建文档名称的对话框

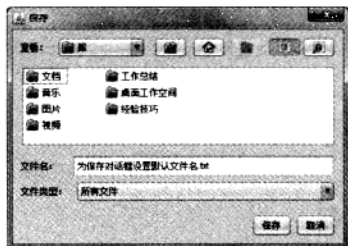


图 14.21 “保存”对话框

实现过程

(1) 在项目中创建窗体类 CustomNameSave。在窗体中添加菜单栏、文本域控件与“保存”按钮。



(2) 编写“新建”菜单项的事件处理方法,在该方法中接收用户输入的新文档名称,然后显示在标签控件中,并激活文本域为可用状态。关键代码如下:

```
protected void do_menuItem_actionPerformed(ActionEvent e) {
    //接收用户输入
    String string = JOptionPane.showInputDialog("请输入新建文档名称");
    if (string==null)
        return;
    label.setText(string);           //用标签控件显示用户输入的文档名称
    textArea.setEnabled(true);     //激活文本域控件
}
```



Note

(3) 编写“保存”按钮的事件处理方法,在该方法中创建文件选择器,然后把用户输入的文档名称创建成文件对象,并设置为文件选择器的 selectedFile 属性,最后显示文件保存对话框。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = label.getText();           //获取标签控件保存的文档名称
    JFileChooser chooser = new JFileChooser(); //创建文件选择器
    File file = new File(text + ".txt");     //用文档名称创建文件对象
    chooser.setSelectedFile(file);           //设置文件选择器的选择文件
    chooser.showSaveDialog(this);           //显示“保存”对话框
    File selectedFile = chooser.getSelectedFile();
    JOptionPane.showMessageDialog(this, "文件保存路径: \n"+selectedFile);
}
```

技术要点

本实例使用了 JFileChooser 类的 getSelectedFile()方法来打开文件保存对话框,但是本实例在“保存”对话框中已经指定了文件名称,这是通过设置文件选择器的 selectedFile 属性来实现的。设置该属性的方法声明如下:

```
public void setSelectedFile(File file)
```

参数说明

file: 指定选中文件。

多学两招:

虽然本实例在打开文件保存对话框之前就创建了文件对象,并设置为文件选择器的属性,但还是要从文件选择器中通过 getSelectedFile()方法重新获取用户选择文件,因为就算用户采用默认文件名直接执行保存,文件选择器也会重新设置文件的路径。

实例 205 支持图片预览的文件选择对话框

(实例位置: 配套资源\SL\14\205)

实例说明

Swing 的普通文件选择对话框在对文件进行选择时,没有给出相应文件的预览效果,这样用户进行操作时会产生一些不便。本实例将实现创建支持图片预览的文件选择对话框,其运行效果如图 14.22 所示。



Note

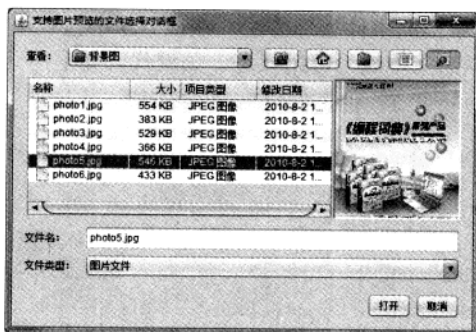


图 14.22 支持图片预览的文件选择对话框

实现过程

(1) 在项目中创建窗体类 PicPreviewFileSelectDialog。把文件选择器作为窗体的控件进行添加，并设置过滤器与图片预览控件。关键代码如下：

```
JFileChooser fileChooser = new JFileChooser();           //创建文件选择器
contentPane.add(fileChooser, BorderLayout.CENTER);      //添加到窗体
paint = new PaintPanel();                               //创建图片预览面板
paint.setBorder(new BevelBorder(BevelBorder.LOWERED, null, null, null, null)); //设置面板的边框
paint.setPreferredSize(new Dimension(150, 300));      //设置预览面板的大小
fileChooser.setAccessory(paint);                        //把面板设置为文件选择器控件
fileChooser.addPropertyChangeListener(new PropertyChangeListener() { //添加选择器的属性事件监听器
    public void propertyChange(PropertyChangeEvent arg0) {
        do_this_propertyChange(arg0);
    }
});
```

fileChooser.setFileFilter(new FileNameExtensionFilter("图片文件", "jpg", "png", "gif")); //设置文件选择器的过滤器

(2) 编写文件选择器的属性改变事件处理方法，当改变选定文件时，这个方法会把图片文件加载到程序中，并设置到图片预览面板的属性进行显示。关键代码如下：

```
protected void do_this_propertyChange(PropertyChangeEvent e) {
    //处理改变选定文件的属性事件处理
    if (JFileChooser.SELECTED_FILE_CHANGED_PROPERTY == e.getPropertyName()) {
        File picfile = (File) e.getNewValue();           //获取选定的文件
        if (picfile != null && picfile.isFile()) {
            try {
                //从文件加载图片
                Image image = getToolkit().getImage(picfile.toURI().toURL());
                paint.setImage(image);                    //设置预览面板的图片
                paint.repaint();                          //刷新预览面板的界面
            } catch (MalformedURLException e1) {
                e1.printStackTrace();
            }
        }
    }
}
```



(3) 继承 JPanel 编写图片预览面板 PaintPanel 类, 重写 paintComponent() 方法, 在该方法中把图片对象绘制到面板上。关键代码如下:

```
protected void paintComponent(Graphics g) {           //重写绘制控件外观
    if (image != null) {
        g.drawImage(image, 0, 0, getWidth(), getHeight(), this); //绘制图片与控件大小相同
    }
    super.paintComponent(g);                             //执行超类方法
}
```



Note

技术要点

本实例的关键技术在于设置文件选择器的 Accessory 控件属性, 这个属性可以把一个控件作为文件选择器的辅助控件, 本实例利用这个辅助控件显示了当前被选中图片文件的预览。设置 Accessory 控件属性的方法声明如下:

```
public void setAccessory(JComponent newAccessory)
```

参数说明

newAccessory: 作为文件选择器的辅助控件。

实例 206 颜色选择对话框

(实例位置: 配套资源\SL\14\206)

实例说明

颜色也是系统资源之一, 它和文件同样重要。在设置颜色值时, 不像文件路径可以通过字符串来表示, 颜色值大多使用对话框进行选择, 总的来说, 颜色选择对话框就是让用户通过视觉来确定颜色值, 而不是通过文本来确定。本实例实现了颜色选择框的应用, 其运行效果如图 14.23 所示。当单击任意一个“选择”按钮时都会弹出“选择颜色”对话框, 如图 14.24 所示。

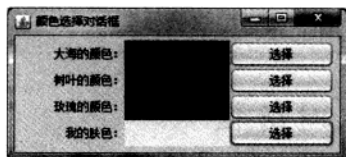


图 14.23 运行效果

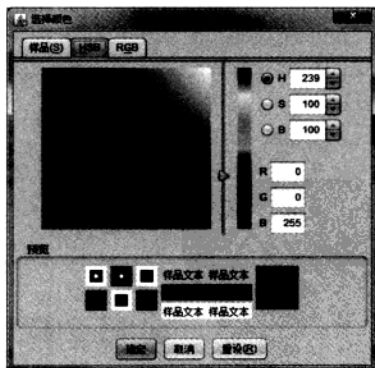


图 14.24 “选择颜色”对话框

实现过程

(1) 在项目中创建窗体类 ColorChooser。在窗体中创建多个标签控件与多个“选择”按钮控件。



(2) 编写“选择”按钮的事件处理方法，在方法体中调用 `setColor()` 方法为窗体上的标签指定背景颜色值。关键代码如下：

```
protected void do_button1_actionPerformed(ActionEvent e) {  
    setColor(label1); //指定标签的颜色设置  
}
```



Note

指点迷津：

由于多个“选择”按钮的事件处理方法相同，所以本实例以其中一个“选择”按钮的事件处理方法为例进行介绍。

(3) 编写 `setColor()` 方法，在该方法体中，首先获取标签控件原来的颜色，然后用这个颜色作为默认值打开颜色选择对话框，最后把用户在对话框中选择的颜色值设置为标签控件的背景色。关键代码如下：

```
private void setColor(JLabel label) {  
    Color color = label.getBackground(); //获取原来的颜色对象  
    //显示“选择颜色”对话框  
    Color newColor = JColorChooser.showDialog(this, "选择颜色", color);  
    label.setBackground(newColor); //把获取的颜色设置为标签的背景色  
}
```

技术要点

本实例的关键技术是使用到了 `JColorChooser` 类的 `showDialog()` 方法，用于显示“选择颜色”对话框，并返回用户选择的颜色值对象。该方法的声明如下：

```
public static Color showDialog(Component component, String title, Color initialColor)  
    throws HeadlessException
```

参数说明

- ☒ `component`：对话框的父级（上级）控件或窗体。
- ☒ `title`：对话框的标题。
- ☒ `initialColor`：对话框初始颜色对象。

多学两招：

Java 在 `Color` 类中定义了很多颜色常量，这些常量都是 `Color` 类的对象，每个对象代表一个常用的颜色值，但是可以使用常量名称标识的颜色有限，那些无法用语言和名称表达的颜色值必须使用更灵活的方式来指定。所以在程序开发中，应该优先使用“选择颜色”对话框为用户提供最大的颜色选择空间。

实例 207 信息输入对话框

（实例位置：配套资源\SL\14\207）

实例说明

GUI 应用程序是与客户交互最灵活的一种应用程序，普通程序窗体中的控件能够与用户完成大部分交互，在特殊情况下还可以弹出各类对话框与用户进行信息交互。本实例利用输入对



话框,接收用户输入要添加联系人的姓名。实例的运行效果如图 14.25 所示。利用对话框将极大地扩展程序交互能力。

实现过程

(1) 在项目中创建窗体类 InfoInputDialog, 在窗体中添加列表控件与文本域控件和“添加”、“删除”两个按钮。

(2) 编写“添加”按钮的事件处理方法。该方法将创建信息输入对话框,用于接收用户输入的姓名,并且对话框设置了默认值为“经理”。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    //显示输入对话框
    String name = JOptionPane.showInputDialog("请输入要添加联系人的姓名: ", "经理");
    DefaultListModel model = (DefaultListModel) list.getModel(); //获取 JList 控件模型
    model.addElement(name); //向模型添加新输入内容
}
```

(3) 编写“删除”按钮的事件处理方法。该方法会根据列表控件的选择来删除对应的选项。关键代码如下:

```
protected void do_button_1_actionPerformed(ActionEvent e) {
    int index = list.getSelectedIndex(); //获取列表控件的选择项索引
    DefaultListModel model = (DefaultListModel) list.getModel(); //获取列表的数据模型
    model.removeElementAt(index); //从模型中删除该索引指定的选项
}
```

技术要点

本实例中使用到了 JOptionPane 类的静态方法实现输入对话框的创建与显示,该方法声明如下:

```
public static String showInputDialog(Object message, Object initialSelectionValue)
```

参数说明:

- ☑ message: 在对话框中的提示信息。
- ☑ initialSelectionValue: 对话框的初始值。

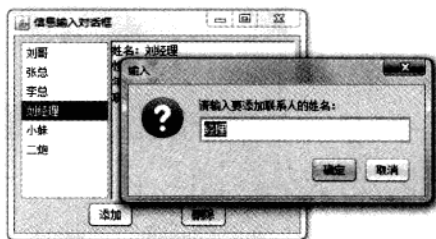


图 14.25 信息输入对话框

实例 208 定制信息对话框

(实例位置: 配套资源\SL\14\208)

实例说明

在程序设计中经常需要使用对话框显示提示信息。Java 内置了信息、错误、警告、询问和普通类型的对话框,这些对话框可以满足大部分程序开发需求,但是对于特殊场景下的程序应用,需要在对话框中显示特定的信息以及特定的按钮,这就需要定制信息对话框的外观。本实例将介绍 JOptionPane 类的另一个更加灵活的定制对话框的使用。实例的运行效果如图 14.26 所示。

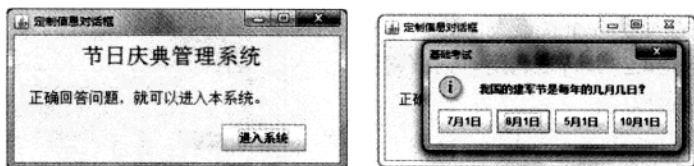


图 14.26 定制信息对话框

实现过程

(1) 在项目中创建窗体类 CustomDialog，在窗体中使用标签控件显示问题文本，并添加一个“进入系统”按钮。

(2) 编写“进入系统”按钮的事件处理方法，该方法将完善自定义对话框所需要的所有参数，并通过 JOptionPane 类的静态方法显示自定义信息的对话框。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent arg0) {  
    //对话框操作项的名称  
    String[] options = new String[] { "7月1日", "8月1日", "5月1日", "10月1日" };  
    String message = "我国的建军节是每年的几月几日? "; //对话框中的信息  
    int num = JOptionPane.showOptionDialog(this, message, "基础考试",  
        JOptionPane.YES_NO_OPTION, JOptionPane.INFORMATION_MESSAGE,  
        null, options, "8月1日"); //显示自定义对话框  
    if (options[num].equals("8月1日")) {  
        JOptionPane.showMessageDialog(this, "恭喜您回答正确。"); //回答正确的提示  
    } else {  
        JOptionPane.showMessageDialog(this, "回答错误，再见。"); //回答错误的提示  
    }  
}
```

技术要点

本实例的关键技术在于对 JOptionPane 类的静态方法 showOptionDialog() 的应用，这个方法通过参数就可以定义对话框的信息、标题、图标、操作项和初始值等对话框属性。该方法的声明如下：

```
public static int showOptionDialog(Component parentComponent, Object message, String title, int  
    optionType, int messageType, Icon icon, Object[] options, Object initialValue)  
    throws HeadlessException
```

该方法的参数说明如表 14.7 所示。

表 14.7 参数说明

参 数 名	说 明
parentComponent	对话框的父窗体
message	对话框中显示的信息文本
title	对话框的标题
optionType	对话框中的信息类型，这决定了对话框中的图标，如果 icon 参数不设置 null 值的话
icon	对话框的图标
options	对话框中操作按钮的名称，该数组决定了对话框中按钮的数量
initialValue	对话框的初始值，与初始值同名的按钮将处于焦点状态



多学两招:

本实例利用 JOptionPane 类的静态方法实现自定义对话框,对话框界面可以有多个操作项,也就是按钮,这些按钮是根据字符串数组参数来遍历生成的,在单击任意一个按钮之后,该按钮对应的字符串数组参数的下标索引将作为对话框的返回值,用这个返回值在字符串数组中提取数组元素,就知道用户单击的是哪个操作按钮了。



Note

实例 209 拦截事件的玻璃窗格

(实例位置: 配套资源\SL\14\209)

实例说明

在软件进行比较耗时的操作时,可以使用玻璃窗格将软件界面暂时锁定,即拦截所有用户的输入事件。本实例模拟一个下载工具,当用户选择一个文件进行下载时,将暂时锁定界面并提示“正在下载”,此时用户无法再选择别的文件或者单击按钮。实例的运行效果如图 14.27 所示。

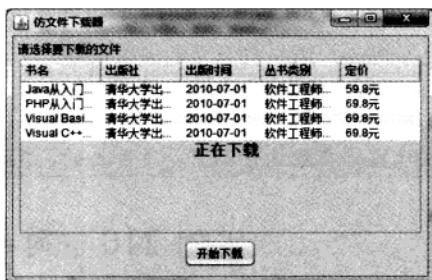


图 14.27 拦截事件的玻璃窗格

实现过程

(1) 编写 GlassPane 类,它继承了 JComponent 类,因此其实例可以作为玻璃窗格。在其构造方法中,首先屏蔽了鼠标事件、键盘事件等。在 paintComponent()方法中简单地在控件上绘制了一个红色字符串。关键代码如下:

```
public class GlassPane extends JComponent {
    private static final long serialVersionUID = 9060636159598343142L;
    public GlassPane() {
        addMouseListener(new MouseAdapter() {           //屏蔽鼠标事件
        });
        addMouseMotionListener(new MouseMotionAdapter() { //屏蔽鼠标拖曳事件
        });
        addKeyListener(new KeyAdapter() {               //屏蔽键盘事件
        });
        setFont(new Font("Default", Font.BOLD, 16));   //设置控件的字体
    }
    @Override
    protected void paintComponent(Graphics g) {
        g.setColor(Color.RED);                          //将画笔换成红色
        g.drawString("正在下载", 190, 130); //在坐标 (190, 130) 处绘制字符串“正在下载”
    }
}
```

(2) 编写 DownloadSoft 类,该类继承了 JFrame。在框架中,主要包括一个表格和一个“开始下载”按钮。表格用来模拟可以下载的资源。编写 do_button_actionPerformed()方法来监听单



击按钮的事件，用来显示玻璃窗格。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {  
    getGlassPane().setVisible(true);  
}  
//显示玻璃窗格
```



Note

指点迷津：

玻璃窗格只能屏蔽用户的输入操作，但是框架是可变的。

技术要点

如果想锁定窗体，则可以使用玻璃窗格。通常根据需求自定义玻璃窗格的对象，然后再将其设置为框架的玻璃窗格，可以使用 `setGlassPane()` 方法来实现。该方法的声明如下：

```
public void setGlassPane(Component glassPane)
```

参数说明

glassPane：用户自定义的玻璃窗格。

脚下留神：

玻璃窗格在默认情况下是不可见的，需要使用 `setVisible(true)` 语句将其设置为可见。

实例 210 简单的每日提示信息

（实例位置：配套资源\SL\14\210）

实例说明

对于一些功能比较复杂的软件，可以在软件启动时弹出一个对话框来显示一些提示信息，如软件的快捷键、软件的使用技巧、软件公司的简介等。本实例使用 `JDialog` 实现了一个每日提示对话框。实例的运行效果如图 14.28 所示。

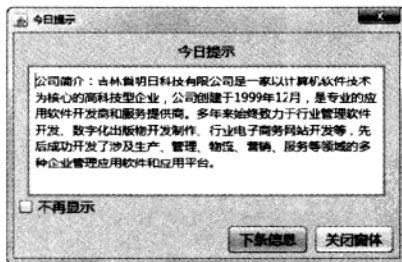


图 14.28 简单的每日提示信息

实现过程

编写 `TipOfDay` 类，该类继承了 `JDialog`，实现了显示提示信息的功能。对话框包含一个标签、一个文本域、一个复选框和两个按钮。关键代码如下：

```
public TipOfDay() {  
    setTitle("\u4ECA\u65E5\u63D0\u793A");  
    setBounds(100, 100, 450, 300);  
    //设置对话框的标题  
    //设置对话框的大小和位置
```



```

getContentPane().setLayout(new BorderLayout()); //设置对话框的布局是边框布局
contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5)); //设置边框是空边框, 宽度是 5
getContentPane().add(contentPanel, BorderLayout.CENTER); //在中央增加面板 contentPanel
contentPanel.setLayout(new BorderLayout(0, 0)); //设置中央面板中空白大小是 0
{
    JPanel panel = new JPanel(); //创建新的 panel 面板
    contentPanel.add(panel, BorderLayout.NORTH); //在 contentPanel 中增加 panel
    {
        JLabel label = new JLabel("\u4ECA\u65E5\u63D0\u793A"); //创建标签
        panel.add(label); //在 panel 中增加标签
    }
}
{
    JPanel panel = new JPanel(); //创建新的 panel 面板
    contentPanel.add(panel, BorderLayout.SOUTH); //在南方增加一个选择框
    panel.setLayout(new BorderLayout(0, 0));
    {
        JCheckBox checkBox = new JCheckBox("\u4E0D\u518D\u663E\u793A");
        panel.add(checkBox);
    }
}
{
    JPanel panel = new JPanel(); //创建新的 panel 面板
    contentPanel.add(panel, BorderLayout.WEST); //在西方增加一个空面板占位
}
{
    JPanel panel = new JPanel(); //创建新的 panel 面板
    contentPanel.add(panel, BorderLayout.EAST); //在东方增加一个空面板占位
}
{
    JScrollPane scrollPane = new JScrollPane();
    contentPanel.add(scrollPane, BorderLayout.CENTER);
    {
        JTextArea textArea = new JTextArea(); //利用文本域来显示主要的信息
        //省略文本信息代码
        scrollPane.setViewportView(textArea);
    }
}
{
    JPanel buttonPane = new JPanel(); //创建新的 panel 面板
    buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
    getContentPane().add(buttonPane, BorderLayout.SOUTH); //增加按钮面板 buttonPane
    {
        JButton okButton = new JButton("\u4E0B\u6761\u4FE1\u606F");
        okButton.setActionCommand("OK");
        buttonPane.add(okButton); //增加“下条信息”按钮
        getRootPane().setDefaultButton(okButton);
    }
}
{
    JButton cancelButton = new JButton("\u5173\u95ED\u7A97\u4F53");
}

```



```
cancelButton.setActionCommand("Cancel");
buttonPane.add(cancelButton);           //增加“关闭窗体”按钮
```

```
}
}
```

技术要点

本实例涉及的技术要点请参见实例 201。

实例 211 震动效果的提示信息

(实例位置: 配套资源\SL\14\211)

实例说明

在软件的使用过程中,如果能增加一些动态效果是很有益的。例如在 QQ 2010 版中就有震动的窗体的例子。Java 的 Swing 也能做成这种效果吗?答案是肯定的。本实例将实现一个震动效果的对话框。实例的运行效果如图 14.29 所示。

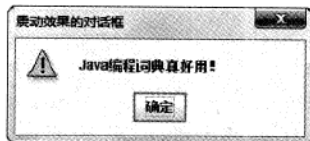


图 14.29 震动效果的提示信息

实现过程

编写 ShakeDialog 类,该类定义了 4 个方法:构造方法用来获得对话框对象;startShake()方法用来实现震动效果,震动时间是 1 秒钟;stopShake()方法用来关闭震动效果并将对话框恢复到原来的位置;main()方法用来进行测试。关键代码如下:

```
public class ShakeDialog {
    private JDialog dialog;
    private Point start;           //保存对话框的初始位置
    private Timer shakeTimer;
    public ShakeDialog(JDialog dialog) {           //在构造方法中获得对话框对象
        this.dialog = dialog;
    }
    public void startShake() {           //开始震动方法
        final long startTime = System.currentTimeMillis(); //获得程序运行的起始时间
        start = dialog.getLocation();           //获得对话框的初始位置
        shakeTimer = new Timer(10, new ActionListener() { //每隔 10 毫秒启动改变对话框坐标事件
            @Override
            public void actionPerformed(ActionEvent e) {
                long elapsed = System.currentTimeMillis() - startTime; //获得程序运行的时间
                Random random = new Random(elapsed); //以运行时间为种子创建随机数对象
                int change = random.nextInt(50); //获得一个小于 50 个随机数整数
                dialog.setLocation(start.x + change, start.y + change); //随机改变坐标
            }
        });
    }
}
```




```

        if (elapsed >= 1000) {                //如果程序运行时间大于 1 秒钟则停止
            stopShake();
        }
    }
});
shakeTimer.start();                          //启动 Timer
}
public void stopShake() {                    //停止震动方法
    shakeTimer.stop();                        //停止 Timer
    dialog.setLocation(start);               //恢复对话框的坐标
    dialog.repaint();                        //重新绘制对话框
}
public static void main(String[] args) {     //测试方法
    JOptionPane pane = new JOptionPane("Java 编程词典真好用!",
    JOptionPane.WARNING_MESSAGE);
    JDialog d = pane.createDialog(null, "震动效果的对话框"); //获得对话框对象
    ShakeDialog sd = new ShakeDialog(d);
    d.pack();                               //按对话框内的控件来绘制对话框
    d.setModal(false);                      //关闭模态
    d.setVisible(true);                     //设为可见
    sd.startShake();                         //开始震动
}
}

```

指点迷津:

可以将该类作为工具类使用，只需要传递要震动的对话框即可。

技术要点

Timer 类可以用于在指定时间间隔触发一个或多个事件。本实例使用的方法如表 14.8 所示。

表 14.8 Timer 类的常用方法

方 法 名	作 用
Timer(int delay, ActionListener listener)	Timer 的构造方法，用于每隔 delay 毫秒触发事件 listener
start()	启动 Timer，使它开始向其侦听器发送动作事件
stop()	停止 Timer，使它停止向其侦听器发送动作事件

脚下留神:

Java API 中共有 3 个 Timer 类，其功能和用法各不相同，请读者注意区别。

实例 212 制作圆形布局管理器

(实例位置: 配套资源\SL\14\212)

实例说明

尽管 Java 的 API 中实现了十余种布局管理器，有时却不能完全满足不同用户的需求，此



时可以考虑自己实现一个布局管理器。圆形是日常生活中最常见的几何形状之一，本实例演示如何实现圆形布局管理器。实例的运行效果如图 14.30 所示。

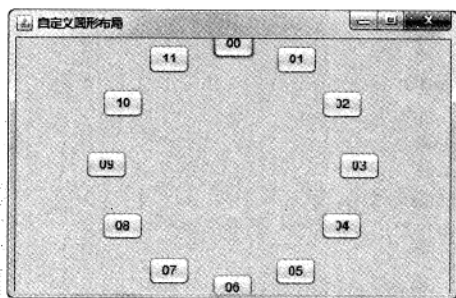


图 14.30 圆形布局管理器

实现过程

编写 `CircleLayout` 类，该类继承了 `LayoutManager`。其中重点实现了 `layoutContainer()` 方法，用来实现如何放置容器中控件的功能，关键代码如下：

```
public void layoutContainer(Container parent) {
    double centerX = parent.getBounds().getCenterX();           //获得容器中心的 X 坐标
    double centerY = parent.getBounds().getCenterY();           //获得容器中心的 Y 坐标
    Insets insets = parent.getInsets();                          //获得容器默认边框对象
    double horizon = centerX - insets.left;                      //获得水平可用长度的一半
    double vertical = centerY - insets.top;                      //获得垂直可用长度的一半
    double radius = horizon > vertical ? vertical : horizon;     //取小的为圆形半径
    int count = parent.getComponentCount();                     //获得容器中控件的个数
    for (int i = 0; i < count; i++) {                           //依次设置所有可见控件的位置和大小
        Component component = parent.getComponent(i);
        if (component.isVisible()) {
            Dimension size = component.getPreferredSize();     //大小使用其最佳大小
            double angle = 2 * Math.PI * i / count;             //获得角度的大小
            double x = centerX + radius * Math.sin(angle);       //获得圆周点的 X 坐标
            double y = centerY - radius * Math.cos(angle);       //获得圆周点的 Y 坐标
            //重新设置控件的位置和大小
            component.setBounds((int) x - size.width / 2, (int) y - size.height / 2, size.width, size.height);
        }
    }
}
```

指点迷津：

可以将该类作为工具类使用，只需要传递要震动的对话框即可。

技术要点

`LayoutManager` 接口是为如何布局控件的容器定制的。Swing 的绘制架构假定 `JComponent` 的子控件不发生重叠。如果 `JComponent` 的布局管理器允许子控件重叠，则它必须重写 `isOptimizedDrawingEnabled()` 方法。`LayoutManager` 接口定义的方法及其说明如表 14.9 所示。



表 14.9 LayoutManager 接口中的方法

方 法 名	作 用
addLayoutComponent(String name, Component comp)	如果布局管理器使用 per-component 字符串, 则将控件 comp 添加到布局, 并将它与 name 指定的字符串关联
layoutContainer(Container parent)	布置 parent 控件
minimumLayoutSize(Container parent)	给定指定容器所包含的控件, 计算该容器的最小大小维数
preferredLayoutSize(Container parent)	给定指定容器所包含的控件, 计算该容器的首选大小维数
removeLayoutComponent(Component comp)	从布局中移除控件 comp



Note

多学两招:

对于简单的自定义布局管理器, 可以只实现 layoutContainer() 方法。

实例 213 制作阶梯布局管理器

(实例位置: 配套资源\SL\14\213)

实例说明

尽管 Java 的 API 中实现了十余种布局管理器, 有时却不能完全满足不同用户的需求, 此时可以考虑自己实现一个布局管理器。阶梯是日常生活中最常见的几何形状之一, 本实例演示如何实现阶梯布局管理器。实例的运行效果如图 14.31 所示。

实现过程

编写 CircleLayout 类, 该类继承了 LayoutManager。其中重点实现了 layoutContainer() 方法, 用来实现如何放置容器中控件的功能。关键代码如下:

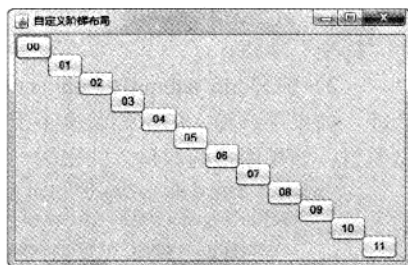


图 14.31 阶梯布局管理器

```

public void layoutContainer(Container parent) {
    Insets insets = parent.getInsets(); //获得容器默认边框对象
    int maxWidth = parent.getWidth() - (insets.left + insets.right); //获得最大可用宽度
    int maxHeight = parent.getHeight() - (insets.top + insets.bottom); //获得最大可用高度
    int count = parent.getComponentCount(); //获得容器中控件的个数
    for (int i = 0; i < count; i++) { //依次设置所有可见控件的位置和大小
        Component component = parent.getComponent(i);
        if (component.isVisible()) {
            Dimension size = component.getPreferredSize(); //大小使用其最佳大小
            int x = maxWidth / count * i; //将宽度分成 count 份根据 i 值调整 X 坐标
            int y = maxHeight / count * i; //将高度分成 count 份根据 i 值调整 Y 坐标
            component.setBounds(x, y, size.width, size.height); //重新设置控件的位置和大小
        }
    }
}

```



技术要点

本实例涉及的技术要点请参见实例 212。



Note

实例 214 密码域控件简单应用

(实例位置: 配套资源\SL\14\214)

实例说明

在用户注册网站的会员时, 通常需要输入用户名、密码、邮箱等。密码需要输入两次以防止用户第一次输入错误。如果两次的密码相同则可以在数据库中创建用户, 否则需要进行修改。本实例演示密码域控件的应用。实例的运行效果如图 14.32 所示。

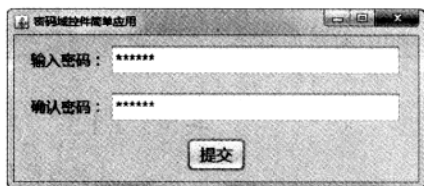


图 14.32 向窗体中添加密码框

实现过程

(1) 编写类 JPasswordFieldTest, 该类继承了 JFrame。在框架中, 包括了两个文本域和一个“提交”按钮。

(2) 编写 do_submitButton_actionPerformed() 方法, 该方法实现了对按钮单击事件的监听功能。如果用户输入的密码长度小于 6, 则发出警告; 如果用户两次输入的密码不一致, 则发出警告; 如果用户两次输入的密码一致, 则提示密码相同。关键代码如下:

```
protected void do_submitButton_actionPerformed(ActionEvent e) {
    char[] password1 = passwordField1.getPassword(); //获得第一个密码域中的内容
    char[] password2 = passwordField2.getPassword(); //获得第二个密码域中的内容
    if (password1.length < 6) { //如果密码的长度小于 6, 则发出警告信息
        JOptionPane.showMessageDialog(this, "密码长度小于 6 位", "",
            JOptionPane.WARNING_MESSAGE);
    } else if (!Arrays.equals(password1, password2)) { //如果密码的长度不同, 则发出警告信息
        JOptionPane.showMessageDialog(this, "两次密码不同", "",
            JOptionPane.WARNING_MESSAGE);
    } else {
        JOptionPane.showMessageDialog(this, "两次密码相同", "",
            JOptionPane.INFORMATION_MESSAGE);
    }
}
```

脚下留神:

在使用完密码内容之后, 可以使用 Arrays 的 fill(char[] a, char val) 方法, 用字符 val 填充整个字符数组 a。

技术要点

密码域继承于文本域, 但是修改了其显示方式。所有用户的输入并不能直接看到, 而是用



一些回显符号替代。典型的回显符号是“*”，用户可以根据需求自行设置。在使用密码域时，最关心的还是如何获得密码域中的文本信息，使用 `getPassword()` 方法可以实现。该方法的声明如下：

```
public char[] getPassword()
```

为了安全起见，在使用之后应该重置字符数组中的内容。

实例 215 文本域设置背景图片

(实例位置：配套资源\SL\14\215)



Note

实例说明

在软件的美化过程中，比较常用的方式之一是使用背景图片。在 Swing 中，除了可以给框架增加背景图片外，还可以给文本域设置背景图片。本实例将演示如何实现该操作。实例的运行效果如图 14.33 所示。

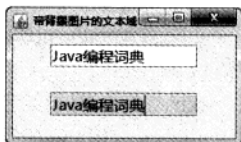


图 14.33 文本域设置背景图片

实现过程

(1) 编写 `BackgroundJTextFieldTest` 类，该类继承了 `JFrame`。在框架中包括了两个文本域。

(2) 编写 `BackgroundJTextField` 类，该类继承了 `JTextField`。在该类的构造方法中利用传递的 `File` 参数获得一个缓冲图像，以此来作为文本框的背景图片。在 `paintComponent()` 方法中，将此背景图片绘制到文本域中，关键代码如下：

```
public class BackgroundJTextField extends JTextField {
    private static final long serialVersionUID = 5810044732894008630L;
    private TexturePaint paint;
    public BackgroundJTextField(File file) {
        super();
        try {
            BufferedImage image = ImageIO.read(file);           //获得缓冲图片
            Rectangle rectangle = new Rectangle(0, 0, image.getWidth(), image.getHeight());
            paint = new TexturePaint(image, rectangle);           //创建 TexturePaint 对象
            setOpaque(false);                                     //将文本域设置成透明的
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;                         //将 g 转型为 Graphics2D
        g2.setPaint(paint);                                       //设置新的颜色模式
        g.fillRect(0, 0, getWidth(), getHeight());              //让图片充满整个区域
        super.paintComponent(g);                                  //调用父类的同名方法
    }
}
```

技术要点

`ImageIO` 类提供了很多读写图片的静态方法，还可以对图片进行简单的编码和解码。本实



例使用该类中的 read() 方法从本地读取图片，该方法的声明如下：

```
public static BufferedImage read(File input) throws IOException
```

参数说明

input: 被读入的文件。

多学两招:

还可以使用 InputStream、URL 等作为 read() 方法的参数。

TexturePaint 类提供一种用被指定为 BufferedImage 的纹理填充 Shape 的方式。因为 BufferedImage 数据由 TexturePaint 对象复制，所以 BufferedImage 对象的大小应该小一些。其构造方法声明如下：

```
public TexturePaint(BufferedImage txtr, Rectangle2D anchor)
```

参数说明

☑ txtr: 具有用于绘制的纹理的 BufferedImage 对象。

☑ anchor: 用户空间中用于定位和复制纹理的 Rectangle2D。

实例 216 文本区设置背景图片

(实例位置: 配套资源\SL\14\216)

实例说明

在软件的美化过程中，比较常用的方式之一是使用背景图片。在 Swing 中，除了可以给框架增加背景图片外，还可以给文本区设置背景图片。本实例将演示如何实现该操作。实例的运行效果如图 14.34 所示。

实现过程

(1) 编写 BackgroundJTextAreaTest 类，该类继承 JFrame。在框架中包括一个自定义文本区。

(2) 编写 BackgroundJTextArea 类，该类继承了 JTextArea。在其构造方法中利用给定的路径获得图片，并将文本区设置成透明的。在 paint() 方法中，将图片绘制到文本区中，关键代码如下：

```
public class BackgroundJTextArea extends JTextArea {
    private static final long serialVersionUID = -4157782271632761973L;
    private Image image;
    public BackgroundJTextArea(String path) {
        ImageIcon imageIcon = new ImageIcon(path); // 获得图片图标
        image = imageIcon.getImage(); // 获得图片
        setOpaque(false); // 将文本区设置成透明的
    }
    @Override
    public void paint(Graphics g) {
```

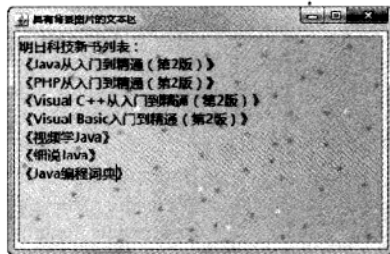


图 14.34 文本区设置背景图片



```
g.drawImage(image, 0, 0, this);
super.paint(g);
```

//绘制图片

技术要点

ImageIcon 类是 Icon 接口的实现,它根据 Image 绘制 Icon。在其构造方法中,提供了根据字符串路径创建图标的方式,该方法的声明如下:

```
public ImageIcon(String filename)
```

参数说明

filename: 指定文件名或路径的字符串。

在获得图标之后,需要获得构成该图标的图片,使用 getImage()方法即可。该方法的声明如下:

```
public Image getImage()
```



Note

实例 217 简单的字符统计工具

(实例位置: 配套资源\SL\14\217)

实例说明

在使用文本编辑软件,如 Word 2007 时,会在软件界面中提示用户软件的总共字符等信息,方便用户掌握文档编写的进度。本实例将模拟 Word 的功能并进行增强,可以实时显示光标所在的位置和用户选择的文本所包含的字符数量。实例的运行效果如图 14.35 所示。

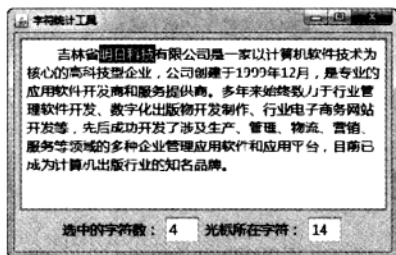


图 14.35 简单的字符统计工具

实现过程

(1) 编写 CharCount 类,该类继承了 JFrame。在框架中包括一个文本区和两个文本域。

(2) 编写 do_textArea_caretUpdate()方法,用来显示光标的变化信息。该方法是由 IDE 自动生成的。关键代码如下:

```
protected void do_textArea_caretUpdate(CaretEvent e) {
    int dot = e.getDot();           //获得光标所在位置
    int mark = e.getMark();         //获得使用鼠标选择时光标的起点位置
    textField1.setText(Math.abs(dot - mark) + ""); //计算用户选择的文本长度并在文本域显示
    textField2.setText(dot + "");   //显示光标所在的位置
}
```




技术要点

CaretListener 接口用于侦听文本控件插入符的位置更改的侦听器。该接口定义了一个 caretUpdate() 方法, 当插入符的位置被更新时该方法被调用, 其声明如下:

```
void caretUpdate(CaretEvent e)
```

参数说明

e: 插入符事件。

CaretEvent 用于通知感兴趣的参与者事件源中的文本插入符已发生更改。该类定义了两个抽象方法, 其声明和说明如下。

- ☑ public abstract int getDot(): 获得插入符的位置。
- ☑ public abstract int getMark(): 获得逻辑选择的另一端的位置。如果没有进行选择, 则此位置将与 dot 相同。

实例 218 能预览图片的复选框

(实例位置: 配套资源\SL\14\218)

实例说明

使用文本域、文本区等控件与用户交互时, 很难解决的一个问题是如何保证用户输入的合法性。对于一些有确定范围的信息, 可以使用复选框来让用户选择。复选框只有两种状态: 选择和未选择, 这样就可以省略校验的代码。本实例根据用户的选择来显示不同的图片。实例的运行效果如图 14.36 所示。

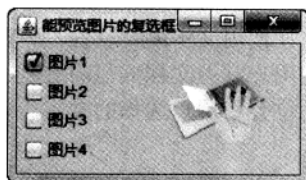


图 14.36 能预览图片的复选框

实现过程

(1) 编写 JCheckBoxTest 类, 该类继承了 JFrame。在框架中包括 4 个复选框和一个用来显示图片的标签。

(2) 编写 do_checkBox1_actionPerformed() 方法, 该方法是 IDE 自动生成的, 用于实现对选择复选框事件的监听。该方法实现了设置标签图片的功能。关键代码如下:

```
protected void do_checkBox1_actionPerformed(ActionEvent e) {  
    if(checkBox1.isSelected()) {  
        ImageIcon icon = new ImageIcon("src/images/1.png");  
        label.setIcon(icon);  
    }  
}
```

技术要点

JCheckBox 类是复选框的实现, 复选框是一个可以被选定和取消选定的项, 它将其状态显示给用户。按照惯例, 可以选定组中任意数量的复选框。其常用方法如表 14.10 所示。



表 14.10 复选框的常用方法

方法名	作用
JCheckBox(String text)	创建一个带文本的、最初未被选中的复选框
addActionListener(ActionListener l)	将一个 ActionListener 添加到复选框中
isSelected()	判断复选框是否被选中
setMnemonic(int mnemonic)	设置当前模型上的键盘助记符
setSelected(boolean b)	设置复选框被选中



Note

指点迷津:

JCheckBox 是 AbstractButton 的间接子类, 因此继承了很多 AbstractButton 中定义的有用的方法。

实例 219 简单的投票计数软件

(实例位置: 配套资源\SL\14\219)

实例说明

日常生活中, 经常听到少数服从多数这句话。那么怎么知道哪个是少数, 哪个是多数呢? 通常是通过投票完成的。本实例实现一个简单的投票计数软件。实例的运行效果如图 14.37 所示。

实现过程

(1) 编写 VoteSystem 类, 该类继承了 JFrame。在框架中, 共包括 4 个复选框、4 个进度条、4 个标签和两个按钮。“提交”按钮用于重新计算投票的结果。“刷新”按钮用于重置复选框。

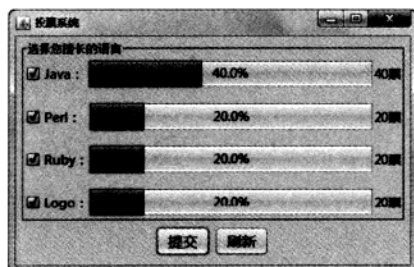


图 14.37 简单的投票计数软件

(2) 编写 do_submitButton_actionPerformed() 方法, 该方法首先获得历史投票数, 然后根据用户在复选框的选择结果重新计算投票结果并在进度条和标签中显示。关键代码如下:

```
protected void do_submitButton_actionPerformed(ActionEvent e) {
    String text1 = label1.getText();           //获得标签中的文本
    int number1 = Integer.parseInt(text1.substring(0, text1.length() - 1)); //获得票数
    String text2 = label2.getText();           //获得标签中的文本
    int number2 = Integer.parseInt(text2.substring(0, text2.length() - 1)); //获得票数
    String text3 = label3.getText();           //获得标签中的文本
    int number3 = Integer.parseInt(text3.substring(0, text3.length() - 1)); //获得票数
    String text4 = label4.getText();           //获得标签中的文本
    int number4 = Integer.parseInt(text4.substring(0, text4.length() - 1)); //获得票数
    if (checkBox1.isSelected()) {              //如果复选框被选中
        number1++;                             //票数加 1
        label1.setText(number1 + "票");         //更新标签
    }
    if (checkBox2.isSelected()) {              //如果复选框被选中
```



Note

```

        number2++;
        label2.setText(number2 + "票");
    }
    if (checkBox3.isSelected()) {
        number3++;
        label3.setText(number3 + "票");
    }
    if (checkBox4.isSelected()) {
        number4++;
        label4.setText(number4 + "票");
    }
    double total = number1 + number2 + number3 + number4;
    progressBar1.setString(number1 * 100 / total + "%");
    progressBar1.setValue(number1);
    progressBar2.setString(number2 * 100 / total + "%");
    progressBar2.setValue(number2);
    progressBar3.setString(number3 * 100 / total + "%");
    progressBar3.setValue(number3);
    progressBar4.setString(number4 * 100 / total + "%");
    progressBar4.setValue(number4);
}

```

//票数加 1
//更新标签
//如果复选框被选中
//票数加 1
//更新标签
//如果复选框被选中
//票数加 1
//更新标签
//计算总共的票数
//在进度条上显示所在比例的文本信息
//在进度条上显示票数
//在进度条上显示所在比例的文本信息
//在进度条上显示票数
//在进度条上显示所在比例的文本信息
//在进度条上显示票数
//在进度条上显示所在比例的文本信息
//在进度条上显示票数

技术要点

本实例应用到的技术要点可参考实例 218。

实例 220 单选按钮的简单应用

(实例位置: 配套资源\SL\14\220)

实例说明

在使用复选框时, 用户可以选择任意多个选项, 有时却只希望用户选择选项组中的一个。最典型的例子就是性别, 通常希望用户在男女之间选择一个。当用户选择一项之后, 前次选择会自动取消。在 Swing 中, 通常使用单选按钮来实现该功能。本实例利用单选按钮来浏览图片。实例的运行效果如图 14.38 所示。

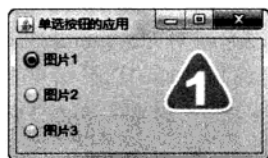


图 14.38 通过单选按钮选择图片

实现过程

(1) 编写 JRadioButtonTest 类, 该类继承了 JFrame。在框架中包括 3 个单选按钮和一个用来显示图片的标签。

(2) 编写 do_radioButton1_actionPerformed() 方法, 该方法是 IDE 自动生成的, 用于实现对选择单选按钮事件的监听。该方法实现了设置标签图片的功能。关键代码如下:

```

protected void do_radioButton1_actionPerformed(ActionEvent e) {
    if (radioButton1.isSelected()) {
        ImageIcon icon = new ImageIcon("src/images/1.png");
    }
}

```

//如果单选按钮被选中
//创建图片图表



```
label.setIcon(icon);
```

```
//设置图标
```

技术要点

JRadioButton 用于实现一个单选按钮，此单选按钮可被选中或取消选中，并可为用户显示其状态。与 ButtonGroup 对象配合使用可创建一组单选按钮，一次只能选中其中的一个单选按钮。（创建一个 ButtonGroup 对象并用其 add() 方法将 JRadioButton 对象包含在此组中。）其常用方法如表 14.11 所示。

表 14.11 单选按钮的常用方法

方 法 名	作 用
JRadioButton(String text)	创建一个具有指定文本的状态为未选中的单选按钮
addActionListener(ActionListener l)	将一个 ActionListener 添加到单选按钮中
isSelected()	判断单选按钮是否被选中
setMnemonic(int mnemonic)	设置当前模型上的键盘助记符
setSelected(boolean b)	设置单选按钮被选中

实例 221 能显示图片的组合框

（实例位置：配套资源\SL\14\221）

实例说明

在屏幕空间有限的情况下，使用单选按钮并不合适，因为需要列出所有的选项，此时可以考虑使用组合框。组合框类包括一个文本域和一个下拉列表两部分。对于普通的组合框使用非常简单，将用户的选项组成一个数组或向量传递给组合框的构造方法即可。本实例用来实现在组合框中显示图片，这样可以使界面更加美观。实例的运行效果如图 14.39 所示。

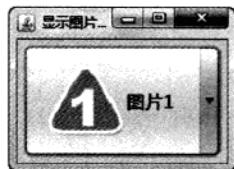


图 14.39 显示图片的组合框


实现过程

（1）编写 ComboBoxRenderer 类，该类继承了 JLabel 并且实现了 ListCellRenderer。该类用于生成组合框中的各个选项，关键代码如下：

```
public class ComboBoxRenderer extends JLabel implements ListCellRenderer {
    private static final long serialVersionUID = -318939036460656104L;
    private Map<String, ImageIcon> content;           //保存图片和其说明
    public ComboBoxRenderer(Map<String, ImageIcon> content) {
        this.content = content;
        setOpaque(true);                             //设置标签为不透明
        setHorizontalAlignment(CENTER);              //水平方向居中对齐
        setVerticalAlignment(CENTER);                  //垂直方向居中对齐
    }
    @Override
```





 **Note**

```
public Component getListCellRendererComponent(JList list, Object value, int index, boolean
isSelected, boolean cellHasFocus) {
    String key = (String)value;           //将组合框的一个值转换成字符串
    if (isSelected) {                     //根据是否处于选中状态而更改外观
        setBackground(list.getSelectionBackground());
        setForeground(list.getSelectionForeground());
    } else {
        setBackground(list.getBackground());
        setForeground(list.getForeground());
    }
    setText(key);                         //设置标签的文本
    setIcon(content.get(key));            //设置标签的图标
    setFont(list.getFont());             //设置标签的字体
    return this;
}
```

多学两招:

通过继承 JLabel, 可以很方便地显示文本和图标, 读者可以根据自己的需求选择合适的类继承。

(2) 编写 JComboBoxTest 类, 该类继承了 JFrame。在框架中显示一个组合框。其构造方法中增加了构造组合框的方法, 关键代码如下:

```
public JComboBoxTest() {
    setTitle("\u663E\u793A\u56FE\u7247\u7EC4\u5408\u6846"); //设置框架的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置框架在关闭时退出
    setBounds(100, 100, 200, 150); //设置显示位置和大小
    contentPane = new JPanel(); //创建面板对象
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5)); //设置面板边距
    contentPane.setLayout(new BorderLayout(0, 0)); //设置面板布局
    setContentPane(contentPane);
    Map<String, ImageIcon> content = new LinkedHashMap<String, ImageIcon>();
    content.put("图片 1", new ImageIcon("src/images/1.png")); //增加由图标说明和图标组成的映射
    content.put("图片 2", new ImageIcon("src/images/2.png")); //增加由图标说明和图标组成的映射
    content.put("图片 3", new ImageIcon("src/images/3.png")); //增加由图标说明和图标组成的映射
    JComboBox comboBox = new JComboBox(content.keySet().toArray()); //利用键值构造组合框
    ComboBoxRenderer renderer = new ComboBoxRenderer(content); //创建渲染器
    comboBox.setRenderer(renderer); //设置渲染器
    comboBox.setMaximumRowCount(3); //设置组合框最多显示 3 行可选项
    comboBox.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置组合框字体
    contentPane.add(comboBox, BorderLayout.CENTER); //将组合框布局在框架中央
}
```

技术要点

JComboBox 是将按钮或可编辑字段与下拉列表组合的控件。用户可以从下拉列表中选择值, 下拉列表在用户请求时显示。如果使组合框处于可编辑状态, 则组合框将包括用户可在其中输入值的可编辑字段。其常用方法如表 14.12 所示。



表 14.12 JComboBox 的常用方法

方 法 名	作 用
JComboBox(Object[] items)	创建包含指定数组中元素的 JComboBox
addActionListener(ActionListener l)	添加 ActionListener
addItem(Object anObject)	为组合框添加项
getItemCount()	返回组合框中的项数
getRenderer()	返回用于显示 JComboBox 字段中所选项的渲染器
getSelectedIndex()	返回列表中与给定项匹配的第一个选项
getSelectedItem()	返回当前所选项
isEditable()	如果 JComboBox 可编辑, 则返回 true
removeAllItems()	从项列表中移除所有项
removeItem(Object anObject)	从项列表中移除项
removeItemAt(int anIndex)	移除 anIndex 处的项
setEditable(boolean aFlag)	确定 JComboBox 字段是否可编辑
setMaximumRowCount(int count)	设置 JComboBox 显示的最大行数
setRenderer(ListCellRenderer aRenderer)	设置渲染器, 该渲染器用于绘制列表项和选择的项
setSelectedIndex(int anIndex)	选择索引 anIndex 处的项
setSelectedItem(Object anObject)	将组合框显示区域中所选项设置为参数中的对象

本实例还使用了 ListCellRenderer 接口, 它表示可用作“橡皮图章”以绘制 JList 中单元格的控件。该接口定义了一个 getListCellRendererComponent() 方法, 该方法返回一个配置好的控件来显示特定值。该方法的声明如下:

```
getListCellRendererComponent(JList list, Object value, int index, boolean isSelected, boolean cellHasFocus)
```

该方法的返回值是 Component, 各个参数的说明如表 14.13 所示。

表 14.13 getListCellRendererComponent() 方法的参数说明

参 数 名	说 明
list	正在绘制的 JList
value	由 list.getModel().getElementAt(index) 返回的值
index	单元格索引
isSelected	如果选择了指定的单元格, 则为 true
cellHasFocus	如果指定的单元格拥有焦点, 则为 true

实例 222 使用滑块来选择日期

(实例位置: 配套资源\SL\14\222)

实例说明

当可以选择的选项很多时, 使用单选按钮并不理想, 因为需要创建大量的按钮, 此时可以考虑使用滑块。滑块可以让用户在一组离散值中进行选择。本实例将使用滑块来选择日期, 实例的运行效果如图 14.40 所示。



Note

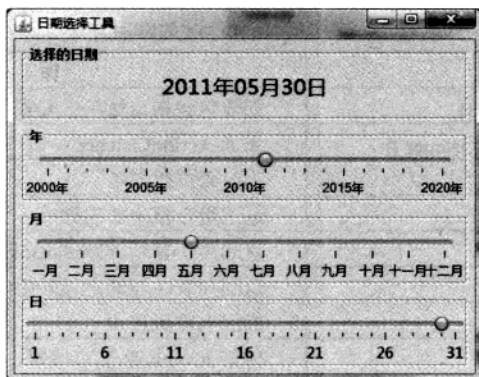


图 14.40 使用滑块来选择日期

实现过程

(1) 编写 `do_this_windowActivated()` 方法, 该方法用于监听窗体活动事件。在该方法中, 对滑块进行了基本设置, 如最大值、最小值、起始值等, 并将时间设置成当前时间。关键代码如下:

```
protected void do_this_windowActivated(WindowEvent e) {
    yearSlider.setMaximum(2020);           //将 yearSlider 滑块的最大值设置成 2020
    yearSlider.setMinimum(2000);           //将 yearSlider 滑块的最小值设置成 2000
    yearSlider.setMajorTickSpacing(5);      //将 yearSlider 滑块的主刻度设置成 5
    yearSlider.setMinorTickSpacing(1);      //将 yearSlider 滑块的副刻度设置成 1
    yearSlider.setValue(calendar.get(Calendar.YEAR)); //将 yearSlider 滑块的值设置成当前年
    Dictionary<Integer, Component> yearLabel = new Hashtable<Integer, Component>();
    yearLabel.put(2000, new JLabel("2000 年")); //为 2000 增加标签 "2000 年"
    yearLabel.put(2005, new JLabel("2005 年")); //为 2005 增加标签 "2005 年"
    yearLabel.put(2010, new JLabel("2010 年")); //为 2010 增加标签 "2010 年"
    yearLabel.put(2015, new JLabel("2015 年")); //为 2015 增加标签 "2015 年"
    yearLabel.put(2020, new JLabel("2020 年")); //为 2020 增加标签 "2020 年"
    yearSlider.setLabelTable(yearLabel);     //为 yearSlider 增加标签
    yearSlider.addChangeListener(cl);        //为 yearSlider 增加监听
    monthSlider.setMaximum(12);              //将 monthSlider 滑块的最大值设置成 12
    monthSlider.setMinimum(1);               //将 monthSlider 滑块的最小值设置成 1
    monthSlider.setMajorTickSpacing(1);      //将 monthSlider 滑块的主刻度设置成 1
    monthSlider.setValue(calendar.get(Calendar.MONTH) + 1); //将 monthSlider 滑块的值设成当月
    String[] months = (new DateFormatSymbols()).getShortMonths(); //获得本地月份字符串数组
    Dictionary<Integer, Component> monthLabel = new Hashtable<Integer, Component>(12);
    for (int i = 0; i < 12; i++) {
        monthLabel.put(i + 1, new JLabel(months[i])); //为 1~12 增加标签
    }
    monthSlider.setLabelTable(monthLabel);   //为 daySlider 增加标签
    monthSlider.addChangeListener(cl);       //为 monthSlider 增加监听
    daySlider.setMaximum(calendar.getMaximum(Calendar.DAY_OF_MONTH)); //最大值设成当月天数
    daySlider.setMinimum(1);                //将 daySlider 滑块的最小值设置成 1
    daySlider.setMajorTickSpacing(5);        //将 daySlider 滑块的主刻度设置成 5
    daySlider.setMinorTickSpacing(1);        //将 daySlider 滑块的副刻度设置成 1
    daySlider.setValue(calendar.get(Calendar.DATE)); //将 daySlider 滑块的值设置成当天
}
```



```
daySlider.addChangeListener(cl);           //为 daySlider 增加监听
dateLabel.setText(dateFormat.format(new Date())); //用标签显示当前时间
}
```

脚下留神:

在设置标尺标签时要注意不能将 JLabel 写成 Label, 否则在程序运行后并不会显示标签。



Note

(2) 编写内部类 DateListener, 该类继承了 ChangeListener, 用来监听滑块的变化事件, 根据用户选择的不同日期来更新标签的内容。关键代码如下:

```
private class DateListener implements ChangeListener {
    @Override
    public void stateChanged(ChangeEvent e) {
        calendar.set(yearSlider.getValue(), monthSlider.getValue() - 1, 1);
        int maxDays = calendar.getActualMaximum(Calendar.DAY_OF_MONTH); //获得月最大天数
        if (daySlider.getMaximum() != maxDays) {
            daySlider.setValue(Math.min(daySlider.getValue(), maxDays)); //设置滑块的值
            daySlider.setMaximum(maxDays); //将滑块的最大值修改成当前月的最大天数
            daySlider.repaint(); //重新绘制日期滑块
        }
        calendar.set(yearSlider.getValue(), monthSlider.getValue() - 1,
            daySlider.getValue()); //将日期设置成用户当前选择的日期
        dateLabel.setText(dateFormat.format(calendar.getTime())); //更新标签的内容
    }
}
```

技术要点

JSlider 是一个让用户以图形方式在有界区间内通过移动滑块来选择值的控件。当用户滑动滑块时, 其值会在最大与最小值之间变化, 还可以给滑块增加标尺、标尺标签等进行修饰。其常用方法如表 14.14 所示。

表 14.14 滑块的常用方法

方 法 名	作 用
JSlider(int min, int max, int value)	用指定的最小值、最大值和初始值创建一个水平滑块
addChangeListener(ChangeListener l)	将一个 ChangeListener 添加到滑块
getValue()	从 BoundedRangeModel 返回滑块的当前值
setFont(Font font)	设置控件的字体
setInverted(boolean b)	指定为 true, 则反转滑块显示的值范围
setMajorTickSpacing(int n)	此方法设置主刻度标记的间隔
setMaximum(int maximum)	将滑块的最大值设置为 maximum
setMinimum(int minimum)	将滑块的最小值设置为 minimum
setMinorTickSpacing(int n)	此方法设置次刻度标记的间隔
setPaintLabels(boolean b)	确定是否在滑块上绘制标签
setPaintTicks(boolean b)	确定是否在滑块上绘制刻度标记
setPaintTrack(boolean b)	确定是否在滑块上绘制滑道
setSnapToTicks(boolean b)	指定为 true, 则滑块解析为最靠近用户放置滑块处的刻度标记的值



脚下留神:

如果希望显示标尺标签, 则必须调用 `setPaintLabels()` 方法将其设置成 `true`。



Note

实例 223 模仿记事本的菜单栏

(实例位置: 配套资源\SL\14\223)

实例说明

在 Windows 操作系统中, 自带了一款简单的文本编辑工具: 记事本。记事本主要由两部分组成, 即菜单栏和文本区。菜单栏实现了各种常用的功能, 文本区用于让用户输入文本。本实例将实现一个类似记事本的菜单栏。实例的运行效果如图 14.41 所示。

实现过程

编写 `Notepad` 类, 该类继承自 `JFrame`。在其构造方法中, 增加了一个菜单栏。在菜单栏中增加了 Windows 的记事本中各个菜单项。关键代码如下:

```
public Notepad() {  
    //省略设置框架属性的代码  
    JMenuBar menuBar = new JMenuBar(); //创建菜单栏  
    setJMenuBar(menuBar); //在框架中增加菜单栏  
    JMenu fileMenu = new JMenu("\u6587\u4ef6(F)"); //创建名为“文件”的菜单  
    fileMenu.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置菜单的字体  
    menuBar.add(fileMenu); //将菜单添加到菜单栏中  
    JMenuItem newMenuItem = new JMenuItem("\u65b0\u5efa(N)..."); //创建新的菜单项  
    newMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置菜单项的字体  
    fileMenu.add(newMenuItem); //将菜单项添加到菜单中  
    JMenuItem openMenuItem = new JMenuItem("\u6253\u5f00(O)..."); //创建新的菜单项  
    openMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置菜单的字体  
    fileMenu.add(openMenuItem); //将菜单项添加到菜单中  
    JMenuItem saveMenuItem = new JMenuItem("\u4fdd\u5b58(S)"); //创建新的菜单项  
    saveMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置菜单的字体  
    fileMenu.add(saveMenuItem); //将菜单项添加到菜单中  
    JMenuItem saveAsMenuItem = new JMenuItem("\u5b58\u5165\u5176\u4ed9(A)..."); //创建新的菜单项  
    saveAsMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置菜单的字体  
    fileMenu.add(saveAsMenuItem); //将菜单项添加到菜单中  
    JSeparator separator1 = new JSeparator(); //创建分隔符  
    fileMenu.add(separator1); //将分隔符添加到菜单中  
    JMenuItem pageSetMenuItem = new JMenuItem("\u9875\u9762\u8bbe\u7f6e(U)...");  
    pageSetMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置菜单的字体  
    fileMenu.add(pageSetMenuItem); //将分隔符添加到菜单中  
    JMenuItem printMenuItem = new JMenuItem("\u6253\u5370(P)..."); //创建新的菜单项  
    printMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置菜单的字体  
    fileMenu.add(printMenuItem); //将菜单项添加到菜单中  
    JSeparator separator2 = new JSeparator(); //创建分隔符
```

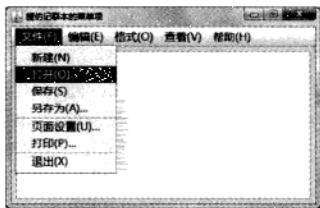


图 14.41 模仿记事本的菜单栏



```
fileMenu.add(separator2); //将分隔符添加到菜单中
JMenuItem exitMenuItem = new JMenuItem("\u9000\u51FA(X)"); //创建新的菜单项
exitMenuItem.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置菜单的字体
fileMenu.add(exitMenuItem); //将菜单项添加到菜单中
//省略其他菜单和文本区代码
}
```

指点迷津:

setFont()方法是在 JComponent 类中定义的, 因此可以在其子类中使用。

技术要点

在 Swing 中使用菜单的第一步是创建一个菜单栏保存各个菜单, 并将菜单栏添加到框架上。其示例代码如下:

```
JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);
```

第二步开始创建各个菜单及其菜单项, 并将菜单项添加到菜单中。为了分类, 可以使用分隔符将功能相近的菜单项分隔后添加到菜单中。示例代码如下:

```
JMenu fileMenu = new JMenu("\u6587\u4EF6(F)");
menuBar.add(fileMenu);
JMenuItem newMenuItem = new JMenuItem("\u65B0\u5EFA(N)");
fileMenu.add(newMenuItem);
```

指点迷津:

菜单栏是可以添加到框架的任意位置的, 按照惯例, 通常将菜单栏添加到容器的顶部。

实例 224 自定义纵向的菜单栏

(实例位置: 配套资源\SL\14\224)

实例说明

在使用软件时, 其菜单栏通常是位于软件窗体的顶部的。如果因为界面设计等方面的原因, 需要将菜单栏放置在窗体左侧, 则可以自定义一个纵向的菜单栏。本实例将实现这个功能。实例的运行效果如图 14.42 所示。

实现过程

(1) 编写 HorizontalMenu 类, 该类继承了 JMenu。在该类的构造方法中, 设置了弹出菜单的布局是水平布局。重写其 getMinimumSize()方法使其最小值正好显示整个控件。重写其 setPopupMenuVisible()方法, 设置弹出菜单的显示位置。关键代码如下:

```
public class HorizontalMenu extends JMenu {
    private static final long serialVersionUID = 1943739671316999698L;
```

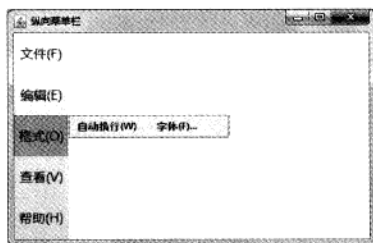


图 14.42 定义纵向的菜单栏



管理器

```

public HorizontalMenu(String label) {
    super(label); //调用父类的构造方法
    JPopupMenu popupMenu = getPopupMenu(); //获得菜单对象的弹出菜单
    popupMenu.setLayout(new BorderLayout(popupMenu, BorderLayout.LINE_AXIS)); //修改布局
}
@Override
public Dimension getMinimumSize() {
    return getPreferredSize(); //将控件的最小范围设置成显示控件的最佳范围
}
@Override
public void setPopupMenuVisible(boolean b) {
    if (b != isPopupMenuVisible()) {
        if ((b == true) && isShowing()) { //如果菜单处于显示状态
            if (getParent() instanceof JPopupMenu) {
                getPopupMenu().show(this, 0, getHeight()); //修改弹出菜单的显示位置
            } else {
                getPopupMenu().show(this, getWidth(), 0); //修改弹出菜单的显示位置
            }
        } else {
            getPopupMenu().setVisible(false); //设置弹出菜单不可见
        }
    }
}
}

```

(2) 编写 HorizontalMenuTest 类，该类继承自 JFrame。在其构造方法中，增加了菜单栏、菜单等，并且修改了内容窗格的布局。关键代码如下：

```

public HorizontalMenuTest() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置框架在退出时的状态
    setBounds(100, 100, 450, 300); //设置框架的大小和显示的位置
    Container contentPane = getContentPane(); //获得内容窗格
    contentPane.setBackground(Color.WHITE); //将内容窗格的背景颜色设置成白色
    JMenuBar menuBar = new JMenuBar(); //建立菜单栏
    menuBar.setLayout(new BorderLayout(menuBar, BorderLayout.PAGE_AXIS)); //修改菜单栏布局
    contentPane.add(menuBar, BorderLayout.WEST); //在内容窗格增加菜单栏
    JMenu fileMenu = new HorizontalMenu("文件(F)"); //增加菜单项
    fileMenu.add("新建(N)"); //增加菜单项
    fileMenu.add("打开(O)..."); //增加菜单项
    fileMenu.add("保存(S)"); //增加菜单项
    fileMenu.add("另存为(A)..."); //增加菜单项
    fileMenu.add("页面设置(U)..."); //增加菜单项
    fileMenu.add("打印(P)..."); //增加菜单项
    fileMenu.add("退出(X)"); //增加菜单项
    menuBar.add(fileMenu); //将菜单增加到菜单栏中
    //省略其他菜单
}

```

技术要点

Java 中菜单栏的主体是菜单，用 JMenu 对象表示。通过重写其 setPopupMenuVisible() 方法，



可以设置菜单项弹出的位置, 该方法的声明如下:

```
public void setPopupMenuVisible(boolean b)
```

参数说明

b: 一个 boolean 值, true 表示菜单可见, false 表示隐藏。

为了让重写后的菜单显示得更加好看, 重写其 `getMinimumSize()` 方法, 该方法用来设置控件的最小值。其声明如下:

```
public Dimension getMinimumSize()
```



Note

多学两招:

当用户选择菜单时, 将触发动作事件, 可以通过调用 `addActionListener()` 方法来监听该事件。在事件监听器中, 可以完成该菜单项需要实现的功能, 如打开新文件、保存文件、退出程序等。通常只监听菜单项而不会监听菜单。

实例 225 复选框与单选按钮菜单项

(实例位置: 配套资源\SL\14\225)

实例说明

复选框和单选按钮为用户的输入提供了便利, 其实它们也可以用在菜单中。复选框菜单项和单选按钮菜单项分别对应于复选框和单选按钮, 其使用的方式也是类似的。本实例将演示它们的用法。实例的运行效果如图 14.43 所示。

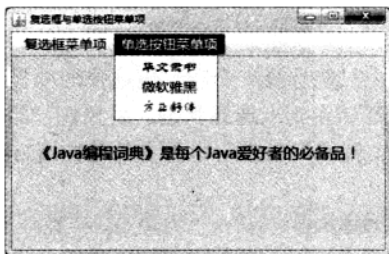


图 14.43 复选框与单选按钮菜单项

实现过程

(1) 编写类 `FontChooser`, 该类继承了 `JFrame`。在框架的菜单栏中有两个菜单项, 分别用来演示复选框菜单项和单选按钮菜单项。在面板中显示带字符串的标签。

(2) 使用匿名类创建 `listener` 对象, 用来监听复选框菜单项被选择的事件, 用于实现是否让字体变粗和斜体的功能。关键代码如下:

```
private ActionListener listener = new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int mode = 0; //利用整数保存字体的状态
        if (bold.isSelected()) {
            mode += Font.BOLD; //如果加粗复选框菜单项被选择, 则让 mode 值发生变化
        }
    }
}
```




Note

```
}  
if (italic.isSelected()) {  
    mode += Font.ITALIC;    //如果斜体复选框菜单项被选择, 则让 mode 值发生变化  
}  
Font font = label.getFont();    //获得标签正在使用的字体  
label.setFont(new Font(font.getName(), mode, font.getSize()));    //更新标签的字体  
}  
};
```

(3) 对于不同的单选按钮菜单项, 其实现的功能是类似的, 在此选择显示为“华文隶书”的单选按钮进行讲解。编写 `do_radioButtonMenuItem_actionPerformed()` 方法, 用于监听单选按钮菜单项被选择的事件。关键代码如下:

```
protected void do_radioButtonMenuItem_actionPerformed(ActionEvent e) {  
    Font font = label.getFont();    //获得标签正在使用的字体  
    label.setFont(new Font("华文隶书", font.getStyle(), font.getSize()));    //更新标签的字体  
}
```

技术要点

`JCheckBoxMenuItem` 代表可以被选定或取消选定的菜单项。如果被选定, 菜单项的旁边通常会出现一个复选标记。如果未被选定或被取消选定, 菜单项的旁边就没有复选标记。像常规菜单项一样, 复选框菜单项可以有与之关联的文本或图标, 或者二者兼而有之。本实例使用以字符串为参数的构造方法创建复选框菜单项, 该方法的声明如下:

```
JCheckBoxMenuItem(String text)
```

参数说明

text: `CheckBoxMenuItem` 的文本。

`JRadioButtonMenuItem` 是一个单选按钮菜单项的实现。`JRadioButtonMenuItem` 是属于一组菜单项中的一个菜单项, 该组中只能选择一个项。被选择的项显示其选择状态。选择此项的同时, 其他任何以前被选择的项都切换到未选择状态。要控制一组单选按钮菜单项的选择状态, 请使用 `ButtonGroup` 对象。本实例使用以字符串为参数的构造方法创建单选按钮菜单项, 该方法的声明如下:

```
JRadioButtonMenuItem(String text)
```

参数说明

text: `RadioButtonMenuItem` 的文本。

实例 226 包含图片的弹出菜单

(实例位置: 配套资源\SL\14\226)

实例说明

除了固定的菜单栏, 还有一种比较特殊的菜单: 弹出菜单。在 Windows 操作系统的桌面上, 单击鼠标右键就会出现一个弹出式菜单。该菜单可以用来排列桌面图片、创建文件或文件夹等。本实例将演示如何在 Swing 框架中创建包含图片的弹出式菜单。实例的运行效果如图 14.44 所示。



图 14.44 包含图片的弹出菜单



Note

实现过程

(1) 编写 `PopupMenuTest` 类，该类继承自 `JFrame`。在框架中只增加了一个标签，用于显示用户在弹出式菜单上选择的的操作。关键代码如下：

```
public PopupMenuTest() {
    //省略设置框架属性的代码
    JPopupMenu popupMenu = new JPopupMenu(); //创建弹出式菜单
    contentPane.setComponentPopupMenu(popupMenu); //为面板增加弹出式菜单
    JMenuItem cut = new JMenuItem("\u526A\u5207"); //创建新菜单项
    cut.setIcon(new ImageIcon(PopupMenuTest.class.getResource("/images/cut.png")));
    cut.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置菜单项字体
    cut.addActionListener(listener); //增加监听
    popupMenu.add(cut); //增加菜单项
    //省略其他菜单项和标签的代码
}
```

(2) 使用匿名类创建 `listener` 对象，用来监听弹出式菜单的菜单项被选择的事件，用于实现改变标签文本的功能。关键代码如下：

```
private ActionListener listener = new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        label.setText(e.getActionCommand()); //设置标签的文本为用户选择的操作
    }
};
```

技术要点

`JPopupMenu` 可以实现弹出菜单，弹出菜单是一个可弹出并显示一系列选项的小窗口。`JPopupMenu` 用于用户在菜单栏上选择菜单项时显示的菜单，还用于当用户选择菜单项并激活它时显示的“右拉式 (pull-right)”菜单。最后，`JPopupMenu` 还可以在想让菜单显示的任何其他位置使用。例如，当用户在指定区域中右击时。创建弹出式菜单的语法如下：

```
JPopupMenu popup = new JPopupMenu();
```

在创建完弹出式菜单后，需要调用其所在控件的 `setComponentPopupMenu()` 方法来使用该菜单。方法的声明如下：

```
public void setComponentPopupMenu(JPopupMenu popup)
```

参数说明

`popup`: 分配给此控件的弹出菜单，可以为 `null`。



实例 227 工具栏的实现与应用

(实例位置: 配套资源\SL\14\227)



Note

实例说明

除了标准菜单和弹出式菜单,还有一种让用户使用鼠标选择操作的方法。对于一些功能非常复杂的软件,可以将一些常用的操作放置在一个工具栏中方便用户使用。本实例将演示如何使用工具栏。实例的运行效果如图 14.45 所示。

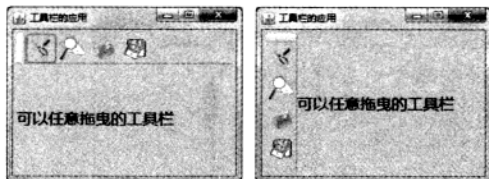


图 14.45 在窗体中添加工具栏

实现过程

编写 `ToolBarTest` 类,该类继承自 `JFrame`。在其构造方法中,创建了工具条并增加了 4 个按钮。构造方法的核心代码如下:

```
public ToolBarTest() {  
    //省略设置框架属性的代码  
    JToolBar toolBar = new JToolBar();  
    contentPane.add(toolBar, BorderLayout.NORTH);  
    JButton cutButton = new JButton("");  
    cutButton.setToolTipText("\u526A\u5207");  
    cutButton.setIcon(new ImageIcon(ToolBarTest.class.getResource("/images/cut.png")));  
    toolBar.add(cutButton);  
    //省略其他按钮和标签的代码  
}  
//创建工具栏  
//设置工具栏的布局  
//新建一个按钮  
//为按钮增加提示信息  
//将按钮增加到工具栏上
```

技术要点

`JToolBar` 提供了一种快速访问程序常用功能的方法。工具栏的特殊之处在于可以随意地移动。其常用方法如表 14.15 所示。

表 14.15 工具栏的常用方法

方 法 名	作 用
<code>add(Component comp)</code>	在工具栏中增加控件
<code>addSeparator()</code>	将默认大小的分隔符添加到工具栏的末尾
<code>setToolTipText(String text)</code>	注册要在工具提示中显示的文本

脚下留神:

只有在采用边框布局和支持 `NORTH`、`EAST`、`SOUTH` 和 `WEST` 方向常量的布局管理器时才可以用工具条。

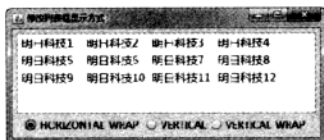


实例 228 修改列表项显示方式

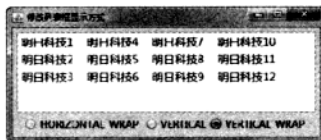
(实例位置: 配套资源\SL\14\228)

实例说明

当可供用户选择的选项比较多时,使用单选按钮或复选框会占用较多的空间,此时可以考虑使用列表。列表可以将若干选项组织起来,根据显示的列表项个数可以调节其占用的空间。本实例将演示列表布局的用法。实例的运行效果如图 14.46 所示。



(a) 水平排列



(b) 垂直排列

图 14.46 修改列表项的显示方式

实现过程

(1) 编写类 `JListTest`, 该类继承自 `JFrame`。在框架中包括了一个列表和一个单选按钮组。单选按钮组包含了 3 个单选按钮, 用来表示列表的 3 种布局方式。

(2) 编写 `do_this_windowActivated()` 方法, 用来监听窗体激活事件, 并在该方法中设置了列表的数据。关键代码如下:

```
protected void do_this_windowActivated(WindowEvent e) {
    String[] listData = new String[12];           //创建一个含有 12 个元素的数组
    for(int i=0;i<listData.length;i++) {
        listData[i] = "明日科技" + (i+1);        //为数组中各个元素赋值
    }
    list.setListData(listData);                  //为列表增加列表项
}
```

(3) 编写 `do_radioButton1_actionPerformed()` 方法, 用来监听单选按钮 `radioButton1` 被选中事件, 该方法用来修改列表的布局方式并更新界面。关键代码如下:

```
protected void do_radioButton1_actionPerformed(ActionEvent e) {
    list.setLayoutOrientation(JList.HORIZONTAL_WRAP); //修改列表的布局方式
    scrollPane.revalidate();                          //更新界面
}
```

技术要点

`JList` 类是显示对象列表并且允许用户选择一个或多个项的控件。在为列表增加列表项时, 可以使用对象数组或列表模型。列表包含了很多方法, 本实例使用的方法如表 14.16 所示。

表 14.16 列表常用方法

方法名	作用
<code>setLayoutOrientation(int layoutOrientation)</code>	定义布置列表单元的方式
<code>setListData(Object[] listData)</code>	根据一个对象数组构造只读 <code>ListModel</code> , 然后对此模型调用 <code>setModel</code>
<code>setVisibleRowCount(int visibleRowCount)</code>	根据不同的布局方式, 设置可见的行或列数





指点迷津:

使用对象数组作为列表中的列表项意味着列表不仅可以用来显示字符串,也可以显示其他类型的对象。

列表有 3 种常见的布局方式,使用 3 个不同的域变量表示,其说明如表 14.17 所示。

表 14.17 列表的布局方式

常 量 名	作 用
HORIZONTAL_WRAP	指示“报纸样式”布局,单元按先水平方向后垂直方向排列
VERTICAL	指示单个列中单元的垂直布局;默认布局
VERTICAL_WRAP	指示“报纸样式”布局,单元按先垂直方向后水平方向排列

实例 229 列表项与提示信息

(实例位置: 配套资源\SL\14\229)

实例说明

有时出于节约空间等原因,列表项并不能很好地表明其含义,此时如果为列表项增加提示信息会很有用的。然而默认的列表项提示信息方法有些复杂,本实例将自定义一个能显示提示信息的列表。实例的运行效果如图 14.47 所示。

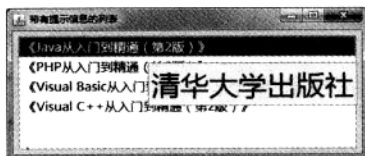


图 14.47 列表元素的提示信息

实现过程

(1) 编写 ToolTipList 类,该类继承自 JList。该类定义了一个以二维数组为参数的构造方法并重写了 getToolTipText()方法。在重写的方法中,为不同的列表项指定了不同的提示信息。关键代码如下:

```
public class ToolTipList extends JList {
    private static final long serialVersionUID = -5334116242803068391L; //随机的序列化标识
    private Object[][] data; //定义一个二维数组保存传递的参数
    public ToolTipList(Object[][] data) {
        this.data = data;
        Object[] listData = new Object[data.length]; //定义一个一维数组保存列表项
        for (int i = 0; i < listData.length; i++) {
            listData[i] = data[i][0]; //获得列表项
        }
        setListData(listData); //设置列表项
    }
    @Override
    public String getToolTipText(MouseEvent event) {
        int index = locationToIndex(event.getPoint()); //获得鼠标所在位置的列表项的索引
        if (index > -1) {
            return "<html><font face=微软雅黑 size=16 color=red>" + data[index][1] + "</font></html>"; //返回提示信息
        }
    }
}
```




```

    } else {
        return super.getToolTipText(event);
    }
}
}

```

指点迷津:

可以使用 HTML 标签来修改显示的提示信息的字体、大小、颜色等。



Note

(2) 编写 `do_this_windowActivated()` 方法, 用来监听窗体激活事件, 并在该方法中初始化了一个自定义列表对象并将其添加到滚动面板中。关键代码如下:

```

protected void do_this_windowActivated(WindowEvent e) {
    String[][] data = new String[4][2];           //定义一个4行2列的二维数组
    data[0][0] = "《Java 从入门到精通 (第2版)》"; //初始化数据
    data[0][1] = "清华大学出版社";               //初始化数据
    data[1][0] = "《PHP 从入门到精通 (第2版)》"; //初始化数据
    data[1][1] = "清华大学出版社";               //初始化数据
    data[2][0] = "《Visual Basic 从入门到精通 (第2版)》"; //初始化数据
    data[2][1] = "清华大学出版社";               //初始化数据
    data[3][0] = "《Visual C++从入门到精通 (第2版)》"; //初始化数据
    data[3][1] = "清华大学出版社";               //初始化数据
    JList list = new ToolTipList(data);            //创建自定义列表
    list.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置列表项的字体
    scrollPane.setViewportViewView(list);             //将列表添加到滚动面板中
}

```

指点迷津:

可以在定义二维数组时直接将其初始化, 本实例为了让读者阅读方便才这样写的, 并不实用。

技术要点

通常情况下, 各列表项的提示信息是不同的。因此, 可以使用一个二维数组或映射来保存列表项及提示信息, 然后通过重写 `getToolTipText()` 方法来为不同的列表项选择不同的提示信息。该方法的声明如下:

```
public String getToolTipText(MouseEvent event)
```

参数说明

event: 用于获取工具提示文本的 `MouseEvent`。

实例 230 表头与列的高度设置

(实例位置: 配套资源\SL\14\230)

实例说明

在默认情况下, 表格中表头和表体的单元格高度是固定的。如果要修改单元格的外观, 如放大字体, 则会隐藏部分表格的内容, 此时就需要修改单元格的高度。Swing 中的表格对于表



头和表体的处理方式是不同的。本实例将演示如何自定义表头和表体的高度。实例的运行效果如图 14.48 所示。

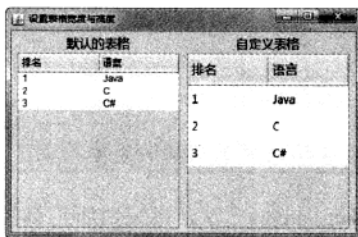


图 14.48 自定义表头和表体的高度

实现过程

(1) 编写类 `ResizeTableTest`，该类继承了 `JFrame`。在框架中包含两个表格，分别用来演示不同的设置方式。关键代码如下：

```
public ResizeTableTest() {
    //省略与自定义表格无关的其他代码
    table2 = new JTable();
    table2.setFont(new Font("微软雅黑", Font.PLAIN, 14));           //修改表体的字体
    table2.setRowHeight(35);                                         //修改表体的高度
    JTableHeader header = table2.getTableHeader();                  //获得表头
    header.setFont(new Font("微软雅黑", Font.PLAIN, 16));           //修改表头的字体
    header.setPreferredSize(new Dimension(header.getWidth(), 40)); //修改表头的高度
    scrollPane2.setViewportView(table2);                                //显示表
}
```

脚下留神：

在设置表格的高度时要考虑边框的大小，通常为 1 像素，可以使用 `setRowMargin()` 方法修改。

(2) 编写 `do_this_windowActivated()` 方法，用来监听窗体激活事件。在该方法中，定义了一个默认的表格模型，并让两个表格同时使用这个表格模型。关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultTableModel model = new DefaultTableModel();           //创建表格模型
    model.setRowCount(0);                                         //将表格模型中的数据清空
    model.setColumnIdentifiers(new Object[] { "排名", "语言" }); //设置表头
    model.addRow(new Object[] { "1", "Java" });                 //增加行
    model.addRow(new Object[] { "2", "C" });                    //增加行
    model.addRow(new Object[] { "3", "C#" });                   //增加行
    table1.setModel(model);                                       //为表格设置表格模型
    table2.setModel(model);                                       //为表格设置表格模型
}
```

技术要点

`JTable` 控件用来显示和编辑常规二维单元表。`JTable` 有很多用来自定义其呈现和编辑的工具，同时提供了这些功能的默认设置，从而可以轻松地设置简单表。本实例使用其 `setRowHeight()`



方法来设置所有行的高度，该方法的声明如下：

```
public void setRowHeight(int rowHeight)
```

参数说明

rowHeight: 新的行高。

指点迷津：

setRowHeight()方法还有一个重载版本，可以设置指定行的高度。

对于表头，并没有直接修改其高度的方法，为此需要使用 getTableHeader() 方法获得 JTableHeader 对象。该方法的声明如下：

```
public JTableHeader getTableHeader()
```

然后使用从 JComponent 继承的 setPreferredSize() 方法来设置其大小，该方法的声明如下：

```
public void setPreferredSize(Dimension preferredSize)
```

参数说明

preferredSize: 新的首选大小。

多学两招：

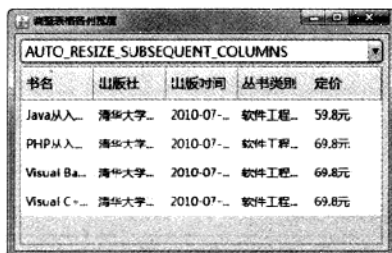
如果不希望修改表头的宽度，则可以先使用 getWidth() 方法获得表头的宽度并进行保存。

实例 231 调整表格各列的宽度

(实例位置：配套资源\SL\14\231)

实例说明

默认情况下，当用户调整一列的大小时，其他各列的大小也会随之改变。以便将表格的各列显示在框架中。为了适应不同的情况，表格提供了其他几个常量来设置表格列的变化方式。本实例将演示它们的用法。实例的运行效果如图 14.49 所示。



(a) 默认模式



(b) 禁用模式

图 14.49 调整表格各列的宽度

实现过程

(1) 编写 do_this_windowActivated() 方法，用来监听窗体激活事件，并在该方法中使用表格模型初始化表格的数据。关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
```

```
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel(); //获得表格模型
```



Note



```
tableModel.setRowCount(0); //将表格模型中的数据清空
tableModel.setColumnIdentifiers(new Object[] { "书名", "出版社", "出版时间", "丛书类别",
"定价" }); //设置表头
tableModel.addRow(new Object[] { "Java 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "59.8 元" }); //增加行
tableModel.addRow(new Object[] { "PHP 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
tableModel.addRow(new Object[] { "Visual Basic 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
tableModel.addRow(new Object[] { "Visual C++从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
table.setModel(tableModel); //更新表格模型
}
```

多学两招:

可以使用表格模型的 `setRowCount()` 方法来清空表格中的数据。

(2) 编写 `do_comboBox_actionPerformed()` 方法, 用来监听组合框动作事件, 并在该方法中根据用户选择的组合框项来重新设置表格列的调整模式。关键代码如下:

```
protected void do_comboBox_actionPerformed(ActionEvent e) {
//使用映射来保存组合框中字符串与表格调整模式之间的对应关系
Map<String, Integer> columnModel = new HashMap<String, Integer>();
columnModel.put("AUTO_RESIZE_ALL_COLUMNS", JTable.AUTO_RESIZE_ALL_COLUMNS);
columnModel.put("AUTO_RESIZE_LAST_COLUMN", JTable.AUTO_RESIZE_LAST_COLUMN);
columnModel.put("AUTO_RESIZE_NEXT_COLUMN", JTable.AUTO_RESIZE_NEXT_COLUMN);
columnModel.put("AUTO_RESIZE_OFF", JTable.AUTO_RESIZE_OFF);
columnModel.put("AUTO_RESIZE_SUBSEQUENT_COLUMNS", JTable.AUTO_RESIZE_
SUBSEQUENT_COLUMNS);
String text = (String) comboBox.getSelectedItemAt(); //获得用户的选择项
table.setAutoResizeMode(columnModel.get(text)); //设置调整模式
}
```

技术要点

当表格中一列的大小发生变化时, `JTable` 类共提供了 5 种模式来调节其他列的变化。这些方式的说明如表 14.18 所示。

表 14.18 表格常用域说明

域 名	作 用
<code>AUTO_RESIZE_ALL_COLUMNS</code>	在所有的调整大小操作中, 按比例调整所有的列
<code>AUTO_RESIZE_LAST_COLUMN</code>	在所有的调整大小操作中, 只对最后一列进行调整
<code>AUTO_RESIZE_NEXT_COLUMN</code>	在 UI 中调整了一个列时, 对其下一列进行相反方向的调整
<code>AUTO_RESIZE_OFF</code>	不自动调整列的宽度; 使用滚动条
<code>AUTO_RESIZE_SUBSEQUENT_COLUMNS</code>	在 UI 调整中, 更改后续列以保持总宽度不变; 此为默认行为



多学两招:

表格中的各个域都是静态的,因此可以直接使用 JTable 类进行调用。

为了使用这些模式,需要使用 setAutoResizeMode()方法来进行设置,该方法的声明如下:

```
public void setAutoResizeMode(int mode)
```

参数说明

mode: 表 14.18 中的一个域。



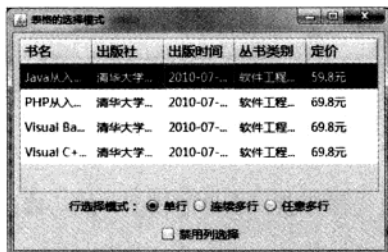
Note

实例 232 设置表格的选择模式

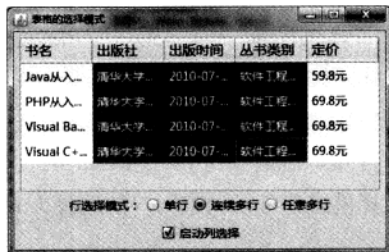
(实例位置: 配套资源\SL\14\232)

实例说明

对于一个二维的表格而言,其选择模式有很多种。以行为例,可以选择一行、连续几行、任意几行,对于列同理。此外,对于单元格也可以设置成选择一个单元格、选择一个连续区域的单元格或选择一个不连续区域的单元格等。本实例将演示它们的用法。实例的运行效果如图 14.50 所示。



(a) 单行选择



(b) 多行选择

图 14.50 设置表格的选择模式

实现过程

(1) 编写类 TableSelectModeTest, 该类继承了 JFrame。在框架中包含一个表格、一个单选按钮组和一个复选框。单选按钮组包括“单行”、“连续多行”和“任意多行”3个单选按钮。通过单选按钮和复选框的组合,来实现不同的选择方式。

(2) 编写 do_this_windowActivated()方法,用来监听窗体激活事件,并在该方法中使用表格模型初始化表格的数据。关键代码如下:

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel(); //获得表格模型
    tableModel.setRowCount(0); //将表格模型中的数据清空
    tableModel.setColumnIdentifiers(new Object[] { "书名", "出版社", "出版时间", "丛书类别",
"定价" }); //设置表头
    tableModel.addRow(new Object[] { "Java 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "59.8 元" }); //增加行
    tableModel.addRow(new Object[] { "PHP 从入门到精通 (第 2 版)", "清华大学出版社",
```



```
"2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
tableModel.addRow(new Object[] { "Visual Basic 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
tableModel.addRow(new Object[] { "Visual C++从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
table.setModel(tableModel); //更新表格模型
}
```

(3) 编写 `do_rowRadioButton1_actionPerformed()` 方法, 用来监听选中“单行”单选按钮事件。在该方法中, 设置了表格行的选择模式是选择单行。关键代码如下:

```
protected void do_rowRadioButton1_actionPerformed(ActionEvent e) {
    table.getSelectionModel().setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
}
```

(4) 编写 `do_checkBox_actionPerformed()` 方法, 用来监听复选框被选中事件。在该方法中, 启动或禁用表格的列选择功能, 并修改了复选框的文本。关键代码如下:

```
protected void do_checkBox_actionPerformed(ActionEvent e) {
    if (checkBox.isSelected()) {
        checkBox.setText("启动列选择"); //修改复选框的文本内容为“启动列选择”
        table.setColumnSelectionAllowed(true); //启动列选择
    } else {
        checkBox.setText("禁用列选择"); //修改复选框的文本内容为“禁止列选择”
        table.setColumnSelectionAllowed(false); //禁止列选择
    }
}
```

技术要点

利用选择模式, 可以调整用户的选择方式。首先需要获得表格的模型, 然后修改其选择模式, 代码如下:

```
table.getSelectionModel().setSelectionMode(mode);
```

`mode` 是 `ListSelectionModel` 接口定义的选择模式, 其详细说明如表 14.19 所示。

表 14.19 列表选择模式常量

常量名	作用
<code>MULTIPLE_INTERVAL_SELECTION</code>	一次选择一个或多个连续的索引范围
<code>SINGLE_INTERVAL_SELECTION</code>	一次选择一个连续的索引范围
<code>SINGLE_SELECTION</code>	一次选择一个列表索引

指点迷津:

表格的默认选择模式是 `SINGLE_SELECTION`。

对于列的选择, 在默认情况下是禁用的, 需要使用 `setColumnSelectionAllowed()` 方法启用。该方法的声明如下:

```
public void setColumnSelectionAllowed(boolean columnSelectionAllowed)
```

参数说明

`columnSelectionAllowed`: 如果此模型允许列选择, 则为 `true`。



实例 233 为表头增添提示信息

(实例位置: 配套资源\SL\14\233)

实例说明

为了节约空间,表头通常使用一些缩写形式。作为对表头含义的补充说明,可以为表头添加提示信息。默认情况下,可以使用 `setToolTipText()` 方法设置统一的提示信息,这通常被用来设置默认提示信息。本实例将实现让不同的表头显示不同的提示信息。实例的运行效果如图 14.51 所示。

Title	Press	Date	Category	Price
Java从入...	清华大学...	2010...		8元
PHP从入...	清华大学...	2010...		8元
Visual Ba...	清华大学...	2010-07...	软件工程...	69.8元
Visual C+...	清华大学...	2010-07...	软件工程...	69.8元

(a) Date 列的提示信息

Title	Press	Date	Category	Price
Java从入...	清华大...		软件工程...	59.8元
PHP从入...	清华大...		软件工程...	69.8元
Visual Ba...	清华大...	2010-07...	软件工程...	69.8元
Visual C+...	清华大...	2010-07...	软件工程...	69.8元

(b) Press 列的提示信息

图 14.51 实例运行效果

实现过程

(1) 编写类 `ToolTipHeader`, 该类继承了 `JTableHeader`, 并重写了其 `getToolTipText()` 方法, 实现根据鼠标所在位置来返回适当的提示信息的功能。关键代码如下:

```
public class ToolTipHeader extends JTableHeader {
    private static final long serialVersionUID = 6694115973725345619L;
    private String[] toolTips;
    public ToolTipHeader(TableColumnModel model) {
        super(model); //初始化表头
    }
    public void setToolTips(String[] toolTips) {
        this.toolTips = toolTips; //获得提示信息数组
    }
    @Override
    public String getToolTipText(MouseEvent event) {
        int index = columnAtPoint(event.getPoint()); //获得鼠标所在位置
        if (index != -1) { //如果鼠标位于表头
            return "<html><font face=微软雅黑 size=16 color=red>" + toolTips[index] +
                "</font></html>"; //返回鼠标所在位置的提示信息
        } else {
            return ""; //返回空字符串
        }
    }
}
```



**多学两招:**

在窗体中也可以使用 HTML 标签来修改显示的提示信息的字体、大小、颜色等。

(2) 编写类 `ToolTipHeaderTableExample`, 该类继承了 `JFrame`。在构造方法中, 初始化了表格数据, 并为其设置了自定义的表头。构造方法的核心代码如下:

Note

```
public ToolTipHeaderTableExample() {
    //省略与表格无关的代码
    table = new JTable(); //创建表格对象
    table.setFont(new Font("微软雅黑", Font.PLAIN, 14)); //设置表体的字体
    table.setRowHeight(30); //设置表体的高度
    scrollPane.setViewportView(table); //在滚动面板上显示表格
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel(); //获得表格模型
    tableModel.setRowCount(0); //将表格模型中的数据清空
    tableModel.setColumnIdentifiers(new Object[] { "Title", "Press", "Date", "Category", "Price" }); //设置表头

    tableModel.addRow(new Object[] { "Java 从入门到精通 (第 2 版)", "清华大学出版社",
    "2010-07-01", "软件工程师入门丛书", "59.8 元" }); //增加行
    tableModel.addRow(new Object[] { "PHP 从入门到精通 (第 2 版)", "清华大学出版社",
    "2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
    tableModel.addRow(new Object[] { "Visual Basic 从入门到精通 (第 2 版)", "清华大学出版社",
    "2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
    tableModel.addRow(new Object[] { "Visual C++从入门到精通 (第 2 版)", "清华大学出版社",
    "2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
    table.setModel(tableModel); //更新表格模型
    String[] tips = { "书名", "出版社", "出版时间", "丛书类别", "定价" }; //创建提示信息数组
    ToolTipHeader header = new ToolTipHeader(table.getColumnModel()); //创建新表头
    header.setFont(new Font("微软雅黑", Font.PLAIN, 16)); //设置表头的字体
    header.setPreferredSize(new Dimension(header.getWidth(), 30)); //设置表头的高度
    header.setToolTips(tips); //设置提示信息数组
    table.setTableHeader(header); //设置表头
}
```

技术要点

在 `JTableHeader` 类中提供了 `getToolTipText()` 方法, 它可以用来为不同的表头设置不同的提示信息。该方法的声明如下:

```
public String getToolTipText(MouseEvent event)
```

参数说明

event: 标识正确渲染器和正确提示的事件位置。

指点迷津:

`getToolTipText()` 方法还有一个没有参数的重载形式。

为了能够让不同的表头显示不同的提示信息, 需要使用 `columnAtPoint()` 方法获得鼠标所在位置的索引。该方法的声明如下:

```
public int columnAtPoint(Point point)
```

参数说明

point: 鼠标所在的位置。



指点迷津:

如果鼠标所在位置位于表格外, 则返回-1。

实例 234 单元格的粗粒度排序

(实例位置: 配套资源\SL\14\234)



Note

实例说明

在使用表格时, 会出现根据不同的列来对表格进行排序的情况。例如可以根据价格的不同来对各种书籍进行排序。本实例将演示如何实现简单的排序功能。实例的运行效果如图 14.52 所示。

书名	出版社	出版时间	丛书类别	定价
Java从入门到精通 (第2版)	清华大学出版社	2010-07-01	软件工程	59.8元
PHP从入门到精通 (第2版)	清华大学出版社	2010-07-01	软件工程	69.8元
Visual Basic从入门到精通 (第2版)	清华大学出版社	2010-07-01	软件工程	69.8元
Visual C++从入门到精通 (第2版)	清华大学出版社	2010-07-01	软件工程	69.8元

(a) 升序排序

书名	出版社	出版时间	丛书类别	定价
PHP从入门到精通 (第2版)	清华大学出版社	2010-07-01	软件工程	69.8元
Visual Basic从入门到精通 (第2版)	清华大学出版社	2010-07-01	软件工程	69.8元
Visual C++从入门到精通 (第2版)	清华大学出版社	2010-07-01	软件工程	69.8元
Java从入门到精通 (第2版)	清华大学出版社	2010-07-01	软件工程	59.8元

(b) 降序排序

图 14.52 单元格的粗粒度排序

多学两招:

在初始化表格数据时, 可以将 `Object` 类型的数据添加到表格中。然而排序时, 如果当前列中的数据实现了 `Comparable` 接口, 则根据其实现的方法进行排序; 否则是将列中的数据转换成字符串来排序的, 因此不能进行细粒度排序, 如颜色的深浅。

实现过程

编写 `do_this_windowActivated()` 方法, 用来监听窗体激活事件, 并在该方法中初始化了表格中的数据并启动了排序功能。关键代码如下:

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel(); //获得表格模型
    tableModel.setRowCount(0); //将表格模型中的数据清空
    tableModel.setColumnIdentifiers(new Object[] { "书名", "出版社", "出版时间", "丛书类别",
"定价" }); //设置表头
    tableModel.addRow(new Object[] { "Java 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "59.8 元" }); //增加行
    tableModel.addRow(new Object[] { "PHP 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
    tableModel.addRow(new Object[] { "Visual Basic 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
    tableModel.addRow(new Object[] { "Visual C++从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" }); //增加行
    table.setModel(tableModel); //更新表格模型
    table.setAutoCreateRowSorter(true); //启动排序功能
}
```



技术要点

默认情况下表格是不支持排序的,然而使用 `setAutoCreateRowSorter()` 方法就可以让表格根据列来排序。该方法的声明如下:

```
public void setAutoCreateRowSorter(boolean autoCreateRowSorter)
```

参数说明

`autoCreateRowSorter`: 是否应该自动创建 `RowSorter`。

指点迷津:

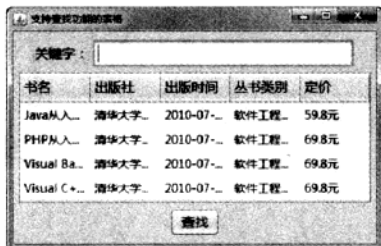
`setAutoCreateRowSorter()` 方法仅实现了粗粒度的排序,要想实现细粒度排序需要使用 `TableRowSorter` 类。

实例 235 实现表格的查找功能

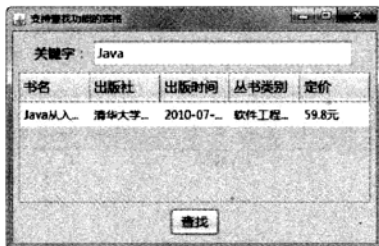
(实例位置: 配套资源\SL\14\235)

实例说明

对于数据量较大的表格,通常会为用户提供查找功能,这可以让用户快速找到自己需要的行。如果没有符合用户需求的行则进行提示,节约用户的时间。本实例将演示如何在表格中实现这个功能。实例的运行效果如图 14.53 所示。



(a) 查找前



(b) 查找后

图 14.53 表格的查找功能

实现过程

(1) 编写类 `SearchTable`, 该类继承了 `JFrame`。在框架中包含了一个文本域、一个表格和一个“查找”按钮。文本域用于获得用户输入的关键字,“查找”按钮用于在表格中查找用户输入的关键字。

(2) 编写 `do_this_windowActivated()` 方法,用来监听窗体激活事件。在该方法中,初始化了表格的数据,并为表格设置了 `RowSorter`。关键代码如下:

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel(); //获得表格模型
    tableModel.setRowCount(0); //将表格模型中的数据清空
    tableModel.setColumnIdentifiers(new Object[] { "书名", "出版社", "出版时间", "丛书类别",
        "定价" }); //设置表头
    tableModel.addRow(new Object[] { "Java 从入门到精通 (第 2 版)", "清华大学出版社",
```



```

"2010-07-01", "软件工程师入门丛书", "59.8 元" });           //增加行
    tableModel.addRow(new Object[] { "PHP 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" });           //增加行
    tableModel.addRow(new Object[] { "Visual Basic 从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" });           //增加行
    tableModel.addRow(new Object[] { "Visual C++从入门到精通 (第 2 版)", "清华大学出版社",
"2010-07-01", "软件工程师入门丛书", "69.8 元" });           //增加行
    sorter.setModel(tableModel);                               //为 TableRowSorter 对象增加表格模型
    table.setRowSorter(sorter);                                //设置 RowSorter
}

```



Note

多学两招:

为 TableRowSorter 设置了表格模型之后,就不需要再对表格设置表格模型了。

(3) 编写 do_button_actionPerformed()方法,用来监听单击“查找”按钮事件。在该方法中,使用用户在文本域输入的关键字过滤表格内容。关键代码如下:

```

protected void do_button_actionPerformed(ActionEvent e) {
    sorter.setRowFilter(RowFilter.regexFilter(textField.getText())); //实现过滤
}

```

regexFilter()方法支持参数为 null 和空字符串,因此不用对用户的输入进行校验。

技术要点

RowFilter 用于从模型中过滤条目,使得这些条目不会在视图中显示。例如,一个与 JTable 关联的 RowFilter 可能只允许包含带指定字符串的列的那些行。条目的含义取决于控件类型。例如,当过滤器与 JTable 关联时,一个条目对应于一行;当过滤器与 JTree 关联时,一个条目对应于一个节点。本实例使用其 regexFilter()方法来实现文本过滤。该方法的声明如下:

```
public static <M,I> RowFilter<M,I> regexFilter(String regex,int... indices)
```

参数说明

- ☒ regex: 在其上进行过滤的正则表达式。
- ☒ indices: 要检查的值的索引如果没有提供,则计算所有的值。

实例 236 应用网格布局设计计算器窗体

(实例位置: 配套资源\SL\14\236)

实例说明

在包含两个及以上控件的容器中,肯定要涉及如何布局的问题。Java 中使用布局管理器来管理容器中控件的位置和大小。每个容器都有自己的默认布局,读者可以根据需要进行修改。本实例演示如何使用网格布局。实例的运行效果如图 14.54 所示。

实现过程

编写 Calculator 类,该类继承了 JFrame 类,实现了一个



图 14.54 定义计算器窗体



简单的计算器。关键代码如下：

```
public Calculator() {  
    setTitle("\u8BA1\u7B97\u5668");           //设置框架标题  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //设置框架关闭属性  
    setLocationByPlatform(true);               //设置框架的位置由操作系统决定  
    contentPane = new JPanel();  
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
    contentPane.setLayout(new BorderLayout(0, 0)); //设置框架采用边框布局  
    setContentPane(contentPane);  
    JPanel displayPanel = new JPanel();  
    contentPane.add(displayPanel, BorderLayout.NORTH); //将保存文本域的面板添加到框架北部  
    //省略文本域相关代码  
    JPanel buttonPanel = new JPanel();  
    contentPane.add(buttonPanel, BorderLayout.CENTER); //将保存按钮的面板添加到框架中央  
    buttonPanel.setLayout(new GridLayout(4, 4, 5, 5)); //设置保存按钮的面板为网格布局  
    //省略按钮相关代码  
    pack();                                       //重新绘制框架使其容纳所有控件  
}
```

指点迷津：

向网格布局的容器添加控件时，顺序是从第一行第一列开始，依次填满，然后开始填第二行第一列。

技术要点

网格布局类似一个表格，其每个单元格的大小是相同的。图 14.54 显示的计算器程序就使用了网格布局来管理其按钮。当框架缩放时，按钮也会发生缩放，但是各个按钮还是大小相同的。通常在创建 GridLayout 对象时指明行数和列数，构造方法 GridLayout(int rows, int cols, int hgap, int vgap)的各参数说明如表 14.20 所示。

表 14.20 网格布局构造方法参数说明

参 数 名	说 明
rows	新网格布局对象的行数
cols	新网格布局对象的列数
hgap	各格子之间的水平距离
vgap	各格子之间的垂直距离

第15章

多线程

本章读者可以学到如下实例：

- » 实例 237 查看线程的运行状态
- » 实例 238 查看 JVM 中的线程名
- » 实例 239 查看和修改线程优先级
- » 实例 240 休眠当前线程
- » 实例 241 终止指定线程
- » 实例 242 线程的插队运行
- » 实例 243 使用方法实现线程同步
- » 实例 244 使用特殊域变量实现线程同步
- » 实例 245 简单的线程通信
- » 实例 246 新建有返回值的线程
- » 实例 247 使用线程池优化多线程编程
- » 实例 248 哲学家的就餐问题



实例 237 查看线程的运行状态

(实例位置: 配套资源\SL\15\237)



Note

实例说明

线程共有以下 6 种状态: 新建、运行(可运行)、阻塞、等待、计时等待和终止。当使用 new 操作符创建新线程时, 线程处于“新建”状态。当调用 start()方法时, 线程处于运行(可运行)状态。当线程需要获得对象的内置锁, 而该锁正被其他线程拥有, 线程处于阻塞状态。当线程等待其他线程通知调度表可以运行时, 该线程处于等待状态。对于一些含有时间参数的方法, 如 Thread 类的 sleep()方法, 可以使线程处于计时等待状态。当 run()方法运行完毕或出现异常时, 线程处于终止状态。本实例实现查看线程的运行状态。实例的运行效果如图 15.1 所示。

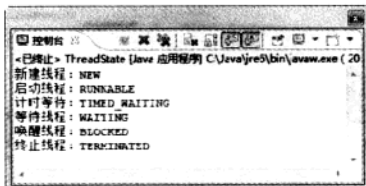


图 15.1 查看线程的运行状态

脚下留神:

多线程程序的运行与底层操作系统密切相关, 如果读者的运行结果与图片不同是很正常的。

实现过程

(1) 编写类 ThreadState, 该类实现了 Runnable 接口。在该类中定义了 3 个方法: waitForASecond()方法用于将当前线程暂时等待 0.5 秒, waitForYears()方法用于将当前线程永久等待, notifyNow()方法用于通知等待状态的线程运行。在 run()方法中, 运行了 waitForASecond()和 waitForYears()方法, 关键代码如下:

```
public class ThreadState implements Runnable {
    public synchronized void waitForASecond() throws InterruptedException {
        wait(500);           //使当前线程等待 0.5 秒或其他线程调用 notify()或 notifyAll()方法
    }
    public synchronized void waitForYears() throws InterruptedException {
        wait();              //使当前线程永久等待, 直到其他线程调用 notify()或 notifyAll()方法
    }
    public synchronized void notifyNow() throws InterruptedException {
        notify();            //唤醒由调用 wait()方法进入等待状态的线程
    }
    public void run() {
        try {
            waitForASecond(); //在新线程中运行 waitForASecond()方法
            waitForYears();   //在新线程中运行 waitForYears()方法
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```



(2) 编写类 Test 进行测试, 在 main()方法中, 输出了线程的各种不同的状态。关键代码如下:

```
public class Test {
    public static void main(String[] args) throws InterruptedException {
        ThreadState state = new ThreadState();           //创建 State 对象
        Thread thread = new Thread(state);               //利用 State 对象创建 Thread 对象
        System.out.println("新建线程: " + thread.getState()); //输出线程状态
        thread.start();                                   //调用 thread 对象的 start()方法, 启动新线程
        System.out.println("启动线程: " + thread.getState()); //输出线程状态
        Thread.sleep(100);                               //当前线程休眠 0.1 秒, 使新线程运行 waitForASecond()方法
        System.out.println("计时等待: " + thread.getState()); //输出线程状态
        Thread.sleep(1000);                               //当前线程休眠 1 秒, 使新线程运行 waitForYears()方法
        System.out.println("等待线程: " + thread.getState()); //输出线程状态
        state.notifyNow();                                //调用 state 的 notifyNow()方法
        System.out.println("唤醒线程: " + thread.getState()); //输出线程状态
        Thread.sleep(1000);                               //当前线程休眠 1 秒, 使新线程结束
        System.out.println("终止线程: " + thread.getState()); //输出线程状态
    }
}
```



Note

技术要点

使用 Thread 类的 getState()方法可以获得线程的状态, 该方法的返回值是 Thread.State, 它是线程状态的枚举。其枚举常量说明如表 15.1 所示。

表 15.1 Thread.State 的枚举常量说明

枚举常量	描述
NEW	新建状态
RUNNABLE	运行(可运行)状态
BLOCKED	阻塞状态
TIMED_WAITING	休眠状态
WAITING	等待状态
TERMINATED	终止状态

脚下留神:

请读者在编写多线程程序时, 时刻注意线程的状态。不同状态下, 线程能够执行的任务是不同的。

实例 238 查看 JVM 中的线程名

(实例位置: 配套资源\SL\15\238)

实例说明

在 Java 虚拟机 (JVM) 中, 除了用户创建的线程, 还有服务于用户线程的其他线程, 它们根据用途被分配到不同的组中进行管理。本实例演示了如何查看 JVM 中线程的名称及其所在组的名称。实例的运行效果如图 15.2 所示。



图 15.2 查看 JVM 的线程名

脚下留神:

读者可以在本实例的基础上增加想查看的信息,如线程的状态、线程的优先级、是否为守护线程等。

实现过程

编写类 ThreadList, 在该类中包含了 4 个方法: getRootThreadGroups()方法用于获得根线程组, getThreads()方法用于获得指定线程组中所有线程的名称, getThreadGroups()方法用于获得线程组中所有子线程组, main()方法用于测试。关键代码如下:

```
public class ThreadList {
    private static ThreadGroup getRootThreadGroups() { //获得根线程组
        ThreadGroup rootGroup = Thread.currentThread().getThreadGroup(); //获得当前线程组
        while (true) {
            if (rootGroup.getParent() != null) { //如果 getParent()方法的返回值非空则不是根线程组
                rootGroup = rootGroup.getParent(); //获得父线程组
            } else {
                break; //如果到达根线程组则退出循环
            }
        }
        return rootGroup; //返回根线程组
    }

    public static List<String> getThreads(ThreadGroup group) { //获得给定线程组中所有线程名
        List<String> threadList = new ArrayList<String>(); //创建保存线程名的列表
        Thread[] threads = new Thread[group.activeCount()]; //根据活动线程数创建线程数组
        int count = group.enumerate(threads, false); //复制线程到线程数组
        for (int i = 0; i < count; i++) { //遍历线程数组将线程名及其所在组保存到列表中
            threadList.add(group.getName() + "线程组: " + threads[i].getName());
        }
        return threadList; //返回列表
    }

    public static List<String> getThreadGroups(ThreadGroup group) { //获得线程组中子线程组
        List<String> threadList = getThreads(group); //获得给定线程组中线程名
        ThreadGroup[] groups = new ThreadGroup[group.activeGroupCount()]; //创建线程组数组
        int count = group.enumerate(groups, false); //复制子线程组到线程组数据
        for (int i = 0; i < count; i++) { //遍历所有子线程组
            threadList.addAll(getThreads(groups[i])); //利用 getThreads()方法获得线程名列表
        }
        return threadList; //返回所有线程名
    }

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



```
for (String string : getThreadGroups(getRootThreadGroups())) {
    System.out.println(string);           //遍历输出列表中的字符串
}
}
```



Note

技术要点

线程组 (ThreadGroup) 表示一个线程的集合。此外, 线程组也可以包含其他线程组。线程组构成一棵树, 在树中, 除了初始线程组外, 每个线程组都有一个父线程组。允许线程访问有关自己的线程组的信息, 但是不允许它访问有关其线程组的父线程组或其他任何线程组的信息。本实例使用的方法如表 15.2 所示。

表 15.2 ThreadGroup 类的常用方法

方 法 名	作 用
activeCount()	返回此线程组中活动线程的估计数
activeGroupCount()	返回此线程组中活动线程组的估计数
enumerate(Thread[] list, boolean recurse)	把此线程组中的所有活动线程复制到指定数组中
enumerate(ThreadGroup[] list, boolean recurse)	把对此线程组中的所有活动子组的引用复制到指定数组中
getName()	返回此线程组的名称
getParent()	返回此线程组的父线程组

脚下留神:

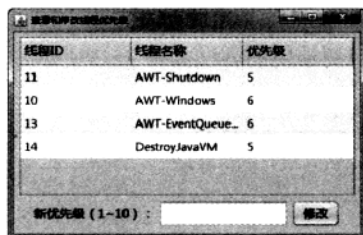
请读者在编写多线程程序时, 时刻注意线程的状态。不同状态下, 线程能够执行的任务是不同的。

实例 239 查看和修改线程优先级

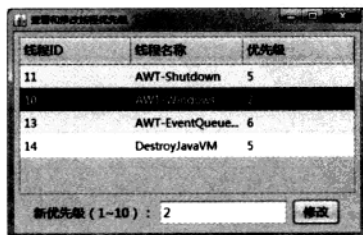
(实例位置: 配套资源\SL\15\239)

实例说明

Java 中每个线程都有优先级属性。默认情况下, 新建线程的优先级与创建该线程的线程优先级相同。每当线程调度器选择要运行的线程时, 通常选择优先级较高的线程。本实例演示如何查看和修改线程的优先级。实例的运行效果如图 15.3 所示。



(a) 初始状态



(b) 修改线程优先级后

图 15.3 实例运行效果



脚下留神:

线程优先级是高度依赖于操作系统的,而且 Sun 对于不同的操作系统提供的虚拟机并不完全相同。



Note

实现过程

(1) 编写类 ThreadPriorityTest, 该类继承了 JFrame。在框架中包含了一个表格, 用来显示当前线程组中运行的线程。一个文本域用来获得用户输入的新线程优先级。“修改”按钮实现了修改优先级的功能, 并更新了表格。

(2) 编写 do_this_windowActivated() 方法, 用来监听窗体激活事件。在该方法中, 使用当前线程所在线程组中的线程 ID、名称和优先级作为表格的数据。关键代码如下:

```
protected void do_this_windowActivated(WindowEvent e) {
    ThreadGroup group = Thread.currentThread().getThreadGroup(); //获得当前线程所在线程组
    Thread[] threads = new Thread[group.activeCount()];           //使用数组保存活动状态的线程
    group.enumerate(threads);                                     //获得所有线程
    DefaultTableModel model = (DefaultTableModel) table.getModel(); //获得表格模型
    model.setRowCount(0);                                         //清空表格模型中的数据
    model.setColumnIdentifiers(new Object[] { "线程 ID", "线程名称", "优先级" }); //定义表头
    for (Thread thread : threads) {                               //增加行数据
        model.addRow(new Object[] { thread.getId(), thread.getName(),
            thread.getPriority() });
    }
    table.setModel(model);                                         //更新表格模型
}
```

(3) 编写 do_button_actionPerformed() 方法, 用来单击“修改”按钮事件。在该方法中, 获得了用户输入的优先级和用户选择的行, 根据用户输入的优先级修改了线程优先级。关键代码如下:

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText();                             //获得用户输入的优先级
    Integer priority = Integer.parseInt(text);                     //将优先级转换成 Integer 对象
    int selectedRow = table.getSelectedRow();                       //获得用户选择的行
    DefaultTableModel model = (DefaultTableModel) table.getModel(); //获得默认表格模型
    model.setValueAt(priority, selectedRow, 2);                   //更改表格中的数据
    repaint();                                                      //重新绘制各个控件
}
```

技术要点

线程 (Thread) 是程序中的执行线程。Java 虚拟机允许应用程序并发地运行多个执行线程。在该类中定义了大量与线程操作相关的方法, 本实例使用的方法如表 15.3 所示。

表 15.3 Thread 类的常用方法

方 法 名	作 用
MAX_PRIORITY	线程可以具有的最高优先级
MIN_PRIORITY	线程可以具有的最低优先级
NORM_PRIORITY	分配给线程的默认优先级
getPriority()	获得线程的优先级
setPriority(int newPriority)	修改线程的优先级



实例 240 休眠当前线程

(实例位置: 配套资源\SL\15\240)



Note

实例说明

在龟兔赛跑的寓言故事中, 原本领先的兔子因为骄傲自满在中途休息而痛失了冠军。其寓意是告诉我们为人要谦虚, 不要骄傲。本实例将使用线程休眠技术来模拟龟兔赛跑的过程, 实例的运行效果如图 15.4 所示。

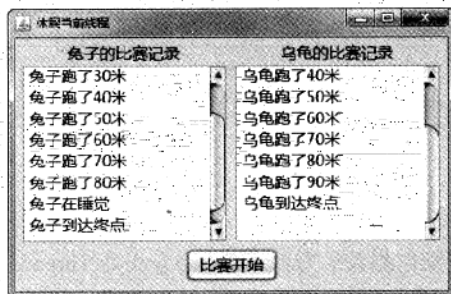


图 15.4 休眠线程

实现过程

(1) 编写类 RaceFrame, 该类继承了 JFrame。在框架中包含了两个文本区域, 用来输出乌龟和兔子的比赛记录, “比赛开始”按钮用来开始比赛。

(2) 编写内部类 Rabbit, 该类实现了 Runnable 接口。在 run() 方法中, 让兔子休眠了 1 秒钟。关键代码如下:

```
private class Rabbit implements Runnable {
    @Override
    public void run() {
        for (int i = 1; i < 11; i++) { // 循环 10 次模拟赛跑的过程
            String text = rabbitTextArea.getText(); // 获得文本域中的信息
            try {
                Thread.sleep(1); // 线程休眠 0.001 秒, 模拟兔子在跑步
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            rabbitTextArea.setText(text + "兔子跑了" + i + "0 米\n"); // 显示兔子的跑步距离
            if (i == 9) {
                rabbitTextArea.setText(text + "兔子在睡觉\n"); // 当跑了 90 米时开始睡觉
                try {
                    Thread.sleep(10000); // 线程休眠 10 秒, 模拟兔子在睡觉
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```




Note

```
    }  
    if(i == 10) {  
        try {  
            Thread.sleep(1);           //线程休眠 0.001 秒，模拟兔子在跑步  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        rabbitTextArea.setText(text + "兔子到达终点\n"); //显示兔子到达了终点  
    }  
}  
}
```

指点迷津:

由于乌龟和兔子在比赛中的行为类似，因此不讲解乌龟的代码。

技术要点

线程 (Thread) 是程序中的执行线程。Java 虚拟机允许应用程序并发地运行多个执行线程。在该类中定义了与线程操作相关的方法。sleep() 方法是 Thread 类的一个静态方法，它有两种形式，具体方法如表 15.4 所示。

表 15.4 Thread 类的 sleep() 方法

方 法 名	作 用
sleep(long millis)	让线程休眠指定的毫秒数
sleep(long millis, int nanos)	让线程休眠指定的毫秒数加纳秒数

实例 241 终止指定线程

(实例位置: 配套资源\SL\15\241)

实例说明

线程因为以下两种原因终止: run() 方法结束和未捕获的异常终止了 run() 方法。Thread 类提供了一个 stop() 方法来强迫线程停止执行, 然而由于其固有的不安全性, 建议不要使用。本实例利用 boolean 值来终止正在运行的线程。实例的运行效果如图 15.5 所示。

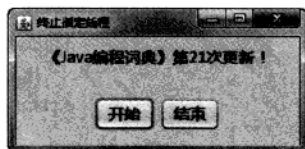


图 15.5 终止指定线程

实现过程

(1) 编写类 StopThreadTest, 该类继承了 JFrame。在框架中包含了一个标签用来显示线程正在运行。“开始”按钮用来运行新的线程并在标签上更新, “结束”按钮用来结束正在更新标签的线程。

(2) 编写类 CounterThread, 该类实现了 Runnable 接口。在 run() 方法中, 使用死循环,



每隔 0.1 秒更新一次标签上的文本信息。关键代码如下：

```
private class CounterThread implements Runnable {
    private int count = 0;
    private boolean stopped = true;
    public void setStopped(boolean stopped) {
        this.stopped = stopped;
    }
    public void run() {
        while (stopped) {                                //用布尔值控制线程运行
            try {
                Thread.sleep(100);                        //线程休眠 0.1 秒
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            lbljava.setText("《Java 编程词典》第" + (count++) + "次更新!"); //更新标签内容
        }
    }
}
```



Note

(3) 编写 do_button1_actionPerformed() 方法，用来监听单击“开始”按钮事件，并在该方法中运行了新的线程。关键代码如下：

```
protected void do_button1_actionPerformed(ActionEvent e) {
    counter = new CounterThread();                //创建 Runnable 对象
    new Thread(counter).start();                  //运行新线程
}
```

(4) 编写 do_button2_actionPerformed() 方法，用来监听单击“结束”按钮事件，并在该方法中结束了更新标签的线程。关键代码如下：

```
protected void do_button2_actionPerformed(ActionEvent e) {
    if (counter == null) {                          //检查 CounterThread 是否被实例化
        JOptionPane.showMessageDialog(this, "先运行线程", "", JOptionPane.WARNING_
MESSAGE);
        return;
    }
    counter.setStopped(false);                      //将布尔值设置为 false
}
```

技术要点

本实例涉及的技术要点请参见实例 240。

实例 242 线程的插队运行

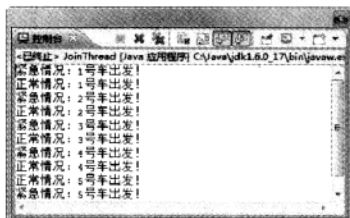
(实例位置：配套资源\SL\15\242)

实例说明

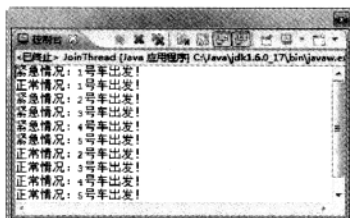
在编写多线程程序时，会遇到让一个线程优先于其他线程运行的情况。此时除了可以设置该线程的优先级高于其他线程外，更直接的方式是使用 Thread 类的 join() 方法。本实例将演示该方法的实际效果，实例的运行效果如图 15.6 所示。



Note



(a) 使用 join() 方法前



(b) 使用 join() 方法后

图 15.6 线程的插队运行

实现过程

(1) 编写类 EmergencyThread, 该类实现了 Runnable 接口。在 run() 方法中, 每隔 0.1 秒输出一条语句。关键代码如下:

```
public class EmergencyThread implements Runnable {
    @Override
    public void run() {
        for (int i = 1; i < 6; i++) {
            try {
                Thread.sleep(100);           //当前线程休眠 0.1 秒实现动态更新
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("紧急情况: " + i + "号车出发! "); //紧急情况下车辆出发
        }
    }
}
```

(2) 编写类 JoinThread 用来进行测试, 在该类中使用 EmergencyThread 创建并运行新的线程。使用 join() 方法让新线程优先于当前线程运行。关键代码如下:

```
public class JoinThread {
    public static void main(String[] args) {
        Thread thread = new Thread(new EmergencyThread()); //创建新线程
        thread.start();                                     //运行新线程
        for (int i = 1; i < 6; i++) {
            try {
                Thread.sleep(100);           //当前线程休眠 0.1 秒实现动态更新
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("正常情况: " + i + "号车出发! "); //正常情况下车辆出发
            try {
                thread.join();               //使用 join() 方法让新创建的线程优先完成
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```



指点迷津:

读者可以注释掉 `thread.join()` 语句对比程序的运行效果。

技术要点

线程 (Thread) 是程序中的执行线程。Java 虚拟机允许应用程序并发地运行多个执行线程。在该类中定义了与线程操作相关的方法。`join()` 方法是 Thread 类的一个静态方法, 它有 3 种形式, 具体方法如表 15.5 所示。

表 15.5 Thread 类的 `join()` 方法

方法名	作用
<code>join()</code>	等待调用该方法的线程终止
<code>join(long millis)</code>	等待调用该方法的线程终止的时间最长为 <code>millis</code> 毫秒
<code>join(long millis, int nanos)</code>	等待调用该方法的线程终止的时间最长为 <code>millis</code> 毫秒加 <code>nanos</code> 纳秒

脚下留神:

如果有线程中断了运行 `join()` 方法的线程, 则抛出 `InterruptedException` 异常。

实例 243 使用方法实现线程同步

(实例位置: 配套资源\SL\15\243)

实例说明

Java 提供了很多方式和工具类来帮助程序员简化多线程的开发, 同步方法是最简单和常用的一种方法。本实例将模拟一个简单的银行系统, 使用两个不同的线程向同一个账户存钱。账户的原始金额是 100 元, 两个线程分别存入 100 元, 并将存钱的方法修改成同步的方法。实例的运行效果如图 15.7 所示。

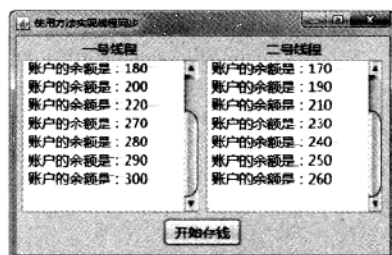


图 15.7 使用方法实现线程同步

实现过程

(1) 继承 `JFrame` 类编写名称为 `SynchronizedBankFrame` 的窗体, 该窗体的主要控件如表 15.6 所示。

表 15.6 窗体的主要控件

控件类型	控件命名	控件用途
<code>JButton</code>	<code>startButton</code>	启动两个线程开始转账
<code>JTextArea</code>	<code>thread1TextArea</code>	显示一号线程的输出结果
	<code>thread2TextArea</code>	显示二号线程的输出结果

(2) 编写内部类 `Bank`, 在该类中定义了一个整型域 `account` 来表示账户, 一个存钱方法





deposit()和一个显示账户余额的方法 getAccount()。关键代码如下：

```
private class Bank {  
    private int account = 100;           //每个账户的初始金额是 100 元  
    public synchronized void deposit(int money) {    //向账户中存入 money 元  
        account += money;  
    }  
    public int getAccount() {           //查询账户余额  
        return account;  
    }  
}
```

脚下留神：

“account += money;”总共分成 3 步执行：读取 account 的值，account 的值和 money 的值求和，再存入 account 的值。在单线程程序中，并没有什么问题。而在多线程程序中，如果两个线程同时读取 account，再先后存入 account，就会少计算一次 money 的值。

(3) 编写内部类 Transfer，它实现了 Runnable 接口，因此可以在新线程中运行，实现了向账户存钱的功能。关键代码如下：

```
private class Transfer implements Runnable {  
    private Bank bank;  
    private JTextArea textArea;  
    public Transfer(Bank bank, JTextArea textArea) {    //初始化变量  
        this.bank = bank;  
        this.textArea = textArea;  
    }  
    public void run() {  
        for (int i = 0; i < 10; i++) {    //向账户中存入 10 次钱  
            bank.deposit(10);           //向账户中存入 10 元钱  
            String text = textArea.getText();    //获得文本域中的文本  
            //在文本域中显示账户中的余额  
            textArea.setText(text + "账户的余额是: " + bank.getAccount() + "\n");  
        }  
    }  
}
```

技术要点

所谓同步方法即有 synchronized 关键字修饰的方法。之所以十几个字母就能解决困难的同步问题，与 Java 的内置锁密切相关。从 1.0 版开始，每个 Java 对象都有一个内置锁。如果方法用 synchronized 关键字声明，内置锁会保护整个方法。即在调用该方法前，需要获得内置锁，否则就处于阻塞状态。最简单的同步方法代码如下：

```
public synchronized void save(){} 
```

指点迷津：

synchronized 关键字也可以修饰静态方法，此时如果调用该静态方法会锁住整个类。



实例 244 使用特殊域变量实现线程同步

(实例位置: 配套资源\SL\15\244)

实例说明

Java 提供了很多方式和工具类来帮助程序员简化多线程的开发。同步方法是最简单和常用的一种方法。本实例将模拟一个简单的银行系统, 使用两个不同的线程向同一个账户存钱。账户的原始金额是 100 元, 两个线程分别存入 100 元, 并演示如何使用 `volatile` 关键字实现同步。实例的运行效果如图 15.8 所示。

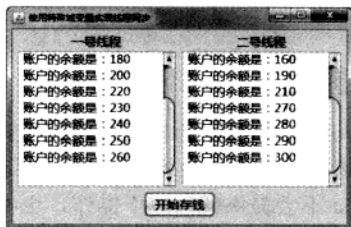


图 15.8 使用特殊域变量实现线程同步

实现过程

(1) 继承 `JFrame` 类编写名称为 `SynchronizedBankFrame` 的窗体, 该窗体的主要控件如表 15.7 所示。

表 15.7 窗体的主要控件

控件类型	控件命名	控件用途
<code> JButton </code>	<code>startButton</code>	启动两个线程开始转账
<code> JTextArea </code>	<code>thread1TextArea</code>	显示一号线线程的输出结果
	<code>thread2TextArea</code>	显示二号线线程的输出结果

(2) 编写内部类 `Bank`, 在该类中定义了一个整型域 `account` 来表示账户, 一个存钱方法 `deposit()` 和一个显示账户余额的方法 `getAccount()`。关键代码如下:

```
private class Bank {
    private volatile int account = 100;           //将域变量用 volatile 修饰
    public void deposit(int money) {              //向账户中存钱
        account += money;
    }
    public int getAccount() {                     //获得账户余额
        return account;
    }
}
```

(3) 编写内部类 `Transfer`, 它实现了 `Runnable` 接口, 因此可以在新线程中运行, 实现了向账户存钱的功能。关键代码如下:

```
private class Transfer implements Runnable {
    private Bank bank;
    private JTextArea textArea;
    public Transfer(Bank bank, JTextArea textArea) { //初始化变量
        this.bank = bank;
        this.textArea = textArea;
    }
}
```





Note

```

public void run() {
    for (int i = 0; i < 10; i++) {
        bank.deposit(10);           //向账户中存入 10 次钱
        String text = textArea.getText(); //向账户中存入 10 元钱
        //在文本域中显示账户中的余额    //获得文本域中的文本
        textArea.setText(text + "账户的余额是: " + bank.getAccount() + "\n");
    }
}

```

技术要点

`volatile` 关键字为域变量的访问提供了一种免锁机制。使用 `volatile` 修饰域相当于告诉虚拟机该域可能会被其他线程更新。因此每次使用该域就要重新计算，而不是使用寄存器中的值。`volatile` 不会提供任何原子操作，也不能用来修饰 `final` 类型的变量。

实例 245 简单的线程通信

(实例位置: 配套资源\SL\15\245)

实例说明

使用多线程编程的一个重要原因就是线程间通信的代价比较小。本实例将模拟一个在线购物系统，当单击“开始交易”按钮时，卖家会向买家发送 5 种明日科技公司出版的 Java 图书，以此来演示如何实现两个线程间的通信。实例的运行效果如图 15.9 所示。

实现过程

(1) 继承 `JFrame` 类编写名称为 `TransactionFrame` 的窗体，该窗体的主要控件如表 15.8 所示。

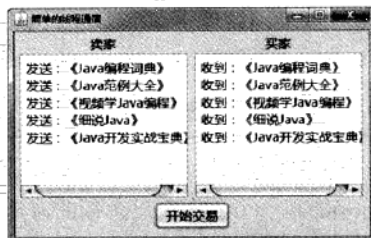


图 15.9 简单的线程通信

表 15.8 窗体的主要控件

控 件 类 型	控 件 命 名	控 件 用 途
<code>JButton</code>	<code>button</code>	实现线程通信的功能
<code>JTextArea</code>	<code>senderTextArea</code>	显示卖家线程的输出结果
	<code>receiverTextArea</code>	显示买家线程的输出结果

(2) 编写内部类 `Sender`，该类实现了 `Runnable` 接口。在 `run()` 方法中，向买家发送了 5 本书并检查买家是否接收到。关键代码如下：

```

private class Sender implements Runnable {
    private String[] products = {"《Java 编程词典》", "《Java 范例大全》", "《视频学 Java 编程》",
    "《细说 Java》", "《Java 开发实战宝典》"}; //模拟商品列表
    private volatile String product; //保存一个商品名称
    private volatile boolean isValid; //保存卖家是否发送商品的状态
    public boolean isValid() { //读取状态

```



Note

```

        return isValid;
    }
    public void setIsValid(boolean isValid) {           //设置状态
        this.isValid = isValid;
    }
    public String getProduct() {                       //获得商品
        return product;
    }
    public void run() {
        for (int i = 0; i < 5; i++) {                  //向买家发送 5 次商品
            while (isValid) {                          //如果已经发送商品就进入等待状态，等待买家接收
                Thread.yield();
            }
            product = products[i];                     //获得一件商品
            String text = senderTextArea.getText();    //获得卖家文本域信息
            senderTextArea.setText(text + "发送: " + product + "\n"); //更新卖家文本域信息
            try {
                Thread.sleep(100);                     //当前线程休眠 0.1 秒实现发送的效果
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            isValid = true;                             //将状态设置为已经发送商品
        }
    }
}

```

(3) 编写内部类 Receiver，该类实现了 Runnable 接口。在 run() 方法中，接收卖家发送的书籍并再次等待卖家发送书籍。关键代码如下：

```

private class Receiver implements Runnable {
    private Sender sender;                             //创建一个对发送者的引用
    public Receiver(Sender sender) {                   //利用构造方法初始化发送者引用
        this.sender = sender;
    }
    public void run() {
        for (int i = 0; i < 5; i++) {                  //接收 5 次商品
            while (!sender.isValid()) {                //如果发送者没有发送商品就进行等待
                Thread.yield();
            }
            String text = receiverTextArea.getText();  //获得卖家文本域信息
            //更新卖家文本域信息
            receiverTextArea.setText(text + "收到: " + sender.getProduct() + "\n");
            try {
                Thread.sleep(1000);                    //线程休眠 1 秒实现动态发送的效果
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            sender.setIsValid(false); //设置卖家发送商品的状态为未发送，这样卖家就可以继续
        }
    }
}

```



技术要点

线程 (Thread) 是程序中的执行线程。Java 虚拟机允许应用程序并发地运行多个执行线程。在该类中定义了大量与线程操作相关的方法。yield() 方法是 Thread 类的静态方法, 用来暂停当前正在执行的线程对象, 并执行其他线程。该方法的声明如下:

```
public static void yield()
```

**Note**

实例 246 新建有返回值的线程

(实例位置: 配套资源\SL\15\246)

实例说明

本实例将演示如何新建有返回值的线程。首先使用 ThreadLocal 来管理一号线和二号线, 它们是分别向账户增加 100 元。在三号线中利用一号、二号的计算结果来算出账户实际的金额。实例的运行效果如图 15.10 所示。



图 15.10 新建有返回值的线程

实现过程

(1) 继承 JFrame 类编写名称为 SynchronizedBankFrame 的窗体, 该窗体的主要控件如表 15.9 所示。

表 15.9 窗体的主要控件

控件类型	控件命名	控件用途
JButton	startButton	启动 3 个线程开始转账
JTextArea	thread1TextArea	显示一号线线程的输出结果
	thread2TextArea	显示二号线线程的输出结果
	thread3TextArea	显示三号线线程的输出结果

(2) 编写类 Transfer, 该类实现了 Callable 接口。在重写 call() 方法时, 将返回值设置成账户的余额。关键代码如下:

```
public class Transfer implements Callable<Integer> {  
    private Bank bank;  
    private JTextArea textArea;  
    public Transfer(Bank bank, JTextArea textArea) {  
        this.bank = bank;  
    }  
    //利用构造方法初始化变量
```



Note

```

        this.textArea = textArea;
    }
    public Integer call() {
        for (int i = 0; i < 10; i++) {           //循环 10 次向账户中存钱
            bank.deposit(10);
            String text = textArea.getText();
            textArea.setText(text + "账户的余额是: " + bank.getAccount() + "\n");
        }
        return bank.getAccount();               //获得账户的余额
    }
}

```

(3) 编写 `do_button_actionPerformed()` 方法, 用来监听单击“开始存钱”按钮事件。在该方法中, 分别获得了两个 `ThreadLocal` 变量的结果并计算最后的存钱结果。关键代码如下:

```

protected void do_button_actionPerformed(ActionEvent arg0) {
    Bank bank = new Bank();
    Transfer transfer1 = new Transfer(bank, thread1TextArea); //创建 Transfer 对象
    Transfer transfer2 = new Transfer(bank, thread2TextArea); //创建 Transfer 对象
    FutureTask<Integer> task1 = new FutureTask<Integer>(transfer1); //创建 FutureTask 对象
    FutureTask<Integer> task2 = new FutureTask<Integer>(transfer2); //创建 FutureTask 对象
    Thread thread1 = new Thread(task1); //创建一号线程
    Thread thread2 = new Thread(task2); //创建二号线程
    thread1.start(); //运行一号线程
    thread2.start(); //运行二号线程
    try {
        int thread1Result = task1.get(); //获得一号线程的计算结果
        int thread2Result = task2.get(); //获得二号线程的计算结果
        //更新三号线程文本域信息
        thread3TextArea.setText(thread3TextArea.getText() + "一号计算结果是: " + thread1Result
+ "\n");
        //更新三号线程文本域信息
        thread3TextArea.setText(thread3TextArea.getText() + "二号计算结果是: " + thread2Result
+ "\n");
        thread3TextArea.setText(thread3TextArea.getText() + "实际的金额是: " + (thread1Result +
thread2Result - 100) + "\n"); //更新三号线程文本域信息
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (ExecutionException e) {
        e.printStackTrace();
    }
}
}

```

技术要点

`Callable<V>` 接口类似于 `Runnable`, 两者都是为那些可能被另一个线程执行的类设计的。但是 `Runnable` 不会返回结果, 并且无法抛出经过检查的异常。实现该接口需要重写 `call()` 方法, 该方法的声明如下:

V call() throws Exception

`Future<V>` 接口表示异步计算的结果。它提供了检查计算是否完成的方法, 以等待计算的完成, 并获取计算的结果。计算完成后只能使用 `get()` 方法来获取结果, 如有必要, 计算完成前



可以阻塞此方法。取消则由 `cancel()` 方法来执行。还提供了其他方法, 以确定任务是正常完成还是被取消了。一旦计算完成, 就不能再取消计算。如果为了可取消性而使用 `Future` 但又不提供可用的结果, 则可以声明 `Future<?>` 形式类型, 并返回 `null` 作为底层任务的结果。现该接口需要重写 `get()` 方法, 该方法的声明如下:

`V get() throws InterruptedException, ExecutionException`



Note

脚下留神:

请读者在处理有返回值线程的问题时, 不要忘记捕获异常。

实例 247 使用线程池优化多线程编程

(实例位置: 配套资源\SL\15\247)

实例说明

Java 中的对象是使用 `new` 操作符创建的, 如果创建大量短生命周期的对象, 这种方式性能非常低下。为了解决这个问题发明了池技术。对于数据库连接有连接池, 对于线程则有线程池。本实例有两种方式创建 1000 个短生命周期的线程, 第一种是普通方式, 第二种是线程池方式。通过时间和内存消耗的对比, 就可以很明显地看出线程池的优势。实例的运行效果如图 15.11 所示。

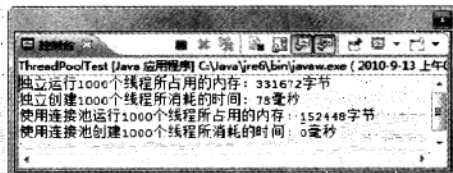


图 15.11 使用线程池优化多线程编程

指点迷津:

使用线程池创建对象的时间是 0 毫秒, 说明线程池是非常高效的。

实现过程

(1) 编写类 `TempThread`, 该类实现了 `Runnable` 接口。在 `run()` 方法中, 进行简单的自增运算。关键代码如下:

```
public class TempThread implements Runnable {           //测试用的 Runnable 接口实现类
    private int id = 0;
    @Override
    public void run() {                                   //run()方法给 id 做自增运算
        id++;
    }
}
```

(2) 编写类 `ThreadPoolTest` 进行测试, 在 `main()` 方法中, 使用两种方式创建了 1000 个线程, 分别输出了创建时间和占用的内存。关键代码如下:

```
public class ThreadPoolTest {
    public static void main(String[] args) {
```




Note

```

Runtime run = Runtime.getRuntime();           //创建 Runtime 对象
run.gc();                                     //运行垃圾回收器, 这样可以减少误差
long freeMemory = run.freeMemory();           //获得当前虚拟机的空闲内存
long currentTime = System.currentTimeMillis(); //获得当前虚拟机的时间
for (int i = 0; i < 1000; i++) {               //独立运行 1000 个线程
    new Thread(new TempThread()).start();
}
//查看内存的变化
System.out.println("独立运行 1000 个线程所占用的内存:" + (freeMemory - run.freeMemory())
+ "字节");
System.out.println("独立创建 1000 个线程所消耗的时间:" + (System.currentTimeMillis()
- currentTime) + "毫秒"); //查看时间的变化
run.gc();                                     //运行垃圾回收器
freeMemory = run.freeMemory();               //获得当前虚拟机的空闲内存
currentTime = System.currentTimeMillis();    //获得当前虚拟机的时间
ExecutorService executorService = Executors.newFixedThreadPool(2); //创建线程池
for (int i = 0; i < 1000; i++) {               //使用线程池运行 1000 个线程
    executorService.submit(new TempThread());
}
System.out.println("使用连接池运行 1000 个线程所占用的内存:" + (freeMemory -
run.freeMemory()) + "字节"); //查看内存的变化
System.out.println("使用连接池创建 1000 个线程所消耗的时间:" + (System.
currentTimeMillis() - currentTime) + "毫秒"); //查看时间的变化
}
}

```

脚下留神:

在使用完线程池后, 需要调用 `ExecutorService` 接口中定义的 `shutdownNow()` 方法终止线程池。

技术要点

`Executors` 类为 `java.util.concurrent` 包中所定义的 `Executor`、`ExecutorService`、`ScheduledExecutorService`、`ThreadFactory` 和 `Callable` 类提供工厂和实用方法。此类支持以下各种方法:

- ☑ 创建并返回设置有常用配置字符串的 `ExecutorService` 的方法。
- ☑ 创建并返回设置有常用配置字符串的 `ScheduledExecutorService` 的方法。
- ☑ 创建并返回“包装的”`ExecutorService` 方法, 它通过使特定于实现的方法不可访问来禁用重新配置。
- ☑ 创建并返回 `ThreadFactory` 的方法, 它可将新创建的线程设置为已知的状态。
- ☑ 创建并返回非闭包形式的 `Callable` 的方法, 这样可将其用于需要 `Callable` 的执行方法中。

本实例使用其 `newFixedThreadPool()` 方法创建一个可重用固定线程数的线程池, 它以共享的无界队列方式来运行这些线程。该方法的声明如下:

```
public static ExecutorService newFixedThreadPool(int nThreads)
```

参数说明

`nThreads`: 池中的线程数。



多学两招:

一个线程池中有多个处于可运行状态的线程,当向线程池中添加 Runnable 或 Callable 接口对象时,就会有一个线程来执行 run()或 call()方法。如果方法执行完毕,则该线程并不终止,而是继续在池中处于可运行状态,以运行新的任务。



Note

实例 248 哲学家的就餐问题

(实例位置: 配套资源\SL\15\248)

实例说明

假设有 5 个哲学家,他们围绕圆桌坐成一圈,每人的右手边有一根筷子。哲学家只有两种状态,即思考和吃饭。当需要吃饭时,他需要用两根筷子。此时很有可能他只有一根或没有筷子,因为旁边的哲学家在就餐。那么他就处于等待状态,如果 5 个哲学家都在等待别人的筷子,程序就进入死锁状态。本实例演示如何解决哲学家就餐问题。实例的运行效果如图 15.12 所示。



图 15.12 哲学家的就餐问题

实现过程

(1) 继承 JFrame 类编写名称为 DiningPhilosophersFrame 的窗体,该窗体的主要控件如表 15.10 所示。

表 15.10 窗体的主要控件

控件类型	控件命名	控件用途
JButton	startButton	启动新线程运行程序
JTextArea	thinkingTextArea	显示处于思考状态的哲学家
	eatingTextArea	显示处于就餐状态的哲学家
	waitingTextArea	显示处于等待状态的哲学家

(2) 本实例实现起来有些复杂,现选择 Philosopher 类的 eating()方法进行讲解。该类根据记录的哲学家的状态来判断他是否需要使用筷子。如果两只筷子都可用则哲学家开始就餐,否则处于等待状态。关键代码如下:

```
public synchronized void eating() {
    if (!state) { //state 是一个布尔值, true 表示哲学家刚才的状态是吃饭, false 表示思考
        if (chopstickArray.get(id).isAvailable()) { //如果哲学家右手边的筷子可用
            if (chopstickArray.getLast(id).isAvailable()) { //如果哲学家左手边的筷子可用
```



Note

```

        chopstickArray.get(id).setAvailable(false);           //设置右手筷子不可用
        chopstickArray.getLast(id).setAvailable(false);       //设置左手筷子不可用
        String text = eatingTextArea.getText();
        eatingTextArea.setText(text + this + " 在吃饭\n"); //显示哲学家在吃饭
        try {
            Thread.sleep(100);                                //吃饭时间设置成 0.1 秒
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    } else { //如果哲学家左手边的筷子不可用，就在相应的文本域中显示等待信息
        String text = waitingTextArea.getText();
        waitingTextArea.setText(text + this + " 在等待 " + chopstickArray.getLast(id) + "\n");
        try {
            wait(new Random().nextInt(100)); //等待小于 0.1 秒时间后检查筷子是否可用
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
} else { //如果哲学家右手边的筷子不可用，就在相应的文本域中显示等待信息
    String text = waitingTextArea.getText();
    waitingTextArea.setText(text + this + " 在等待 " + chopstickArray.get(id) + "\n");
    try {
        wait(new Random().nextInt(100)); //等待小于 0.1 秒时间后检查筷子是否可用
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}
state = true; //设置 state 的值为 true 表示哲学家的状态是吃饭
}

```

技术要点

本实例应用了 `Object` 类中的 `notify()`、`notifyAll()` 和 `wait()` 方法用来控制线程的运行状态。其中 `wait()` 方法有 3 种重载形式，这些方法的说明如表 15.11 所示。

表 15.11 `Object` 类线程相关方法

方 法 名	作 用
<code>notify()</code>	唤醒在此对象监视器上等待的单个线程
<code>notifyAll()</code>	唤醒在此对象监视器上等待的所有线程
<code>wait()</code>	在其他线程调用此对象的 <code>notify()</code> 或 <code>notifyAll()</code> 方法前，导致当前线程等待
<code>wait(long timeout)</code>	在其他线程调用此对象的 <code>notify()</code> 或 <code>notifyAll()</code> 方法，或者超过指定的时间量前，导致当前线程等待
<code>wait(long timeout, int nanos)</code>	在其他线程调用此对象的 <code>notify()</code> 或 <code>notifyAll()</code> 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，导致当前线程等待

指点迷津：

以上方法均在 `Object` 类中，并且都是 `final` 的。读者不要把它们和 `Thread` 类混淆。

第 16 章

网络通信

本章读者可以学到如下实例：

- » 实例 249 获得内网的所有 IP 地址
- » 实例 250 获取网络资源的大小
- » 实例 251 解析网页中的内容
- » 实例 252 网络资源的单线程下载
- » 实例 253 网络资源的多线程下载
- » 实例 254 下载网络资源的断点续传
- » 实例 255 建立服务器套接字
- » 实例 256 建立客户端套接字
- » 实例 257 设置等待连接的超时时间
- » 实例 258 获得 Socket 信息
- » 实例 259 接受和发送 Socket 信息
- » 实例 260 关闭 Socket 缓冲
- » 实例 261 使用 Socket 通信
- » 实例 262 防止 Socket 传递汉字乱码
- » 实例 263 使用 Socket 传递对象
- » 实例 264 使用 Socket 传输图片
- » 实例 265 使用 Socket 传输音频
- » 实例 266 使用 Socket 传输视频
- » 实例 267 一个服务器与一个客户端通信
- » 实例 268 一个服务器与多个客户端通信
- » 实例 269 客户端一对多通信
- » 实例 270 客户端一对一通信
- » 实例 271 基于 Socket 的数据库编程
- » 实例 272 使用 Proxy 创建代理服务器
- » 实例 273 使用 ProxySelector 选择代理服务器
- » 实例 274 聊天室服务器端
- » 实例 275 聊天客户端



实例 249 获得内网的所有 IP 地址

(实例位置: 配套资源\SL\16\249)

实例说明

在进行网络编程时,有时需要对局域网内的所有主机进行遍历,为此需要获得内网的所有 IP 地址,本实例将演示如何在 Java 应用程序中,获得内网的所有 IP 地址。运行程序,单击窗体上的“显示所有 IP”按钮,将在文本域中显示内网中所有主机的 IP 地址,效果如图 16.1 所示。

实现过程

(1) 在 Eclipse 中新建项目 249,在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 GainAllIpFrame。

(3) 在 GainAllIpFrame 窗体的内容面板顶部位置放一个面板控件,并在该面板控件上放两个命令按钮,然后在内容面板的中部位置放一个滚动面板控件,并在滚动面板上放一个文本域控件,完成窗体界面的设置。

(4) 在 GainAllIpFrame 窗体类中定义一个 gainAllIp()方法,用于获得内容中的所有 IP 地址,并追加到文本域中显示。

(5) 在 GainAllIpFrame 窗体类中定义一个 PingIpThread 线程类,用于 Ping 指定的 IP 地址,并判断其是否为有效的内网 IP 地址,如果是就添加到集合对象中。关键代码如下:

```
public String ip; //表示 IP 地址的成员变量
public PingIpThread(String ip) { //参数为需要判断的 IP 地址
    this.ip = ip;
}
public void run() {
    try {
        //获得所 ping 的 IP 进程, -w 280 是等待每次回复的超时时间, -n 1 是要发送的回显请求数
        Process process = Runtime.getRuntime().exec(
            "ping " + ip + " -w 280 -n 1");
        InputStream is = process.getInputStream(); //获得进程的输入流对象
        InputStreamReader isr = new InputStreamReader(is); //创建 InputStreamReader 对象
        BufferedReader in = new BufferedReader(isr); //创建缓冲字符流对象
        String line = in.readLine(); //读取信息
        while (line != null) {
            if (line != null && !line.equals("")) {
                if (line.substring(0, 2).equals("来自"))
                    || (line.length() > 10 && line.substring(0, 10)
                        .equals("Reply from")) { //判断是 ping 通过的 IP 地址

```



图 16.1 获得内网的所有 IP 地址



Note



Note

```

        pingMap.put(ip, "true");           //向集合中添加 IP
    }
    }
    line = in.readLine();                 //再读取信息
}
} catch (IOException e) {
}
}

```

技术要点

本实例通过获得本机的 IP 地址所属的网段，然后 Ping 网络中的 IP 地址通过输入流对象读取所 Ping 结果，并判断是否为内网的 IP 地址，从而实现了本实例的功能。

(1) 获得本机的 IP 地址所属的网段，可以通过如下代码实现：

```

InetAddress host = InetAddress.getLocalHost();    //获得本机的 InetAddress 对象
String hostAddress = host.getHostAddress();      //获得本机的 IP 地址
int pos = hostAddress.lastIndexOf(".");          //获得 IP 地址中最后一个点的位置
String wd = hostAddress.substring(0, pos + 1);    //对本机的 IP 进行截取，获得网段

```

(2) Ping 网络中的 IP 地址，通过输入流对象读取所 Ping 结果，并判断是否为内网的 IP 地址。关键代码如下：

```

//获得所 Ping 的 IP 进程，-w 280 是等待每次回复的超时时间，-n 1 是要发送的回显请求数
Process process = Runtime.getRuntime().exec("ping " + ip + " -w 280 -n 1");
InputStream is = process.getInputStream();        //获得进程的输入流对象
InputStreamReader isr = new InputStreamReader(is); //创建 InputStreamReader 对象
BufferedReader in = new BufferedReader(isr);     //创建缓冲字符流对象
String line = in.readLine();                     //读取信息
while (line != null) {
    if (line != null && !line.equals("")) {
        //是 Ping 通的 IP 地址
        if (line.substring(0, 2).equals("来自") || (line.length() > 10 && line.substring(0, 10).equals(
("Reply from")))) {
            pingMap.put(ip, "true");           //向集合中添加 IP
        }
    }
    line = in.readLine();                     //再读取信息
}
}

```

脚下留神：

本实例在 Windows 7 系统下调试通过。

实例 250 获取网络资源的大小

(实例位置：配套资源\SL\16\250)

实例说明

本实例演示了如何获取网络资源的大小，运行程序，在输入网址标签右侧的文本框中输入要访问的网站网址，单击“获得大小”按钮，将获取网络资源的大小，效果如图 16.2 所示。

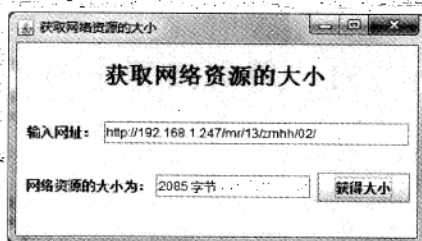


图 16.2 获取网络资源的大小

实现过程

- (1) 在 Eclipse 中新建项目 250，在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承 JFrame 类的 InternetSizeFrame 窗体类。
- (3) 在 InternetSizeFrame 窗体类中，将内部面板设置为绝对布局，然后在窗体上添加标签、文本框和命令按钮控件，完成窗体界面的设计。
- (4) 在 InternetSizeFrame 窗体类中，定义获取网络资源大小的 netSourceSize() 方法。关键代码如下：

```
public long netSourceSize(String sUrl) throws Exception{
    URL url = new URL(sUrl);           //创建 URL 对象
    URLConnection urlConn = url.openConnection(); //获得网络连接对象
    urlConn.connect();                 //打开到 url 引用资源的通信链接
    return urlConn.getContentLength(); //以字节为单位返回资源的大小
}
```

- (5) 在窗体类 InternetSizeFrame 的“获得大小”按钮事件中，添加获取网络资源大小，并将网络资源大小显示在文本框中的事件代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        String address = tf_address.getText().trim(); //获得输入的网址
        try {
            long len = netSourceSize(address); //调用方法获取网络资源的大小
            tf_size.setText(String.valueOf(len)+" 字节"); //在文本框中显示网络资源的大小
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
});
```

技术要点

本实例主要是通过 URLConnection 类的 getContentLength() 方法获取网络资源的大小。下面是获取网络资源大小的代码：

```
URLConnection urlConn = url.openConnection(); //获得网络连接对象
long size = urlConn.getContentLength(); //以字节为单位获取网络资源的大小
```

指点迷津：

上面代码中的 url 是一个有效的 URL 对象。



实例 251 解析网页中的内容

(实例位置: 配套资源\SL\16\251)



Note

实例说明

本实例演示了如何在 Java 应用程序中解析网页的内容。运行程序, 在输入网址标签右侧的文本框中输入网址, 单击“解析网页”按钮, 将在文本域中显示解析的网页内容, 效果如图 16.3 所示。

实现过程

(1) 在 Eclipse 中新建项目 251, 在项目下创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承 JFrame 类的 InternetContentFrame 窗体类。

(3) 在 InternetContentFrame 窗体类中, 创建用于解析网页内容的 getURLConnection() 方法。关键代码如下:

```
public Collection<String> getURLConnection(String urlString) {  
    URL url = null; //声明 URL  
    URLConnection conn = null; //声明 URLConnection  
    Collection<String> urlCollection = new ArrayList<String>(); //创建集合对象  
    try {  
        url = new URL(urlString); //创建 URL 对象  
        conn = url.openConnection(); //获得连接对象  
        conn.connect(); //打开到 URL 引用资源的通信链接  
        InputStream is = conn.getInputStream(); //获取流对象  
        InputStreamReader in = new InputStreamReader(is, "UTF-8"); //转换为字符流  
        BufferedReader br = new BufferedReader(in); //创建缓冲流对象  
        String nextLine = br.readLine(); //读取信息, 解析网页  
        while (nextLine != null) {  
            urlCollection.add(nextLine); //解析网页的全部内容, 添加到集合中  
            nextLine = br.readLine(); //读取信息, 解析网页  
        }  
    } catch (Exception ex) {  
        ex.printStackTrace();  
    }  
    return urlCollection;  
}
```

(4) 在 InternetContentFrame 窗体类中为“解析网页”按钮添加事件代码, 实现调用 getURLConnection() 方法解析网页内容并显示在文本域中。该按钮的事件代码如下:

```
button.addActionListener(new ActionListener() {  
    public void actionPerformed(final ActionEvent e) {  
        String address = tf_address.getText().trim(); //获得输入的网址
```

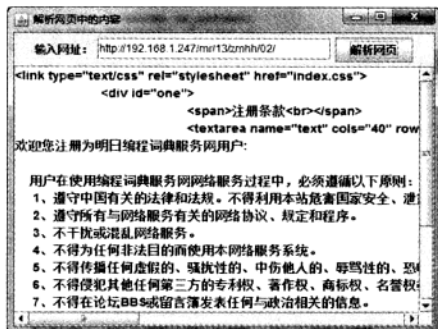


图 16.3 解析网页中的内容



```
Collection urlCollection = getURLCollection(address); //调用方法，获得网页内容的集合对象
Iterator it = urlCollection.iterator();              //获得集合的迭代器对象
while(it.hasNext()){
    ta_content.append((String)it.next()+"\n");        //在文本域中显示解析的内容
}
});
```

技术要点

本实例主要是通过 URLConnection 类的 getInputStream() 方法获得网页资源的输入流对象，然后从该输入流中读取信息，完成解析网页内容的操作。

使用 URLConnection 类的 getInputStream() 方法，可以获得网页资源的输入流对象，获得网页资源的输入流对象，可以通过如下代码实现：

```
URLConnection conn = url.openConnection();          //获得网页资源的连接对象
InputStream is = conn.getInputStream();               //获得网页资源的输入流对象
```

指点迷津：

上面代码中的 url 是一个有效的网页资源的 URL 对象。

实例 252 网络资源的单线程下载

(实例位置：配套资源\SL\16\252)

实例说明

本实例实现了网络资源的单线程下载，也就是在一个线程中完成网络资源的下载。运行程序，输入下载资源的网址，单击窗体上的“单击开始下载”按钮，将下载网络资源，并在下载完毕后进行提示，效果如图 16.4 和图 16.5 所示。

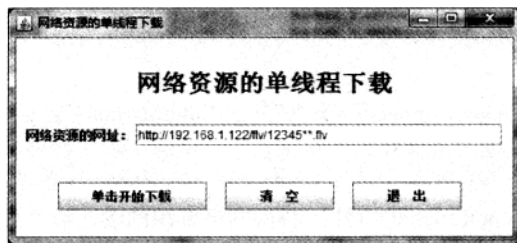


图 16.4 网络资源的单线程下载界面

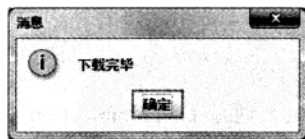


图 16.5 提示下载完毕

实现过程

- (1) 在 Eclipse 中新建项目 252，在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承 JFrame 类的 SingleThreadDownloadFrame 窗体类。
- (3) 在 SingleThreadDownloadFrame 窗体类的内容面板上，添加相应控件完成窗体界面的设计。
- (4) 在 SingleThreadDownloadFrame 窗体类中定义 download() 方法，用于根据参数 urlAddr





指定的地址，完成网络资源的单线程下载。关键代码如下：

```
public void download(String urlAddr){           //从指定网址下载文件
    try {
        URL url = new URL(urlAddr);           //创建 URL 对象
        URLConnection urlConn = url.openConnection(); //获得连接对象
        urlConn.connect();                     //打开到 URL 引用资源的通信链接
        InputStream in = urlConn.getInputStream(); //获得输入流对象
        String filePath = url.getFile();        //获得完整路径
        int pos = filePath.lastIndexOf("/");    //获得路径中最后一个斜杠的位置
        String fileName = filePath.substring(pos+1); //截取文件名
        FileOutputStream out = new FileOutputStream("C:"+fileName); //创建输出流对象
        byte[] bytes = new byte[1024];        //声明存放下载内容的字节数组
        int len = in.read(bytes);              //从输入流中读取内容
        while (len != -1){
            out.write(bytes,0,len);            //将读取的内容写到输出流
            len = in.read(bytes);              //继续从输入流中读取内容
        }
        out.close();                          //关闭输出流
        in.close();                          //关闭输入流
        JOptionPane.showMessageDialog(null, "下载完毕");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

(5) 在 SingleThreadDownloadFrame 窗体上为“单击开始下载”按钮添加事件代码，用于调用 download()方法。该按钮的事件代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        String address = tf_address.getText().trim(); //获得网址
        download(address);                             //下载文件
    }
});
```

技术要点

本实例的实现主要是在主线程中，利用 URLConnection 对象的 getInputStream()方法获得网络资源的输入流对象，并将读取的信息通过输出流保存到用户主机上，从而实现了网络资源的单线程下载。

(1) 通过 URLConnection 对象的 getInputStream()方法，获得网络资源的输入流对象，用于读取网络资源的信息。

(2) 使用 FileOutputStream 类创建输出流对象，然后使用该类的 write()方法，将从输入流获得的网络资源保存到磁盘上，实现了网络资源的单线程下载。关键代码如下：

```
FileOutputStream out = new FileOutputStream("C:"+fileName); //创建输出流对象
byte[] bytes = new byte[1024];                             //声明存放下载内容的字节数组
int len = in.read(bytes);                                    //从输入流中读取内容
while (len != -1){
    out.write(bytes,0,len);                                  //将读取的内容写到输出流
    len = in.read(bytes);                                    //继续从输入流中读取内容
}
```



指点迷津:

上面代码中的 fileName 是需要下载的网络资源的名称。

实例 253 网络资源的多线程下载

(实例位置: 配套资源\SL\16\253)



Note

实例说明

本实例实现了网络资源的多线程下载, 也就是主线程启动后, 再通过单独的线程完成网络资源的下载。运行程序, 输入下载资源的网址, 单击窗体上的“下载”按钮, 将完成网络资源下载, 并在下载完毕后进行提示, 效果如图 16.6 和图 16.7 所示。



图 16.6 网络资源的多线程下载界面

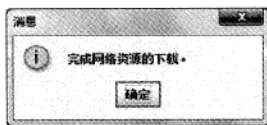


图 16.7 提示完成下载

实现过程

(1) 在 Eclipse 中新建项目 253, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个实现 Runnable 接口的线程类 DownMultiThread, 用于实现网络资源的下载。该线程类中 run() 方法的关键代码如下:

```
RandomAccessFile out = new RandomAccessFile(desFile, "rw");//创建可读写的流对象
out.seek(startPos);                                     //指定读写的开始标记
InputStream in = conn.getInputStream();                   //获得网络资源的输入流对象
BufferedInputStream bin = new BufferedInputStream(in);  //创建输入缓冲流对象
byte[] buff = new byte[2048];                          //创建字节数组
int len = -1;                                           //声明存放读取字节数的变量
len=bin.read(buff);                                    //读取到内容并添加到字节数组
while (len!=-1){
    out.write(buff,0,len);                             //写入磁盘文件
    len=bin.read(buff);                                 //读取到内容并添加到字节数组
}
```

(3) 在 com.mingrisoft 包中再创建一个继承 JFrame 类的 MultiThreadDownFrame 窗体类, 在该类中定义一个 download() 方法, 用于实现通过线程类 DownMultiThread 下载网络资源。关键代码如下:

```
public void download(String url, String dest, int threadNum)
    throws Exception {
    URL downURL = new URL(url);                          //创建网络资源的 URL
    HttpURLConnection conn = (HttpURLConnection) downURL.openConnection();//打开网络连接
    long fileLength = -1;                                //用于存储文件长度的变量
    int stateFlagCode = conn.getResponseCode();          //获得连接状态标记代码
    if (stateFlagCode == 200) {                          //网络连接正常
```



Note

```

        fileLength = conn.getContentLength();           //获得文件的长度
        conn.disconnect();                             //取消网络连接
    }
    if (fileLength > 0) {
        long byteCounts = fileLength / threadNum + 1;   //计算每个线程的字节数
        File file = new File(dest);                     //创建目标文件的 File 对象
        int i = 0;
        while (i < threadNum) {
            long startPosition = byteCounts * i;         //定义开始位置
            long endPosition = byteCounts * (i + 1);     //定义结束位置
            if (i == threadNum - 1) {
                DownMultiThread fileThread = new DownMultiThread(url, file,
                    startPosition, 0);                   //创建 DownMultiThread 线程的实例
                new Thread(fileThread).start();          //启动线程对象
            } else {
                DownMultiThread fileThread = new DownMultiThread(url, file,
                    startPosition, endPosition);         //创建 DownMultiThread 线程的实例
                new Thread(fileThread).start();          //启动线程对象
            }
            i++;
        }
        JOptionPane.showMessageDialog(null, "完成网络资源的下载。");
    }
}

```

(4) 在 MultiThreadDownFrame 窗体类的“下载”按钮事件中，添加调用 download()方法的代码，完成网络资源的多线程下载。该按钮的事件代码如下：

```

String address = tf_address.getText(); //获得网络资源地址
download(address, "c:\\01.flv", 2);   //调用 download()方法，将下载的网络资源保存到磁盘

```

技术要点

本实例的实现主要是通过创建线程类，然后在下载时指定线程数，并通过 RandomAccessFile 类的 seek()方法定位下一个写入点，然后通过 write()方法写入文件，从而完成网络资源的多线程下载。

(1) 通过实现 Runnable 接口创建 DownMultiThread 类，在该类的成员声明区定义网络资源地址、需要写入的目标文件对象、写入的开始位置和结束位置 4 个成员变量，并通过构造方法为其赋值。关键代码如下：

```

public class DownMultiThread implements Runnable {
    private String sUrl = "";           //网络资源地址
    private File desFile;                //需要写入的目标文件对象
    private long startPos;               //写入的开始位置
    private long endPos;                 //写入的结束位置
    public DownMultiThread(String sUrl, File desFile, long startPos, long endPos) {
        this.sUrl = sUrl;               //网络资源地址
        this.desFile = desFile;          //需要写入的目标文件对象
        this.startPos = startPos;        //写入的开始位置
        this.endPos = endPos;            //写入的结束位置
    }
}

```




```
//省略了其他代码
```

```
}
```

(2) 使用 `RandomAccessFile` 类的 `seek()` 方法定位下一个写入点, 并通过 `write()` 方法将网络资源写入文件。关键代码如下:

```
RandomAccessFile out = new RandomAccessFile(desFile, "rw");//创建可读写的流对象
out.seek(startPos); //指定读写的开始标记
out.write(buff,0,len); //写入磁盘文件
```



Note

指点迷津:

上面代码中的 `buff` 是存储网络资源的字节数组, `len` 是数组中所存储内容的实际长度。

实例 254 下载网络资源的断点续传

(实例位置: 配套资源\SL\16\254)

实例说明

本实例实现了网络资源的断点续传功能。运行程序, 输入下载资源的网址, 然后按回车键, 将显示网络资源的大小、上次读取到的字节位置以及未读取的字节数, 输入下载的起始位置和结束位置, 单击窗体上的“开始下载”按钮, 开始下载网络资源, 如果没有下载完成, 可以接着上次的下载位置继续下载, 直到全部资源下载完成, 弹出消息框并退出系统, 效果如图 16.8 和图 16.9 所示。



图 16.8 下载网络资源的断点续传

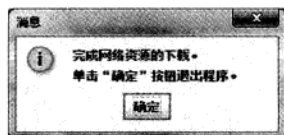


图 16.9 显示完成下载消息框

实现过程

- (1) 在 Eclipse 中新建项目 254, 在项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中创建一个继承 `JFrame` 类的 `BreakPointSuperveneFrame` 窗体类。
- (3) 在 `BreakPointSuperveneFrame` 窗体类的内容面板上, 添加相应控件完成窗体界面的设计。
- (4) 在 `BreakPointSuperveneFrame` 窗体类中定义 `download()` 方法, 用于实现网络资源的断点续传。关键代码如下:

```
public void download(long startPosition, long endPosition) {
    try {
        URL url = new URL(urlAddress); //获得网络资源的 URL
        HttpURLConnection connection = (HttpURLConnection) url
            .openConnection(); //获得连接对象
        connection.setRequestProperty("User-Agent", "NetFox"); //设置请求属性
```




Note

下载的资源

```
String rangeProperty = "bytes=" + startPosition + "-"; //定义请求范围属性
if (endPosition > 0) {
    rangeProperty += endPosition; //调整请求范围属性
}
connection.setRequestProperty("RANGE", rangeProperty); //设置请求范围属性
connection.connect(); //连接网络资源
InputStream in = connection.getInputStream(); //获得输入流对象
String file = url.getFile(); //获得文件对象
String name = file.substring(file.lastIndexOf('/') + 1); //获得文件名
FileOutputStream out = new FileOutputStream("c:/\" + name, true); //创建输出流对象, 保存

byte[] buff = new byte[2048]; //创建字节数组
int len = 0; //定义存储读取内容长度的变量
len = in.read(buff); //读取内容
while (len != -1) {
    out.write(buff, 0, len); //写入磁盘
    len = in.read(buff); //读取内容
}
out.close(); //关闭流
in.close(); //关闭流
connection.disconnect(); //断开连接
if (readToPos > 0 && readToPos == totalLength) {
    JOptionPane.showMessageDialog(null, "完成网络资源的下载。\\n 单击“确定”按钮退出程序。");
    System.exit(0);
}
} catch (Exception e) {
    e.printStackTrace();
}
}
```

(5) 在 BreakPointSuperveneFrame 窗体上, 为“网络资源的地址”标签右侧的文本框添加动作事件, 用于获得文件大小和初始化窗体上的控件。关键代码如下:

```
tf_address.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        try {
            urlAddress = tf_address.getText().trim();
            URL url = new URL(urlAddress); //获得网络资源的 URL
            HttpURLConnection connection = (HttpURLConnection) url
                .openConnection(); //获得连接对象
            connection.connect(); //连接网络资源
            totalLength = connection.getContentLength(); //获得网络资源的长度
            connection.disconnect(); //断开连接
            tf_totalLength.setText(String.valueOf(totalLength)); //显示总长度
            tf_readToPos.setText("0"); //显示上次读取到的位置
            residuaryLength = totalLength; //未读内容为文件总长度
            tf_residuaryLength.setText(String.valueOf(residuaryLength)); //显示未读内容
        } catch (MalformedURLException e1) {
            e1.printStackTrace();
        } catch (IOException e2) {
```



Note

```

        e2.printStackTrace();
    }

}

});

```

(6) 在 BreakPointSuperveneFrame 窗体上, 为“开始下载”按钮添加动作事件, 用于根据输入的起始位置和结束位置完成网络资源的断点续传。关键代码如下:

```

button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        if (totalLength == 0) {
            JOptionPane.showMessageDialog(null,
                "没有网络资源。\\n\\n 请输入正确的网址, 然后回车。");
            return;
        }
        long startPos = 0; //起始位置
        long endPos = 0; //结束位置
        try {
            startPos = Long.parseLong(tf_startPos.getText().trim()); //起始位置
            endPos = Long.parseLong(tf_endPos.getText().trim()); //结束位置
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, "输入的起始位置或结束位置不正确。");
            return;
        }
        readToPos = endPos; //记录读取到的位置
        residuaryLength = totalLength - readToPos; //记录未读内容的大小
        tf_readToPos.setText(String.valueOf(readToPos)); //显示读取到的位置
        tf_residuaryLength.setText(String.valueOf(residuaryLength)); //显示未读字节数
        tf_startPos.setText(String.valueOf(readToPos)); //设置下一个读取点的开始位置
        tf_endPos.setText(String.valueOf(totalLength)); //设置下一个读取点的结束位置
        tf_endPos.requestFocus(); //使结束位置文本框获得焦点
        tf_endPos.selectAll(); //选择结束位置文本框中的全部内容, 方便输入结束位置值
        download(startPos, endPos); //调用方法进行下载
    }
});

```

技术要点

本实例的实现主要是通过设置请求参数 RANGE 实现的, 通过该参数可以指定下载网络资源的字节区间, 从而可以实现每次下载部分网络资源的功能。

例如, 将该参数设置为“RANGE: bytes = 0-1024”, 就表示将网络资源中从 0~1024 之间的内容下载到客户机; 如果将该参数设置为“RANGE: bytes =1024-”, 就表示将网络资源中从 1024 到结束位置的内容全部下载到客户机。要设置 RANGE 属性可以通过如下代码实现:

```

connection.setRequestProperty("User-Agent", "NetFox"); //设置请求属性
String rangeProperty = "bytes=" + startPosition + "-"; //定义请求范围属性
if (endPosition > 0) {
    rangeProperty += endPosition; //调整请求范围属性
}
connection.setRequestProperty("RANGE", rangeProperty); //设置请求范围属性

```

**指点迷津:**

上面代码中的 connection 是连接到网络资源的 HttpURLConnection 对象, startPosition 是下载的起始位置, endPosition 是下载的结束位置。

**Note****实例 255 建立服务器套接字**

(实例位置: 配套资源\SL\16\255)

实例说明

在进行网络编程时,需要先运行服务器程序,服务器程序运行后,等待客户程序连接,因此需要在服务器程序中建立服务器套接字,并等待客户程序的连接。本实例演示了如何创建服务器套接字对象。运行程序,效果如图 16.10 所示。

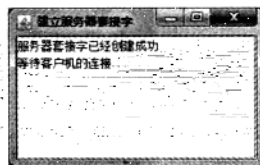


图 16.10 建立服务器套接字

实现过程

- (1) 在 Eclipse 中新建项目 255, 在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 ServerSocketFrame。
- (3) 在 ServerSocketFrame 窗体的内容面板上添加一个 JScrollPane 滚动面板控件, 然后在滚动面板的视图上放置一个 JTextArea 文本域控件, 文本域控件的名称为 ta_info, 用于显示服务器套接字的创建信息, 以及与客户端的连接信息。
- (4) 实现创建服务器套接字和等待客户端连接的代码定义在 getServer() 方法中。关键代码如下:

```
public void getServer() {
    try {
        server = new ServerSocket(1978);           //实例化 Socket 对象
        ta_info.append("服务器套接字已经创建成功\n"); //输出信息
        while (true) {
            ta_info.append("等待客户端的连接.....\n"); //输出信息
            socket = server.accept();                 //等客户端连接来实例化 Socket 对象
            ta_info.append("连接成功.....\n");       //输出信息
        }
    } catch (Exception e) {
        e.printStackTrace();                       //输出异常信息
    }
}
```

技术要点

本实例的实现主要是通过 ServerSocket 类创建绑定到指定端口的服务器套接字对象, 然后调用 ServerSocket 类的 accept() 方法监听客户端的连接。

1. ServerSocket 类的构造方法

通过 ServerSocket 类的构造方法, 可以创建绑定到指定端口的服务器套接字对象, 其定义



如下:

```
public ServerSocket(int port) throws IOException
```

参数说明

- ☒ port: 服务器套接字绑定的端口号。
- ☒ 异常: 使用该构造方法需要处理 IO 异常。

2. Accept()方法

通过 ServerSocket 类的 accept()方法,可以监听客户端的连接,并返回一个套接字对象。该方法的定义如下:

```
public Socket accept() throws IOException
```

参数说明

- ☒ 返回值: 与客户端通信的套接字对象。
- ☒ 异常: 使用该方法需要处理 IO 异常。



Note

实例 256 建立客户端套接字

(实例位置: 配套资源\SL\16\256)

实例说明

在进行网络编程时,为了与服务器端进行通信,需要创建客户端套接字,本实例演示了如何创建客户端套接字对象。运行程序,效果如图 16.11 所示。



图 16.11 建立客户端套接字

脚下留神:

本实例单独运行会发生异常,因此需要先运行实例 255 “建立服务器套接字”,然后再运行本实例,这样程序就不会发生异常了。

实现过程

- (1) 在 Eclipse 中新建项目 256,在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 ClientSocketFrame。
- (3) 在 ClientSocketFrame 窗体的内容面板上添加一个 JScrollPane 滚动面板控件,然后在滚动面板的视图上放置一个 JTextArea 文本域控件,文本域控件的名称为 ta_info,用于显示客户端与服务器的连接信息。
- (4) 实现创建客户端套接字,并与服务器连接的代码定义在 connect()方法中。关键代码如下:

```
private void connect() {                                     //连接套接字方法
    ta_info.append("尝试连接.....\n");                     //在文本域中显示提示信息
}
```



```

try {
    socket = new Socket("192.168.1.122", 1978);
    ta_info.append("完成连接。\\n");
} catch (Exception e) {
    e.printStackTrace();
}

```

//捕捉异常
//实例化 Socket 对象
//在文本域中显示提示信息
//输出异常信息



Note

技术要点

本实例的实现主要是通过 Socket 类创建到指定服务器和端口的套接字对象,从而实现与服务器的连接。

通过 Socket 类的构造方法,可以创建到指定服务器和端口的套接字对象,其定义如下:

```
public Socket(String host, int port) throws UnknownHostException, IOException
```

参数说明

- ☒ host: 是要连接服务器的主机名或 IP 地址。
- ☒ port: 连接到服务器的端口号。
- ☒ 异常: 使用该构造方法需要处理 IO 异常和未知主机异常。

实例 257 设置等待连接的超时时间

(实例位置: 配套资源\SL\16\257)

实例说明

在进行网络编程时,由于进行网络连接是比较消耗资源的,因此,可以对连接的等待时间进行设置,如果在规定的时间没有进行连接,则进行其他的处理。运行程序,等待 10 秒钟后,将弹出消息框提示连接超时,效果如图 16.12 所示。



图 16.12 设置等待连接的超时时间

实现说明

- (1) 在 Eclipse 中新建项目 257,在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 ConnectionTimeoutSetFrame。
- (3) 在 ConnectionTimeoutSetFrame 窗体的内容面板上添加一个 JScrollPane 滚动面板控件,然后在滚动面板的视图上放置一个 JTextArea 文本域控件,文本域控件的名称为 ta_info,用于显示服务器等待客户端连接的相关信息,并显示等待连接超时的信息。
- (4) 实现创建服务器套接字和设置等待连接超时时间的代码定义在 getServer()方法中。关键代码如下:

```

public void getServer() {
    try {
        server = new ServerSocket(1978);
        server.setSoTimeout(10000);
    }
}

```

//实例化 Socket 对象
//设置连接超时时间为 10 秒



```

        ta_info.append("服务器套接字已经创建成功\n");    //输出信息
        while (true) {                                  //如果套接字是连接状态
            ta_info.append("等待客户机的连接.....\n");    //输出信息
            server.accept();                              //等待客户机连接
        }
    } catch (SocketTimeoutException e) {
        ta_info.append("连接超时.....");
        JOptionPane.showMessageDialog(null, "连接超时.....");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```



Note

技术要点

本实例主要是通过 `ServerSocket` 类的实例,调用 `setSoTimeout()` 方法设置连接的等待超时时间,从而实现设置等待连接的超时时间的。`ServerSocket` 类的 `setSoTimeout()` 方法的定义如下:

```
public void setSoTimeout(int timeout) throws SocketException
```

参数说明

- ☒ `timeout`: 以毫秒为单位指定等待连接的超时时间。
- ☒ 异常: 使用该方法需要处理 `SocketException` 异常。

实例 258 获得 Socket 信息

(实例位置: 配套资源\SL\16\258)

实例说明

本实例演示在网络通信程序中,如何获取远程服务器和客户机的 IP 地址与端口号。运行程序,效果如图 16.13 和图 16.14 所示,其中图 16.13 是服务器端程序,用于等待客户端的连接;图 16.14 是客户端程序,用于获得远程服务器和客户机的 IP 地址与端口号。

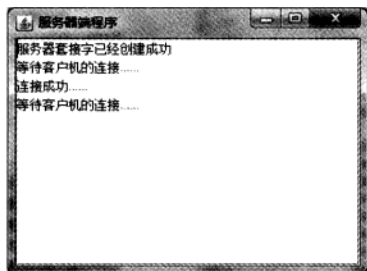


图 16.13 服务器端程序

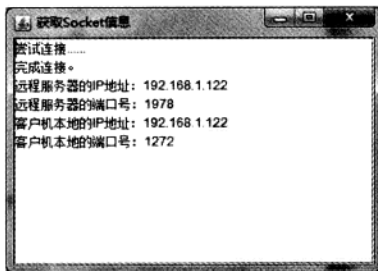


图 16.14 获取 Socket 信息

实现过程

- (1) 在 Eclipse 中新建项目 258, 在项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中创建一个继承自 `JFrame` 类的服务器窗体类 `ServerSocketFrame` 和一个客户端窗体类 `ClientSocketFrame`。



(3) 在服务器窗体类 `ServerSocketFrame` 的内容面板上添加一个 `JScrollPane` 滚动面板控件, 然后在滚动面板的视图上放置一个 `JTextArea` 文本域控件, 文本域控件的名称为 `ta_info`, 用于显示客户端与服务器的连接信息。

(4) 在客户端窗体类 `ClientSocketFrame` 的内容面板上添加一个 `JScrollPane` 滚动面板控件, 然后在滚动面板的视图上放置一个 `JTextArea` 文本域控件, 文本域控件的名称为 `ta_info`, 用于显示远程服务器和客户机的 IP 地址与端口号。

(5) 客户端窗体类 `ClientSocketFrame` 中, 用于显示远程服务器和客户机的 IP 地址与端口号的代码定义在 `connect()` 方法中。关键代码如下:

```
private void connect() {                                //连接套接字方法
    ta.append("尝试连接.....\n");                      //文本域中提示信息
    try {                                                //捕捉异常
        socket = new Socket("192.168.1.122", 1978);      //实例化 Socket 对象
        ta.append("完成连接.\n");                        //文本域中提示信息
        InetAddress netAddress = socket.getInetAddress(); //获得远程服务器的地址
        String netIp = netAddress.getHostAddress();       //获得远程服务器的 IP 地址
        int netPort = socket.getPort();                   //获得远程服务器的端口号
        InetAddress localAddress = socket.getLocalAddress(); //获得客户端的地址
        String localIp = localAddress.getHostAddress();    //获得客户端的 IP 地址
        int localPort = socket.getLocalPort();            //获得客户端的端口号
        ta.append("远程服务器的 IP 地址: " + netIp + "\n");
        ta.append("远程服务器的端口号: " + netPort + "\n");
        ta.append("客户机本地的 IP 地址: " + localIp + "\n");
        ta.append("客户机本地的端口号: " + localPort + "\n");
    } catch (Exception e) {                             //输出异常信息
        e.printStackTrace();
    }
}
```

技术要点

本实例的实现主要是通过 `Socket` 类的 `getInetAddress()` 方法获得远程服务器的地址, 使用 `getLocalAddress()` 方法获得客户端的地址实现的。

1. `getInetAddress()` 方法

通过 `Socket` 类的 `getInetAddress()` 方法, 可以获得远程服务器的地址, 进而可以获得远程服务器的 IP 地址和端口号。该方法的定义如下:

```
public InetAddress getInetAddress()
```

返回值: 此套接字连接到的远程 IP 地址; 如果套接字是未连接的, 则返回 `null`。

2. `getLocalAddress()` 方法

通过 `Socket` 类的 `getLocalAddress()` 方法, 可以获得客户端的地址, 进而可以获得客户端的 IP 地址和端口号。该方法的定义如下:

```
public InetAddress getLocalAddress()
```

返回值: 将套接字绑定到的本地地址; 如果尚未绑定套接字, 则返回 `InetAddress.anyLocalAddress()`。



实例 259 接收和发送 Socket 信息

(实例位置: 配套资源\SL\16\259)

实例说明

本实例演示在网络通信程序中, 如何实现信息的发送与接收。运行程序, 服务器端程序效果如图 16.15 所示, 客户端程序如图 16.16 所示。在图 16.16 所示的客户端程序下面的文本框中输入信息, 然后按回车键, 就会将所输入的信息发送到服务器端, 并在图 16.15 所示的服务器端显示接收到的信息。

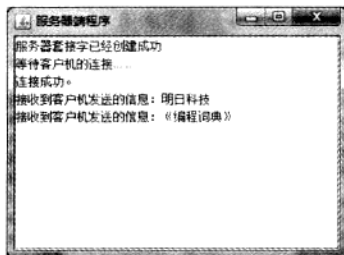


图 16.15 服务器端程序

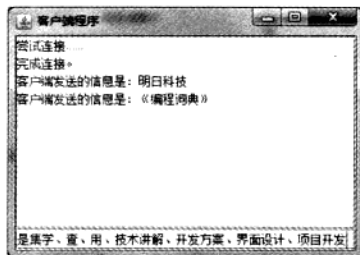


图 16.16 客户端程序

实现过程

- (1) 在 Eclipse 中新建项目 259, 在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame。
- (3) 在服务器窗体类 ServerSocketFrame 的内容面板上添加一个 JScrollPane 滚动面板控件, 然后在滚动面板的视图上放置一个 JTextArea 文本域控件, 文本域控件的名称为 ta_info, 用于显示客户端与服务器的连接信息, 以及显示接收到客户端发送的信息。
- (4) 在客户端窗体类 ClientSocketFrame 的内容面板的中部位置添加一个 JScrollPane 滚动面板控件, 然后在滚动面板的视图上放置一个 JTextArea 文本域控件, 文本域控件的名称为 ta_info, 用于显示连接信息, 以及客户端本身发送的信息; 再向客户端窗体的内容面板的底部位置添加一个 JTextField 文本框, 文本框的名称为 tf_send, 用于向服务器端发送信息。

服务器端程序的主要代码

- (5) 在服务器窗体类 ServerSocketFrame 的成员声明区, 声明输入流、服务器套接字等成员变量, 以方便在其他方法中使用。关键代码如下:

```
private BufferedReader reader;           //声明 BufferedReader 对象
private ServerSocket server;             //声明 ServerSocket 对象
private Socket socket;                   //声明 Socket 对象
```

- (6) 在服务器窗体类 ServerSocketFrame 中定义 getServer() 方法, 用于创建服务器端套接字、监听客户端程序, 以及创建输入流对象用于接收客户端发送的信息。关键代码如下:

```
public void getServer() {
    try {
        server = new ServerSocket(1978);           //实例化 ServerSocket 对象
```





Note

```

ta_info.append("服务器套接字已经创建成功\n"); //输出信息
while (true) { //如果套接字是连接状态
    ta_info.append("等待客户机的连接.....\n"); //输出信息
    socket = server.accept(); //实例化 Socket 对象
    ta_info.append("连接成功。 \n"); //输出信息
    reader = new BufferedReader(new InputStreamReader(socket
        .getInputStream())); //实例化 BufferedReader 对象
    getClientInfo(); //调用 getClientInfo()方法, 该方法也是一个自定义的成员方法
}
} catch (Exception e) {
    e.printStackTrace(); //输出异常信息
}
}

```

(7) 在服务器窗体类 `ServerSocketFrame` 中再定义一个 `getClientInfo()` 方法, 用于接收客户端发送的信息, 以及关闭输入流对象和套接字对象。关键代码如下:

```

private void getClientInfo() {
    try {
        while (true) { //如果套接字是连接状态
            ta_info.append("接收到客户机发送的信息: " + reader.readLine() + "\n"); //获得客户
            端信息
        }
    } catch (Exception e) {
        ta_info.append("客户端已退出。 \n"); //输出异常信息
    } finally {
        try {
            if (reader != null) {
                reader.close(); //关闭流
            }
            if (socket != null) {
                socket.close(); //关闭套接字
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```

客户端程序的主要代码

(8) 在客户端窗体类 `ClientSocketFrame` 的成员声明区, 声明输出流和套接字对象, 以方便在其他方法中使用。关键代码如下:

```

private PrintWriter writer; //声明 PrintWriter 类对象
private Socket socket; //声明 Socket 对象

```

(9) 在客户端窗体类 `ClientSocketFrame` 中定义 `connect()` 方法, 用于创建套接字对象和输出流对象, 以及在文本域中显示连接信息。关键代码如下:

```

private void connect() { //连接套接字方法
    ta_info.append("尝试连接.....\n"); //文本域中提示信息
    try { //捕捉异常
        socket = new Socket("192.168.1.122", 1978); //实例化 Socket 对象
        writer = new PrintWriter(socket.getOutputStream(), true); //创建输出流对象
        ta_info.append("完成连接。 \n"); //文本域中提示信息
    }
}

```



```

    } catch (Exception e) {
        e.printStackTrace();           //输出异常信息
    }
}

```

(10) 在客户端窗体类 ClientSocketFrame 中为文本框 tf_send 添加事件代码, 实现向服务器端发送信息, 其事件代码如下:

```

tf_send.addActionListener(new ActionListener() {           //绑定事件
    public void actionPerformed(ActionEvent e) {
        writer.println(tf_send.getText());                 //将文本框中的信息写入流
        ta_info.append("客户端发送的信息是: " + tf_send.getText() + "\n"); //将文本框中的信息
显示在文本域中
        tf_send.setText("");                               //将文本框清空
    }
});

```



Note

脚下留神:

要使本程序能够正常运行, 必须要先运行服务器端程序, 然后再运行客户端程序, 否则程序会发生异常。

技术要点

本实例主要是通过 Socket 类的 `getInputStream()` 方法获得输入流对象, 使用 `getOutputStream()` 方法获得输出流对象实现的。

1. `getInputStream()` 方法

通过 Socket 类的 `getInputStream()` 方法, 可以获得输入流对象, 该输入流对象用于接收对方发送的信息。该方法的定义如下:

```
public InputStream getInputStream() throws IOException
```

参数说明

- ☒ 返回值: 从此套接字读取字节的输入流, 用于接收对方发送的信息。
- ☒ 异常: 使用该方法需要处理 IO 异常。

2. `getOutputStream()` 方法

通过 Socket 类的 `getOutputStream()` 方法, 可以获得输出流对象, 该输出流对象用于向对方发送信息。该方法的定义如下:

```
public OutputStream getOutputStream() throws IOException
```

参数说明

- ☒ 返回值: 将字节写入此套接字的输出流, 用于向对方发送信息。
- ☒ 异常: 使用该方法需要处理 IO 异常。

实例 260 关闭 Socket 缓冲

(实例位置: 配套资源\SL\16\260)

实例说明

在网络应用程序中, 对于一般的网络通信, 是允许数据延迟的, 而对于一些特殊的网络应



用,例如需要实时监控的网络程序以及游戏,就必须禁用数据延迟,如果允许数据延迟,将会极大地影响数据传输的速度,本实例将演示如何关闭 Socket 缓冲。运行程序,效果如图 16.17 所示。

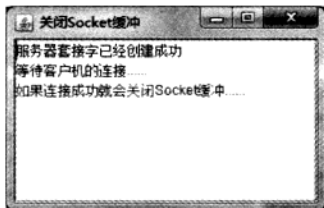


图 16.17 关闭 Socket 缓冲

实现过程

- (1) 在 Eclipse 中新建项目 260, 在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 ServerSocketFrame。
- (3) 在 ServerSocketFrame 窗体的内容面板上添加一个 JScrollPane 滚动面板控件, 然后在滚动面板的视图上放置一个 JTextArea 文本域控件, 文本域控件的名称为 ta_info, 用于显示服务器等待客户端连接的相关信息, 并显示如果连接成功就会关闭 Socket 缓冲的信息。
- (4) 实现创建服务器套接字和设置关闭 Socket 缓冲的代码定义在 getServer()方法中。关键代码如下:

```
public void getserver() {  
    try {  
        server = new ServerSocket(1978);           //实例化 Socket 对象  
        ta_info.append("服务器套接字已经创建成功\n"); //输出信息  
        while (true) {  
            ta_info.append("等待客户端的连接.....\n"); //输出信息  
            ta_info.append("如果连接成功就会关闭 Socket 缓冲.....\n"); //输出信息  
            socket = server.accept();                //实例化 Socket 对象  
            socket.setTcpNoDelay(true);              //关闭 Socket 缓冲, 提高数据传输速度  
            ta_info.append("已经关闭 Socket 缓冲.....\n"); //输出信息  
        }  
    } catch (Exception e) {  
        e.printStackTrace();                        //输出异常信息  
    }  
}
```

脚下留神:

本实例是在服务器端设置关闭套接字缓冲的, 由于关闭客户端套接字缓冲的方法与关闭服务器端套接字的方法相同, 所以可以在创建客户端套接字后, 调用 setTcpNoDelay()方法关闭客户端 Socket 缓冲。

技术要点

本实例主要是通过 Socket 类的实例, 调用 setTcpNoDelay()方法, 将参数设置为 true 来关闭 Socket 缓冲的。该方法的定义如下:

```
public void setTcpNoDelay(boolean on) throws SocketException
```



参数说明

- ☒ on: 为 true 时表示关闭 Socket 缓冲, 为 false 表示启用 Socket 缓冲。
- ☒ 异常: 使用该方法需要处理 SocketException 异常。

实例 261 使用 Socket 通信

(实例位置: 配套资源\SL\16\261)



Note

实例说明

本实例使用套接字实现了服务器端与客户端的通信。运行程序, 在服务器端的文本框中输入信息, 然后按回车键, 客户端就会收到服务器端发送的信息; 在客户端的文本框中输入信息, 然后按回车键, 服务器端就会收到客户端发送的信息, 发送信息后的效果如图 16.18 和图 16.19 所示。

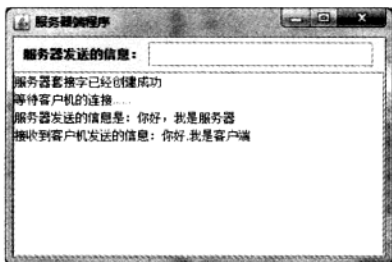


图 16.18 服务器端程序

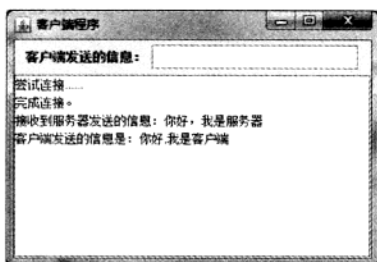


图 16.19 客户端程序

实现过程

(1) 在 Eclipse 中新建项目 261, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame。

(3) 在服务器窗体类 ServerSocketFrame 的内容面板中央添加一个 JScrollPane 滚动面板控件, 然后在滚动面板的视图上放置一个 JTextArea 文本域控件, 文本域控件的名称为 ta_info, 用于显示客户端与服务器的连接信息, 以及显示接收到客户端发送的信息; 再在服务器窗体的内容面板上部位置添加一个 JPanel 面板控件, 并在面板上添加一个 JLabel 标签和一个 JTextField 文本框, 文本框的名称为 tf_send, 用于向客户端发送信息。

(4) 在客户端窗体类 ClientSocketFrame 的内容面板的中部位置添加一个 JScrollPane 滚动面板控件, 然后在滚动面板的视图上放置一个 JTextArea 文本域控件, 文本域控件的名称为 ta_info, 用于显示连接信息, 以及客户端本身发送的信息; 再向客户端窗体的内容面板的上部位置添加一个 JPanel 面板控件, 并在面板上添加一个 JLabel 标签和一个 JTextField 文本框, 文本框的名称为 tf_send, 用于向用户服务器端发送信息。

服务器端程序的主要代码

(5) 在服务器窗体类 ServerSocketFrame 的成员声明区, 声明输出流、输入流、服务器套接字等成员变量, 以方便在其他方法中使用。关键代码如下:

```
private PrintWriter writer;           //声明 PrintWriter 类对象
private BufferedReader reader;        //声明 BufferedReader 对象
```




```
private ServerSocket server;           //声明 ServerSocket 对象
private Socket socket;                 //声明 Socket 对象
```

(6) 在服务器窗体类 `ServerSocketFrame` 中定义 `getServer()` 方法, 用于创建服务器端套接字、监听客户端程序, 以及创建向客户端发送信息的输出流对象、创建用于接收客户端发送信息的输入流对象。关键代码如下:

```
public void getServer() {
    try {
        server = new ServerSocket(1978);           //实例化 Socket 对象
        ta_info.append("服务器套接字已经创建成功\n"); //输出信息
        while (true) {
            ta_info.append("等待客户机的连接.....\n"); //输出信息
            socket = server.accept();                //实例化 Socket 对象
            reader = new BufferedReader(new InputStreamReader(socket
                .getInputStream()));                //实例化 BufferedReader 对象
            writer = new PrintWriter(socket.getOutputStream(), true);
            getClientInfo();                        //调用 getClientInfo() 方法
        }
    } catch (Exception e) {
        e.printStackTrace();                      //输出异常信息
    }
}
```

(7) 在服务器窗体类 `ServerSocketFrame` 中再定义一个 `getClientInfo()` 方法, 用于接收客户端发送的信息, 以及关闭输入流对象和套接字对象。关键代码如下:

```
private void getClientInfo() {
    try {
        while (true) {
            String line = reader.readLine();        //如果套接字是连接状态
                                                    //读取客户端发送的信息
            if (line != null)
                ta_info.append("接收到客户机发送的信息: " + line + "\n"); //获得客户端信息
        }
    } catch (Exception e) {
        ta_info.append("客户端已退出。 \n");        //输出异常信息
    } finally {
        try {
            if (reader != null) {
                reader.close();                      //关闭流
            }
            if (socket != null) {
                socket.close();                      //关闭套接字
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

(8) 在服务器端窗体类 `ServerSocketFrame` 中为文本框 `tf_send` 添加事件代码, 实现向客户端发送信息, 其事件代码如下:

```
tf_send.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
```



```

writer.println(tf_send.getText());           //将文本框中的信息写入流
ta_info.append("服务器发送的信息是: " + tf_send.getText() + "\n"); //将文本框中的信息
显示在文本域中
tf_send.setText("");                         //将文本框清空
    }
};

```

客户端程序的主要代码

(9) 在客户端窗体类 ClientSocketFrame 的成员声明区, 声明输出流、输入流和套接字对象, 以方便在其他方法中使用。关键代码如下:

```

private PrintWriter writer;           //声明 PrintWriter 类对象
private BufferedReader reader;        //声明 BufferedReader 对象
private Socket socket;                //声明 Socket 对象

```

(10) 在客户端窗体类 ClientSocketFrame 中定义 connect() 方法, 用于创建套接字对象、输入流和输出流对象, 以及在文本域中显示连接信息和接收服务器端发送的信息。关键代码如下:

```

private void connect() {               //连接套接字方法
    ta_info.append("尝试连接.....\n"); //文本域中提示信息
    try {                               //捕捉异常
        socket = new Socket("192.168.1.122", 1978); //实例化 Socket 对象
        while (true) {
            writer = new PrintWriter(socket.getOutputStream(), true); //创建输出流对象
            reader = new BufferedReader(new InputStreamReader(socket
                .getInputStream())); //实例化 BufferedReader 对象
            ta_info.append("完成连接.\n"); //文本域中提示信息
            getClientInfo();
        }
    } catch (Exception e) {
        e.printStackTrace();           //输出异常信息
    }
}

```

(11) 在客户端窗体类 ClientSocketFrame 中再定义一个 getClientInfo() 方法, 用于接收服务器端发送的信息, 以及关闭输入流对象和套接字对象。关键代码如下:

```

private void getClientInfo() {
    try {
        while (true) { //如果套接字是连接状态
            if (reader != null) {
                String line = reader.readLine(); //读取服务器发送的信息
                if (line != null)
                    ta_info.append("接收到服务器发送的信息: " + line + "\n"); //获得客户端信息
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if (reader != null) {
                reader.close(); //关闭流
            }
        }
    }
}

```



Note



Note

```
        if (socket != null) {  
            socket.close();           //关闭套接字  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
}
```

(12) 在客户端窗体类 ClientSocketFrame 中为文本框 tf_send 添加事件代码, 实现向服务器端发送信息, 其事件代码如下:

```
tf_send.addActionListener(new ActionListener() {           //绑定事件  
    public void actionPerformed(ActionEvent e) {  
        writer.println(tf_send.getText());                 //将文本框中的信息写入流  
        ta_info.append("客户端发送的信息是: " + tf_send.getText()  
            + "\n");                                         //将文本框中的信息显示在文本域中  
        tf_send.setText("");                               //将文本框清空  
    }  
});
```

技术要点

本实例的实现主要是通过 Socket 类的 `getInputStream()` 方法获得输入流对象, 并借助 `InputStreamReader` 类将输入流对象转换为 `BufferedReader` 对象读取接收到的信息; 使用 `getOutputStream()` 方法获得输出流对象, 并创建了 `PrintWriter` 对象发送信息。

1. `getInputStream()` 方法

通过 Socket 类的 `getInputStream()` 方法, 可以获得输入流对象, 该输入流对象用于接收对方发送的信息。该方法的定义如下:

```
public InputStream getInputStream() throws IOException
```

参数说明

- ☑ 返回值: 从此套接字读取字节的输入流, 用于接收对方发送的信息。
- ☑ 异常: 使用该方法需要处理 IO 异常。

2. `InputStreamReader` 类的构造方法

`InputStreamReader` 类是字节流通向字符流的桥梁, 该类的构造方法定义如下:

```
public InputStreamReader(InputStream in)
```

参数说明

in: 字节输入流对象。

3. `BufferedReader` 类的构造方法

`BufferedReader` 类是缓冲字符输入流, 可以通过 `InputStreamReader` 类的实例作为 `BufferedReader` 类构造方法的参数创建 `BufferedReader` 对象。`BufferedReader` 类的构造方法定义如下:

```
public BufferedReader(Reader in)
```

参数说明

in: 字符输入流对象。



Note

4. getOutputStream()方法

通过 Socket 类的 getOutputStream()方法,可以获得输出流对象,该输出流对象用于向对方发送信息。该方法的定义如下:

```
public OutputStream getOutputStream() throws IOException
```

参数说明

- ☒ 返回值: 将字节写入此套接字的输出流,用于向对方发送信息。
- ☒ 异常: 使用该方法需要处理 IO 异常。

5. PrintWriter 类的构造方法

通过 PrintWriter 类的构造方法,用通过 Socket 类的 getOutputStream()方法获得的输出流对象作为参数,创建 PrintWriter 对象。PrintWriter 类的构造方法定义如下:

```
public PrintWriter(OutputStream out, boolean autoFlush)
```

参数说明

- ☒ out: 用通过 Socket 类的 getOutputStream()方法获得的输出流对象。
- ☒ autoFlush: 如果为 true,则 println()、printf()或 format()方法将刷新输出缓冲区,反之则不刷新输出缓冲区。

实例 262 防止 Socket 传递汉字乱码

(实例位置: 配套资源\SL\16\262)

实例说明

在使用套接字进行网络编程时,使用 Socket 传递汉字有时会出现乱码。本实例将讲解如何防止出现汉字乱码的问题,运行程序,效果如图 16.20 和图 16.21 所示。

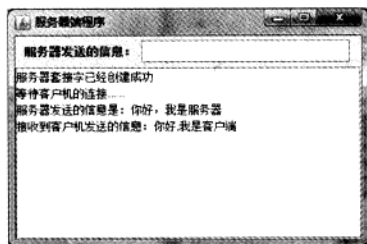


图 16.20 防止传递汉字乱码服务器端

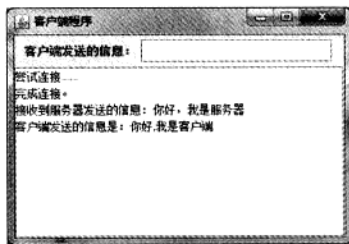


图 16.21 防止传递汉字乱码客户端

实现过程

(1) 在 Eclipse 中新建项目 262,在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame。

(3) 在服务器窗体类 ServerSocketFrame 的内容面板中央添加一个 JScrollPane 滚动面板控件,然后在滚动面板的视图上放置一个 JTextArea 文本域控件,文本域控件的名称为 ta_info,用于显示客户端与服务器的连接信息,以及显示接收到客户端发送的信息;再在服务器窗体的内容面板上部位置添加一个 JPanel 面板控件,并在面板上添加一个 JLabel 标签和一个 JTextField



文本框，文本框的名称为 `tf_send`，用于向客户端发送信息。

(4) 在客户端窗体类 `ClientSocketFrame` 的内容面板中部位置添加一个 `JScrollPane` 滚动面板控件，然后在滚动面板的视图上放置一个 `JTextArea` 文本域控件，文本域控件的名称为 `ta_info`，用于显示连接信息，以及客户端本身发送的信息；再向客户端窗体的内容面板上部位置添加一个 `JPanel` 面板控件，并在面板上添加一个 `JLabel` 标签和一个 `JTextField` 文本框，文本框的名称为 `tf_send`，用于向服务器端发送信息。

服务器端程序的主要代码

(5) 在服务器窗体类 `ServerSocketFrame` 中定义 `getServer()` 方法，用于创建服务器端套接字、监听客户端程序，以及创建向客户端发送信息的输出流对象和用于接收客户端发送信息的输入流对象。关键代码如下：

```
public void getServer() {
    try {
        server = new ServerSocket(1978);           //实例化 Socket 对象
        ta_info.append("服务器套接字已经创建成功\n"); //输出信息
        while (true) {
            ta_info.append("等待客户机的连接.....\n"); //如果套接字是连接状态 //输出信息
            socket = server.accept();                //实例化 Socket 对象
            reader = new BufferedReader(new InputStreamReader(socket
                .getInputStream(), "UTF-8"));        //实例化 BufferedReader 对象
            out = new OutputStreamWriter(socket.getOutputStream(), "UTF-8"); //实例化
OutputStreamWriter 对象
            writer = new PrintWriter(out, true);      //实例化 PrintWriter 对象
            getClientInfo();                          //调用 getClientInfo()方法
        }
    } catch (Exception e) {
        e.printStackTrace();                        //输出异常信息
    }
}
```

客户端程序的主要代码

(6) 在客户端窗体类 `ClientSocketFrame` 中定义 `connect()` 方法，用于创建套接字对象、输入流和输出流对象，以及在文本域中显示连接信息和接收服务器端发送的信息。关键代码如下：

```
private void connect() {
    ta_info.append("尝试连接.....\n"); //连接套接字方法 //文本域中提示信息
    try {
        socket = new Socket("192.168.1.122", 1978); //捕捉异常 //实例化 Socket 对象
        while (true) {
            out = new OutputStreamWriter(socket.getOutputStream(), "UTF-8"); //实例化
OutputStreamWriter 对象
            writer = new PrintWriter(out, true); //实例化 PrintWriter 对象
            reader = new BufferedReader(new InputStreamReader(socket
                .getInputStream(), "UTF-8")); //实例化 BufferedReader 对象
            ta_info.append("完成连接.\n"); //文本域中提示信息
            getClientInfo(); //调用方法
        }
    } catch (Exception e) {
        e.printStackTrace(); //输出异常信息
    }
}
```



技术要点

之所以会出现传递汉字乱码,是因为发送数据和接收数据所用的编码不同,因此,为了解决传递汉字乱码,只需要在创建输入流和输出流对象时使用相同的编码,在使用 `OutputStreamWriter` 类和 `InputStreamReader` 类创建对象时,在构造方法中指定相同的编码。



Note

1. `OutputStreamWriter` 类

`OutputStreamWriter` 类是字节流通向字符流的桥梁,该类的构造方法定义如下:

```
public OutputStreamWriter(OutputStream out, String charsetName) throws UnsupportedEncodingException
```

参数说明

- ☒ `out`: 字节输出流对象。
- ☒ `charsetName`: 受支持的字符集名称。
- ☒ 异常: 使用该方法需要处理 `UnsupportedEncodingException` 异常。

2. `InputStreamReader` 类

`InputStreamReader` 类是字节流通向字符流的桥梁,该类的构造方法定义如下:

```
public InputStreamReader(InputStream in, String charsetName) throws UnsupportedEncodingException
```

参数说明

- ☒ `in`: 字节输入流对象。
- ☒ `charsetName`: 受支持的字符集名称。
- ☒ 异常: 使用该方法需要处理 `UnsupportedEncodingException` 异常。

实例 263 使用 Socket 传递对象

(实例位置: 配套资源\SL\16\263)

实例说明

在使用套接字进行网络编程时,有时需要通过 `Socket` 传递对象。本实例讲解了如何通过套接字传递对象,运行程序,效果如图 16.22 和图 16.23 所示。

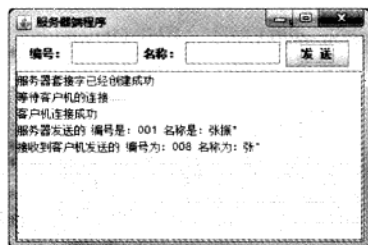


图 16.22 使用 `Socket` 传递对象服务器端

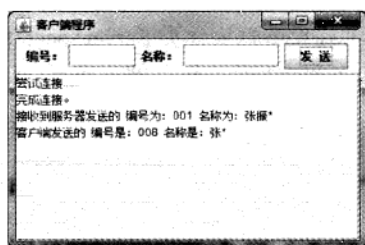


图 16.23 使用 `Socket` 传递对象客户端

实现过程

- (1) 在 Eclipse 中新建项目 263, 在项目中创建 `com.mingrisoft` 包。
- (2) 在 `com.mingrisoft` 包中创建一个实现 `Serializable` 接口的序列化类 `Student`, 该类的代



代码如下:

```
public class Student implements Serializable {           //序列化对象类
    private String id;                                   //类的成员变量
    private String name;                                 //类的成员变量
    public String getId() {                               //成员变量的 getter()方法
        return id;
    }
    public void setId(String id) {                       //成员变量的 setter()方法
        this.id = id;
    }
    public String getName() {                             //成员变量的 getter()方法
        return name;
    }
    public void setName(String name) {                   //成员变量的 setter()方法
        this.name = name;
    }
}
```

(3) 在项目中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame。

(4) 在服务器窗体类 ServerSocketFrame 的内容面板中央添加一个 JScrollPane 滚动面板控件, 然后在滚动面板的视图上放置一个 JTextArea 文本域控件, 文本域控件的名称为 ta_info, 用于显示客户端与服务器的连接信息, 以及显示接收到客户端发送的信息; 再在服务器窗体的内容面板上部位置添加一个 JPanel 面板控件, 并在面板上添加两个 JLabel 标签和两个 JTextField 文本框, 文本框的名称分别为 tf_id 和 tf_name, 用于为 Student 类创建的对象传递参数值, 并将 Student 对象发送到客户端。

(5) 在客户端窗体类 ClientSocketFrame 的内容面板中部位置添加一个 JScrollPane 滚动面板控件, 然后在滚动面板的视图上放置一个 JTextArea 文本域控件, 文本域控件的名称为 ta_info, 用于显示客户端与服务器的连接是否成功, 以及显示接收到服务器端发送的信息; 再在客户端窗体的内容面板上部位置添加一个 JPanel 面板控件, 并在面板上添加两个 JLabel 标签和两个 JTextField 文本框, 文本框的名称分别为 tf_id 和 tf_name, 用于为 Student 类创建的对象传递参数值, 并将 Student 对象发送到服务器端。

服务器端程序的主要代码

(6) 在服务器窗体类 ServerSocketFrame 中定义 getServer()方法, 用于创建服务器端套接字、监听客户端程序, 以及创建向客户端发送信息的输出流对象和创建用于接收客户端发送信息的输入流对象。关键代码如下:

```
server = new ServerSocket(1978);                        //实例化 Socket 对象
ta_info.append("服务器套接字已经创建成功\n");          //输出信息
while (true) {                                          //如果套接字是连接状态
    ta_info.append("等待客户机的连接.....\n");        //输出信息
    socket = server.accept();                            //实例化 Socket 对象
    ta_info.append("客户机连接成功\n");                 //输出信息
    out = new ObjectOutputStream(socket.getOutputStream()); //创建输出流对象
    in = new ObjectInputStream(socket.getInputStream());   //创建输入流对象
    getClientInfo();                                    //调用 getClientInfo()方法
}
```

(7) 在服务器窗体类 ServerSocketFrame 中定义 getClientInfo()方法, 用于接收客户端发送



的信息。关键代码如下：

```
while (true) {
    Student stud = (Student)in.readObject();           //如果套接字是连接状态
    if (stud!=null)                                     //将接收到的内容转换为 Student 类型
        ta_info.append("接收到客户机发送的 编号为: " + stud.getId() + " 名称为: "
+stud.getName() + "\n");
}
```



Note

客户端程序的主要代码

(8) 在客户端窗体类 ClientSocketFrame 中，“发送”按钮的事件用于向服务器传递对象。

“发送”按钮的事件代码如下：

```
Student stud = new Student();           //创建 Student 对象
stud.setId(tf_id.getText());            //将文本框中的内容设置为 Student 对象的 id 值
stud.setName(tf_name.getText());        //将文本框中的内容设置为 Student 对象的 name 值
try {
    out.writeObject(stud);              //将 Student 对象发送到服务器
} catch (IOException e1) {
    e1.printStackTrace();              //打印异常信息
}
//将文本框中信息显示在文本域中
ta_info.append("客户端发送的 编号是: " + tf_id.getText() + " 名称是: " + tf_name.getText() + "\n");
tf_id.setText(null);                    //将文本框清空
tf_name.setText(null);                  //将文本框清空
```

指点迷津：

上面代码使用套接字获得的输出流对象，将 Student 类创建的序列化对象发送到服务器端。

技术要点

在 Java 中使用 Socket 传递对象时，该对象必须是序列化的，在 Java 中实现 Serializable 接口的类，创建的对象就是序列化对象，可以通过 Socket 进行传递，从而实现了使用 Socket 传递对象的功能。

Serializable 接口在 java.io 包中，该接口没有方法，只是用于标识对象是可序列化的。该接口的定义如下：

```
public interface Serializable
```

使用 Serializable 接口，可以创建序列化类。

例如，创建序列化类 Student，可以用如下代码实现：

```
public class Student implements Serializable {           // 序列化对象类
    //省略了类的成员
}
```

实例 264 使用 Socket 传输图片

(实例位置：配套资源\SL\16\264)

实例说明

在使用套接字进行网络编程时，有时需要通过 Socket 传输图片。本实例讲解了如何通过套



接字传输图片，运行程序，在服务器端选择图片，单击“发送”按钮，就会将图片发送到客户端，效果如图 16.24 和图 16.25 所示，也可以在客户端选择图片，单击“发送”按钮，向服务器端发送图片。



Note



图 16.24 服务器端选择图片并发送

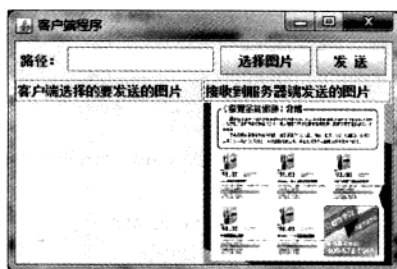


图 16.25 客户端显示接收到的图片

实现过程

(1) 在 Eclipse 中新建项目 264，在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame，并完成服务器和客户端窗体界面的设计。

服务器程序的主要代码

(3) 在服务器窗体类 ServerSocketFrame 中定义 getServer()方法，用于创建服务器端套接字、监听客户端程序，以及创建向客户端发送信息的输出流对象和创建用于接收客户端发送信息的输入流对象。关键代码如下：

```
server = new ServerSocket(1978);           //实例化 Socket 对象
while (true) {                             //如果套接字是连接状态
    socket = server.accept();               //实例化 Socket 对象
    out = new DataOutputStream(socket.getOutputStream()); //获得输出流对象
    in = new DataInputStream(socket.getInputStream());    //获得输入流对象
    getClientInfo();                       //调用 getClientInfo()方法
}
```

(4) 在服务器窗体类 ServerSocketFrame 中定义 getClientInfo()方法，用于接收客户端发送的图片。关键代码如下：

```
long lengths = in.readLong();               //读取图片文件的长度
byte[] bt = new byte[(int) lengths];        //创建字节数组
for (int i = 0; i < bt.length; i++) {
    bt[i] = in.readByte();                  //读取字节信息并存储到字节数组
}
receiveImg = new ImageIcon(bt).getImage(); //创建图像对象
receiveImagePanel.repaint();               //重新绘制图像
```

客户端程序的主要代码

(5) 在客户端窗体类 ClientSocketFrame 中，“发送”按钮的事件用于向服务器传输图片。“发送”按钮的事件代码如下：

```
DataInputStream inStream = null;           //定义数据输入流对象
if (imgFile != null) {
    lengths = imgFile.length();             //获得选择图片的大小
```



```

        inStream = new DataInputStream(new FileInputStream(imgFile)); //创建输入流对象
    } else {
        JOptionPane.showMessageDialog(null, "还没有选择图片文件。");
        return;
    }
    out.writeLong(lengths); //将文件的大小写入输出流
    byte[] bt = new byte[(int) lengths]; //创建字节数组
    int len = -1;
    while ((len = inStream.read(bt)) != -1) { //将图片文件读取到字节数组
        out.write(bt); //将字节数组写入输出流
    }

```



Note

技术要点

本实例通过使用 `DataInputStream` 类的 `read()` 方法，将图片文件读取到字节数组，然后使用 `DataOutputStream` 类从 `DataOutput` 类继承的 `write()` 方法输出字节数组，从而实现了使用 `Socket` 传输图片的功能。

1. `read()` 方法

使用 `DataInputStream` 类的 `read()` 方法，可以将文件中的信息读取到字节数组。该方法的定义如下：

```
public final int read(byte[] b) throws IOException
```

参数说明

- ☒ `b`：存储读取信息的字节数组。
- ☒ 返回值：读取到的字节总数，如果已经是文件尾，则返回-1。
- ☒ 异常：使用该方法需要处理 `IOException` 异常。

2. `write()` 方法

使用 `DataOutputStream` 类从 `DataOutput` 类继承的 `write()` 方法，可以输出字节数组。该方法的定义如下：

```
void write(byte[] b) throws IOException
```

参数说明

- ☒ `b`：需要写入输出流中的字节数组。
- ☒ 异常：使用该方法需要处理 `IOException` 异常。

实例 265 使用 `Socket` 传输音频

(实例位置：配套资源\SL\16\265)

实例说明

在使用套接字进行网络编程时，有时需要通过 `Socket` 传输音频文件。本实例讲解了如何通过套接字传输音频文件，运行程序，在服务器端选择音频文件，单击“发送”按钮，就会将音频文件发送到客户端，在客户端弹出的“保存”对话框中指定文件的保存位置和文件名，单击“保存”对话框中的“保存”按钮，完成音频文件的传输，效果如图 16.26 和图 16.27 所示，



也可以在客户端选择音频文件，单击“发送”按钮，向服务器端发送音频文件。



图 16.26 服务器端选择音频并发送

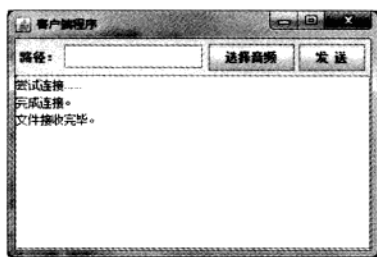


图 16.27 客户端显示文件接收完毕

实现过程

(1) 在 Eclipse 中新建项目 265，在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame，并完成服务器和客户端窗体界面的设计。

服务器端程序的主要代码

(3) 在服务器窗体类 ServerSocketFrame 中定义 getServer()方法，用于创建服务器端套接字、监听客户端程序，以及创建向客户端发送信息的输出流对象和创建用于接收客户端发送信息的输入流对象，并显示相应的信息。关键代码如下：

```
server = new ServerSocket(1978);           //实例化 Socket 对象
ta_info.append("服务器套接字已经创建成功\n"); //输出信息
ta_info.append("等待客户端的连接.....\n"); //输出信息
socket = server.accept();                  //实例化 Socket 对象
ta_info.append("客户端连接成功.....\n");   //输出信息
while (true) {                             //如果套接字是连接状态
    if (socket != null && !socket.isClosed()) {
        out = new DataOutputStream(socket.getOutputStream()); //获得输出流对象
        in = new DataInputStream(socket.getInputStream());      //获得输入流对象
        getClientInfo();                                     //调用 getClientInfo()方法
    } else {
        socket = server.accept();                          //实例化 Socket 对象
    }
}
```

(4) 在服务器窗体类 ServerSocketFrame 中定义 getClientInfo()方法，用于接收客户端发送的音频文件，并弹出“保存”对话框，保存接收到的音频文件。关键代码如下：

```
String name = in.readUTF();                //读取文件名
long lengths = in.readLong();              //读取文件的长度
byte[] bt = new byte[(int) lengths];      //创建字节数组
for (int i = 0; i < bt.length; i++) {
    bt[i] = in.readByte();                 //读取字节信息并存储到字节数组
}
FileDialog dialog = new FileDialog(ServerSocketFrame.this, "保存"); //创建对话框
dialog.setMode(FileDialog.SAVE);          //设置对话框为“保存”对话框
dialog.setFile(name);                     //设置“保存”对话框显示的文件名
```




```

dialog.setVisible(true);           //显示“保存”对话框
String path = dialog.getDirectory(); //获得文件的保存路径
String newFileName = dialog.getFile(); //获得保存的文件名
if (path == null || newFileName == null) {
    return;
}
String pathAndName = path + "\\ " + newFileName; //文件的完整路径
FileOutputStream fOut = new FileOutputStream(pathAndName); //创建输出流对象
fOut.write(bt); //向输出流写信息
fOut.flush(); //更新缓冲区
fOut.close(); //关闭输出流对象
ta_info.append("文件接收完毕。\\n");

```

客户端程序的主要代码

(5) 在客户端窗体类 ClientSocketFrame 中,“发送”按钮的事件用于向服务器传输音频文件。“发送”按钮的事件代码如下:

```

DataInputStream inStream = null; //定义数据输入流对象
if (file != null) {
    lengths = file.length(); //获得所选择音频文件的大小
    inStream = new DataInputStream(new FileInputStream(file)); //创建输入流对象
} else {
    JOptionPane.showMessageDialog(null, "还没有选择音频文件。");
    return;
}
out.writeUTF(fileName); //写入音频文件名
out.writeLong(lengths); //将文件的大小写入输出流
byte[] bt = new byte[(int) lengths]; //创建字节数组
int len = -1; //用于存储读取到的字节数
while ((len = inStream.read(bt)) != -1) { //将音频文件读取到字节数组
    out.write(bt); //将字节数组写入输出流
}
out.flush(); //更新输出流对象
out.close(); //关闭输出流对象
ta_info.append("文件发送完毕。\\n");

```

技术要点

本实例也是使用 DataInputStream 类的 read()方法和 DataOutputStream 类从 DataOutput 类继承的 write()方法实现了对音频文件的读写操作,与实例 264 不同的是,本实例使用“保存”对话框,将接收到的音频文件保存到接收方的主机上。

(1) 有关 DataInputStream 类的 read()方法,以及 DataOutputStream 类从 DataOutput 类继承的 write()方法,请参考实例 264。

(2) 使用文件保存对话框,在接收方保存所接收到的音频文件,在 Java 中将 FileDialog 对象的模式设置为 FileDialog.SAVE,这样该对话框就是文件保存对话框。创建文件保存对话框的代码如下:

```

FileDialog dialog = new FileDialog(ServerSocketFrame.this, "保存"); //创建对话框
dialog.setMode(FileDialog.SAVE); //设置对话框模式为“保存”对话框

```





实例 266 使用 Socket 传输视频

(实例位置: 配套资源\SL\16\266)



Note

实例说明

在使用套接字进行网络编程时,有时需要通过 Socket 传输视频文件。本实例讲解了如何通过套接字传输视频文件,运行程序,在服务器端或客户端选择视频文件,单击“发送”按钮,就会将视频文件发送到对方,并弹出“保存”对话框,然后由用户指定文件的保存位置和文件名,单击对话框中的“保存”按钮,完成视频文件的传输。图 16.28 就是接收到视频文件时弹出的“保存”对话框。

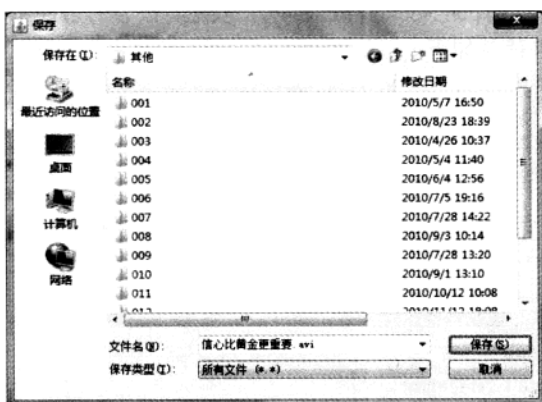


图 16.28 用于保存接收到视频文件的“保存”对话框

实现过程

- (1) 在 Eclipse 中新建项目 266, 在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame, 并完成服务器和客户端窗体界面的设计。
- (3) 在服务器窗体类 ServerSocketFrame 中定义 getServer()方法, 用于创建服务器端套接字、监听客户端程序, 以及创建向客户端发送信息的输出流对象和创建用于接收客户端发送信息的输入流对象, 并显示相应的信息。
- (4) 在服务器窗体类 ServerSocketFrame 中定义 getClientInfo()方法, 用于接收客户端发送的视频文件, 并弹出“保存”对话框, 保存接收到的视频文件。
- (5) 在客户端窗体类 ClientSocketFrame 中定义连接服务器的 connect()方法, 接收服务器信息的 getServerInfo()方法, 以及在“发送”按钮的事件添加向服务器传输视频文件代码, 完成客户端程序的功能。

指点迷津:

由于本实例的代码与实例 265 基本相同, 只是在选择要发送的文件时, 所显示的文件类型不同, 所以这里不再给出, 如果需要可以查看源程序代码。



技术要点

本实例与实例 265 基本相同,所不同的是本实例在发送视频文件时,文件选择对话框中指定的,用于选择要发送的文件类型不同而已。为了方便用户操作,在发送不同类型的文件时,为文件选择对话框指定相应的文件类型。下面是为文件选择对话框指定音频和视频的代码。

(1) 为文件选择对话框指定音频类型的文件,关键代码如下:

```
JFileChooser fileChooser = new JFileChooser();           //创建文件选择器
//创建音频过滤器
FileFilter filter = new FileNameExtensionFilter("音频文件 (WAV/MIDI/MP3/AU)", "WAV", "MID",
"MP3", "AU");
fileChooser.setFileFilter(filter);                       //设置过滤器
int flag = fileChooser.showOpenDialog(null);             //显示打开对话框
```

指点迷津:

上面代码打开的文件选择对话框中,只显示文件类型是 WAV、MID、MP3 和 AU 类型的音频文件,这样可以方便用户对音频文件进行选择。

(2) 为文件选择对话框指定视频类型的文件,关键代码如下:

```
JFileChooser fileChooser = new JFileChooser();           //创建文件选择器
//创建视频过滤器
FileFilter filter = new FileNameExtensionFilter("视频文件 (AVI/MPG/DAT/RM)", "AVI", "MPG",
"DAT", "RM");
fileChooser.setFileFilter(filter);                       //设置过滤器
int flag = fileChooser.showOpenDialog(null);             //显示打开对话框
```

指点迷津:

上面代码打开的文件选择对话框中,只显示文件类型是 AVI、MPG、DAT 和 RM 类型的视频文件,这样可以方便用户对视频文件进行选择。

实例 267 一个服务器与一个客户端通信

(实例位置: 配套资源\SL\16\267)

实例说明

在使用套接字进行网络编程时,需要在服务器和客户端之间进行通信。本实例讲解了如何实现一个服务器与一个客户端进行通信,运行程序,效果如图 16.29 和图 16.30 所示。

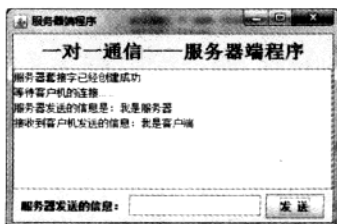


图 16.29 服务器端发送和接收信息的效果

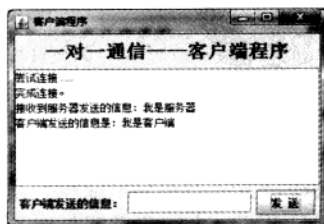


图 16.30 客户端接收和发送信息的效果





实现过程

(1) 在 Eclipse 中新建项目 267，在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame。

服务器端程序的主要代码

(3) 在服务器窗体类 ServerSocketFrame 中定义 getServer() 方法，用于创建服务器端套接字、监听客户端程序、创建向客户端发送信息的输出流对象，以及创建用于接收客户端发送信息的输入流对象。关键代码如下：

```
server = new ServerSocket(1978);           //实例化 Socket 对象
ta_info.append("服务器套接字已经创建成功\n"); //输出信息
while (true) {                             //如果套接字是连接状态
    ta_info.append("等待客户机的连接.....\n"); //输出信息
    socket = server.accept();                //实例化 Socket 对象
    reader = new BufferedReader(new InputStreamReader(socket
        .getInputStream()));                //实例化 BufferedReader 对象
    writer = new PrintWriter(socket.getOutputStream(), true); //实例化 PrintWriter 对象
    getClientInfo();                        //调用 getClientInfo()方法
}
```

(4) 在服务器窗体类 ServerSocketFrame 中，“发送”按钮用于向客户端发送信息。该按钮的事件代码如下：

```
writer.println(tf_send.getText());           //将文本框中的信息写入流
ta_info.append("服务器发送的信息是：" + tf_send.getText() + "\n"); //将文本框中的信息显示在文本域中
tf_send.setText("");                         //将文本框清空
```

客户端程序的主要代码

(5) 在客户端窗体类 ClientSocketFrame 中定义 connect() 方法，用于创建与服务器连接的套接字对象、输入流对象和输出流对象、在文本域中显示与服务器的连接信息和接收到服务器端发送的信息。关键代码如下：

```
socket = new Socket("192.168.1.122", 1978); //实例化 Socket 对象
while (true) {
    writer = new PrintWriter(socket.getOutputStream(), true); //实例化 PrintWriter 对象
    reader = new BufferedReader(new InputStreamReader(socket
        .getInputStream())); //实例化 BufferedReader 对象
    ta_info.append("完成连接.\n"); //文本域中提示信息
    getServerInfo();
}
```

(6) 在客户端窗体类 ClientSocketFrame 中，“发送”按钮用于向服务器端发送信息。该按钮的事件代码如下：

```
writer.println(tf_send.getText());           //将文本框中的信息写入流
ta_info.append("客户端发送的信息是：" + tf_send.getText() + "\n"); //将文本框中的信息显示在文本域中
tf_send.setText("");                         //将文本框清空
```

技术要点

本实例在服务器端和客户端没有使用线程来处理接收到的信息，所以当有多个客户连接到服务器时，只有第一个客户能够与服务器进行通信，而其他客户必须等待，只有第一个客户退出，服务器才能与下一个客户进行通信，依此类推，只有前一个客户退出，服务器才能与下一



个客户进行通信。

(1) 服务器端通过 getClientInfo()方法来接收客户端发送的信息, 关键代码如下:

```
private void getClientInfo() {
    try {
        while (true) {
            String line = reader.readLine();           //读取客户端发送的信息
            if (line != null)
                ta_info.append("接收到客户机发送的信息: " + line + "\n"); //显示客户端发送的信息
        }
    } catch (Exception e) {
        ta_info.append("客户端已退出。 \n");           //输出异常信息
    }
}
```



Note

(2) 客户端通过 getServerInfo()方法来接收服务器端发送的信息, 关键代码如下:

```
private void getServerInfo() {
    try {
        while (true) {
            if (reader != null) {
                String line = reader.readLine();       //读取服务器发送的信息
                if (line != null)
                    ta_info.append("接收到服务器发送的信息: " + line + "\n"); //显示服务器端
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

实例 268 一个服务器与多个客户端通信

(实例位置: 配套资源\SL\16\268)

实例说明

在使用套接字进行网络编程时, 需要在服务器和客户端之间进行通信。本实例讲解了如何通过一个服务器与多个客户端进行通信, 运行程序, 服务器启动后, 启动两个客户端程序, 然后通过服务器向客户端发送信息, 两个客户端都会收到服务器发送的信息, 效果如图 16.31 和图 16.32 所示。

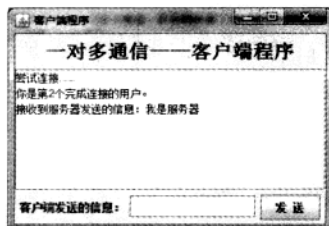
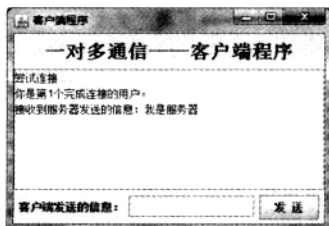


图 16.31 第一个客户端接收到的服务器信息 图 16.32 第二个客户端接收到的服务器信息



实现过程

(1) 在 Eclipse 中新建项目 268，在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame。

服务器端程序的主要代码

(3) 在服务器窗体类 ServerSocketFrame 中定义 getServer()方法，用于创建服务器端套接字、监听客户端程序、向客户端发送信息的输出流对象、向客户端发送连接用户的套接字索引，以及创建并启动线程对象，用于接收客户端发送的信息。关键代码如下：

```
server = new ServerSocket(1978);           //实例化 Socket 对象
ta_info.append("服务器套接字已经创建成功\n"); //输出信息
while (true) {                             //如果套接字是连接状态
    socket = server.accept();                //实例化 Socket 对象
    counts++;                               //计算连接客户总数
    ta_info.append("第" + counts + "个客户连接成功\n"); //输出信息
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true); //创建输出流对象
    out.println(String.valueOf(counts - 1)); //向客户端发送套接字索引
    vector.add(socket);                     //存储客户端套接字对象
    new ServerThread(socket).start();       //创建并启动线程
}
```

客户端程序的主要代码

(4) 在客户端窗体类 ClientSocketFrame 中定义 connect()方法，用于创建与服务器连接的套接字对象、输入流对象和输出流对象、从服务器读取客户端的索引，以及在文本域中显示是第几个与服务器连接的用户和接收服务器端发送的信息。关键代码如下：

```
socket = new Socket("192.168.1.122", 1978); //实例化 Socket 对象
while (true) {
    writer = new PrintWriter(socket.getOutputStream(), true); //创建输出流对象
    reader = new BufferedReader(new InputStreamReader(socket.getInputStream())); //实例化
    //BufferedReader 对象
    index = Integer.parseInt(reader.readLine()); //获得客户登录服务器的索引值
    ta_info.append("你是第" + (index + 1) + "个完成连接的用户。 \n"); //文本域中提示信息
    getServerInfo(); //调用接收服务器信息的方法
}
```

技术要点

本实例在服务器端通过线程来处理不同客户发送的信息，所以当有多个客户连接到服务器时，服务器会为每个客户建立一个线程来处理接收到的信息，由于通过线程来处理接收到每个客户的信息，因此不会产生阻塞，从而实现了一个服务器与多个客户端通信，并且在关闭客户端窗体时，会向服务器端发送退出客户的索引。

(1) 在服务器端创建线程类 ServerThread，用于接收客户端发送的信息，以及处理客户端的退出信息。该线程类中 run()方法的关键代码如下：

```
public void run() {
    try {
        if (socket != null) { //套接字不为空时实例化 BufferedReader 对象
```



Note

端信息

```

reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
int index = -1;                                //存储退出的客户端索引值
try {
    while (true) {                             //如果套接字是连接状态
        String line = reader.readLine();        //读取客户端信息
        try {
            index = Integer.parseInt(line);     //获得退出的客户端索引值
        } catch (Exception ex) {
            index = -1;
        }
        if (line != null) {
            ta_info.append("接收到客户机发送的信息:" + line + "\n"); //获得客户
        }
    }
} catch (Exception e) {
    if (index != -1) {
        vector.set(index, null);               //将退出的客户端套接字设置为 null
        ta_info.append("第" + (index + 1) + "个客户端已经退出。\\n"); //输出异常信息
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}

```

(2) 客户端窗体的关闭事件, 用于向服务器发送退出客户的索引值。客户端窗体的关闭事件代码如下:

```

addWindowListener(new WindowAdapter() {
    public void windowClosing(final WindowEvent e) { //窗体关闭事件
        writer.println(String.valueOf(index));      //向服务器端发送退出客户的索引值
    }
});

```

实例 269 客户端一对多通信

(实例位置: 配套资源\SL\16\269)

实例说明

在使用套接字进行网络编程时, 有时需要在不同的客户端之间进行通信, 其中有一种通信方式就是一个客户端与其他的多个客户端进行通信。本实例讲解了如何实现一个客户端与其他多个客户端进行通信, 运行程序, 服务器启动后, 启动 3 个客户端程序, 然后通过第一个客户端向另外两个客户端发送信息, 则另外的两个客户端都会收到服务器发送的信息, 效果如图 16.33 和图 16.34 所示。



Note

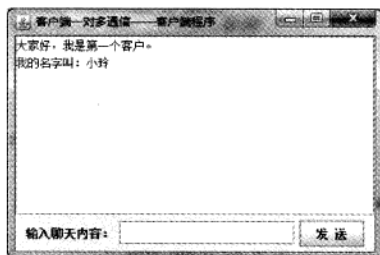


图 16.33 第二个客户端接收到第一个客户端的信息 图 16.34 第三个客户端接收到第一个客户端的信息

实现过程

- (1) 在 Eclipse 中新建项目 269，在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ClientOneToMany_ServerFrame 和一个客户端窗体类 ClientOneToMany_ClientFrame。
- (3) 在服务器窗体类 ClientOneToMany_ServerFrame 中定义 createSocket() 方法，用于创建服务器端套接字、监听客户端程序，以及创建并启动线程对象并将接收到客户端发送的信息转发给其他客户端。关键代码如下：

```
server = new ServerSocket(1978);
while (true) {
    ta_info.append("等待新客户连接.....\n");
    socket = server.accept();                //创建套接字对象
    vector.add(socket);                     //将套接字对象添加到向量对象中
    ta_info.append("客户端连接成功。" + socket + "\n");
    new ServerThread(socket).start();       //创建并启动线程对象
}
```

- (4) 在客户端窗体类 ClientOneToMany_ClientFrame 中定义 createClientSocket() 方法，用于创建与服务器连接的套接字对象、输出流对象，以及启动线程对象接收服务器端转发的信息。关键代码如下：

```
Socket socket = new Socket("192.168.1.122", 1978); //创建套接字对象
out = new PrintWriter(socket.getOutputStream(), true); //创建输出流对象
new ClientThread(socket).start(); //创建并启动线程对象
```

技术要点

本实例主要是在服务器端通过线程对客户端发送的信息进行监听，当有客户端发送信息时，就会将该信息发送给其他已经登录到服务器的客户端，但是不会向发送方发送该信息，在客户端也通过线程来监听服务器转发的信息。

- (1) 在服务器端创建线程类 ServerThread，用于接收客户端发送的信息，并转发给其他已经连接到服务器的客户端。该线程类中 run() 方法的关键代码如下：

```
while (true) {
    String info = in.readLine();           //读取信息
    for (Socket s : vector) {              //遍历所有客户端套接字对象
        if (s != socket) {                 //如果不是发送信息的套接字对象
            PrintWriter out = new PrintWriter(s.getOutputStream(), true); //创建输出流对象
            out.println(info);              //发送信息
        }
    }
}
```



Note

```
out.flush();           //刷新输出缓冲区
```

(2) 在客户端创建线程类 ClientThread, 用于接收服务器端转发的客户端信息, 并在客户端的文本域中显示接收到的信息。该线程类中 run() 方法的关键代码如下:

```
while (true) {
    String info = in.readLine();           //读取信息
    ta_info.append(info + "\n");           //在文本域中显示信息
    if (info.equals("88")) {                //如果接收到的信息是 88, 就结束线程的执行
        break;                             //结束线程
    }
}
```

实例 270 客户端一对一通信

(实例位置: 配套资源\SL\16\270)

实例说明

在使用套接字进行网络编程时, 有时需要在不同的客户端之间进行通信, 其中有一种通信方式就是一个客户端与另一个指定的客户端进行通信。本实例讲解了如何实现一个客户端与另一个指定的客户端进行通信, 运行程序, 服务器启动后, 启动 3 个客户端程序, 并分别以 aaa、bbb 和 ccc 进行登录, 然后在左侧的用户列表中选择接收信息的用户, 输入聊天信息即可发送到目标用户, 效果如图 16.35 和图 16.36 所示。

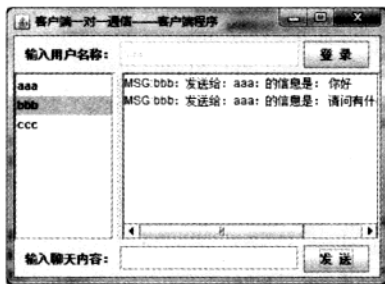


图 16.35 客户端 aaa 接收到 bbb 的信息

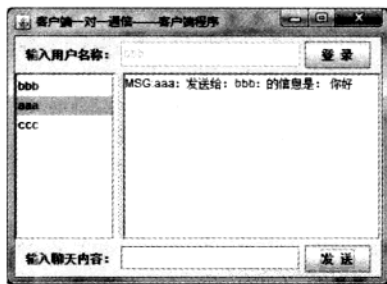


图 16.36 客户端 bbb 接收到 aaa 的信息

实现过程

(1) 在 Eclipse 中新建项目 270, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ClientOneToOne_ServerFrame 和一个客户端窗体类 ClientOneToOne_ClientFrame。

(3) 在服务器窗体类 ClientOneToMany_ServerFrame 中定义 ServerThread 线程类, 在该线程类的 run() 方法中有一部分代码用于为客户端添加用户列表, 关键代码如下:

```
if (info.startsWith("用户: ")) {           //添加登录用户到客户端列表
    key = info.substring(3, info.length()); //获得用户名并作为键使用
    map.put(key, socket);                  //添加键值对
```



Note

```

Set<String> set = map.keySet();           //获得集合中所有键的 Set 视图
Iterator<String> keyIt = set.iterator();   //获得所有键的迭代器
while (keyIt.hasNext()) {
    String receiveKey = keyIt.next();       //获得表示接收信息的键
    Socket s = map.get(receiveKey);         //获得与该键对应的套接字对象
    PrintWriter out = new PrintWriter(s
        .getOutputStream(), true);         //创建输出流对象
    Iterator<String> keyIt1 = set.iterator(); //获得所有键的迭代器
    while (keyIt1.hasNext()) {
        String receiveKey1 = keyIt1.next(); //获得键，用于向客户端添加用户列表
        out.println(receiveKey1);           //发送信息
        out.flush();                         //刷新输出缓冲区
    }
}

} else {                                   //转发接收到的消息
    //这里省略了转发接收消息的代码
}

```

(4) 在服务器窗体类 ClientOneToMany_ServerFrame 的线程类 ServerThread 中，在 run() 方法中有一部分代码用于转发客户端发送的消息，关键代码如下：

```

if (info.startsWith("用户: ")) {           //添加登录用户到客户端列表
    //这里省略了向客户端添加用户列表的代码
} else {                                     //转发接收的消息
    key = info.substring(info.indexOf(": 发送给: ") + 5, info
        .indexOf(": 的信息是: "));           //获得接收方的 key 值，即接收方的用户名
    String sendUser = info.substring(0, info
        .indexOf(": 发送给: "));           //获得发送方的 key 值，即发送方的用户名
    Set<String> set = map.keySet();           //获得集合中所有键的 Set 视图
    Iterator<String> keyIt = set.iterator(); //获得所有键的迭代器
    while (keyIt.hasNext()) {
        String receiveKey = keyIt.next();     //获得表示接收信息的键
        if (key.equals(receiveKey)
            && !sendUser.equals(receiveKey)) //如果是发送方，但不是用户本身
            Socket s = map.get(receiveKey); //获得与该键对应的套接字对象
        PrintWriter out = new PrintWriter(s
            .getOutputStream(), true);         //创建输出流对象

        out.println("MSG:"+info);             //发送信息
        out.flush();                             //刷新输出缓冲区
    }
}
}

```

(5) 在客户端窗体类 ClientOneToOne_ClientFrame 中定义 ClientThread 线程类，用于对接收到服务器的信息进行处理，如果是登录用户就添加到用户列表中，如果是消息就追加到文本域中。该线程类的 run() 方法实现该功能的关键代码如下：

```

if (!info.startsWith("MSG:")) {
    boolean itemFlag = false; //标记是否为列表框添加列表项，为 true 不添加，为 false 添加
    for (int i = 0; i < model.getSize(); i++) {
        if (info.equals((String) model.getElementAt(i))) {

```



Note

```

        itemFlag = true;           //标记为 true, 表示是用户
    }
}
if (!itemFlag) {
    model.addElement(info);       //添加列表项
} else {
    itemFlag = false;           //标记设置为 false
}
} else {
    ta_info.append(info + "\n");   //在文本域中显示信息
    if (info.equals("88")) {
        break;                   //结束线程
    }
}
}

```

技术要点

本实例主要是在服务器端通过线程对客户端发送的信息进行监听, 并对登录用户和消息分别进行处理。如果是登录用户, 就将所有用户添加到客户端的用户列表中; 如果是消息, 就转发给指定的用户; 客户端则通过线程对接收到的信息进行处理, 如果是登录用户就添加到用户列表中, 如果是消息就追加到文本域中。

(1) 在服务器端创建线程类 `ServerThread`, 用于对登录用户和消息分别进行处理。如果是登录用户, 就将所有用户添加到客户端的用户列表中; 如果是消息就转发给指定的用户。该线程类 `run()` 方法中用于判断是登录用户还是消息的关键代码如下:

```

if (info.startsWith("用户: ")) { //添加登录用户到客户端列表
    //省略了向客户端用户列表添加登录用户的代码
} else { //转发接收的消息
    //省略了向客户端指定用户发送信息的代码
}
}

```

(2) 在客户端创建线程类 `ClientThread`, 用于对接收到的信息进行处理, 如果是登录用户就添加到用户列表中, 如果是消息就追加到文本域中。该线程类 `run()` 方法中用于判断接收到的是登录用户还是消息的关键代码如下:

```

if (!info.startsWith("MSG:")) { //如果接收到的不是消息, 则是登录用户
    //省略了向用户列表添加登录用户的代码
} else { //如果接收到的是消息
    ta_info.append(info + "\n"); //在文本域中显示信息
    if (info.equals("88")) {
        break; //结束线程
    }
}
}

```

实例 271 基于 Socket 的数据库编程

(实例位置: 配套资源\SL\16\271)

实例说明

本实例实现了基于 Socket 的数据库编程, 运行程序, 在客户端输入信息, 单击“保存”按



钮, 通过 Socket 将信息发送到服务器端, 服务器端接收到信息后, 将数据信息保存到数据库中, 将反馈给客户端程序, 效果如图 16.37 和图 16.38 所示。

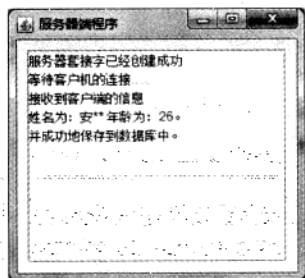


图 16.37 将客户端发送的信息保存到数据库

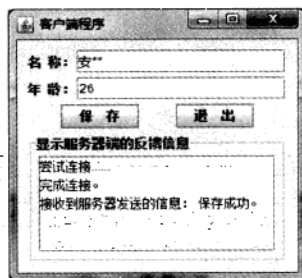


图 16.38 获得服务器端保存成功的反馈信息

实现过程

- (1) 在 Eclipse 中新建项目 271, 在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个 DAO 类, 用于加载数据库驱动及建立到数据库的连接。关键代码如下:

```
public DAO() {
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");           //加载数据库驱动
    } catch (ClassNotFoundException e) {
        JOptionPane.showMessageDialog(null, "数据库驱动加载失败。\\n" + e.getMessage());
    }
}

public static Connection getConn() {
    try {
        Connection conn = null;                                   //定义数据库连接
        //数据库 db_picture.mdb 的 URL
        String url = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=src/database/
db_picture.mdb";
        String username = "";                                     //数据库的用户名
        String password = "";                                    //数据库密码
        conn = DriverManager.getConnection(url, username, password); //建立连接
        return conn;                                             //返回连接
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "数据库连接失败。\\n" + e.getMessage());
        return null;
    }
}
```

- (3) 在项目中创建一个继承自 JFrame 类的服务器窗体类 DatabaseServerFrame 和一个客户端窗体类 DatabaseClientFrame。

- (4) 在服务器窗体类 DatabaseServerFrame 中定义一个 getClientInfo() 方法, 用于接收客户端发送的信息, 并将接收到的信息保存到数据库中, 然后反馈给客户端程序。关键代码如下:

```
BufferedReader reader;                                         //声明 BufferedReader 对象
while (true) {                                                 //如果套接字是连接状态
    reader = new BufferedReader(new InputStreamReader(socket
```




Note

```

        .getInputStream()); //实例化 BufferedReader 对象
String line = reader.readLine(); //读取客户端信息
if (line != null) {
    String[] value = new String[2]; //创建数组
    value[0] = line.substring(0, line.indexOf(":data:")); //获得姓名
    value[1] = line.substring(line.indexOf(":data:") + 6); //获得年龄
    ta_info.append("接收到客户端的信息\n姓名为: "+value[0]+" 年龄为: "+value[1]+".\n");
    try {
        Connection conn = DAO.getConn(); //获得数据库连接
        String sql = "insert into tb_employee (name,age) values(?,?)"; //定义 SQL 语句
        PreparedStatement ps = conn.prepareStatement(sql); //创建 PreparedStatement 对象,并
        ps.setString(1, value[0]); //为第 1 个参数赋值
        ps.setInt(2, Integer.parseInt(value[1])); //为第 2 个参数赋值
        int flag = ps.executeUpdate(); //执行 SQL 语句, 获得更新记录数
        ps.close(); //关闭 PreparedStatement 对象
        conn.close(); //关闭连接
        if (flag > 0) {
            ta_info.append("并成功地保存到数据库中。.\n");
            writer.println("保存成功。"); //向客户端输出保存成功的信息
        } else {
            writer.println("保存失败。.\n"); //向客户端输出保存成功的信息
        }
    } catch (SQLException ee) {
        writer.println("保存失败。.\n" + ee.getMessage()); //向客户端输出保存成功的信息
    }
}
}
}

```

传递 SQL 语句

指点迷津:

由于本实例的其他代码与前几个实例相似, 这里不再给出, 如果需要请查看源程序代码。

技术要点

本实例通过 Socket 将客户端的信息发送到服务器端, 然后服务器端对接收到的信息进行处理, 并通过 JDBC-ODBC 桥的方式连接到 Access 数据库, 然后通过 JDBC 技术将数据信息保存到数据库中。

- (1) 使用 JDBC-ODBC 桥加载数据库驱动, 其实现代码如下:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //加载数据库驱动
```

- (2) 连接到项目中的 src/database/db_picture.mdb 数据库的代码如下:

```

Connection conn = null; //定义数据库连接
//数据库 db_picture.mdb 的 URL
String url = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=src/database/db_picture.mdb ";
String username = ""; //数据库的用户名
String password = ""; //数据库密码
conn = DriverManager.getConnection(url, username, password); //建立连接

```




实例 272 使用 Proxy 创建代理服务器

(实例位置: 配套资源\SL\16\272)



Note

实例说明

本实例实现了使用 Proxy 创建的代理服务器访问网络资源, 运行程序, 输入代理服务器的地址和端口号, 然后输入要访问的网站网址, 就会在“访问结果”标签下面的文本域中显示相应的信息, 效果如图 16.39 所示。

实现过程

(1) 在 Eclipse 中新建项目 272, 在项目创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 UserProxyFrame, 并完成窗体界面的设计。

(3) 在窗体类 UserProxyFrame 中定义 accessUrl() 方法, 用于实现通过代理访问网络资源。关键代码如下:

```
url = new URL(accessAddress); //创建 URL 对象
proxy = new Proxy(Proxy.Type.HTTP, new InetSocketAddress(proxyAddress, proxyPort)); //创建代理
urlConn = url.openConnection(proxy); //通过代理打开连接
scanner = new Scanner(urlConn.getInputStream(), "UTF8"); //通过流创建扫描器
ta_show.setText(null); //清空文本域的内容
StringBuffer sb = new StringBuffer(); //创建字符串缓存
while (scanner.hasNextLine()) { //判断扫描器是否有数据
    String line = scanner.nextLine(); //从扫描器获得一行数据
    sb.append(line + "\n"); //向字符串缓存添加信息
}
if (sb != null) {
    ta_show.append(sb.toString()); //在文本域中显示信息
}
```

(4) 在输入网址文本框的动作事件中, 添加用于调用 accessUrl() 方法的代理, 将读取的网络资源显示到文本域中, 其事件代码如下:

```
try {
    String proxyAddress = tf_proxyAddress.getText().trim(); //代理服务器的地址
    int proxyPort = Integer.parseInt(tf_proxyPort.getText().trim()); //代理服务器的端口
    String accessAddress = tf_accessAddress.getText().trim(); //需要打开的网站网址
    accessUrl(proxyAddress, proxyPort, accessAddress); //调用方法, 使用代理访问网站
} catch (Exception ex) {
    JOptionPane.showMessageDialog(null, "输入的信息有误。");
}
```

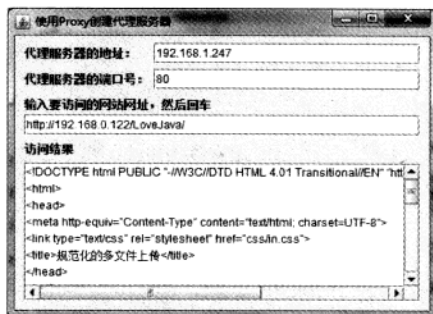


图 16.39 使用 Proxy 创建代理服务器



技术要点

本实例使用 Proxy 类创建代理, 并在使用 URL 类的 openConnection() 方法打开连接时, 将该代理传递给 openConnection() 方法, 从而实现了使用代理服务器的功能。

(1) 使用 Proxy 类可以创建代理, 创建代理的代码如下:

```
proxy = new Proxy(Proxy.Type.HTTP, new InetSocketAddress(proxyAddress, proxyPort)); //创建代理
```

指点迷津:

上面代码中的 Proxy.Type.HTTP 用于指定代理类型为 HTTP 代理, InetSocketAddress 是代理套接字地址, 用于指定代理服务器和端口号。

(2) 为文件选择对话框指定视频类型的文件, 代码如下:

```
URLConnection urlConn = url.openConnection(proxy); //通过代理服务器打开连接
```

指点迷津:

上面代码中的 url 是要访问网站的 URL 对象, proxy 是所创建的代理, urlConn 是所要打开连接的 URLConnection 对象。

实例 273 使用 ProxySelector 选择代理服务器

(实例位置: 配套资源\SL\16\273)

实例说明

本实例实现了使用 ProxySelector 选择代理服务器的功能, 运行程序, 输入代理服务器的地址, 然后输入要访问的网站网址, 就会在“访问结果”标签下面的文本域中显示相应的信息, 效果如图 16.40 所示。

实现过程

(1) 在 Eclipse 中新建项目 273, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 ProxySelectorFrame, 并完成窗体界面的设计。

(3) 在窗体类 ProxySelectorFrame 中定义 setProxyInfo() 方法, 用于为本地系统设置可以选择的代理。关键代码如下:

```
public void setProxyInfo(String proxyAddress) {
    Properties properties = System.getProperties(); //获得系统属性对象
    properties.setProperty("http.proxyHost", proxyAddress); //设置 HTTP 服务使用的代理服务器地址
    properties.setProperty("http.proxyPort", "80"); //设置 HTTP 服务使用的代理服务器端口
```



图 16.40 使用 ProxySelector 选择代理服务器





服务器地址

```

properties.setProperty("https.proxyHost", proxyAddress); //设置安全 HTTP 服务使用的代理
properties.setProperty("https.proxyPort", "443");//设置安全 HTTP 服务使用的代理服务器端口
properties.setProperty("ftp.proxyHost", proxyAddress); //设置 FTP 访问的代理服务器主机
properties.setProperty("ftp.proxyPort", "21"); //设置 FTP 访问的代理服务器端口
properties.setProperty("socks.ProxyHost", proxyAddress); //设置 socks 代理服务器的地址
properties.setProperty("socks.ProxyPort", "1080"); //设置 socks 代理服务器的端口
}

```

(4) 在窗体类 ProxySelectorFrame 中定义 removeProxyInfo()方法, 用于从本地系统中移除所设置的代理。关键代码如下:

```

public void removeProxyInfo() {
    Properties properties = System.getProperties(); //获得系统属性对象
    properties.remove("http.proxyHost"); //清除 HTTP 服务使用的代理服务器地址
    properties.remove("http.proxyPort"); //清除 HTTP 服务使用的代理服务器端口
    properties.remove("https.proxyHost"); //清除安全 HTTP 服务使用的代理服务器地址
    properties.remove("https.proxyPort"); //清除安全 HTTP 服务使用的代理服务器端口
    properties.remove("ftp.proxyHost"); //清除 FTP 访问的代理服务器主机
    properties.remove("ftp.proxyPort"); //清除 FTP 访问的代理服务器端口
    properties.remove("socksProxyHost"); //清除 socks 代理服务器的地址
    properties.remove("socksProxyPort"); //清除 socks 代理服务器的端口
}

```

(5) 在窗体类 ProxySelectorFrame 中定义 useHttpAccess()方法, 用于测试使用 HTTP 服务时所选择的代理服务器。关键代码如下:

```

public void useHttpAccess(String accessAddress) throws Exception{
    URL url = new URL(accessAddress); //创建 URL 对象
    URLConnection urlConn = url.openConnection(); //自动调用系统设置的 HTTP 代理服务
    器, 并打开连接
    Scanner scanner = new Scanner(urlConn.getInputStream(), "utf8");//通过流创建扫描对象
    StringBuffer sb = new StringBuffer(); //创建字符串缓存
    while (scanner.hasNextLine()) { //如果扫描器中有信息
        sb.append(scanner.nextLine()+"\n"); //读取代理服务器上的网页内容, 并添加到字符串
        缓存中
    }
    if (sb!=null) {
        ta_info.append(sb.toString()); //显示网页内容
    }
}

```

(6) 在输入网址文本框的动作事件中, 实现了用于设置代理、进行测试和移除代理的功能, 其事件代码如下:

```

try {
    String proxyAddress = tf_proxyAddress.getText().trim(); //代理服务器地址
    setProxyInfo(proxyAddress); //设置本地代理
    String accessAddress = tf_accessAddress.getText().trim(); //获得需要访问的网址
    useHttpAccess(accessAddress); //调用方法, 进行 HTTP 访问
    removeProxyInfo(); //清除本地代理
} catch (Exception ex) {
}

```



技术要点

在 Java 中可以通过实现 ProxySelector 类, 指定连接到 URL 引用的网络资源时, 选择要使用的代理服务器, 但是如果为本地系统设置了多个代理服务器, 则可以不实现 ProxySelector 类, 系统会使用默认的 ProxySelector 选择代理服务器。本实例就是使用默认的 ProxySelector 选择代理服务器, 方法是根据所访问的内容选择所使用的代理, 使用默认的 ProxySelector 选择所使用的代理。关键代码如下:

```
Properties properties = System.getProperties(); //获得系统属性对象
properties.setProperty("http.proxyHost", proxyAddress); //设置 HTTP 服务使用的代理服务器地址
properties.setProperty("http.proxyPort", "80"); //设置 HTTP 服务使用的代理服务器端口
properties.setProperty("https.proxyHost", proxyAddress); //设置安全 HTTP 服务使用的代理服务器地址
properties.setProperty("https.proxyPort", "443"); //设置安全 HTTP 服务使用的代理服务器端口
properties.setProperty("ftp.proxyHost", proxyAddress); //设置 FTP 访问的代理服务器主机
properties.setProperty("ftp.proxyPort", "21"); //设置 FTP 访问的代理服务器端口
properties.setProperty("socks.ProxyHost", proxyAddress); //设置 socks 代理服务器的地址
properties.setProperty("socks.ProxyPort", "1080"); //设置 socks 代理服务器的端口
```

指点迷津:

上面代码指定了 4 种类型的代理, 并使用了各服务的默认端口, 其中所使用的代理包括 HTTP 服务使用的代理、安全 HTTP 服务使用的代理、FTP 服务使用的代理以及 socks 服务使用的代理, 这样设置完代理, 程序就会根据访问的内容选择相应的代理。

实例 274 聊天室服务器端

(实例位置: 配套资源\SL\16\274)

实例说明

本实例实现了聊天室服务器端的功能, 运行程序, 服务器端等待客户端的连接, 并显示客户端的连接信息, 图 16.41 是有 3 个客户端连接到服务器, 然后有一个客户端退出的效果。

实现过程

(1) 在 Eclipse 中新建项目 274, 在项目中创建 com.mingrisoft 包。

(2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ChatServerFrame, 然后在窗体上添加一个滚动面板, 并在滚动面板上添加一个文本域控件, 完成窗体界面的设计。

(3) 在服务器窗体类 ChatServerFrame 的成员声明区定义一个 Hashtable 对象, 用于存储登录用户的用户名和套接字对象。关键代码如下:

```
//用于存储连接到服务器的用户和客户端套接字对象
private Hashtable<String, Socket> map = new Hashtable<String, Socket>();
```

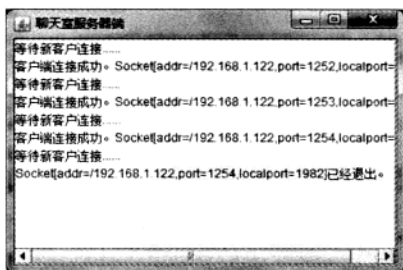


图 16.41 聊天室服务器端





(4) 在服务器窗体类 ChatServerFrame 中定义 createSocket() 方法, 用于创建服务器套接字对象、获得连接到服务器的客户端套接字对象以及启动线程对象对客户端发送的信息进行处理。关键代码如下:

```
public void createSocket() {
    try {
        server = new ServerSocket(1982);           //创建服务器套接字对象
        while (true) {
            ta_info.append("等待新客户连接.....\n");
            socket = server.accept();              //获得套接字对象
            ta_info.append("客户端连接成功。" + socket + "\n");
            new ServerThread(socket).start();      //创建并启动线程对象
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

(5) 在服务器窗体类 ChatServerFrame 中, 定义内部线程类 ServerThread, 用于对客户端的连接信息以及发送的信息进行处理和转发。关键代码如下:

```
class ServerThread extends Thread {
    //省略了部分代码
    public void run() {
        try {
            ObjectInputStream ins = new ObjectInputStream(socket.getInputStream()); //获得输入
            //流对象

            while (true) {
                //省略了部分代码
                if (v != null && v.size() > 0) {
                    for (int i = 0; i < v.size(); i++) {
                        String info = (String) v.get(i);           //读取信息
                        String key = "";
                        if (info.startsWith("用户: ")) {           //添加登录用户到客户端列表
                            //省略了部分代码
                        } else if (info.startsWith("退出: ")) {
                            //省略了部分代码
                        } else {
                            //转发接收的消息
                            key = info.substring(info.indexOf(": 发送给: ") + 5,
                                                    info.indexOf(": 的信息是: ")); //获得接收方的 key 值,
                            //即接收方的用户名

                            String sendUser = info.substring(0, info
                                                                .indexOf(": 发送给: ")); //获得发送方的 key 值, 即发
                            //送方的用户名

                            Set<String> set = map.keySet(); //获得集合中所有键的 Set 视图
                            Iterator<String> keyIt = set.iterator(); //获得所有键的迭代器
                            while (keyIt.hasNext()) {
                                String receiveKey = keyIt.next(); //获得表示接收信息的键
                                //与接收用户相同, 但不是发送用户
                                if (key.equals(receiveKey) && !sendUser.equals(receiveKey)) {
                                    Socket s = map.get(receiveKey); //获得与该键对应的套
```




接字对象

```
//创建输出流对象
PrintWriter out = new PrintWriter(s.getOutputStream(), true);
out.println("MSG:" + info);    //发送信息
out.flush();                  //刷新输出缓冲区
```

```
}
}
}
}
}
} catch (IOException e) {
    ta_info.append(socket + "已经退出。\\n");
}
}
}
```



Note

技术要点

本实例使用 `Hashtable` 类来存储连接到服务器的用户名和套接字对象，并使用 `String` 类的 `startsWith()` 方法判断客户端发送信息的类型，从而实现了向服务器端添加登录用户、发送退出信息、通过服务器转发客户端发送的信息等功能，最终完成了聊天室服务器端程序的开发。

1. `Hashtable` 类的构造方法

`Hashtable` 类实现了一个哈希表，用于存储键值映射关系，任何非 `null` 对象都可以用来作键和值使用。本实例中用到该类的构造方法的定义如下：

```
public Hashtable()
```

使用默认的初始容量 11 和加载因子 0.75 创建一个新的空哈希表。

2. `startsWith()` 方法

`String` 类的 `startsWith()` 方法用于判断当前字符串，是否以参数指定的字符串为前缀。该方法的定义如下：

```
public boolean startsWith(String prefix)
```

参数说明

prefix: 指定的前缀字符串。

返回值：如果该方法是以指定的前缀开始，则返回 `true`，否则返回 `false`。

实例 275 聊天室客户端

(实例位置：配套资源\SL\16\275)

实例说明

本实例实现了聊天室客户端，运行程序，用户登录服务器后，可以从用户列表中选择单个用户进行聊天，也可以选择多个用户进行聊天，其中选择单个用户进行聊天的效果如图 16.42 和图 16.43 所示。



Note



图 16.42 阳光**与开心**的聊天记录



图 16.43 开心**与阳光**的聊天记录

实现过程

- (1) 在 Eclipse 中新建项目 275，在项目中创建 com.mingrisoft 包。
- (2) 在 com.mingrisoft 包中创建一个继承自 JFrame 类的客户端窗体类 ChatClientFrame，用于进行用户登录、发送聊天信息和显示聊天信息，在该类中完成窗体界面的设计。
- (3) 在客户端窗体类 ChatClientFrame 中定义 createClientSocket() 方法，用于创建套接字对象、输出流对象以及启动线程对象对服务器转发的信息进行处理。关键代码如下：

```
public void createClientSocket() {
    try {
        Socket socket = new Socket("192.168.1.122", 1982);           //创建套接字对象
        out = new ObjectOutputStream(socket.getOutputStream());      //创建输出流对象
        new ClientThread(socket).start();                             //创建并启动线程对象
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

- (4) 在客户端窗体类 ChatClientFrame 中定义内部线程类 ClientThread，用于对服务器端转发的信息进行处理，并显示在相应的控件中，该类的关键代码将在技术要点中给出。

- (5) 在客户端窗体类 ChatClientFrame 中，为“登录”按钮添加实现用户登录功能的代码。“登录”按钮的事件代码如下：

```
if (loginFlag) {
    JOptionPane.showMessageDialog(null, "在同一窗口只能登录一次。");
    return;
}
String userName = tf_newUser.getText().trim();           //获得登录用户名
Vector v = new Vector();                                 //定义向量，用于存储登录用户
v.add("用户: " + userName);                             //添加登录用户
try {
    out.writeObject(v);                                   //将用户向量发送到服务器
    out.flush();                                          //刷新输出缓冲区
} catch (IOException ex) {
    ex.printStackTrace();
}
tf_newUser.setEnabled(false);                            //禁用用户文本框
loginFlag = true;                                        //将已登录标记设置为 true
```



(6) 在客户端窗体类 ChatClientFrame 中, 定义发送聊天信息的 send() 方法。关键代码如下:

```
private void send() {
    if (!loginFlag) {
        JOptionPane.showMessageDialog(null, "请先登录。");
        return; //如果用户没登录则返回
    }
    String sendUserName = tf_newUser.getText().trim(); //获得登录用户名
    String info = tf_send.getText(); //获得输入的发送信息
    if (info.equals("")) {
        return; //如果没输入信息则返回, 即不发送
    }
    Vector<String> v = new Vector<String>(); //创建向量对象, 用于存储发送的消息
    Object[] receiveUserNames = user_list.getSelectedValues(); //获得选择的用户数组
    if (receiveUserNames.length <= 0) {
        return; //如果没选择用户则返回
    }
    for (int i = 0; i < receiveUserNames.length; i++) {
        String msg = sendUserName + ": 发送给: " + (String) receiveUserNames[i]
            + ": 的信息是: " + info; //定义发送的信息
        v.add(msg); //将信息添加到向量
    }
    try {
        out.writeObject(v); //将向量写入输出流, 完成信息的发送
        out.flush(); //刷新输出缓冲区
    } catch (IOException e) {
        e.printStackTrace();
    }
    DateFormat df = DateFormat.getDateInstance(); //获得 DateFormat 实例
    String dateString = df.format(new Date()); //格式化为日期
    df = DateFormat.getTimeInstance(DateFormat.MEDIUM); //获得 DateFormat 实例
    String timeString = df.format(new Date()); //格式化为时间
    String sendUser = tf_newUser.getText().trim(); //获得发送信息的用户
    String receiveInfo = tf_send.getText().trim(); //显示发送的信息
    //在文本域中显示信息
    ta_info.append(" " + sendUser + " " + dateString + " " + timeString + "\n " + receiveInfo + "\n");
    tf_send.setText(null); //清空文本框
    ta_info.setSelectionStart(ta_info.getText().length() - 1); //设置选择的起始位置
    ta_info.setSelectionEnd(ta_info.getText().length()); //设置选择的结束位置
    tf_send.requestFocus(); //使发送信息文本框获得焦点
}
}
```



Note

技术要点

通过线程对接收到的信息进行处理, 其中分为 3 种情况, 第一种接收到的是登录用户, 第二种接收到的是退出提示, 第三种接收到的是消息。实现这 3 种功能的关键代码如下:

```
String info = in.readLine().trim(); //读取信息
if (info.startsWith("MSG:")) { //接收到的不是消息
    if (info.startsWith("退出: ")) { //接收到的是退出消息
        model.removeElement(info.substring(3)); //从用户列表中移除用户
    }
}
```



表示添加



Note

```
} else { //接收到的的是登录用户
    boolean itemFlag = false; //标记是否为列表框添加列表项, 为 true 表示不添加, 为 false
    for (int i = 0; i < model.getSize(); i++) { //对用户列表进行遍历
        if (info.equals((String) model.getElementAt(i))) { //如果用户列表中存在该用户名
            itemFlag = true; //设置为 true, 表示不添加到用户列表
            break; //结束 for 循环
        }
    }
    if (!itemFlag) {
        model.addElement(info); //将登录用户添加到用户列表
    }
} else { //如果获得的是消息, 则在文本域中显示接收到的消息
    DateFormat df = DateFormat.getDateInstance(); //获得 DateFormat 实例
    String dateString = df.format(new Date()); //格式化为日期
    df = DateFormat.getTimeInstance(DateFormat.MEDIUM); //获得 DateFormat 实例
    String timeString = df.format(new Date()); //格式化为时间
    String sendUser = info.substring(4, info.indexOf(": 发送给: ")); //获得发送信息的用户
    String receiveInfo = info.substring(info.indexOf(": 的信息是: ") + 6); //获得接收到的信息
    //在文本域中显示信息
    ta_info.append(" " + sendUser + " " + dateString + " " + timeString + "\n " + receiveInfo + "\n");
    ta_info.setSelectionStart(ta_info.getText().length() - 1); //设置选择的起始位置
    ta_info.setSelectionEnd(ta_info.getText().length()); //设置选择的结束位置
    tf_send.requestFocus(); //使发送信息文本框获得焦点
}
```

第17章

数据库操作

本章读者可以学到如下实例：

- » 实例 276 JDBC 连接 MySQL 数据库
- » 实例 277 连接 SQL Server 2005 数据库
- » 实例 278 JDBC 连接 Oracle 数据库
- » 实例 279 获取 SQL Server 指定数据库中的数据表信息
- » 实例 280 获取 MySQL 指定数据库中的数据表名称
- » 实例 281 查看数据表结构
- » 实例 282 动态维护投票数据库
- » 实例 283 SQL Server 数据备份
- » 实例 284 SQL Server 数据恢复
- » 实例 285 MySQL 数据备份
- » 实例 286 MySQL 数据恢复
- » 实例 287 动态附加数据库
- » 实例 288 生成 SQL 数据库脚本
- » 实例 289 表中字段的描述信息
- » 实例 290 将员工信息添加到数据表
- » 实例 291 添加数据时使用数据验证
- » 实例 292 插入用户登录日志信息
- » 实例 293 生成有规律的编号
- » 实例 294 生成无规律的编号
- » 实例 295 插入数据时过滤危险字符
- » 实例 296 复选框保存到数据库
- » 实例 297 把数据复制到另一张表中
- » 实例 298 批量插入数据
- » 实例 299 更新指定记录
- » 实例 300 在删除数据时给出提示信息



实例 276 JDBC 连接 MySQL 数据库

(实例位置: 配套资源\SL\17\276)



Note

实例说明

MySQL 数据库以其易于使用和管理、跨平台、开源等优点备受程序员的青睐, 在使用 MySQL 数据库时, 首先要建立与 MySQL 数据库的连接。本实例向大家介绍的是利用 JDBC 技术与 MySQL 数据库建立连接, 运行效果如图 17.1 所示。

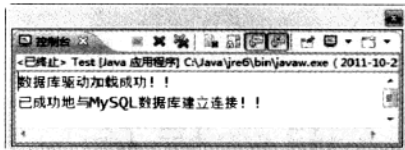


图 17.1 JDBC 连接 MySQL 数据库

实现过程

创建类 CreateMySQL, 用于建立与 MySQL 数据库的连接。在该类中定义 getConnection() 方法, 以数据库连接对象 Connection 作为返回值。关键代码如下:

```
public Connection getConnection() {
    try {
        Class.forName("com.mysql.jdbc.Driver");           //加载 MySQL 数据库驱动
        System.out.println("数据库驱动加载成功!! ");
        String url = "jdbc:mysql://localhost:3306/db_database17"; //定义连接数据库的 url
        String user = "root"; //定义连接数据库的用户名
        String passWord = "111"; //定义连接数据库的密码
        conn = DriverManager.getConnection(url, user, passWord);
        System.out.println("已成功地与 MySQL 数据库建立连接!! ");
    } catch (Exception e) {
        e.printStackTrace();
    }
    return conn;
}
```

技术要点

目前, MySQL 的 JDBC 包主要包括 Jconnector 和 org.git.mm.mysql。

1. Jconnector 包

Jconnector 包是 MySQL 官方网站公布的, 其更新速度比较快, 很多程序员都使用该包。

2. org.git.mm.mysql 包

org.git.mm.mysql 包是国外一些 Java 爱好者编写的, 出现的时间比较长, 国际化程度做得比较好, 而且对中文支持也比较好。本实例使用的就是该包。

连接 MySQL 数据库的驱动程序, 代码如下:

```
org.git.mm.mysql.Driver
```

URL 地址的代码如下:

```
jdbc:mysql://IP:PORT/databaseName?user=UserName&password=PWD&useUnicode=true
```

参数说明



- ☑ IP: 是指 MySQL 主机的 IP 地址。
- ☑ PORT: 是指 MySQL 主机的端口号, 3306 为安装 MySQL 时的默认端口号。
- ☑ useUnicode: 用于设置是否使用 Unicode 输出。

多学两招:

要使用 JDBC 技术操作数据库, 首先要向项目中添加数据库驱动, 怎样来理解数据库驱动呢? 例如有的机器安装摄像头时, 需要安装摄像头的驱动; 有的机器安装 U 盘就要装相应的 U 盘驱动, 这样用户的机器才能识别相应的设备。数据库的驱动类似于摄像头的驱动或者 U 盘的驱动, 当机器上安装了相应的数据库后, 需要安装相应的数据库驱动后机器才能识别这种数据库 (如 MySQL、SQL Server)。



Note

实例 277 连接 SQL Server 2005 数据库

(实例位置: 配套资源\SL\17\277)

实例说明

相比 SQL Server 2000 数据库, SQL Server 2005 数据库有了很大的改进, 因此获取 SQL Server 2005 的数据库的连接与获取 SQL Server 2000 数据库的连接有一些差距。本实例介绍通过 JDBC 技术与 SQL Server 2005 数据库建立连接, 运行效果如图 17.2 所示。

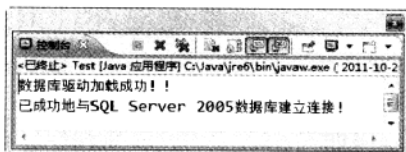


图 17.2 连接 SQL Server 2005 数据库

实现过程

在项目中创建类 CreateConn, 用于获取与 SQL Server 2005 数据库的连接, 在该类中定义连接数据库的 getConnection() 方法。关键代码如下:

```
private Connection conn; //定义 Connection 对象
public Connection getConnection(){ //定义连接数据库方法
    try {
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver"); //加载数据库驱动
        System.out.println("数据库驱动加载成功!");
        String url = "jdbc:sqlserver://localhost:1433;DatabaseName=db_database22"; //定义连接数据库 URL

        String userName = "sa";
        String passWord = "";
        conn = DriverManager.getConnection(url, userName, passWord); //获取数据库连接
        if(conn != null){
            System.out.println("已成功地与 SQL Server 2005 数据库建立连接!");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return conn;
}
```




技术要点

连接 SQL Server 2005 数据库应用的驱动程序是 sqljdbc.jar, 可以到 microsoft 的官方网站上下载, 网址为 <http://www.microsoft.com>。很多初学者会使用连接 2000 数据库的代码来连接 2005, 结果导致一些问题。连接 2000 与连接 2005 的驱动与 URL 的差别如表 17.1 所示。



Note

表 17.1 SQL Server 2000 与 SQL Server 2005 的连接信息

数 据 库	驱 动	URL
SQL Server 2000	com.microsoft.jdbc.sqlserver.SQLServerDriver	jdbc:microsoft:sqlserver://localhost:1433; DatabaseName=数据库名称
SQL Server 2005	com.microsoft.sqlserver.jdbc.SQLServerDriver	jdbc:sqlserver://localhost:1433; DatabaseName=数据库名称

多学两招:

要保证与 SQL Server 2005 数据库创建连接, 必须保证开启 TCP/IP 服务, 设置 TCP 端口为 1433, 因为 SQL 服务器默认是禁用的并且端口号没有配置, 所以要重新设置。

实例 278 JDBC 连接 Oracle 数据库

(实例位置: 配套资源\SL\17\278)

实例说明

Oracle 数据库的数据管理功能比较强大, 对计算机的要求比较高。但 Oracle 数据库在 IT 行业所占的地位比较重要, 要求程序员必须学会使用, 使用 Oracle 数据库的前提是与数据库建立连接。本实例将向大家介绍应用 JDBC 技术连接 Oracle 数据库, 实例的运行效果如图 17.3 所示。

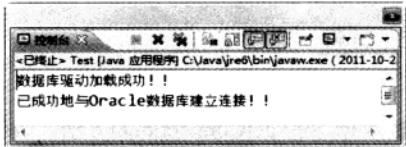


图 17.3 JDBC 连接 Oracle 数据库

实现过程

在项目中创建 CreateOracle 类, 在该类中定义连接数据库的 getConnection()方法, 该方法以 Connection 对象作为返回值。关键代码如下:

```
public Connection getConnection() {
    Connection conn = null;
    try {
        Class.forName("oracle.jdbc.driver.OracleDriver");           //加载数据库驱动
        System.out.println("数据库驱动加载成功!");                 //输出的信息
        String url = "jdbc:oracle:thin:@localhost:1521:orc17";        //获取连接 URL
        String user = "system";                                       //连接用户名
        String password = "aaa";                                     //连接密码
        Connection con = DriverManager.getConnection(url, user, password); //获取数据库连接
        if (con != null) {
            System.out.println("已成功地与 Oracle 数据库建立连接!");
        }
    }
}
```



```

    } catch (Exception e) {
        e.printStackTrace();
    }
    return conn;
}

```

//返回 Connection 实例

指点迷津:

SID 是数据库的唯一标识符, 是在建立一个数据库时系统自动赋予的一个初始 ID。



Note

技术要点

通过 JDBC 连接数据库仍然可分为两步: 加载数据库驱动和获取数据库连接。本实例连接 Oracle 数据库应用的驱动是 classes12.jar。加载数据库驱动的代码如下:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

在获取数据库连接时, 需要定义连接数据库的 URL, 连接 Oracle 与连接 SQL Server 数据库的 URL 有较大差异, 本实例获取数据库连接的 URL 地址, 代码如下:

```
String url="jdbc:oracle:thin:@localhost:1521:orc17";
```

参数说明

- ☒ @: 分隔符。
- ☒ 1521: 数据库端口。
- ☒ Orc17: 数据库名或 SID。

多学两招:

SID 和数据库名都是数据库的唯一标识符, 但是在作用上就有很大的差别。SID 主要用于一些 DBA 操作以及与操作系统交互, 从操作系统的角度访问实例名, 必须通过 Oracle SID (操作系统的环境变量), SID 在注册表中也存在。数据库名是在安装数据库、创建新的数据库、创建数据库控制文件、修改数据库结构、备份与恢复数据库时都需要使用到的。

实例 279 获取 SQL Server 指定数据库中的数据表信息

(实例位置: 配套资源\SL\17\279)

实例说明

在程序开发时, 有时需要创建大量的数据表。要获取数据库中的所有数据表信息, 可以使用 java.sql 包中的 DatabaseMetaData 接口, 该接口中提供了获取数据库综合信息的方法。本实例将实现获取 SQL Server 指定数据库中的所有数据表信息, 将其结果在控制台上输出, 实例的运行效果如图 17.4 所示。

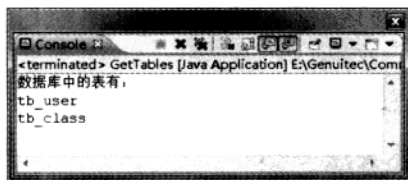


图 17.4 SQL Server 数据库中指定库中的所有表名



实现过程

(1) 创建类 GetTables, 在该类中首先定义连接数据库的 getConn() 方法, 该方法以 Connection 对象作为返回值。关键代码如下:

```
public static Connection getConn() {
    try {
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver"); //加载数据库驱动
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    String url = "jdbc:sqlserver://localhost:1433;DatabaseName=db_database17"; //连接数据库 URL
    String userName = "sa"; //连接数据库的用户名
    String passWord = ""; //连接数据库密码
    try {
        conn = DriverManager.getConnection(url, userName, passWord); //获取数据库连接
        if (conn != null) {
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return conn; //返回 Connection 对象
}
```

(2) 在该类中定义查询数据库的 GetRs() 方法, 该方法以 resultSet 作为返回值, 将数据库中的所有表获取出来。关键代码如下:

```
public static ResultSet GetRs() {
    try {
        String[] tableType = {"TABLE"}; //指定要进行查询的表类型
        Connection conn = getConn(); //调用与数据库建立连接的方法
        DatabaseMetaData databaseMetaData = conn.getMetaData(); //获取 DatabaseMetaData 实例
        ResultSet resultSet = databaseMetaData.getTables(null, null, "%", tableType); //获取数据库
        //中所有数据表集合
        return resultSet;
    } catch (SQLException e) {
        System.out.println("记录数量获取失败!");
        return null;
    }
}
```

指点迷津:

getTables() 方法获取的 ResultSet 数据库集合表由多个列组成, 其中列名称为 TABLE_NAME 的数据列, 用来存储数据库中的所有数据表集合。

技术要点

使用 getTables() 方法, 可获取指定数据库中所有数据表名, 该方法返回 ResultSet 数据结果集。该方法的声明语法如下:

```
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)
```

getTables() 方法的参数说明如表 17.2 所示。



表 17.2 getTables()方法的参数说明

参 数	类 型	描 述
catalog	String	类别名称, 必须与存储在数据库中的类别名称匹配
schemaPattern	String	模式名称, 必须与存储在数据库中的模式名称匹配
tableNamePattern	String	表名称模式, 它必须与存储在数据库中的表名称匹配
types	String[]	要包括的表类型所组成的列表



Note

多学两招:

本实例实现的是将数据库中的用户表检索出来, 在 SQL Server 数据库中还包括一些系统表。

系统表 sysobjects 用于记录在数据库内创建的每个对象 (约束、默认值、日志、规则、存储过程等)。该表中的 name 字段记录了所有对象的名称。

系统表 sysprocesses 用于保存关于运行在 Microsoft® SQL Server™ 上的进程信息。这些进程可以是客户端进程或系统进程。该表中的 smallid 字段记录了所有表的字段 ID 号, value 字段记录所有表字段的描述信息。

系统表 syscolumns 用于记录每个表和视图中的每列, 以及存储过程中的每个参数。该表位于每个数据库中。该表中的 name 字段记录了所有表的记录名。

实例 280 获取 MySQL 指定数据库中的数据表名称

(实例位置: 配套资源\SL\17\280)

实例说明

在实际开发中有时需要获取指定数据库的数据表, 本实例向大家介绍的是如何获取 MySQL 数据库中的数据表名称。运行本实例, 在控制台上显示指定数据库下的数据表, 如图 17.5 所示。

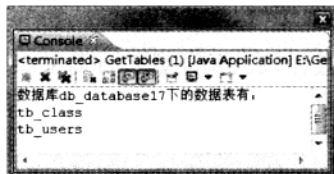


图 17.5 db_database17 数据库下所有的表

实现过程

(1) 创建类 GetTables, 在该类中定义查询数

据的方法, 首先定义连接数据库的 getConnection() 方法, 具体方法读者可参考配套资源中的源程序, 这里不再赘述。

(2) 在该类中定义查询数据库的 listDB() 方法, 用于查询数据库中的所有数据表。关键代码如下:

```
public ResultSet listDB() {
    String sql = "show tables;";           //定义查询数据 SQL 语句
    try {
        conn = getConnection();           //获取数据库连接
        Statement stmt = conn.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY); //实例化 Statement 对象
        ResultSet rs = stmt.executeQuery(sql); //执行查询 SQL 语句
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



技术要点

在 MySQL 中通过 SHOW 语句可以列出指定数据库中的数据表，其声明语法如下：

```
SHOW TABLES [FROM databaseName] [LIKE expression];
```

参数说明

- ☒ **databaseName** 子句：可选参数，用于指定要获取数据表的数据库。省略该子句，则列举当前打开数据库中的数据表，如果当前没有打开数据库，则返回错误信息。
- ☒ **expression** 子句：可选参数，用于设置列举条件。**expression** 是一个字符型表达式，可以包括通配符，如百分号（代表多个字符）、下划线（代表一个字符）等。

多学两招：

要保证与 SQL Server 2005 数据库创建连接，必须保证开启 TCP/IP 服务，设置 TCP 端口为 1433，因为 SQL 服务器默认是禁用的并且端口号没有配置，所以要重新设置。

实例 281 查看数据表结构

（实例位置：配套资源\SL\17\281）

实例说明

数据表结构是数据库维护的主要操作，查看数据表结构有助于操作者了解软件的内部体系结构。本实例将实现查看用户所选的数据表结构，运行程序，可在窗体中将用户选择的数据表结构显示在表格中，如图 17.6 所示。

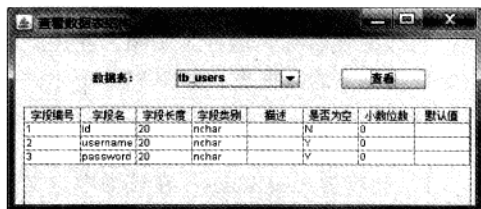


图 17.6 查看数据表结构

实现过程

- （1）创建类 **GetFrame**，用于定义查询数据库的方法。在该类中首先定义连接数据库的 **Con()** 方法，具体代码读者可参考配套资源中的源程序。
- （2）本实例实现将数据库中所有数据表显示在下拉列表中。需要在类 **GetFrame** 中创建获取数据库中所有数据表的 **GetRs()** 方法，读者可参考实例 279。
- （3）在该类中定义 **GetRs()** 方法，指定 SQL 语句。关键代码如下：

```
public ResultSet GetRs(final String SQL) {
    try {
        Connection Con = Con();
        //获取数据库连接
```




```

Statement Smt = Con
    .createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATABLE);    //获取 Statement 对象
ResultSet Rs = Smt.executeQuery(SQL);    //执行查询语句, 获取查询结果集
return Rs;
} catch (SQLException e) {
    System.out.println("记录数量获取失败!");
    return null;
}
}

```



Note

(4) 在该类中定义获取数据表结构的 getMessage()方法, 用于查询数据表结构, 以 List 作为返回值。关键代码如下:

```

public List getMessage(String tableName) {
    List list = new ArrayList();    //定义保存返回值的 List 集合
    String SQL = "SELECT syscolumns.ColId AS [字段编号],syscolumns.name AS [字段名],
        + "syscolumns.length AS [字段长度], "
        + "systypes.name AS [字段类别], "
        + "sys.extended_properties.[value] AS [描述], "
        + "CASE syscolumns.isnullable WHEN '1' THEN 'Y' ELSE 'N' END AS [是否为空], "
        + "ISNULL(COLUMNPROPERTY(syscolumns.id, syscolumns.name, 'Scale'), 0)
AS [小数位数], "
        + "syscomments.text AS [默认值], "
        + "CASE WHEN EXISTS (SELECT 1 FROM sysobjects WHERE xtype = 'PK'
AND name IN "
        + "(SELECT name FROM sysindexes WHERE indid IN "
        + "(SELECT indid "
        + "FROM sysindexkeys "
        + "WHERE id = syscolumns.id AND colid = syscolumns.colid))) "
        + "THEN '√' ELSE " END AS [主键] "
        + "FROM syscolumns "
        + "INNER JOIN systypes "
        + "ON syscolumns.xtype = systypes.xtype "
        + "LEFT JOIN sysobjects ON syscolumns.id = sysobjects.id "
        + "LEFT OUTER JOIN sys.extended_properties ON "
        + "( sys.extended_properties.minor_id = syscolumns.colid "
        + "AND sys.extended_properties.major_id = syscolumns.id) "
        + "LEFT OUTER JOIN syscomments ON syscolumns.cdefault = syscomments.id "
        + "WHERE (systypes.name <> 'sysname') "
        + "AND syscolumns.id IN (SELECT id FROM SYSOBJECTS WHERE xtype =
'U' AND NAME = '"
        + tableName + "')";    //定义查询 SQL 语句
    ResultSet res = GetRs(SQL);    //调用执行 SQL 语句方法
    ResultSetMetaData Rsmd;    //获取 ResultSetMetaData 方法
    try {
        Rsmd = res.getMetaData();    //实例化 ResultSetMetaData 对象
        while (res.next()) {    //循环遍历查询结果集
            Student student = new Student();    //创建与数据库对应的 JavaBean 对象
            student.setId(res.getString("字段编号"));    //设置对象属性

```




```

student.setName(res.getString("字段名"));
student.setType(res.getString("字段类别"));
student.setAcquiescence(res.getString("默认值"));
student.setDepict(res.getString("描述"));
student.setDigit(res.getString("小数位数"));
student.setLength(res.getString("字段长度"));
student.setIfNull(res.getString("是否为空"));
list.add(student); //将对象添加到 List 集合中
}
} catch (SQLException e) {
    e.printStackTrace();
}
return list; //返回 List 集合
}

```

技术要点

本实例主要应用 SQL Server 系统表 Sysobjects（系统对象）、Syscolumns（字段）、sys.extended_properties（描述）、Systypes（字段类别）和 Syscomments（默认值）等相对应的表关系显示出表的结构。这些系统表都可以通过对象编号和相关表编号进行关联。

实例 282 动态维护投票数据库

（实例位置：配套资源\SL\17\282）

实例说明

数据库维护只指向数据库中添加和删除数据库表中的字段，本实例以投票数据库为例，向大家介绍如何通过 Java 程序实现数据库表的维护。实例的运行效果如图 17.7 所示。

实现过程

（1）在项目中创建窗体类 BallotUtilFrame，该类继承自 JFrame 类，实现窗体类，向该窗体中添加选项卡面板、标签控件、文本框控件与按钮控件等实现传统布局。

（2）在项目中创建工具类 BallotUtil，在该类中定义删除数据表中字段与添加数据表中字段的方法，该方法有两个 String 类型的参数，分别用于指定字段名称和字段的数据类型。添加数据表字段方法的代码如下：

```

public void addField(String fieldName,String type) {
    conn = getConn(); //获取数据库连接
    try {
        Statement statement = conn.createStatement(); //获取 Statement 方法
        String sql = "alter table tb_ballot add "+fieldName+" "+type; //向数据表中添加字段
        statement.executeUpdate(sql); //执行更新数据表 SQL 语句
        conn.close(); //关闭数据库连接
    }
}

```

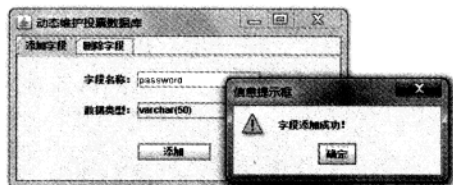


图 17.7 为表中添加一个字段



```

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

(3) 在工具类中定义删除数据表中字段的方法，该方法有一个 String 类型的参数，用于指定要删除的数据表的字段。关键代码如下：

```

public void deleteField(String fieldName) {
    conn = getConn(); //获取数据库连接
    try {
        Statement statement = conn.createStatement();
        String sql = "alter table tb_ballot drop column "+fieldName; //定义从数据库中删除字段
        statement.executeUpdate(sql); //执行删除操作
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

SQL 语句



Note

技术要点

本实例主要使用 ALTER TABLE 语句中的 ADD 和 DROP 子句。

1. ADD 子句

向数据表中添加字段。其语法格式如下：

```

ALTER TABLE table
ADD [ < column_definition > ] column_name AS computed_column_expression

```

参数说明

- ☒ column_definition: 字段的定义。
- ☒ column_name: 字段的名字。
- ☒ computed_column_expression: 计算字段的表达式。

2. DROP 子句

删除数据表中的字段。其语法格式如下：

```

ALTER TABLE table
DROP constraint_name

```

参数说明

- ☒ table: 需要删除字段的数据表名。
- ☒ constraint_name: 需要删除字段的名称。

多学两招：

ALTER TABLE 语句对具有架构绑定视图的表执行时，所受限制与当前在更改具有简单索引的表时所受的限制相同。添加列是允许的。但是，不允许删除或更改参与架构绑定视图的表中的列。如果 ALTER TABLE 语句要求更改用在架构绑定视图中的列，更改操作将失败，并且 SQL Server 将引发一条错误信息。



实例 283 SQL Server 数据备份

(实例位置: 配套资源\SL\17\283)

实例说明

对于大型的或者安全性较高的应用程序,都需要具备数据库的备份和恢复功能,这样可防止当数据库因某些意外原因被破坏或删除时,给企业的经济效益带来巨大的损失。解决该问题的唯一途径就是及时对数据库进行备份。本实例实现将用户选择的数据库进行备份,当用户单击“备份”按钮,即可生成备份文件,并保存到 C 盘。实例的运行效果如图 17.8 所示。



图 17.8 备份 db_database17 数据库

实现过程

- (1) 在项目中创建类 BackUpFrame, 该类继承自 JFrame 类, 实现窗体类。
- (2) 创建类 BackupData, 用于定义数据库备份方法, 在该类中定义连接数据库的 Con() 方法, 具体代码读者可参考配套资源中的源程序, 这里不再赘述。
- (3) 在该类中定义 getDatabase() 方法, 用于获取所有数据库方法, 返回值为 List 集合对象。关键代码如下:

```
public List getDatabase() {  
    List list = new ArrayList();           //定义 List 集合对象  
    Connection con = Con();               //获取数据库连接  
    Statement st;                         //定义 Statement 对象  
    try {  
        st = con.createStatement();       //实例化 Statement 对象  
                                           //指定查询所有数据库方法  
        ResultSet rs = st.executeQuery("select name from dbo.sysdatabases");  
        while (rs.next()) {               //循环遍历查询结果集  
            list.add(rs.getString(1));    //将查询数据添加到 List 集合中  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    return list;                          //返回查询结果  
}
```

- (4) 定义执行数据库的备份方法 getBak(), 该方法包含两个 String 类型的参数, 分别用于定义要进行备份的数据库和备份文件的保存名称。关键代码如下:

```
public void getBak(String databaseName, String databasePath) {  
    Connection con = Con();               //获取数据库连接  
    Statement st;  
    try {  
        st = con.createStatement();       //实例化 Statement 对象  
        st.executeUpdate("backup database " + databaseName + " to disk=" "  
            + databasePath + """);        //指定数据库备份 SQL 语句  
    }  
}
```



```
con.close(); //关闭连接
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

技术要点

本实例运用 SQLDMO.backup 对象完成整个系统数据库的备份,这使得在系统或数据库发生故障(如硬盘发生故障)时可以重建系统。备份整个数据库的语法如下:

```
BACKUP DATABASE { database_name | @database_name_var }
TO < backup_device > [ ,...n ]
[ WITH
    [ BLOCKSIZE = { blocksize | @blocksize_variable } ]
    [ [ , ] DESCRIPTION = { 'text' | @text_variable } ]
    [ [ , ] DIFFERENTIAL ]
    [ [ , ] EXPIREDATE = { date | @date_var }
      | RETAINDAYS = { days | @days_var } ]
    [ [ , ] PASSWORD = { password | @password_variable } ]
    [ [ , ] FORMAT | NOFORMAT ]
    [ [ , ] { INIT | NOINIT } ]
    [ [ , ] MEDIADescription = { 'text' | @text_variable } ]
    [ [ , ] MEDIANAME = { media_name | @media_name_variable } ]
    [ [ , ] MEDIAPASSWORD = { mediapassword | @mediapassword_variable } ]
    [ [ , ] NAME = { backup_set_name | @backup_set_name_var } ]
    [ [ , ] { NOSKIP | SKIP } ]
    [ [ , ] { NOREWIND | REWIND } ]
    [ [ , ] { NOUNLOAD | UNLOAD } ]
    [ [ , ] RESTART ]
    [ [ , ] STATS [= percentage] ]
]
```

备份数据库的参数说明如表 17.3 所示。

表 17.3 备份数据库的参数说明

参 数	说 明
DATABASE	指定一个完整的数据库备份。假如指定了一个文件和文件组的列表,那么仅有这些被指定的文件和文件组被备份
{ database_name @database_name_var }	指定了一个数据库,从该数据库中对事务日志、部分数据库或完整的数据库进行备份。如果作为变量 (@database_name_var) 提供,则可将该名称指定为字符串常量 (@database_name_var = database name) 或字符串数据类型 (ntext 或 text 数据类型除外) 的变量
< backup_device >	指定备份操作时要使用的逻辑或物理备份设备。可以是下列一种或多种形式。{ logical_backup_device_name } { @logical_backup_device_name_var } : 是由 sp_addumpdevice 创建的备份设备的逻辑名称,数据库将备份到该设备中,其名称必须遵守标识符规则。如果将其作为变量 (@logical_backup_device_name_var) 提供,则可将该备份设备名称指定为字符串常量 (@logical_backup_device_name_var = logical backup device name) 或字符串数据类型 (ntext 或 text 数据类型除外) 的变量



续表

参 数	说 明
n	表示可以指定多个备份设备的占位符。备份设备数目的上限为 64
BLOCKSIZE = { blocksize @blocksize_variable }	用字节数来指定物理块的大小。在 Windows NT 系统上，默认设置是设备的默认块大小。一般情况下，当 SQL Server 选择适合于设备的块大小时不需要此参数。在基于 Windows 2000 的计算机上，默认设置是 65536（64KB，是 SQL Server 支持的最大大小）
DESCRIPTION = { 'text' @text_variable }	指定描述备份集的自由格式文本。该字符串最长可以有 255 个字符
DIFFERENTIAL	指定数据库备份或文件备份应该与上一次完整备份后改变的数据库或文件部分保持一致。差异备份一般会比完整备份占用更少的空间。对于上一次完整备份时备份的全部单个日志，使用该选项可以不必再进行备份
EXPIREDATE = { date @date_var }	指定备份集到期和允许被重写的日期。如果将该日期作为变量（@date_var）提供，则可以将该日期指定为字符串常量（@date_var = date）、字符串数据类型变量（ntext 或 text 数据类型除外）、smalldatetime 或者 datetime 变量，并且该日期必须符合已配置的系统 datetime 格式
RETAIN_DAYS = { days @days_var }	指定必须经过多少天才可以重写该备份媒体集。假如用变量（@days_var）指定，该变量必须为整型
PASSWORD = { password @password_variable }	为备份集设置密码。PASSWORD 是一个字符串。如果为备份集定义了密码，必须提供这个密码才能对该备份集执行任何还原操作
FORMAT	指定应将媒体头写入用于此备份操作的所有卷。任何现有的媒体头都被重写。FORMAT 选项使整个媒体内容无效，并且忽略任何现有的内容
NOFORMAT	指定媒体头不应写入所有用于该备份操作的卷中，并且不要重写该备份设备，除非指定了 INIT
INIT	指定应重写所有备份集，但是保留媒体头。如果指定了 INIT，将重写那个设备上的所有现有的备份集数据
NOINIT	表示备份集将追加到指定的磁盘或磁带设备上，以保留现有的备份集。NOINIT 是默认设置
MEDIADESCRIPTION = { 'text' @text_variable }	指明媒体集的自由格式文本描述，最多为 255 个字符
MEDIA_NAME = { media_name @media_name_variable }	为整个备份媒体集指明媒体名，最多为 128 个字符。假如指定了 MEDIA_NAME，则它必须与以前指定的媒体名相匹配，该媒体名已存在于备份卷中。假如没有指定 MEDIA_NAME，或指定了 SKIP 选项，将不会对媒体名进行验证检查
MEDIA_PASSWORD = { mediapassword @mediapassword_variable }	为媒体集设置密码。MEDIA_PASSWORD 是一个字符串
NAME = { backup_set_name @backup_set_name_var }	指定备份集的名称。名称最长可达 128 个字符。假如没有指定 NAME，它将为空
NOSKIP	指示 BACKUP 语句在可以重写媒体上的所有备份集之前先检查它们的过期日期
SKIP	禁用备份集过期和名称检查，这些检查一般由 BACKUP 语句执行以防重写备份集



续表

参 数	说 明
NOUNLOAD	指定不在备份后从磁带驱动器中自动卸载磁带。设置始终为 NOUNLOAD, 直到指定 UNLOAD 为止。该选项只用于磁带设备
UNLOAD	指定在备份完成后自动倒带并卸载磁带。启动新用户会话时其默认设置为 UNLOAD。该设置一直保持到用户指定了 NOUNLOAD 时为止。该选项只用于磁带设备
RESTART	指定 SQL Server 重新启动一个被中断的备份操作。因为 RESTART 选项在备份操作被中断处重新启动该操作, 所以它节省了时间。若要重新启动一个特定的备份操作, 可重复整个 BACKUP 语句并且加入 RESTART 选项。不一定非要使用 RESTART 选项, 但是它可以节省时间
STATS [= percentage]	每当另一个 percentage 结束时显示一条消息, 它被用于测量进度。如果省略 percentage, SQL Server 将每完成 10 个百分点显示一条消息

多学两招:

备份数据库是指对数据库或事务日志进行复制。当系统、磁盘或数据库文件损坏时, 可以使用备份文件进行恢复, 防止数据丢失。SQL Server 数据库备份支持 4 种类型, 分别为完全备份、事务日志备份、差异备份、文件和文件组备份。完全备份将整个数据库, 包括表、索引、视图等所有数据库对象都进行备份; 事务日志备份备份时只需要复制自上次备份依赖对数据库做的改变; 差异备份只包含自完全备份依赖所改变的数据库; 文件和文件组备份指备份数据库中的一部分。

实例 284 SQL Server 数据恢复

(实例位置: 配套资源\SL\17\284)

实例说明

对于大型应用程序, 具有数据恢复功能非常重要。因为数据恢复功能可以在数据遭到破坏时将备份的数据恢复到系统中, 保证系统重新正常运转, 从而避免因数据异常丢失所带来的损失。因此建议读者在程序开发中添加数据库恢复功能。本实例实现将数据库恢复至备份时状态, 实例的运行效果如图 17.9 所示。

实现过程

(1) 在项目中创建类 ResumeFrame, 该类继承自 JFrame 类, 实现窗体类。

(2) 创建类 Resume, 用于定义数据恢复方法 getBak()。该方法包含两个 String 类型的参数, 分别用于定义要恢复的数据库和数据库的备份文件。关键代码如下:

```
public void getBak(String databaseName, String databasePath) {
    Connection con = Con(); //获取数据库连接
```



图 17.9 SQL Server 数据恢复



```
Statement st;
try {
    st = con.createStatement();           //实例化 Statement 对象
    st.executeUpdate("restore database " + dbName
        + " from disk=" + dbPath + "");   //指定数据库备份 SQL 语句
    con.close();                         //关闭连接
} catch (SQLException e) {
    e.printStackTrace();
}
```

技术要点

本实例运用 SQLDMO.Restore 对象恢复使用 BACKUP 命令所做的整个数据库备份。RESTORE 的语法声明如下:

```
RESTORE DATABASE { database_name | @database_name_var }
[ FROM < backup_device > [ ,...n ] ]
[ WITH
    [ RESTRICTED_USER ]
    [ [, ] FILE = { file_number | @file_number } ]
    [ [, ] PASSWORD = { password | @password_variable } ]
    [ [, ] MEDIANAME = { media_name | @media_name_variable } ]
    [ [, ] MEDIAPASSWORD = { mediapassword | @mediapassword_variable } ]
    [ [, ] MOVE 'logical_file_name' TO 'operating_system_file_name' ]
    [ ,...n ]
    [ [, ] KEEP_REPLICATION ]
    [ [, ] { NORECOVERY | RECOVERY | STANDBY = undo_file_name } ]
    [ [, ] { NOREWIND | REWIND } ]
    [ [, ] { NOUNLOAD | UNLOAD } ]
    [ [, ] REPLACE ]
    [ [, ] RESTART ]
    [ [, ] STATS [= percentage ] ]
]
```

RESTORE 的参数说明如表 17.4 所示。

表 17.4 RESTORE 的参数说明

参 数	说 明
DATABASE	指定备份还原整个数据库。如果指定了文件和文件组列表,则只还原那些文件和文件组
{database_name @database_name_var}	是将日志或整个数据库还原到的数据库。如果将其作为变量(@database_name_var)提供,则可将该名称指定为字符串常量(@database_name_var = database name)或字符串数据类型(ntext 或 text 数据类型除外)的变量
FROM	指定从中还原备份的备份设备。如果没有指定 FROM 子句,则不会发生备份还原,而是恢复数据库。可用省略 FROM 子句的办法尝试恢复通过 NORECOVERY 选项还原的数据库,或切换到一台备用服务器上。如果省略 FROM 子句,则必须指定 NORECOVERY、RECOVERY 或 STANDBY



续表

参 数	说 明
< backup_device >	指定还原操作要使用的逻辑或物理备份设备。可以是下列一种或多种形式
n	表示可以指定多个备份设备和逻辑备份设备的占位符。备份设备或逻辑备份设备最多可以为 64 个
RESTRICTED_USER	限制只有 db_owner、dbcreator 或 sysadmin 角色的成员才能访问新近还原的数据库
FILE = {file_number @file_number}	标识要还原的备份集。例如, file_number 为 1 表示备份媒体上的第 1 个备份集, file_number 为 2 表示第 2 个备份集
PASSWORD = { password @password_variable }	提供备份集的密码。PASSWORD 是一个字符串。如果在创建备份集时提供了密码, 则从备份集执行还原操作时必须提供密码
MEDIA_NAME = {media_name @media_name_variable}	指定媒体名称。如果提供媒体名称, 该名称必须与备份卷上的媒体名称相匹配, 否则还原操作将终止。如果 RESTORE 语句没有给出媒体名称, 将不对备份卷执行媒体名称匹配检查
MEDIA_PASSWORD = {mediapassword @mediapassword_variable}	提供媒体集的密码。MEDIAPASSWORD 是一个字符串
MOVE 'logical_file_name' TO 'operating_system_file_name'	指定应将给定的 logical_file_name 移到 operating_system_file_name。默认情况下, logical_file_name 将还原到其原始位置。如果使用 RESTORE 语句将数据库复制到相同或不同的服务器上, 则可能需要使用 MOVE 选项重新定位数据库文件以避免与现有文件冲突。可以在不同的 MOVE 语句中指定数据库内的每个逻辑文件
n	占位符, 表示可通过指定多个 MOVE 语句移动多个逻辑文件
KEEP_REPLICATION	指示还原操作在将发布的数据库还原到创建它的服务器以外的服务器上时保留复制设置。当设置复制与日志传送一同使用时, 需使用 KEEP_REPLICATION。这样, 当在备用服务器上还原数据库或日志备份并且恢复数据库时, 可防止删除复制设置。还原备份时若指定了该选项, 则不能选择 NORECOVERY 选项
NORECOVERY	指示还原操作不回滚任何未提交的事务。如果需要应用另一个事务日志, 则必须指定 NORECOVERY 或 STANDBY 选项。如果 NORECOVERY、RECOVERY 和 STANDBY 均未指定, 则默认为 RECOVERY。当还原数据库备份和多个事务日志时, 或在需要多个 RESTORE 语句时 (如在完整数据库备份后进行差异数据库备份), SQL Server 要求在除最后的 RESTORE 语句外的所有其他语句上使用 WITH NORECOVERY 选项
RECOVERY	指示还原操作回滚任何未提交的事务。在恢复进程后即可随时使用数据库。如果安排了后续 RESTORE 操作 (RESTORE LOG 或从差异数据库备份 RESTORE DATABASE), 则应改为指定 NORECOVERY 或 STANDBY
STANDBY = undo_file_name	指定撤销文件名以便可以取消恢复效果。撤销文件的大小取决于因未提交的事务所导致的撤销操作量。如果 NORECOVERY、RECOVERY 和 STANDBY 均未指定, 则默认为 RECOVERY。STANDBY 允许将数据库设定为在事务日志还原期间只能读取, 并且可用于备用服务器情形, 或用于需要在日志还原操作之间检查数据库的特殊恢复情形



Note



续表

参 数	说 明
NOREWIND	指定 SQL Server 在备份操作完成后使磁带保持打开。磁带保持打开将防止其他过程访问磁带。直到颁发 REWIND 或 UNLOAD 语句, 或直到服务器关闭时, 才释放该磁带。通过查询 master 数据库中的 sysopentapes 表可查找当前打开的一系列磁带。NOREWIND 意即 NOUNLOAD。该选项只用于磁带设备。如果对 RESTORE 使用非磁带设备, 将忽略该选项
REWIND	指定 SQL Server 将释放磁带和倒带。如果 NOREWIND 和 REWIND 均未指定, 则默认设置为 REWIND。该选项只用于磁带设备, 如果对 RESTORE 使用非磁带设备, 将忽略该选项
NOUNLOAD	指定不在 RESTORE 后从磁带机中自动卸载磁带。设置始终为 NOUNLOAD, 直到指定 UNLOAD 为止。该选项只用于磁带设备。如果对 RESTORE 使用非磁带设备, 将忽略该选项
UNLOAD	指定在还原完成后自动倒带并卸载磁带。启动新用户会话时其默认设置为 UNLOAD。设置始终为 UNLOAD, 直到指定 NOUNLOAD 为止。该选项只用于磁带设备, 如果对 RESTORE 使用非磁带设备, 将忽略该选项
REPLACE	指定即使存在另一个具有相同名称的数据库, SQL Server 也应该创建指定的数据库及其相关文件。在这种情况下将删除现有的数据库。如果没有指定 REPLACE 选项, 则将进行安全检查以防止意外重写其他数据库
RESTART	指定 SQL Server 应重新启动被中断的还原操作。RESTART 从中断点重新启动还原操作
STATS [= percentage]	每当另一个 percentage 结束时显示一条消息, 并用于测量进度。如果省略 percentage, 则 SQL Server 每完成 10 个百分比显示一条消息

多学两招:

与数据库备份对应, 数据库恢复也支持 4 种类型, 分别是还原整个数据库的完整数据库恢复、完整数据库恢复和差异数据库恢复、事务日志还原、个别文件和文件组恢复。

实例 285 MySQL 数据备份

(实例位置: 配套资源\SL\17\285)

实例说明

在实际开发中, MySQL 数据库应用得非常广泛, 因此掌握 MySQL 数据库的备份技术非常重要。备份 MySQL 数据库的方法很多, 可以通过 MySQL Tools 或者 phpMy Admin 管理工具进行备份, 也可以通过命令进行备份。本实例使用的是 MySQL DUMP 命令备份数据库, 运行本实例, 效果如图 17.10 所示。



图 17.10 在 MySQL 下备份数据库



Note

实现过程

(1) 在项目中创建类 BackFrame, 该类继承自 JFrame 类, 实现窗体类。

(2) 创建类 MySQLConn, 在该类中定义备份 MySQL 数据库的 mysqldump() 方法, 该方法包含两个 String 类型的参数, 分别用于定义要进行备份的数据库和备份数据库的文件名称。关键代码如下:

```
public boolean mysqldump(String database, String path) {           //备份数据库
    try {
        Process p = Runtime.getRuntime().exec(
            "cmd.exe /c mysqldump -uroot -p111 " + database + ">"
            + path + "");                                           //定义进行数据备份的语句
        StringBuffer out1 = new StringBuffer();                  //定义字符串缓冲对象
        byte[] b = new byte[1024];                               //定义字节数组
        for (int i; ((i = p.getInputStream().read(b)) != -1);) {   //将数据写入到指定文件中
            out1.append(new String(b, 0, i));                     //向流中追加数据
        }
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

脚下留神:

在 Java 类中执行命令时, 必须要在命令前加上 “cmd.exe /c”, 然后再接命令 “mysqldump -uroot -p111 "+database+" ">"+path+"”。

技术要点

本实例主要应用 java.lang 软件包中的 Runtime、Process 和 StringBuffer 类。首先通过 getRuntime() 方法获取与当前 Java 应用程序相关的 Runtime 对象, 然后应用 exec() 方法, 执行 MySQL DUMP 命令。接着应用 Process 类中的 getInputStream() 方法获取子进程的输入流, 最后应用 StringBuffer 类中的 append() 方法将流中的数据追加到指定的字符序列中, 完成数据的备份。

(1) getRuntime() 方法的声明语法如下:

```
public static Runtime getRuntime();
```

返回值: 返回与当前 Java 应用程序相关的 Runtime 对象。Runtime 类的大多数方法是实例方法, 并且必须根据当前的运行时对象对其进行调用。



(2) exec()方法的声明语法如下:

```
public Process exec(String command);
```

参数说明:

command: 一条指定的系统命令。

(3) getInputStream()方法的声明语法如下:

```
public abstract InputStream getInputStream();
```

返回值: 获取子进程的输入流。输入流获得由该 Process 对象表示的进程的标准输出流。

返回连接到子进程正常输出的输入流。

(4) append()方法的声明语法如下:

```
public StringBuffer append(String str);
```

参数说明

str: 一个指定的字符串。



Note

多学两招:

在指定备份文件的存储位置和名称时,必须使用“D:\\Data\\db_database04.txt”这样的格式,即应该使用“\\”,不是单“\”。

实例 286 MySQL 数据恢复

(实例位置: 配套资源\\SL\\17\\286)

实例说明

对于大型应用系统来说,具有数据恢复功能非常重要。数据恢复可以在数据遭到破坏时将备份的数据恢复到数据库系统中,保证系统的正常运行,避免数据丢失带来的损失。本实例介绍 MySQL 数据恢复的方法,与 MySQL 数据备份的方法类似,都是使用命令实现。实例的运行效果如图 17.11 所示。



图 17.11 恢复 MySQL 数据库下的数据

实现过程

(1) 在项目中创建类 BackFrame, 该类继承自 JFrame 类, 实现窗体类。

(2) 创建类 ResumeUtil, 在该类中定义恢复数据库的 mysqlresume()方法, 该方法包含两个 String 类型的参数, 分别用于定义要恢复的数据库和备份数据文件的保存地址。关键代码如下:

```
public boolean mysqlresume(String database, String path) {           //恢复数据库
    try {
        Process p = Runtime.getRuntime().exec(
            "cmd.exe /c mysql -uroot -p111 " + database + " <" + path
            + "");                                                     //执行恢复语句
        StringBuffer out1 = new StringBuffer();                     //定义字符串缓冲对象
        byte[] b = new byte[1024];                                   //定义字节数组
        for (int i; ((i = p.getInputStream().read(b)) != -1);) {      //将数据写入到指定文件中
```



```

        out1.append(new String(b, 0, i));           //向流中追加数据
    }
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}

```



Note

技术要点

MySQL 数据库的恢复使用的是 MySQL 命令，其语法格式如下：

```
mysql -uUser -pPass DataBase <Path
```

MySQL 命令的参数说明如表 17.5 所示。

表 17.5 MySQL 命令的参数说明

参 数	说 明
User	数据库用户名
Pass	数据库密码
DataBase	备份的数据库
Path	备份文件存储的位置

多学两招：

mysqldump 是 MySQL 自带的一个非常方便的小工具，存在于 MySQL 安装目录的 bin 下。不是 MySQL 命令不能在 MySQL 中执行。在命令标识符中输入“mysqldump - help”命令可以测试 mysqldump 所支持的选项。

实例 287 动态附加数据库

（实例位置：配套资源\SL\17\287）

实例说明

在项目开发完成时，需要为用户提供安装程序，即使最简单的安装也需要提供数据库导入功能。本实例实现的是通过 Java 应用程序，将 SQL Server 2000 数据库文件附加到本地 SQL Server 服务器中。运行本实例，用户在“数据库名称”文本框中输入附加数据库后的数据库名，并为用户提供了可浏览本地数据库文件与数据库日志文件。实例的运行效果如图 17.12 所示。



图 17.12 附加数据库

实现过程

（1）在项目中创建类 SubjoinFrame，该类继承自 JFrame 类，实现窗体类。



(2) 创建类 SubjoinDate, 在该类中定义附加数据库的 executeUpdate() 方法, 该方法包含 3 个 String 类型的参数, 分别用于定义附加数据库的名称、数据库文件地址与数据库日志文件地址。关键代码如下:

```
public boolean executeUpdate(String dataName,String mPath,String lPath) {
    if (con == null) {
        Connection(); //数据库连接
    }
    try {
        stmt = con.createStatement();
        int iCount = stmt.executeUpdate(
            "EXEC sp_attach_db @dbname='"+dataName+"', @filename1='"+
            mPath+"', @filename2='"+lPath+"'"); //执行数据库附加
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        return false;
    }
    closeConnection(); //调用关闭数据库连接方法
    return true;
}
```

技术要点

本实例主要应用 SQL Server 中的系统存储过程 sp_attach_db 完成。

sp_attach_db 用于将数据库附加到 SQL Server 服务器中, 其语法声明如下:

```
sp_attach_db [ @dbname = ] 'dbname', [ @filename1 = ] 'filename_n' [ ,...16 ]
```

参数说明

- ☒ [@dbname =] 'dbname': 要附加到服务器的数据库的名称。该名称必须是唯一的。dbname 的数据类型为 sysname, 默认值为 null。
- ☒ [@filename1 =] 'filename_n': 数据库文件的物理名称, 包括路径。filename_n 的数据类型为 nvarchar(260), 默认值为 null, 最多可以指定 16 个文件名。参数名称以 @filename1 开始, 递增到 @filename16。文件名列表必须包括主文件, 因为主文件包含指向数据库中其他文件的系统表。该列表还必须包括数据库分离后所有被移动的文件。

多学两招:

本实例在执行数据库附加时, 需要将指定的数据库文件添加到文本框中, 此时要注意限制用户选择文件的文件格式, 如果用户选择的不是数据库文件和数据库日志文件, 系统就会出现异常, 因此为了保证系统的完整性, 在开发本实例时, 要注意对用户选择的文件进行限制。

实例 288 生成 SQL 数据库脚本

(实例位置: 配套资源\SL\17\288)

实例说明

SQL 脚本包含用于创建数据库及其对象的语句描述, 可以从现有数据库中的对象生成脚



本, 然后通过运行该数据库的脚本将这些对象添加到其他数据库。实际上, 这样做是重新创建了整个数据库结构, 以及所有的单个数据库对象。

实现过程

下面以 db_database17 数据库为例介绍生成 SQL 数据库脚本的步骤。

(1) 打开企业管理器, 展开“数据库”节点, 选择生成脚本, 如图 17.13 所示。

(2) 进入欢迎界面后单击“下一步”按钮, 选择所要生成脚本的数据库并单击“下一步”按钮, 如图 17.14 所示。



Note

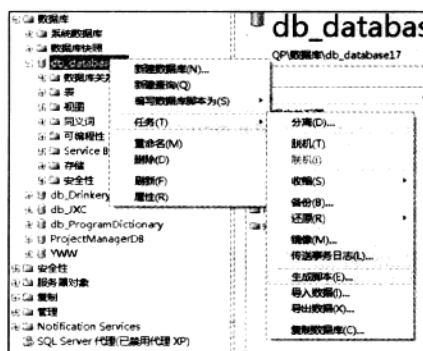


图 17.13 选择生成脚本



图 17.14 选择数据库

(3) 进入选择脚本选项后单击“下一步”按钮。在弹出的界面中选择对象类型, 这里选中“表”复选框后单击“下一步”按钮, 如图 17.15 所示。



图 17.15 选择对象类型

(4) 在弹出的界面中选择所要编写脚本的表后单击“下一步”按钮, 如图 17.16 所示。



Note



图 17.16 选择要编写脚本的表

(5) 在弹出的界面中选择所编写脚本的保存路径后单击“下一步”按钮，如图 17.17 所示。

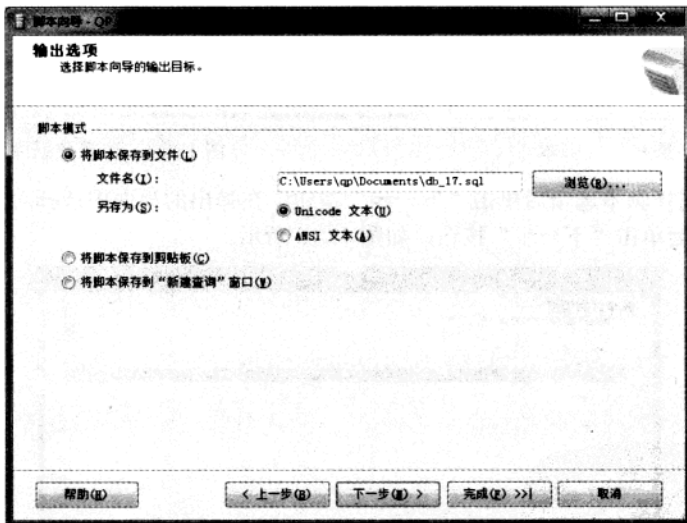


图 17.17 选择要保存的路径

(6) 最后单击“完成”按钮，完成生成 SQL 数据库脚本。

技术要点

SQL 脚本生成的架构可以有許多用途，包括以下方面：

- ☑ 维护备份脚本，该脚本将允许用户重新创建所有用户、组、登录和权限。
- ☑ 创建和更新数据库开发代码。
- ☑ 从现有的架构创建一个测试或开发环境。



实例 289 表中字段的描述信息

(实例位置: 配套资源\SL\17\289)

实例说明

在程序开发时,有时需要创建大量的数据表。要获取数据库中的所有数据表信息,可以使用 java.sql 包中的 DatabaseMetaData 接口,该接口中提供了获取数据库综合信息的方法。本实例实现获取 SQL Server 指定数据库中的所有数据表字段信息,将其结果在控制台上输出。实例的运行效果如图 17.18 所示。

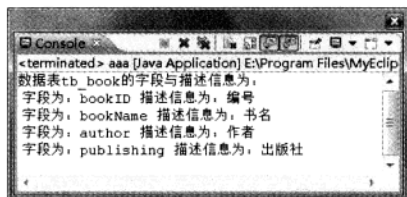


图 17.18 tb_book 表的字段描述信息

实现过程

(1) 创建类 GetDescribe, 在该类中定义获取数据表字段的描述信息方法, 在该类中定义获取数据库的连接方法 getConn(), 具体代码读者可参考配套资源中的源程序, 这里不再赘述。

(2) 在该类中定义 getDescribe()方法, 该方法包含一个 String 类型的参数, 用于指定要查询的数据表, 并以 List 对象作为返回值。关键代码如下:

```
public List getDescribe(String tableName) {
    conn = getConn();                //获取数据库连接
    List list = new ArrayList();      //定义 List 集合对象
    try {
        Statement stmt = conn.createStatement(); //获取 Statement 对象
        ResultSet rest = stmt.executeQuery("select c.name,b.value FROM sysobjects a,sysproperties
b,syscolumns " + "c where a.name='" + tableName + "' and a.id=b.id and b.id=c.id and b.smallid=c.colorder"); //执行查
        询语句
        while(rest.next()){           //循环遍历查询结果集
            Describe describe = new Describe(); //定义的 JavaBean 对象
            describe.setName(rest.getString(1)); //设置对象属性
            describe.setValue(rest.getString(2));
            list.add(describe);        //向集合中添加对象
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return list;
}
```

技术要点

获取 SQL Server 2000 数据库中数据表字段的描述信息,主要用到了数据库的系统表 sysobjects、sysproperties 和 syscolumns。

系统表 sysobjects 用于记录在数据库内创建的每个对象(约束、默认值、日志、规则、存储过程等)。该表中的 name 字段记录了所有对象的名称。





系统表 sysprocesses 用于保存关于运行在 Microsoft® SQL Server™ 上的进程信息。这些进程可以是客户端进程或系统进程。该表中的 smallid 字段记录了所有表的字段 ID 号, value 字段记录所有表字段的描述信息。

系统表 syscolumns 用于记录每个表和视图中的每列, 以及存储过程中的每个参数。该表位于每个数据库中。该表中的 name 字段记录了所有表的记录名。

多学两招:

编写代码后, 可能由于没有注意格式化问题, 使代码显得比较凌乱。如果使用 Eclipse 开发工具, 可以通过快捷键 Shift+Ctrl+F 来对 Eclipse 中的代码进行格式化。

实例 290 将员工信息添加到数据表

(实例位置: 配套资源\SL\17\290)

实例说明

数据录入对每个应用程序来说都是必不可少的。本实例实现的是单条数据的录入。在实现数据录入时, 可以对录入进行验证来保证程序的完整性。本实例实现当用户填写完以“*”标注的信息后, 单击“添加”按钮, 系统会将用户添加的信息写入到数据库中, 实例的运行效果如图 17.19 所示。

实现过程

(1) 在项目中创建类 InsertEmpFrame, 该类继承自 JFrame 类, 实现窗体类。

(2) 创建类 JdbcUtil, 在该类中定义 insertEmp() 方法, 用于实现添加数据, 该方法包含一个与员工表 tb_emp 对应的 JavaBean 类, Emp 为参数。关键代码如下:

```
public void insertEmp(Emp emp){
    conn = getConn();                                //获取数据库连接
    try {
        PreparedStatement statement = conn.prepareStatement("insert into tb_emp values(?,?,?,?,?,?)");
                                                                    //预处理语句
                                                                    //设置预处理语句的参数值
        statement.setString(1, emp.getName());
        statement.setString(2, emp.getSex());
        statement.setInt(3, emp.getAge());
        statement.setString(4, emp.getDept());
        statement.setString(5, emp.getPhone());
        statement.setString(6, emp.getRemark());
        statement.executeUpdate();                            //执行预处理语句
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



图 17.19 添加员工信息



技术要点

本实例在实现数据录入时,使用的是 insert 插入语句。insert 语句的语法格式如下:

```
INSERT INTO 表名[(字段名 1,字段名 2...)]
VALUES(属性值 1,属性值 2, ...)
```

例如,向数据表 tb_emp (包含字段 id,name,sex,department) 中插入数据,代码如下:

```
insert into tb_emp values(2,'lili','女','销售部');
```



Note

多学两招:

本实例在执行插入数据时,使用了预处理语句。与普通 SQL 语句相比,预处理语句有两个优点,一是安全,二是可以提高访问数据库的速度。因此提倡使用预处理语句来操作数据库。

实例 291 添加数据时使用数据验证

(实例位置: 配套资源\SL\17\291)

实例说明

在将数据添加到数据库时,为了确保数据的完整性,在添加数据时,实现数据验证是很常见的功能。本实例实现的是在添加员工信息时,使用数据验证,可以验证要添加的员工信息是否存在数据库中,如果不存在允许添加,如果存在不允许执行添加操作。在“年龄”一栏中也添加了数据验证,只允许用户输入数字。实例的运行效果如图 17.20 所示。



图 17.20 数据验证

实现过程

(1) 创建类 JdbcUtil, 在该类中定义 insertEmp() 方法, 用于实现添加数据, 该方法包含一个与员工表 tb_emp 对应的 JavaBean 类, Emp 为参数。代码同实例 290。

(2) 当用户输入员工姓名后, 单击“验证”按钮后, 系统会调用 JdbcUtil 类的 selectEmpUseName() 方法检验用户输入的员工是否在员工表中存在。selectEmpUseName() 方法的具体代码如下:

```
public int selectEmpUseName(String name){
    conn = getConn();                //获取数据库连接
    Statement statment;
    int id = 0;                       //定义保存返回值的 int 对象
    try {
        statment = conn.createStatement();    //获取 Statement 对象
        String sql = "select id from tb_emp where name = '"+name+"'"; //定义查询 SQL 语句
        ResultSet rest = statment.executeQuery(sql); //执行查询语句获取查询结果集
        while(rest.next()){              //循环遍历查询结果集
            id = rest.getInt(1);          //获取查询结果
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return id;
}
```




Note

```

    }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return id;
}
//返回查询结果

```

技术要点

本实例是通过用户输入的员工姓名来查询员工表,如果员工表中包含相应的信息,则给出相应的提示。在限制年龄文本框只能输入数字时,通常为文本框添加键盘事件。

多学两招:

电话号码包含固定电话和手机号码。很多程序验证用户输入的电话号码是否合法,都是使用正则表达式。正则表达式可以很好地保证验证的完整性。使用正则表达式验证电话号码可以使用这样的表达式“0\d{2,3}[-]?\\d{7,8}|0\d{2,3}\\s?\\d{7,8}|13[0-9]\\d{8}|15[1089]\\d{8}”。

实例 292 插入用户登录日志信息

(实例位置: 配套资源\SL\17\292)

实例说明

几乎所有的应用程序都提供了一个登录窗体,该窗体用于验证用户是否有权限登录此程序。在用户登录的同时也可以将用户的登录信息记录到数据库中。本实例实现将用户登录的用户名、密码、当前登录时间写入到数据库中。实例的运行效果如图 17.21 所示。

实现过程

(1) 在项目中创建类 InsertInfoFrame, 该类继承自 JFrame 类, 实现窗体类。

(2) 在项目中定义类 InsertInfo, 在该类中定义将用户登录信息插入到数据库的 insertUser() 方法。关键代码如下:

```

public void insertUser(User user) {
    conn = getConn(); //获取数据库连接
    try {
        //定义插入数据库的预处理语句
        PreparedStatement statement = conn
            .prepareStatement("insert into tb_user values(?,?,?)");
        statement.setString(1, user.getUserName()); //设置预处理语句参数
        statement.setString(2, user.getPassWord());
        //根据指定格式定义 SimpleDateFormat 对象
        SimpleDateFormat date_time = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String datetime = date_time.format(new Date()); //对当前日期进行格式化
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```



图 17.21 用户登录



```
statement.setString(3, datetime);
statement.executeUpdate();           //执行预处理语句
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

技术要点

本实例实现将用户登录的时间写入到数据库中,并对日期进行了格式化,SimpleDateFormat 类的 format()方法可实现将日期进行格式化。

SimpleDateFormat 类是一个与语言环境有关的方式来格式化和解析日期的具体类。该类的常用构造方法介绍如下。

(1) 用默认的模式和默认语言环境的日期格式符号构造 SimpleDateFormat 类。其声明语法如下:

```
SimpleDateFormat()
```

(2) 用给定的模式和默认语言环境的日期格式符号构造 SimpleDateFormat 类。其声明语法如下:

```
SimpleDateFormat(String pattern);
```

参数说明

pattern: 描述日期和时间格式的模式。

该类的 format()方法可将给定的 Date 格式化为日期/时间字符串。以 String 返回格式化后的字符串。其声明语法如下:

```
format(Date date);
```

参数说明

date: 要格式化为时间字符串的时间值。

多学两招:

SimpleDateFormat 类可以选择任何用户定义的日期-时间格式的模式,日期-时间格式由日期和时间模式字符串指定。在日期和时间模式字符串中,未加引号的字母'A'到'Z'和'a'到'z'被解释为模式字母,用来表示日期或时间字符串元素。文本可以使用单引号(')引起来,以免进行解释。""表示单引号。所有其他字符均不解释;只是在格式化时将它们简单复制到输出字符串,或者在解析时与输入字符串进行匹配。

实例 293 生成有规律的编号

(实例位置: 配套资源\SL\17\293)

实例说明

在程序开发中,经常会遇到这样的情况,数据表中的主键是唯一的,如果要求用户手工输入是很不方便的,也容易产生错误,解决这个问题的最好方法就是让系统自动生成唯一的 ID 号。生成 ID 号可以有多种形式,本实例为创建一个商品信息添加窗体,在该窗体中可以添加商品信息。当用户单击“查看”按钮时,系统会给出所有商品信息,如图 17.22 所示。



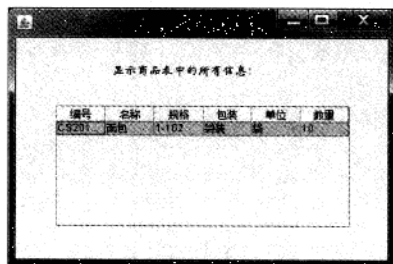


图 17.22 生成有规律的编号

实现过程

(1) 在项目中创建工具类 WareUtil, 在该类中定义向商品中添加数据的 insertWare() 方法, 该方法以与数据库对应的 JavaBean 为参数。关键代码如下:

```
public void insertWare(Ware ware) {
    conn = getConn(); //获取数据库连接
    try {
        //定义插入数据库的预处理语句
        PreparedStatement statement = conn.prepareStatement("insert into tb_ware values(?,?,?,?,?,?,?)");
        //设置预处理语句的参数值
        statement.setString(1, ware.getSID());
        statement.setString(2, ware.getName());
        statement.setString(3, ware.getSpec());
        statement.setString(4, ware.getCasing());
        statement.setString(5, ware.getUnit());
        statement.setString(6, ware.getDate());
        statement.setInt(7, ware.getAmount());
        statement.executeUpdate(); //执行预处理语句
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

(2) 当用户单击商品添加表中的“添加”按钮后, 系统会将用户添加的信息保存在数据库中。“添加”按钮的单击事件中的代码如下:

```
protected void do_insertButton_actionPerformed(ActionEvent arg0) {
    String name = nameTextField.getText(); //获取用户添加的商品名称
    ..... //省略了获取其他商品信息代码
    int count = Integer.parseInt(amountTextField.getText()); //获取用户添加的商品数量
    int ID = 0;
    String sDate = WareUtil.getDateTime(); //调用获取系统时间方法
    List list = util.selectWare(); //获取商品表中全部的商品
    String sid = "";
    for(int i = 0; i < list.size(); i++) { //循环遍历查询结果集
        Ware ware = (Ware)list.get(i); //获取商品
        sid = ware.getSID(); //获取商品编号
    }
    if(list.size() == 0) { //如果商品集合中为空
        sid = "CS"+sDate.replace("-", "")+"00001"; //定义商品编号
    }
    else { //如果商品集合不为空
```



```

sid = sid.trim();
ID = Integer.parseInt(sid.substring(sid.length()-5));    //截取商品编号中的后 5 位
sid = sid.substring(0, sid.length()-5) + String.format("%05d", ID + 1)//商品编号
}
Ware ware = new Ware();                                //定义与商品表对应的 JavaBean 对象
ware.setSID(sid);                                       //设置 JavaBean 编号
.....                                                  //省略了设置其他属性代码
util.insertWare(ware);                                  //添加商品信息
JOptionPane.showMessageDialog(getContentPane(), "数据添加成功!",
    "信息提示框", JOptionPane.CANCEL_OPTION);
}

```



Note

技术要点

商品编号由字母“CG”、系统日期和 5 位数字组成。首先判断采购信息表中的采购编号是否为空，如果为空，则采购编号等于字母“CG”+系统日期+“00001”；如果不为空，则先查找数据表中最大的采购编号，此时采购编号等于字母“CG”+系统日期+5 位数字编码加 1。在实现编码加 1 时，数字前面的 0 被忽略，实现数字前加 0 的格式，可以使用字符串的 format() 方法。

format() 方法用于输出指定格式的字符串，其语法格式如下：

```
String.format("%05d", ID + 1)
```

参数说明

- ☒ ID: 是数字格式的变量，将其加 1 实现编号增值。
- ☒ "%05d": 设定数字的格式的位数是 5 位，如果不足 5 位前面补 0。

多学两招:

本实例中生成的用户编号是根据系统时间确定的，本实例中使用了 SimpleDateFormat 类获取系统时间，该类是一个以与语言环境有关的方式来格式化和解析日期的具体类。在创建该类对象时，可给定时间日期的格式。

实例 294 生成无规律的编号

(实例位置: 配套资源\SL\17\294)

实例说明

在实例 293 中为读者介绍了生成有规律的编号的实例，当然也可以生成无规律的编号，这种无规则的编号，不仅可以用在一张表中作主键，也可以用在其他方面。实例的运行效果如图 17.23 所示。

实现过程

(1) 在项目中创建类添加部门信息的窗体类 InsertDeptFrame，向该类中添加标签、文本框与按钮控件，实现窗体布局，并创建显示部

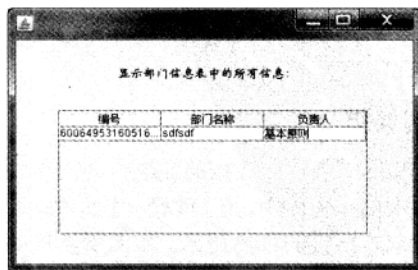


图 17.23 生成无规律的编号



门信息的 `SelectDeptFrame` 类, 向该类中添加标签与表格控件, 实现窗体布局。

(2) 在项目中定义工具类 `DeptUtil`, 在该类中操作部门信息表方法, 该类中的方法不是本实例重点, 用户可参考配套资源中的源程序, 这里不再赘述。

(3) 当用户单击添加部门信息窗体中的“添加”按钮时, 系统会将用户添加的信息保存到数据库中。“添加”按钮的单击事件中的代码如下:

```
protected void do_insertButton_actionPerformed(ActionEvent arg0) {
    String name = deptNameTextField.getText();           //获取用户添加的部门名称
    String person = personTextField.getText();           //获取用户添加的部门负责人
    Random ran = new Random(System.currentTimeMillis()); //根据当前时间的毫秒数创建随机数流
    StringBuilder idb = new StringBuilder();             //创建字符串缓冲对象
    idb.append(String.format("%019d", Math.abs(ran.nextLong())) + String.format("%03d", (int)(Math.random() * 100 % 9) + 1));
    idb = idb.reverse();                                  //对字符串缓冲对象取反
    String id = idb + "" + String.format("%010d", Math.abs(ran.nextInt()));
    Dept dept = new Dept();                               //创建与数据表对应的 JavaBean 对象
    dept.setDid(id);                                     //设置对象属性
    dept.setdName(name);
    dept.setPriName(person);
    DeptUtil deptUtil = new DeptUtil();                  //创建工具类对象
    deptUtil.insertDept(dept);                           //调用添加方法
    JOptionPane.showMessageDialog(getContentPane(),
        "数据添加成功!", "信息提示框", JOptionPane.WARNING_MESSAGE);
}
```

技术要点

用户编号由一个 3 位的整型随机数加上 10 位的整型随机数和一个 19 位的长整型随机数的字符串连接而成, 总长度为 32 位。由于使用该方法可能出现编号重复的情况, 所以实例中对插入编号的唯一性进行了判断, 如果编号重复将重新生成。

多学两招:

生成无规律编号不仅可以作为一张表的主键, 还可以用在其他的程序中作为唯一的标识。例如, 在上传文件中, 为了避免上传文件的名称冲突, 可以使用这样无规律的编号形式。

实例 295 插入数据时过滤危险字符

(实例位置: 配套资源\SL\17\295)

实例说明

程序开发中, 数据的完整性与安全性是必须要考虑的问题, 而对危险字符的过滤不仅可以应用在应用程序中, 还可以应用在 Web 程序中。本实例以在插入数据时过滤掉危险字符为例介绍对危险字符的过滤技术, 本实例实现的是向图书销售表中插入数据, 插入时将图书名称中的危险字符过滤掉。实例的运行效果如图 17.24 所示。



图 17.24 过滤危险字符

实现过程

(1) 在项目中创建窗体类 InsertBookFrame, 该类继承自 JFrame 类, 实现窗体类, 向该窗体中添加标签、文本框与按钮控件, 实现窗体布局。

(2) 创建字符串处理类 DoString, 在该类中定义过滤掉危险字符的方法。关键代码如下:

```
public void dostring(){
    this.checkstr=this.getstr;
    this.checkstr=this.checkstr.replaceAll("&", "&");           //替换字符处理
    this.checkstr=this.checkstr.replaceAll(";", "");
    this.checkstr=this.checkstr.replaceAll("'", "");
    this.checkstr=this.checkstr.replaceAll("<", "<");
    this.checkstr=this.checkstr.replaceAll(">", ">");
    this.checkstr=this.checkstr.replaceAll("--", "");
    this.checkstr=this.checkstr.replaceAll("\\""", "&quot;");
    this.checkstr=this.checkstr.replaceAll("/", "");
    this.checkstr=this.checkstr.replaceAll("%", "");
}
```

技术要点

本实例主要应用了 String 类中的 replaceAll()方法, 其声明语法如下:

```
replaceAll(String source,String replace)
```

参数说明

- ☒ source: 要替换掉的字符串。
- ☒ replace: 用来替代的字符串。

多学两招:

过滤敏感词相信大家都很熟悉, 如果用户输入了敏感词汇, 系统会将这些词汇自动地过滤掉, 防止其他用户看到, 使用本实例中提供的方法, 也可以实现过滤敏感词。

实例 296 复选框保存到数据库

(实例位置: 配套资源\SL\17\296)

实例说明

在用户注册模块中, 要求用户添加个人爱好是十分常见的, 通常情况下, 系统会提供给用





户以复选框的形式来选择。本实例实现用户选择的爱好信息,以字符串的形式保存到数据库中。实例的运行效果如图 17.25 所示。

实现过程

(1) 在项目中创建窗体类 ToStringFrame(), 该类继承自 JFrame 类, 实现窗体类, 在该窗体中添加标签、文本框、复选框与按钮控件, 实现窗体布局。

(2) 在各个复选框控件中添加 ItemEvent, 当用户选择某项爱好时, 会将字符串缓冲区对象做追加处理。关键代码如下:

```
protected void do_checkBox_itemStateChanged(ItemEvent arg0) {
    buff.append(checkBox.getText()+"、");
}
```

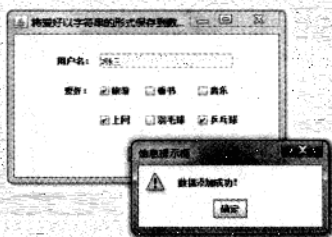


图 17.25 添加员工爱好信息

技术要点

本实例中使用了字符串缓冲区对象 StringBuffer, 当用户选择了一项爱好时, 系统会通过 append() 方法追加内容。在向数据库中添加数据时, 只需要调用该类的 toString() 方法即可。

多学两招:

本实例中使用了“、”分隔符, 对爱好信息进行分隔, 这样方便用户从数据库中读取爱好信息, 本实例中使用了顿号, 当然也可以使用其他的分隔符。

实例 297 把数据复制到另一张表中

(实例位置: 配套资源\SL\17\297)

实例说明

将数据从一张表中复制到另一张表中的情况很多, 例如, 有两张表分别为学生表与优秀学生表, 这两张表具有相同的数据结构, 要求将推举出来的学生信息添加到优秀学生表中。本实例实现了这一功能, 实例的运行效果如图 17.26 所示。

实现过程

(1) 在项目中创建类 CopyFrame, 该类继承自 JFrame 类, 实现窗体类。向该窗体中添加标签、下拉列表、按钮与表格控件, 实现窗体布局。表格控件中显示学习表中信息, 用于可根据该信息选择将哪名学生添加到优秀学生表。

(2) 在项目中创建工具类 CopyDate, 该类用于定义学生表中数据添加到优秀学生表中的 insertStu() 方法, 该方法包含一个 int 类型的参数, 用于定义学生的编号。关键代码如下:

```
public void insertStu(int id) {
    conn = getConn();
```

//获取与数据库的连接



图 17.26 把数据复制到另一张表中



```

try {
    Statement statement = conn.createStatement();           //获取 Statement 对象
    String sql = "insert into tb_excellenceStu select name,sex,specialty,grade from tb_stu where
id = "+id;           //定义插入数据的 SQL 语句
    statement.executeUpdate(sql);           //执行插入语句
} catch (Exception e) {
    e.printStackTrace();
}
}

```



Note

技术要点

本实例使用的是 INSERT 与 SELECT 语句的组合使用,通常以下情况会使用这种组合形式。

(1) 创建查询表

创建查询表主要可以提高检索速度,因为查询时对多个表进行链接操作比较复杂,比简单查询速度要慢,可以创建包含多个表中数据的查找表,这样查询时仅对查找表进行查询,可以加快查询的速度。

(2) 修改表

在使用数据库的过程中,可以发现原先的表有不合理的地方,需要对表进行修改。对于简单的修改使用 alter table 语句,而对于改动较大的表,可以创建一个包含所需列的表,然后结合 INSERT SELECT 语句,将原有表中的数据转换为新表中的数据,这样就不会造成原有数据的丢失。

多学两招:

使用 SELECT 语句选择数据时,不能从被插入数据的表中选择行。

在指定插入的表后所包含的字段数目必须与 SELECT 语句中返回的字段数目相同。

指定插入的表后所包含的字段数据类型必须与 SELECT 语句中返回的字段数据类型对应相同或系统可以自动转换。

实例 298 批量插入数据

(实例位置: 配套资源\SL\17\298)

实例说明

批量向数据表中插入数据是很常见的功能,实现这一功能可以使用多种形式, JDBC 中就有实现批处理的类,本实例向大家介绍的是使用 UNION ALL 语句实现一次向学生表中插入 3 条记录。实例的运行效果如图 17.27 所示。



图 17.27 插入数据成功提示



实现过程

(1) 在项目中创建类 BatchInsert, 在该类中首先定义数据库连接方法 getConn(), 该方法以 Connection 对象作为返回值, 具体代码读者可参考配套资源中的源程序, 这里不再赘述。

(2) 在该类中定义 insertStu() 方法, 该方法包含一个 String 类型的参数, 用于指定要执行的 SQL 语句。关键代码如下:

```
public void insertStu(String sql){
    conn = getConn();                //获取数据库连接
    try {
        Statement statement = conn.createStatement(); //创建 Statement 对象
        statement.executeUpdate(sql);                //执行插入 SQL 语句
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

(3) 在该类的主方法中调用 insertStu() 方法, 实现向学生表中添加数据。关键代码如下:

```
public static void main(String[] args) {
    BatchInsert insert = new BatchInsert(); //创建本类对象
    String sql = "insert tb_stu select '双双','女','生物科学','08d02' " +
        "union all select '王爽','女','计算机应用','08d02' " +
        "union all select '朱莉','女','英语','07d02'"; //定义插入的 SQL 语句
    insert.insertStu(sql); //调用插入数据方法
}
```

技术要点

本实例使用 UNION ALL 语句实现批处理, 该语句的语法格式如下:

```
INSERT tableName
SELECT columnValue,...
UNION ALL
SELECT columnValue,...
```

参数说明

- ☒ tableName: 要添加数据的数据表。
- ☒ columnValue: 要添加数据表中的数据。

实例 299 更新指定记录

(实例位置: 配套资源\SL\17\299)

实例说明

在程序开发中, 实现对数据的更新是一项很常见的操作。本实例向大家演示了如何更新学生表中的特定记录。运行本实例, 首先将学生表中的数据全部显示在窗体中, 用户可在该窗体中选择要修改的学生记录进行修改操作。实例的运行效果如图 17.28 所示。



图 17.28 修改学生信息

实现过程

(1) 在项目中创建类 UpdateStuFrame, 该类继承自 JFrame 类, 实现窗体类。该类用于显示学生表中全部信息方法, 该窗体中包含一个 JTable 控件与两个 JButton 控件。

(2) 在项目中定义类 UpdateStu, 在该类中定义更新数据的 updateStu() 方法, 该方法以与学生表 tb_stu 对应的 JavaBean 对象为参数。关键代码如下:

```
public void updateStu(Stu stu){
    conn = getConn();                                //获取数据库连接
    try {
        PreparedStatement statement = conn.prepareStatement("update tb_stu set name = ?,sex
= ?,grade = ?,specialty = ? where id = ?");          //定义更新 SQL 语句
        statement.setString(1, stu.getName());         //设置预处理语句参数
        statement.setString(2, stu.getSex());
        statement.setString(3, stu.getGrade());
        statement.setString(4, stu.getSpecialty());
        statement.setInt(5, stu.getId());
        statement.execute();                            //执行预处理语句
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

技术要点

本实例在实现数据更新时, 使用了 update 语句, 其语法格式如下:

UPDATE 数据表名 SET 字段名 = 新的字段值 WHERE 条件表达式

例如, 将员工 tb_emp 表中的编号为 2 的员工姓名修改为“葛雷”, 代码如下:

update tb_emp set name = '葛雷' where id = 2;

通过 Statement 实例的 executeUpdate() 方法可实现通过 Java 程序向数据库发送修改 SQL 语句。

多学两招:

本实例是将用户在某窗体中选择要更新的学生信息在另一个窗体中显示, 这样就要求两个窗体之间进行通信。本实例实现两个窗体之间的通信时将信息通过流写入到文本文件中, 再通过流从文本文件中将信息读取出来。



Note



实例 300 在删除数据时给出提示信息

(实例位置: 配套资源\SL\17\300)

实例说明

删除数据表中的数据是一个很常用的技术,在删除数据时,由于删除之后就不能自动恢复,所以在删除数据前给用户相应的提示信息是必要的。本实例实现删除学生表中指定的信息,在删除信息前给出提示信息。实例的运行效果如图 17.29 所示。

实现过程

(1) 在项目中创建类 DeleteFrame, 该类继承 JFrame 类, 实现窗体类。在该类中添加表格控件用于显示学生表信息, 并添加了用于删除数据的按钮控件。

(2) 在项目中定义类 DeleteUtil, 在该类中定义删除数据的 deleteStu() 方法, 该方法以一个 int 类型的对象为参数, 用来指定删除学生的编号。关键代码如下:

```
public void deleteStu(int id){  
    conn = getConn();           //获取数据库连接  
    try {  
        Statement statement = conn.createStatement(); //定义更新 SQL 语句  
        statement.executeUpdate("delete from tb_stu where id= "+id); //执行预处理语句  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```



图 17.29 删除数据提示信息

技术要点

在 SQL 语句中 delete 语句用于删除数据, 通过使用 JDBC 技术实现向数据库中发送 delete 语句可实现删除数据。

delete 语句的语法格式如下:

```
delete from 数据表名 where 条件表达式
```

例如, 将 tb_emp 表中编号为 1024 的员工信息删除。代码如下:

```
delete from tb_emp where id = 1024;
```

使用 Statement 对象的 executeUpdate() 方法可实现向数据库中发送删除语句。

多学两招:

在日常生活中的软件, 有些菜单显示为灰色, 表示该菜单为不可选状态。在 Java 中如何实现菜单的禁用呢?

实现菜单的禁用可通过 JMenuItem 类的 setEnabled() 方法, 将该方法的参数设置为 false, 表示菜单为禁用; 将参数设置为 true, 表示菜单启用状态。