内附光盘

Java 阶梯丛书

J2SE 进阶

■ JavaResearch.org 编著

机械工业出版社
China Machine Press

Java 阶梯丛书

J2SE 进阶

JavaResearch.org 编著

机 械 工 业 出 版 社

本书面向具有一定 Java 基础，希望能够继续深入掌握 J2SE 相关技术的朋友。本书作者都是具有多年丰富开发经验的 Java 开发人员，注重实际开发技术。本书融入了作者长期的学习、开发经验，内容定位在“进阶”，但不求面面俱到，覆盖的内容要求具有实用性和适中的深度，并穿插了对 JDK 源码、Tomcat 源码、Struts 源码、JDOM 使用的剖析，以及 Java 新增特性 logging、prefs、regex、nio、javax.sql 等主题的探讨。本书是提高 J2SE 相关技术的理想读物。

图书在版编目（CIP）数据

J2SE 进阶 / JavaResearch.org 编著.

-北京：机械工业出版社，2004.5

（Java 阶梯丛书）

ISBN 7-111-14369-8

Ⅰ. J… Ⅱ. J… Ⅲ. Java 语言-程序设计

Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2004）第 036668 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

策划编辑：吴宏伟　　责任编辑：王金航　　版式设计：谭奕丽

北京蓝海印刷有限公司印刷·新华书店北京发行所发行

2004 年 5 月第 1 版第 1 次印刷

787mm×1092mm 1/16 • 23.75 印张 • 557 千字

0001-5000 册

定价：39.00 元（含 1CD）

前　言

这是一本落在想象范围内的书。

不知道从什么时候开始，人们对书本的依赖越来越弱。不是说不需要看书，而是寻找答案的时候大部分都求助于网络。这其中可能有许多原因，但是我想有两条比较突出，一是很难从适合的书中找到合适的答案，二是没有多少书能够引起足够的欲望让自己花掉不少银子来将其纳为己有，总觉得花一本书的价钱换来几个章节的满意，会有点不甘心。在开始写这本书之前，我们定下了一个目标：不要自己亲手酿造另一个无人喝彩的“悲剧”。我们是这么想的，相信读过这本书以后，你也会认为我们是这么做的。

这是一朵山腰上的百合，协助你克服登山时的疲劳，提醒你再往上就是山顶的风光。

也许本书的书名“J2SE 进阶”对你来说有点晦涩，其实我们只是想用这样的词汇来表达我们这样一种想法：内容定位在“进阶”，但不求面面俱到，覆盖的内容要求有实用性和适中的深度，这主要体现在以下几个方面。

一、内容新，紧跟最新技术潮流

主要以 J2SE 1.4 为基础，介绍其中最新的技术趋势。比如，在第 1 章中，我们将精力放在 NIO 的应用上，而其他的章节也尽量以这些新特性为讲解重点，在第 5 章中我们就花了相当的篇幅来展示 NIO 带给套接字编程的变化。

二、在深度上做文章

本书虽然以 J2SE 为对象，但是只要一看目录就知道没有做到面面俱到，是的，我们不求面面俱到。我们主要想通过深入到几个关键的技术当中，让读者切实提高自己的技术水平。有的章节平淡中见精彩，比如线程、集合框架以及 JDBC 等，虽然是一些常见的内容，但是我们希望通过自己的一些实际应用经验来让你体会到一些不同的味道。有些章节是在同类书中涉及得比较少的内容，比如高级 UI、XML 处理等。

三、紧扣实际应用

这本书另一个特点就是能够紧密结合实际应用。不论是技术的讲解，还是代码的示例，我们都是尽量从实际应用的角度出发，讲解在实际应用中如何来使用某项技术，实际应用中有哪些需要考虑的方面。我们相信，实际应用的技术才是有生命力的技术，只有学会如何利用技术来解决实际的问题才是真正掌握了某项技术。细心的读者可能会发现，在本书的很多章节中都有专门分析某项技术在一些开源项目中的应用，在这里你可以了解到 Tomcat 中 HTTP 连接器的精妙，也可以感受到 Struts 框架支持国际化设计的魅力。

本书主要由唐子龙、赵科、钱霄、赵涌、王和全、王洪铎、谭逊、颜承和刘军等执笔，作者都是多年的 Java 程序员，在实际的学习、开发中积累了丰富的经验。

- ✧ 唐子龙，高级 Java 程序员和程序分析员。现工作于 Strategic Technology Institute（Sti）。目前兴趣包括算法设计、在线数据分析和软件项目管理。负责第 1 章。
- ✧ 钱霄，ERP/CMS 产品研发工程师。自 2000 年起研究高性能 Java CMS 系统的设计，

曾在 Java 研究组织等网站发表相关文章多篇。目前兴趣方向为高性能企业应用、分布式系统。负责第 2 章。

- ✧ 赵科，SCJP，多年 Java 开发经验，潜心于 Java 相关技术的学习和研究，在《开发系统世界》、Java 研究组织网站等杂志媒体发表文章近 30 篇，参与多部 Java 图书的翻译工作。负责第 3 章。
- ✧ 赵涌，多年的编程经验，从 1998 年开始由 C/C++编程转为 Java 编程，一直在电信领域编写和设计分布式应用软件。现在对 Java 服务器端的模式和架构非常感兴趣。负责第 4、11 章。
- ✧ 王和全，一名普通的软件工程师，主要从事广电行业的大型 J2EE 系统的开发工作。负责第 5、9 章。
- ✧ 王洪铎，软件工程师，一直从事 Web 开发，活跃在电子商务领域，对 Web 服务的框架很感兴趣。负责第 6 章。
- ✧ 谭逊，Java 程序员，兴趣方向主要在 JDBC、JDO 等数据库相关的中间件技术上。负责第 7 章。
- ✧ 颜承，Java 程序员，主持并参与了多项 Java 企业级应用系统的开发，目前兴趣主要集中在 EAI 领域。负责第 8、12 章。
- ✧ 刘军，高级软件工程师和系统设计师，多年的 Java 开发经验，主要从事金融行业的 Java 应用系统设计与开发。负责第 10 章。

另外，参与本书写作的还有刘敏、朱岱、公飞、周键、叶丽、伊言、全烂、刘志春、杨念勇、许先涛、高刚刚、谢志文和罗时飞等。其中，刘敏进行了前期的小组组建策划工作，并对第 6 章进行了修订。Java 研究组织的相关工作人员对本书作了组织、协调工作，并提出了很多中肯的批评和建议。

如果你读完本书仍感到意犹未尽，需要了解相关信息，请访问我们的网站 www.JavaResearch.org。真心祝愿您阅读愉快！

Java 研究组织《J2SE 进阶》写作项目组

目　录

第1章 新I/O

从JSDK1.4版本开始，Java导入一组全新的I/O用户程序界面（API），即新I/O（New I/O）。它主要包括6个包：

- java.nio
- java.nio.channels
- java.nio.channels.spi
- java.nio.charset
- java.nio.charset.spi
- java.util.regex

同时，java.lang、java.net和java.io 3个包有新类的加入和改写，总体大约有85个新类被加入和改写，特别是java.io包，很多类和方法已改写，以充分利用新I/O的性能优势。在JSDK1.4平台上，文件处理和网络服务器程序的性能都得到了明显的改善。

本章主要介绍新I/O类库的基本方法、标准化的编程范例和作者的实际例子。由于篇幅有限，本章难以对该包进行全面的讨论，作者力求做到较为详细地介绍基本技能和技巧，比较新I/O类库和传统I/O类库的异同，着重性能分析。希望读者在读完本章后能较为自如地使用新I/O的API。

1.1 缓冲器（Buffer）和通道（Channel）

本节主要讨论新I/O类库中的两个核心概念：缓冲器（Buffer）和通道（Channel）。首先对比新I/O和传统I/O概念上的异同。新I/O和传统I/O包最重要的差别在于如何对待数据的存取以及数据在程序中和程序间的转移。

传统I/O系统基于两个核心概念：Byte（字节）和Stream（数据流）。系统一次只处理一个字节。具体地说输入流一次生成一个比特，而输出流一次也只输出一个比特。这种模式的好处是概念易于理解，容易过滤数据，容易链接几个经过过滤的不同数据源，同时各个数据源可以用不同的方法处理数据。不利之处是速度太慢。有关此模式的进一步资料可参见API文献和Java Tutorial。

新I/O系统则有两个对应的核心概念：缓冲器（Buffer）和通道（Channel）。每一缓冲器中直接或间接包含一个字节数组（Byte Array）作为数据存储。即是说在新I/O系统中，操作针对字节数组，而不是一个字节。这种模式的主要好处是速度快，因为它利用了操作系统管理文件和内存的方式方法，并且将一些耗时操作直接转嫁给操作系统。相比而言，其概念则不如传统的I/O那么直观，容易理解。

需要强调一点，java.nio类库的目的不是用来替换java.io类库，两者在Java平台上的

角色和目标相同，作用相辅相成。java.nio 类库的作用是增加新的功能和提高原有 API 的性能。就像 javax.swing 类库不能代替 java.awt 类库一样，但 swing 控件有更好的性能和 Look&Feel。且全面改写后的 java.io 类库的速度也已明显提高，例如 InputStream 的组读入 read（byte[] bytearray, int startindex, int length）和 OutputStream 中的组写入 write（ byte[] bytearray, int startindex, int length）。

1.1.1 缓冲器

在新 I/O 类库中，缓冲器是抽象类。它及其衍生类处理所有数据的读写以及相关的运算。缓冲器内部有一个数组（字节数组），并通过此数组实现数据的管理和运算。数组的数据类型取决于缓冲器的定义。另外，缓冲器还控制着操作系统的读写过程。Java 为每一个原始数据类型定义了相应的缓冲器，其中常用的缓冲器是 ByteBuffer。它们是：

- ByteBuffer
- CharBuffer
- ShortBuffer
- IntBuffer
- LongBuffer
- FloatBuffer
- DoubleBuffer

所有原始数据类型的缓冲器都从 Buffer 抽象类继承而来。所有的缓冲器都用相同或相似的方法管理数据，只是所管理的数据类型不同。另外，因为 ByteBuffer 被用于绝大多数标准的 I/O 操作，它有一些特有的方法，如对其他数据类型的读写。

```
public ByteBuffer putInt(int value)
public int getInt(int index)
```

代码 1.1 演示 ByteBuffer 的简单使用，首先生成 4 字节数组和 ByteBuffer，再将字节数组的值赋于 Buffer，然后取出整数并生成一个 Color 对象。（参见配套代码 SimpleByteBufferDemo.java，位于本书配套光盘中的/org/javaresearch/book/j2seimproved/nio 目录中，下同。）

代码 1.1 ByteBuffer 的使用

```
package org.javaresearch.book.j2seimproved.nio;

import java.awt.Color;
import java.nio.ByteBuffer;

public class SimpleByteBufferDemo
{
```

```
    public SimpleByteBufferDemo()
    {
        byte bytes[] = {(byte)0, (byte)255, (byte)0, (byte)0};

        ByteBuffer byteBuffer = ByteBuffer.allocate(4);
        for (int i = 0; i < bytes.length; i++)
        {
            byteBuffer.put( bytes[i]);
        }
        //重设 ByteBuffer 的状态参数
        byteBuffer.flip();

        //取出整数，生成 Color 对象
        Color color = new Color(byteBuffer.getInt(), true );
        System.out.println(color);
        System.out.println(Color.red);
    }

    public static void main(String[] args) throws Exception
    {
        new SimpleByteBufferDemo();
    }
}
```

1.1.2　通道

通道本身是一个界面，功能类似于传统 I/O 中的 Stream。但通道具有双向性，既可以读入，又可以写入。而 Stream 只能进行单方向操作，任何 Stream 类必须是 InputStream，它只允许读，或是 OutputStream，只允许写。通道的双向性反映了操作系统文件读写的特征。例如 UNIX 操作系统下文件的读写就是双向性的。通道有 6 个子界面，如 ByteChannel、ReadableByteChannel 和 WritableByteChannel 等；共有 7 个类直接或间接地实现该界面，如 FileChannel、SelectableChannel 和 SocketChannel 等。详细请参阅 Java API 文献。

1.1.3　缓冲器操作

1.1.3.1　构造缓冲器

各种缓冲器均没有构造函数，生成是通过缓冲器的静态方法来实现的。缓冲器一旦生成，容量不可改变。共有两种构造方法，以 ByteBuffer 为例：

方法 1:

```
ByteBuffer byteBuffer=ByteBuffer.allocate（256）;
```

allocate 函数生成一个数组，并将该数组打包进缓冲器。本例中数组数据类型是 Byte，长度为 256。

方法 2:

```
Byte byteArray[] = new byte[1024]
ByteBuffer byteBuffer = ByteBuffer.wrap(bytearray);
```

如果已经存在一个 byte 数组 byteArray，可以用 wrap()函数生成缓冲器。须注意， 对 byteBuffer 的读写操作可以直接修改存在于 byteArray 中的数据。

对于其他数据类型，如整数，可以用相同的方法生成。另外，还可以将 byteBuffer 看作为其他原始数据类型缓冲器。这种方法生成的缓冲器, 其内部存储使用 JVM 的内存。

另外一种缓冲器称为直接缓冲器（Direct Buffer），它占用操作系统的内存。构造方法如下:

```
ByteBuffer byteBuffer = ByteBuffer.allocateDirect (256);
```

由于直接缓冲器通常与文件读写有关，详细讨论见本章第 1.2.2 节。

1.1.3.2 缓冲器状态

缓冲器内部使用 3 个变量来描述当前状态。

- 位置 position：是指缓冲器当前可以读写的位置。
- 限度 limit：是指缓冲器第一个不可以读写的位置。
- 容量 capacity：是指缓冲器的长度。缓冲器一旦生成这个值，就不会改变。

这三者的关系是 position≤limit≤capacity。

理解这 3 个变量对于正确使用缓冲器至关重要。由于容量 capacity 不随任何操作而改变，我们在讨论中忽略它。下面通过图解来说明。

步骤 1 生成 ByteBuffer byteBuffer=ByteBuffer.allocate（8）。此时缓冲器内部状态如图 1.1 所示。

步骤 2 在 byteBuffer 中存入数据（代码 1.2），

代码 1.2

```
byteBuffer.put((byte)1)
byteBuffer.put((byte)2)
byteBuffer.put((byte)3)
```

此时，position 是 3，limit 等于 8 不变，如图 1.2 所示。如果现在调用 get()函数，会得到 0，这是默认值。

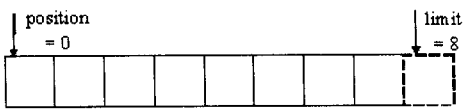

图 1.1 ByteBuffer 内部原始状态

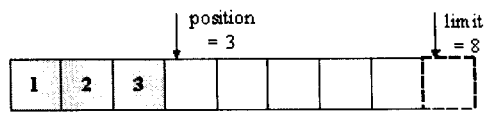

图 1.2 赋值后 ByteBuffer 内部状态

步骤 3　调用 rewind()，将 position 移回到 0，limit 等于 8 不变，如图 1.3 所示。

步骤 4　调用代码 1.3 后，Buffer 内部如图 1.4 所示。

代码 1.3

```
//将状态恢复到步骤 2
byteBuffer.position(3);
//将 position 移到 0，limit 移到 3，
byteBuffer.flip();
```

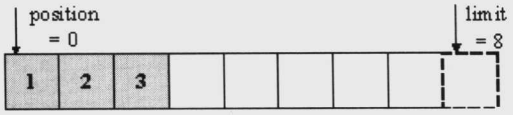

图 1.3　调用 rewind()方法后 ByteBuffer 内部状态

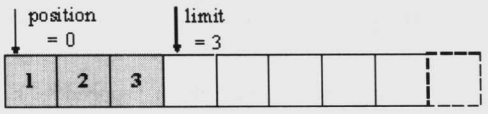

图 1.4　调用代码 1.3 后 ByteBuffer 内部状态

提示

在向缓冲器读/写数据后忘记调用 flip()函数，然后直接用 put()/get()写/读数据可能导致逻辑错误。

步骤 5　调用代码 1.4 方法，Buffer 内部如图 1.5 所示。

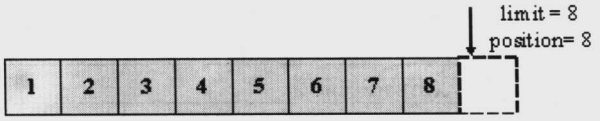

图 1.5　调用代码 1.4 后 ByteBuffer 内部状态

代码 1.4

```
//恢复 limit 到容量
byteBuffer.limit(byteBuffer.capacity());
// 恢复 position 到 3
byteBuffer.position(3);
//输出此时状态
System.out.println(BufferUtils.getBufferMetaText(byteBuffer));
byte counter = 3;
//继续赋值
while( byteBuffer.remaining() > 0 )
{
    byteBuffer.put((byte)(++counter));
}
```

步骤 6. 调用方法 byteBuffer.clear()，可将状态参数恢复到原始值，但是 Buffer 内的值并不改变，如图 1.6 所示。

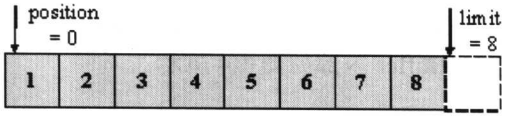

图 1.6　清空后，ByteBuffer 内部状态

另外，代码 1.5 可检查 Buffer 的数值。

代码 1.5

```
StringBuffer buf = new StringBuffer();
for (int i = 0; i <byteBuffer.limit(); i++)
{
    buf.append(byteBuffer.get(i)).append(", ");
}
System.out.println(buf.substring(0, buf.lastIndexOf(",")));
```

以上过程详见配套代码 BufferOperations.java 中的 demoStatusParameter()方法。

Buffer 类中有若干函数与状态参数直接有关，使用时容易混淆，总结列于表 1.1。

表 1.1　Buffer 的状态参数管理方法

方法名	说　明
position()	取出现在的位置
position(int index)	将 index 设为当前位置
limit()	取出现在的限度
limit(int newLimit)	将 newLimit 设为当前限度
clear()	position 设为 0，limit 设为 capacity，取消所有 mark
rewind()	position 设为 0，取消所有 mark
flip()	limit 设为当前 position, position 设为 0

1.1.3.3　读取操作

本节以整数缓冲器（IntBuffer）为例说明，所有方法也适用于其他缓冲器类型。

对于一个缓冲器类，通常有 4 个函数来取数据：

- int getInt()
- IntBuffer get(int[] dst)
- IntBuffer get(int[] dst, int start, int length)
- int getInt(int index)

第一种方法取出当前位置的整数值，并改变 position 参数。

第二种方法将缓冲器中所有数据复制到整数数组，并改变 position 参数。起始于当前位置，长度为目标数组长度。如超出 Buffer 的限制，系统抛出异常 BufferUnderflowException。代码 1.6 摘自配套代码 BufferOperations.java。其中，BufferUtils.getBufferMetaText(intBuffer)是工具函数，参见配套代码中的 BufferUtils.java，其中包含许多常用函数。