



技术畅销书作者撰写，掌握高并发与网络编程基石技术：NIO与Socket

细化到特性级别，涵盖领域中核心技术，包括缓冲区、通道、选择器以及基于Socket的TCP/IP和UDP编程



NIO与Socket 编程技术指南

NIO and Socket Programming Technical Guide

高洪岩 著



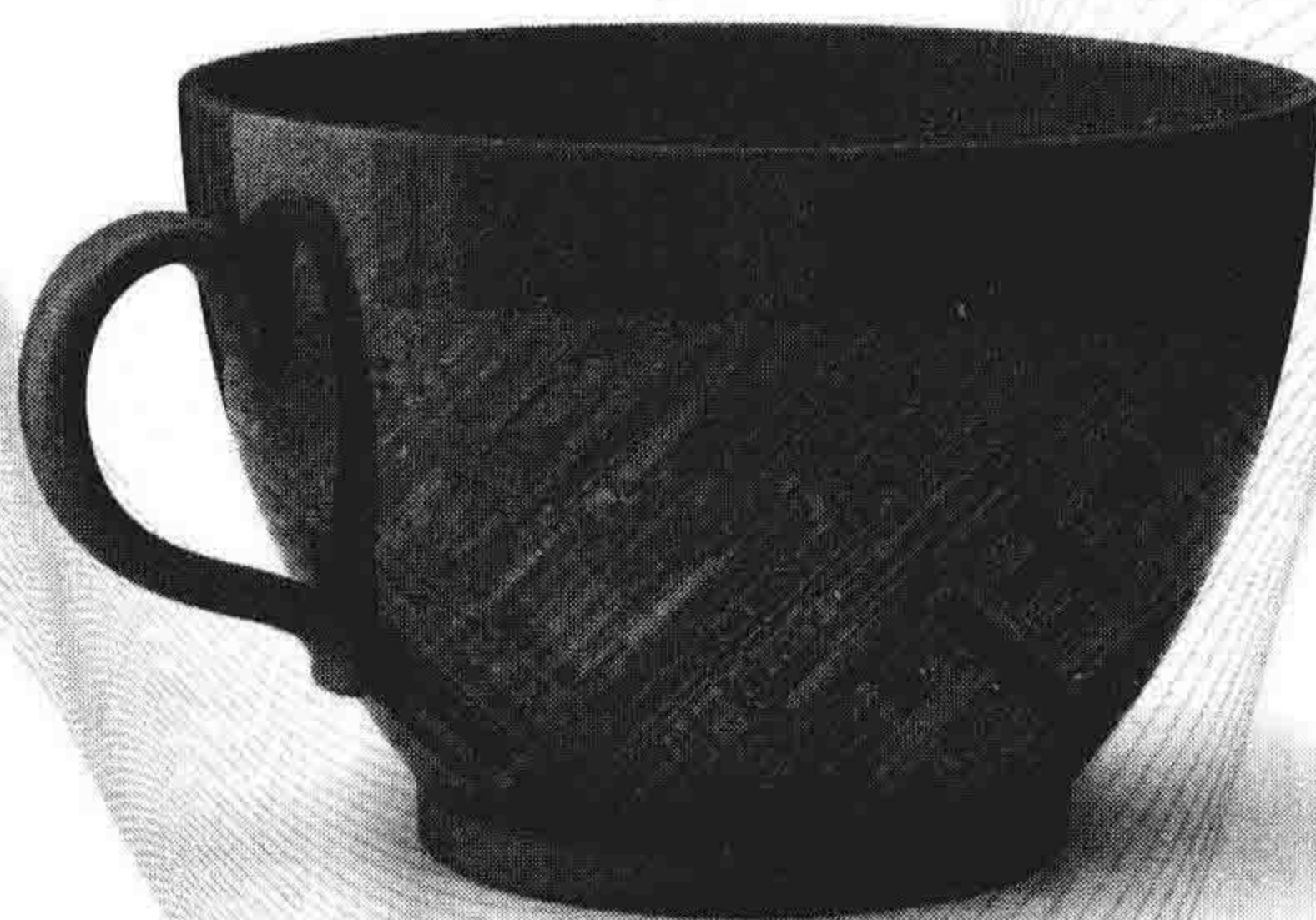
机械工业出版社
China Machine Press



NIO与Socket 编程技术指南

NIO and Socket Programming Technical Guide

高洪岩 著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

NIO 与 Socket 编程技术指南 / 高洪岩著. —北京: 机械工业出版社, 2018.7
(Java 核心技术系列)

ISBN 978-7-111-60406-8

I. N… II. 高… III. JAVA 语言—程序设计—指南 IV. TP312.8—62

中国版本图书馆 CIP 数据核字 (2018) 第 141692 号

NIO 与 Socket 编程技术指南

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 高婧雅

责任校对: 李秋荣

印刷: 三河市宏图印务有限公司

版次: 2018 年 7 月第 1 版第 1 次印刷

开本: 186mm×240mm 1/16

印张: 28.25

书号: ISBN 978-7-111-60406-8

定价: 99.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88379426 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

为什么要写这本书

早在几年前，笔者就曾想过整理一份基于 Java 语言的 NIO 与 Socket 相关的稿件，因为市面上大部分的 Java 书籍都是以 1 章或 2 章的篇幅介绍 NIO 与 Socket 技术，并没有完整地覆盖该技术的知识点，而限于当时的时间及精力，一直没有如愿。

机会终于来了，公司要搭建基础构架知识体系，我负责公司该技术方向的培训，这重燃了我对 NIO 和 Socket 技术的热情。在学习 Java 技术的过程中，当学习了 Java SE/Java EE 之后想探索更深层次的技术，如大数据、分布式和高并发类等，可能会遇到针对 NIO、Socket 的学习，但 NIO 和 Socket 技术的学习并不像 JDBC 一样简单，学习 NIO 和 Socket 时可能要遇到很多的问题。为了在该技术领域有更高的追求，我将 NIO 和 Socket 的技术点以教案的方式进行了整理，并在公司中与同事一起进行学习和交流，同事的反响非常热烈。若干年前的心愿终于达成，同事们也很期待这本书能早日出版发行，那样他们就有真正的纸质参考资料了。希望本书能够受到其他学习 NIO 和 Socket 的读者喜爱，这是我最大的心愿。

本书介绍 NIO 和 Socket 开发中最值得关注的内容，并给出个人的一些想法和见解，希望拓宽读者的学习思路。

在学习 NIO 和 Socket 技术之前，建议先了解一下多线程与并发相关的知识，这对设计和理解代码有非常大的帮助。多线程方面的资料推荐《Java 多线程编程核心技术》，并发相关的资料推荐《Java 并发编程：核心方法与框架》，这两本书都是笔者编著的，希望可以给读者带来一些帮助。

本书特色

在本书写作的过程中，我尽量做到言简意赅，并且全部用演示案例的方式来讲解技术知

识点，使读者看到代码及运行结果后就可以知道此项目要解决的是什么问题。这类似于网络中的博客风格，让读者用最短的时间学习知识点，明白知识点的应用方式及使用时的注意事项，取得快速学习并解决相应问题的效果。

读者对象

- Java 程序员
- 系统架构师
- 大数据开发者
- 其他对 NIO 和 Socket 技术感兴趣的人员

如何阅读本书

本着实用、易懂的学习原则，本书通过 6 章内容来介绍 Java 多线程相关的技术。

第 1 章介绍 NIO 技术中的缓冲区，包括 Buffer、ByteBuffer、CharBuffer 类的核心 API 的使用。

第 2 章介绍 NIO 技术中的 Channel（通道）类的继承关系、核心接口的作用，并重点介绍 FileChannel 类的使用，以增加读者对 NIO 操作 File 类的熟悉度。

第 3 章介绍如何使用 NetworkInterface 类获得网络接口的信息，包括 IP 地址、子网掩码等，还会介绍 InetAddress 和 InterfaceAddress 类的常见 API。如果进行 Java 开发，且基于 Socket 技术，那么这章可以给你需要的信息。

第 4 章介绍如何使用 Java 语言实现 Socket 通信。Socket 通信是基于 TCP/IP 和 UDP 实现的。另外，将介绍 ServerSocket、Socket、DatagramSocket 和 DatagramPacket 类的全部 API。只有熟练掌握 Socket 技术后，在阅读相关网络框架的源代码时才不会迷茫。也就是说，如果读者想要进行 Java 高性能后台处理，那么必须要学习 Socket，并且它是进行细化学习的基础。

第 5 章介绍 NIO 技术中最重要的 Selector（选择器）技术。NIO 技术的核心——多路复用就是在此章体现的。学习这章内容需要有 Socket 的编程基础，这就是为什么在前面用两章篇幅来介绍 Java 的 Socket 编程的原因。同步非阻塞可以大幅度提升程序运行的效率，就在此章体会一下吧。

第 6 章介绍 AIO。AIO 是异步 IO，NIO 是非阻塞 IO。AIO 在 NIO 的基础上实现了异步执行、回调处理等高级功能，可以在不同的场景使用 AIO 或 NIO，可以说 NIO 和 AIO 是 Java 高级程序员、架构师等必须要掌握的技术。

勘误和支持

由于笔者的水平有限，加之编写仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正，期待能够得到你们的真挚反馈，在技术之路上互勉共进。若读者想与我进行技术交流，可发电子邮件到 279377921@qq.com。

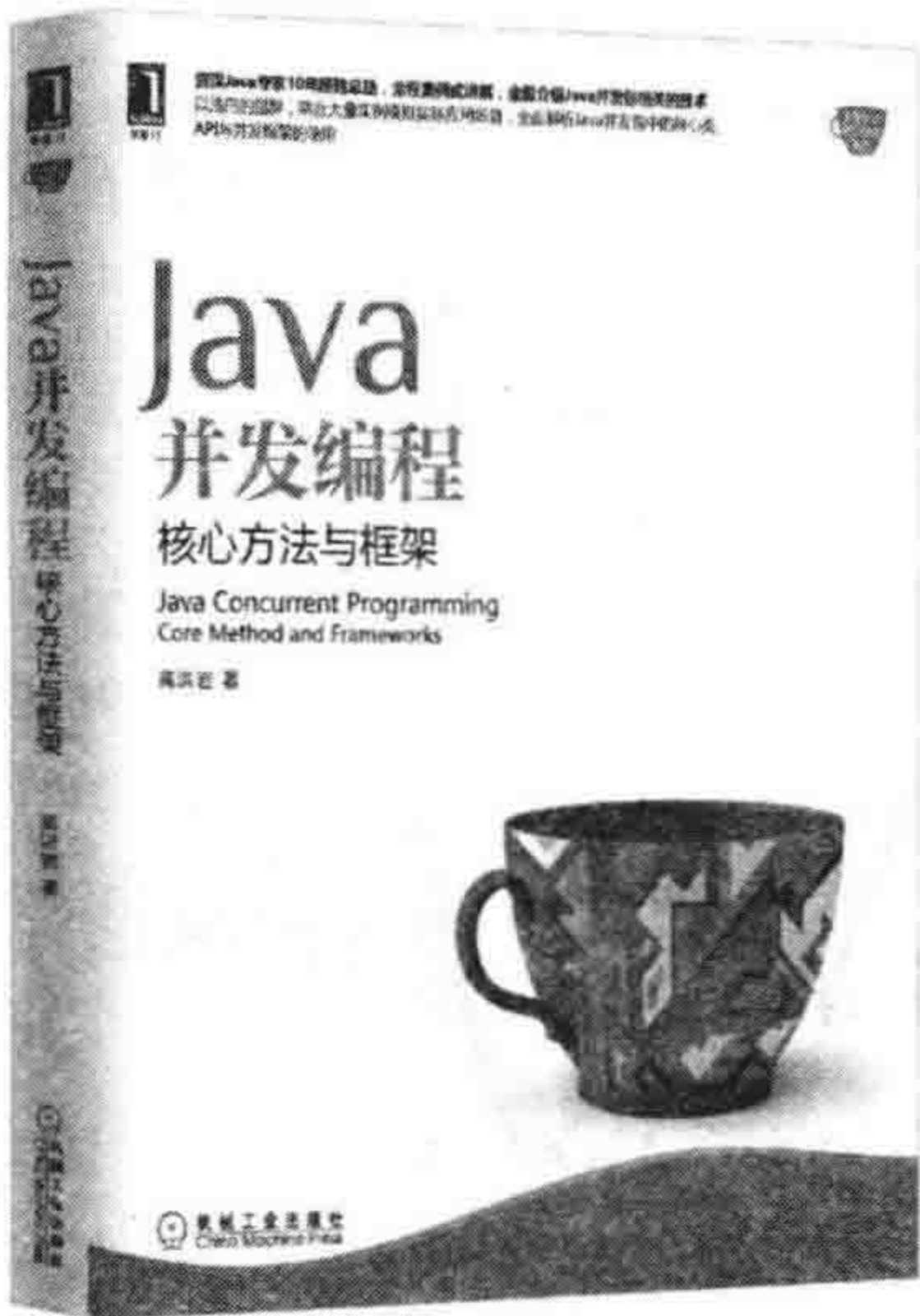
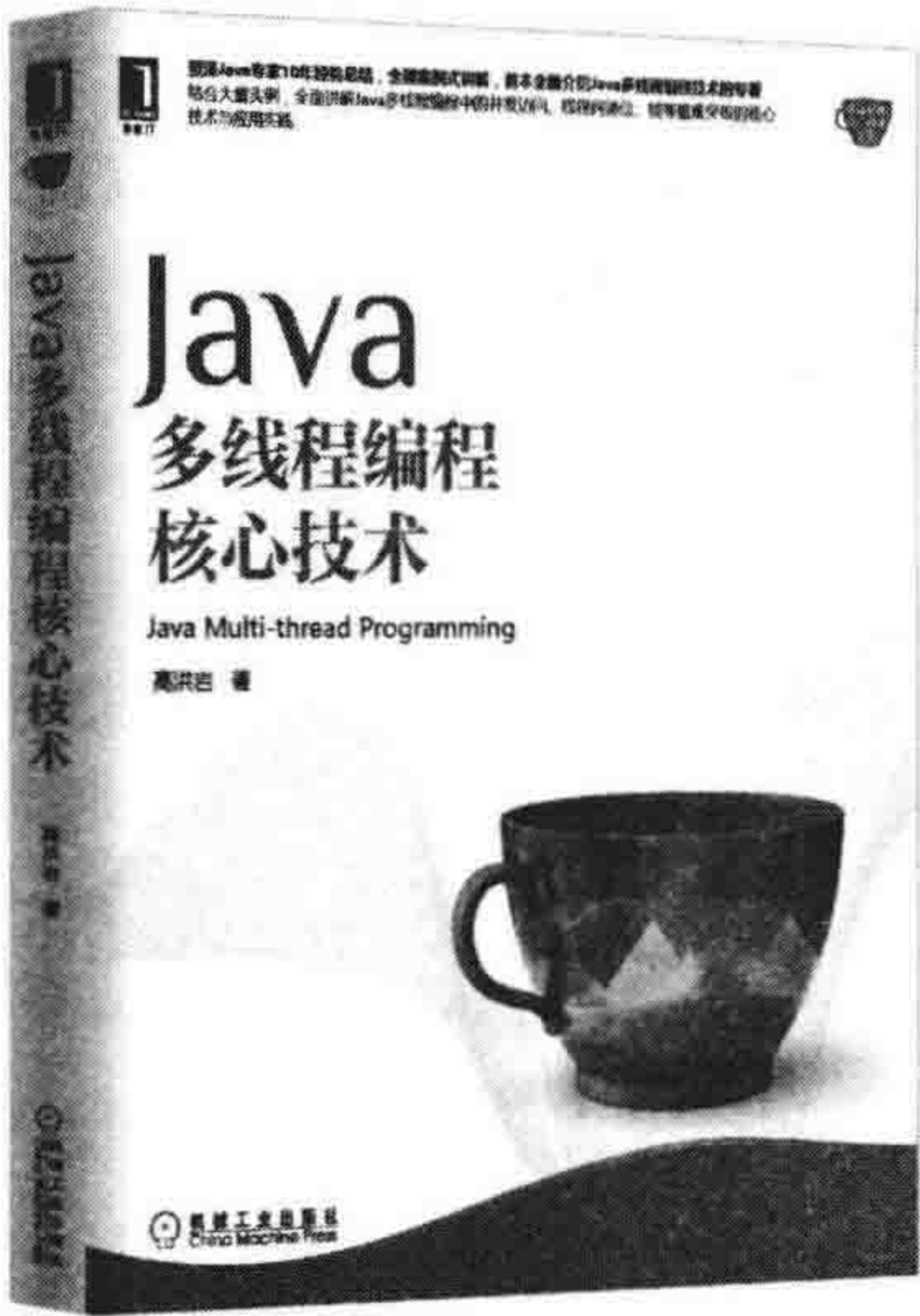
致谢

感谢所在单位领导的支持与厚爱，使我在技术道路上更有信心。

感谢机械工业出版社华章公司的高婧雅，始终支持我的写作，你是我最爱的编辑。因为你们的鼓励和帮助，所以我才会如此顺利地完成了这本书的写作。

高洪岩

推荐阅读



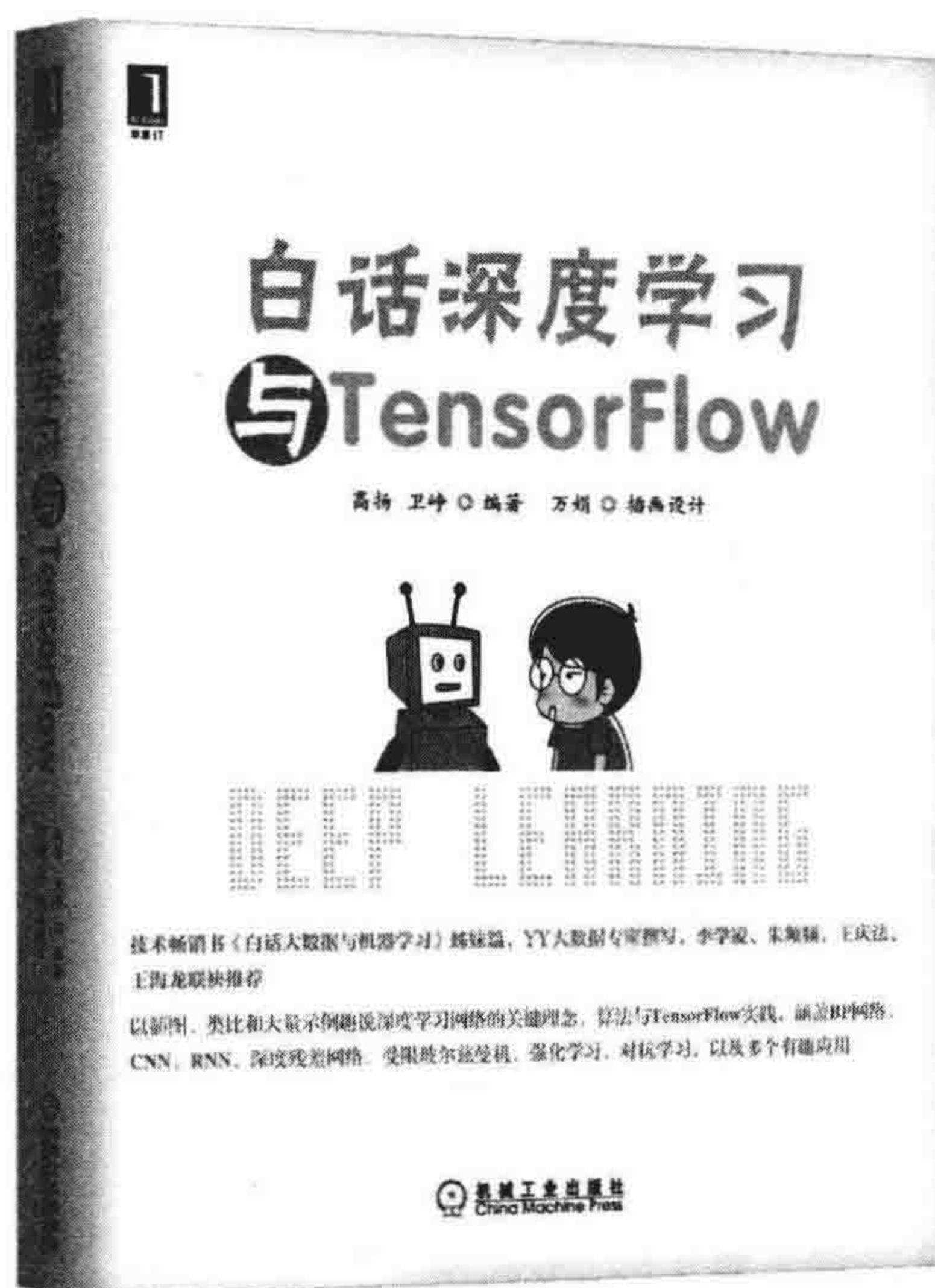
Java多线程编程核心技术

作者：高洪岩 ISBN：978-7-111-50206-7 定价：69.00元

Java并发编程：核心方法与框架

作者：高洪岩 ISBN：978-7-111-53521-8 定价：79.00元

推荐阅读

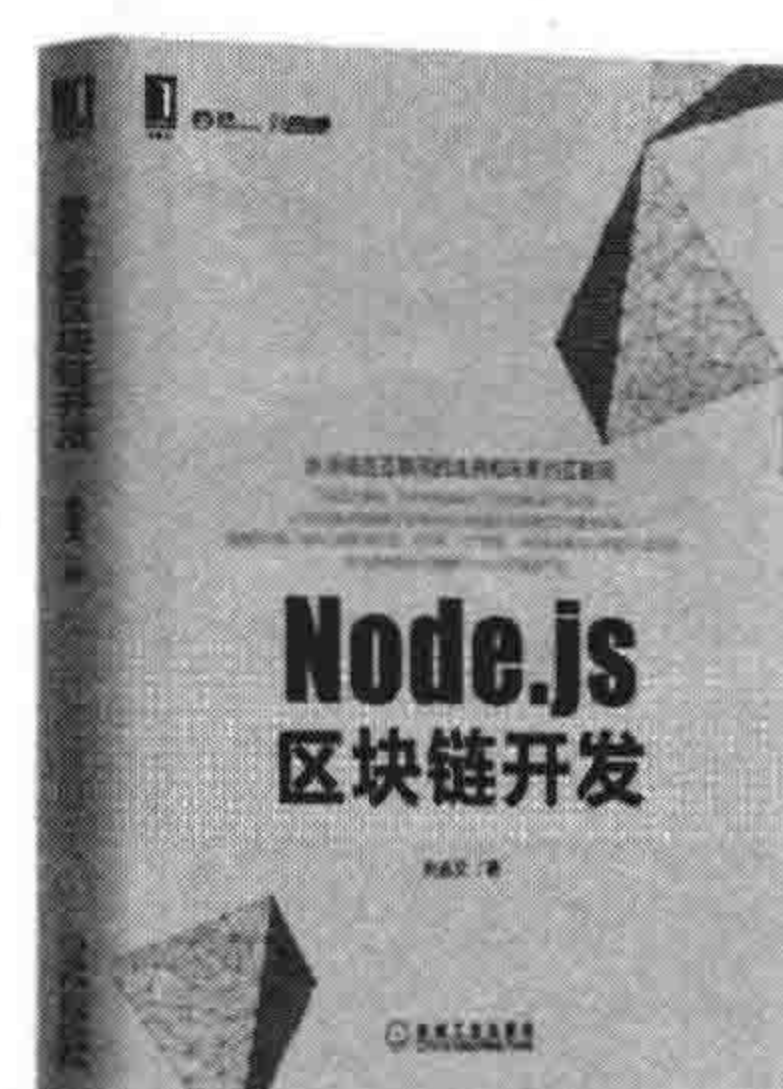
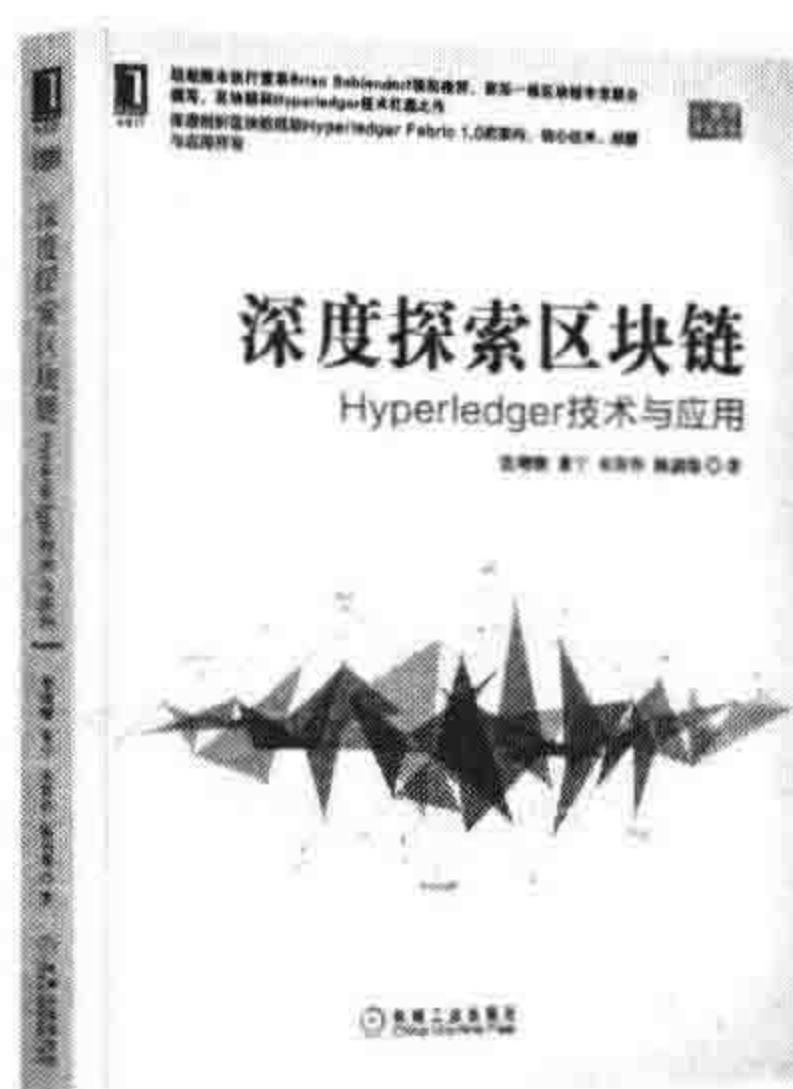
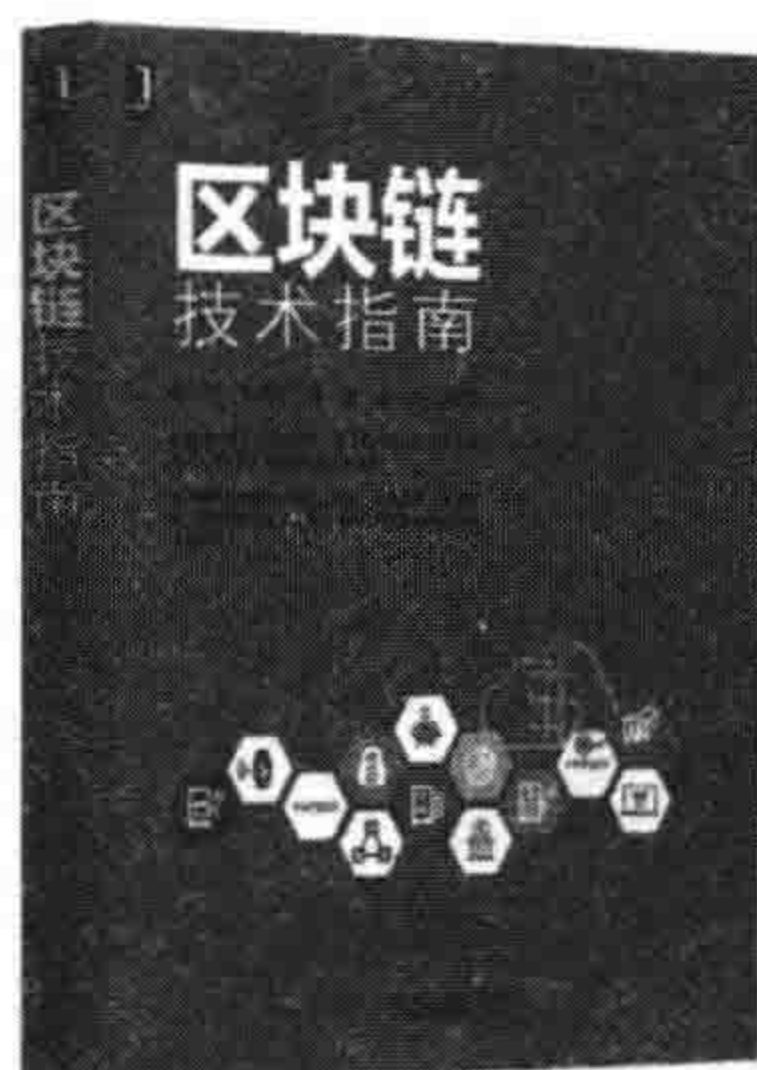


白话深度学习与TensorFlow

书号: 978-7-111-57457-6 作者: 高扬 定价: 69.00元

《白话大数据与机器学习》姊妹篇，YY大数据专家多年实践沉淀之作，
简单易懂，轻松入门

推荐阅读



目 录 *Contents*

前 言

第 1 章 缓冲区的使用 1

1.1 NIO 概述 5

1.2 缓冲区介绍 6

1.3 Buffer 类的使用 7

1.3.1 包装数据与获得容量 7

1.3.2 限制获取与设置 10

1.3.3 位置获取与设置 12

1.3.4 剩余空间大小获取 13

1.3.5 使用 Buffer mark() 方法处理 标记 14

1.3.6 知识点细化测试 15

1.3.7 判断只读 22

1.3.8 直接缓冲区 22

1.3.9 还原缓冲区的状态 23

1.3.10 对缓冲区进行反转 24

1.3.11 判断是否有底层实现的数组 28

1.3.12 判断当前位置与限制之间是否有 剩余元素 29

1.3.13 重绕缓冲区 30

1.3.14 获得偏移量 32

1.3.15 使用 List.toArray(T[]) 转成数组 类型 33

1.4 ByteBuffer 类的使用 34

1.4.1 创建堆缓冲区与直接缓冲区 35

1.4.2 直接缓冲区与非直接缓冲区的运 行效率比较 37

1.4.3 包装 wrap 数据的处理 39

1.4.4 put(byte b) 和 get() 方法的使用与 position 自增特性 40

1.4.5 put(byte[] src, int offset, int length) 和 get(byte[] dst, int offset, int length) 方法的使用 41

1.4.6 put(byte[] src) 和 get(byte[] dst) 方 法的使用 46

1.4.7 put(int index, byte b) 和 get(int index) 方法的使用与 position 不变 49

1.4.8 put(ByteBuffer src) 方法的使用 50

1.4.9 putType() 和 getType() 方法的 使用 51

1.4.10 slice() 方法的使用与 arrayOffset() 为非 0 的测试 53

1.4.11	转换为 CharBuffer 字符缓冲区 及中文的处理	54	2.2.3	ReadableByteChannel 接口的 介绍	84
1.4.12	转换为其他类型的缓冲区	58	2.2.4	ScatteringByteChannel 接口的 介绍	85
1.4.13	设置与获得字节顺序	63	2.2.5	WritableByteChannel 接口的 介绍	86
1.4.14	创建只读缓冲区	65	2.2.6	GatheringByteChannel 接口的 介绍	87
1.4.15	压缩缓冲区	65	2.2.7	ByteChannel 接口的介绍	88
1.4.16	比较缓冲区的内容	66	2.2.8	SeekableByteChannel 接口的 介绍	89
1.4.17	复制缓冲区	70	2.2.9	NetworkChannel 接口的介绍	90
1.4.18	对缓冲区进行扩容	72	2.2.10	MulticastChannel 接口的介绍	91
1.5	CharBuffer 类的 API 使用	73	2.2.11	InterruptibleChannel 接口的 介绍	92
1.5.1	重载 append(char)/append(Char- Sequence)/append(CharSequence, start, end) 方法的使用	73	2.3	AbstractInterruptibleChannel 类的 介绍	93
1.5.2	读取相对于当前位置的给定 索引处的字符	74	2.4	FileChannel 类的使用	95
1.5.3	put(String src)、int read(CharBuffer target) 和 subSequence(int start, int end) 方法的使用	74	2.4.1	写操作与位置的使用	97
1.5.4	static CharBuffer wrap(Char- Sequence csq, int start, int end) 方法的使用	76	2.4.2	读操作	100
1.5.5	获得字符缓冲区的长度	76	2.4.3	批量写操作	106
1.6	小结	77	2.4.4	批量读操作	109
第 2 章 通道和 FileChannel 类的使用			2.4.5	部分批量写操作	117
2.1	通道概述	78	2.4.6	部分批量读操作	120
2.2	通道接口的层次结构	80	2.4.7	向通道的指定 position 位置 写入数据	128
2.2.1	AsynchronousChannel 接口的 介绍	82	2.4.8	读取通道指定位置的数据	130
2.2.2	AsynchronousByteChannel 接口 的介绍	84	2.4.9	设置位置与获得大小	135
			2.4.10	截断缓冲区	136
			2.4.11	将数据传输到其他可写入 字节通道	138

- 2.4.12 将字节从给定可读取字节通道
传输到此通道的文件中 141
- 2.4.13 执行锁定操作 145
- 2.4.14 FileLock lock() 方法的使用 ... 160
- 2.4.15 获取通道文件给定区域的
锁定 160
- 2.4.16 FileLock tryLock() 方法的
使用 162
- 2.4.17 FileLock 类的使用 162
- 2.4.18 强制将所有对通道文件的更新
写入包含文件的存储设备 ... 165
- 2.4.19 将通道文件区域直接映射到
内存 167
- 2.4.20 打开一个文件 174
- 2.4.21 判断当前通道是否打开 181
- 2.5 小结 182

第 3 章 获取网络设备信息 183

- 3.1 NetworkInterface 类的常用方法 ... 184
 - 3.1.1 获得网络接口的基本信息 186
 - 3.1.2 获取 MTU 大小 189
 - 3.1.3 子接口的处理 190
 - 3.1.4 获得硬件地址 192
 - 3.1.5 获得 IP 地址 194
 - 3.1.6 InterfaceAddress 类的使用 200
 - 3.1.7 判断是否为点对点设备 202
 - 3.1.8 是否支持多播 202
- 3.2 NetworkInterface 类的静态
方法 204
 - 3.2.1 根据索引获得 NetworkInterface
对象 204

- 3.2.2 根据网络接口名称获得
NetworkInterface 对象 204
- 3.2.3 根据 IP 地址获得
NetworkInterface 对象 205
- 3.3 小结 205

第 4 章 实现 Socket 通信 206

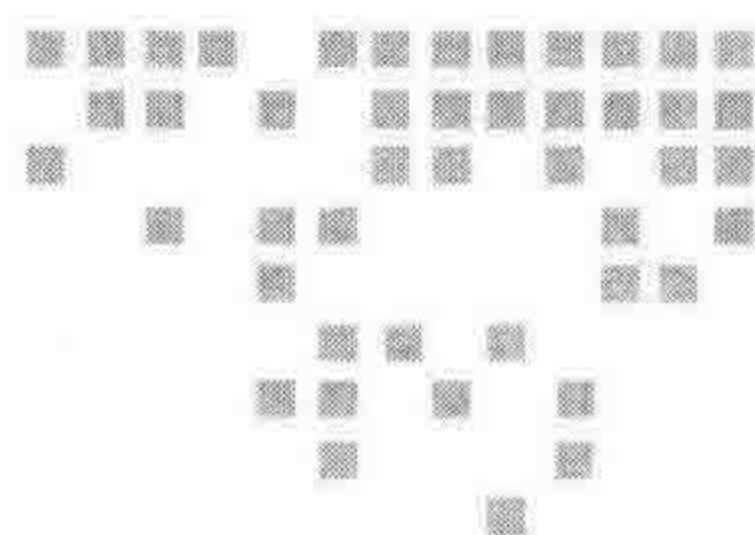
- 4.1 基于 TCP 的 Socket 通信 206
 - 4.1.1 验证 ServerSocket 类的
accept() 方法具有阻塞特性 ... 207
 - 4.1.2 验证 Socket 中 InputStream
类的 read() 方法也具有阻塞
特性 210
 - 4.1.3 客户端向服务端传递字符串 ... 212
 - 4.1.4 服务端向客户端传递字符串 ... 213
 - 4.1.5 允许多次调用 write() 方法进行
写入操作 215
 - 4.1.6 实现服务端与客户端多次的
往来通信 216
 - 4.1.7 调用 Stream 的 close() 方法
造成 Socket 关闭 219
 - 4.1.8 使用 Socket 传递 PNG 图片
文件 221
 - 4.1.9 TCP 连接的 3 次“握手”
过程 222
 - 4.1.10 标志位 SYN 与 ACK 值的自
增特性 225
 - 4.1.11 TCP 断开连接的 4 次“挥手”
过程 226
 - 4.1.12 “握手”的时机与立即传数据
的特性 227

4.1.13 结合多线程 Thread 实现 通信	228	4.3.5 获得远程 InetAddress 与远程 SocketAddress() 地址	264
4.1.14 服务端与客户端互传对象 以及 I/O 流顺序问题	231	4.3.6 套接字状态的判断	265
4.2 ServerSocket 类的使用	233	4.3.7 开启半读与半写状态	266
4.2.1 接受 accept 与超时 Timeout	233	4.3.8 判断半读半写状态	268
4.2.2 构造方法的 backlog 参数 含义	235	4.3.9 Socket 选项 TcpNoDelay	270
4.2.3 参数 backlog 的默认值	237	4.3.10 Socket 选项 SendBufferSize	274
4.2.4 构造方法 ServerSocket (int port, int backlog, InetAddress bindAddr) 的使用	238	4.3.11 Socket 选项 Linger	276
4.2.5 绑定到指定的 Socket 地址	240	4.3.12 Socket 选项 Timeout	287
4.2.6 绑定到指定的 Socket 地址并 设置 backlog 数量	242	4.3.13 Socket 选项 OOBInline	288
4.2.7 获取本地 SocketAddress 对象 以及本地端口	243	4.3.14 Socket 选项 KeepAlive	291
4.2.8 InetAddress 类的使用	244	4.3.15 Socket 选项 TrafficClass	293
4.2.9 关闭与获取关闭状态	247	4.4 基于 UDP 的 Socket 通信	294
4.2.10 判断 Socket 绑定状态	248	4.4.1 使用 UDP 实现 Socket 通信	295
4.2.11 获得 IP 地址信息	249	4.4.2 测试发送超大数据量的包导致 数据截断的情况	297
4.2.12 Socket 选项 ReuseAddress	249	4.4.3 Datagram Packet 类中常用 API 的使用	299
4.2.13 Socket 选项 ReceiveBuffer- Size	257	4.4.4 使用 UDP 实现单播	300
4.3 Socket 类的使用	259	4.4.5 使用 UDP 实现广播	301
4.3.1 绑定 bind 与 connect 以及端口生 成的时机	259	4.4.6 使用 UDP 实现组播	303
4.3.2 连接与超时	261	4.5 小结	305
4.3.3 获得远程端口与本地端口	262	第 5 章 选择器的使用	306
4.3.4 获得本地 InetAddress 地址与 本地 SocketAddress 地址	263	5.1 选择器与 I/O 多路复用	306
		5.2 核心类 Selector、SelectionKey 和 SelectableChannel 的关系	307
		5.3 通道类 AbstractInterruptibleChannel 与接口 InterruptibleChannel 的 介绍	310
		5.4 通道类 SelectableChannel 的介绍	311

5.5	通道类 <code>AbstractSelectableChannel</code> 的介绍	313	5.7.19	返回此通道所支持的操作	332
5.6	通道类 <code>ServerSocketChannel</code> 与接口 <code>NetworkChannel</code> 的介绍	313	5.7.20	执行 <code>Connect</code> 连接操作	333
5.7	<code>ServerSocketChannel</code> 类、 <code>Selector</code> 和 <code>SelectionKey</code> 的使用	315	5.7.21	判断此通道上是否正在进行连接操作	336
5.7.1	获得 <code>ServerSocketChannel</code> 与 <code>ServerSocket</code> <code>socket</code> 对象	316	5.7.22	完成套接字通道的连接过程	338
5.7.2	执行绑定操作	317	5.7.23	类 <code>FileChannel</code> 中的 <code>long transferTo(position, count, WritableByteChannel)</code> 方法的使用	340
5.7.3	执行绑定操作与设置 <code>backlog</code>	317	5.7.24	方法 <code>public static SocketChannel open(SocketAddress remote)</code> 与 <code>SocketOption</code> 的执行顺序	342
5.7.4	阻塞与非阻塞以及 <code>accept()</code> 方法的使用效果	318	5.7.25	传输大文件	344
5.7.5	获得 <code>Selector</code> 对象	320	5.7.26	验证 <code>read</code> 和 <code>write</code> 方法是非阻塞的	346
5.7.6	执行注册操作与获得 <code>SelectionKey</code> 对象	321	5.8	<code>Selector</code> 类的使用	348
5.7.7	判断注册的状态	322	5.8.1	验证 <code>public abstract int select()</code> 方法具有阻塞性	350
5.7.8	将通道设置成非阻塞模式再注册到选择器	323	5.8.2	<code>select()</code> 方法不阻塞的原因和解决办法	351
5.7.9	使用 <code>configureBlocking(false)</code> 方法解决异常	323	5.8.3	出现重复消费的情况	353
5.7.10	判断打开的状态	324	5.8.4	使用 <code>remove()</code> 方法解决重复消费问题	355
5.7.11	获得阻塞锁对象	325	5.8.5	验证产生的 <code>set1</code> 和 <code>set2</code> 关联的各自对象一直是同一个	356
5.7.12	获得支持的 <code>SocketOption</code> 列表	325	5.8.6	<code>int selector.select()</code> 方法返回值的含义	360
5.7.13	获得与设置 <code>SocketOption</code>	327	5.8.7	从已就绪的键集中获得通道中的数据	362
5.7.14	获得 <code>SocketAddress</code> 对象	327	5.8.8	对相同的通道注册不同的相关事件返回同一个 <code>SelectionKey</code>	363
5.7.15	阻塞模式的判断	328			
5.7.16	根据 <code>Selector</code> 找到对应的 <code>SelectionKey</code>	328			
5.7.17	获得 <code>SelectorProvider</code> 对象	329			
5.7.18	通道注册与选择器	330			

5.8.9 判断选择器是否为打开 状态	365	5.10.4 将通道加入组播地址	400
5.8.10 获得 SelectorProvider provider 对象	365	5.10.5 将通道加入组播地址且接收 指定客户端数据	402
5.8.11 返回此选择器的键集	366	5.11 Pipe.SinkChannel 和 Pipe. SourceChannel 类的使用	403
5.8.12 public abstract int select(long timeout) 方法的使用	367	5.12 SelectorProvider 类的使用	406
5.8.13 public abstract int selectNow() 方法的使用	368	5.13 小结	407
5.8.14 唤醒操作	369	第 6 章 AIO 的使用	408
5.8.15 测试若干细节	370	6.1 AsynchronousFileChannel 类的 使用	408
5.9 SelectionKey 类的使用	380	6.1.1 获取此通道文件的独占锁	409
5.9.1 判断是否允许连接 SelectableChannel 对象	381	6.1.2 获取通道文件给定区域的锁	410
5.9.2 判断是否已准备好进行读取	383	6.1.3 实现重叠锁定	412
5.9.3 判断是否已准备好进行写入	384	6.1.4 返回此通道文件当前大小与 通道打开状态	413
5.9.4 返回 SelectionKey 关联的 选择器	386	6.1.5 CompletionHandler 接口的 使用	413
5.9.5 在注册操作时传入 attachment 附件	387	6.1.6 public void failed (Throwable exc, A attachment) 方法调用 时机	414
5.9.6 设置 attachment 附件	389	6.1.7 执行指定范围的锁定与传入 附件及整合接口	415
5.9.7 获取与设置此键的 interest 集合	390	6.1.8 执行锁定与传入附件及整合 接口 CompletionHandler	416
5.9.8 判断此键是否有效	392	6.1.9 lock (position, size, shared, attachment, CompletionHandler) 方法的特点	418
5.9.9 获取此键的 ready 操作集合	392	6.1.10 读取数据方式 1	420
5.9.10 取消操作	395	6.1.11 读取数据方式 2	420
5.10 DatagramChannel 类的使用	396	6.1.12 写入数据方式 1	421
5.10.1 使用 DatagramChannel 类 实现 UDP 通信	398		
5.10.2 连接操作	399		
5.10.3 断开连接	400		

6.1.13	写入数据方式 2	422	6.2.3	重复读与重复写出现异常	428
6.2	AsynchronousServerSocketChannel 和 AsynchronousSocketChannel 类 的使用	422	6.2.4	读数据	429
6.2.1	接受方式 1	425	6.2.5	写数据	433
6.2.2	接受方式 2	427	6.3	同步、异步、阻塞与非阻塞之间 的关系	436
			6.4	小结	437



缓冲区的使用

学习 NIO 能更加接近架构级的技术体系，对未来的职业发展有非常好的促进作用。

当你看到以上这段文字的时候，笔者要恭喜你，因为你正在往 Java 高性能、高并发、高吞吐量技术的道路上迈进，也就代表着未来是有可能将自己的职业规划定位在 Java 高级程序员、Java 资深工程师，以及技术经理、技术总监或首席技术官（CTO）这类职位上。这些职位对 Java 技术的掌握是有一定要求和标准的，至少笔者认为要将自己对技术的关注点从 SSH、SSM 分离出去，落脚在多线程、并发处理、NIO 及 Socket 技术上，因为这些技术是开发 Java 高性能服务器必须要掌握的，甚至有些第三方的优秀框架也在使用这些技术。先不说自己开发框架，即使想要读懂第三方框架的源代码，也要掌握上面提到的多线程、并发处理、NIO 及 Socket 这 4 种核心技术。当你正在进行 SSH、SSM 这类 Web 开发工作时，想要往更高的层次发展，笔者的其他两本书《Java 多线程编程核心技术》和《Java 并发编程：核心方法与框架》，以及本书一定会带给你非常大的帮助，因为这些内容是 Java SE 技术中的核心，是衡量一个 Java 程序员是否合格的明显标志。

在正式开始介绍 NIO 之前，先简要介绍一下 Java SE 中的 4 大核心技术：多线程、并发处理、Socket 和 NIO。如果你是这些技术的初学者，那么这将帮助你了解这些技术及其用途，以及它们的应用场景。

（1）多线程

可以这样说，高性能的解决方案一定离不开多线程，它可以使 1 个 CPU 几乎在同一时间运行更多的任务。在指定的时间单位内运行更多的任务，其实就是大幅度提高运行效率，让软件运行更流畅，处理的数据更多，以提升使用软件时的用户体验。在 Java 中，使用 Thread 类来实现多线程功能的处理。在学习多线程时，要注意同步与异步的区别，也就是着重观察 synchronized 关键字在不同代码结构中的使用效果。另外，多线程的随机性，以及