

Netty、Redis、 ZooKeeper高并发实战

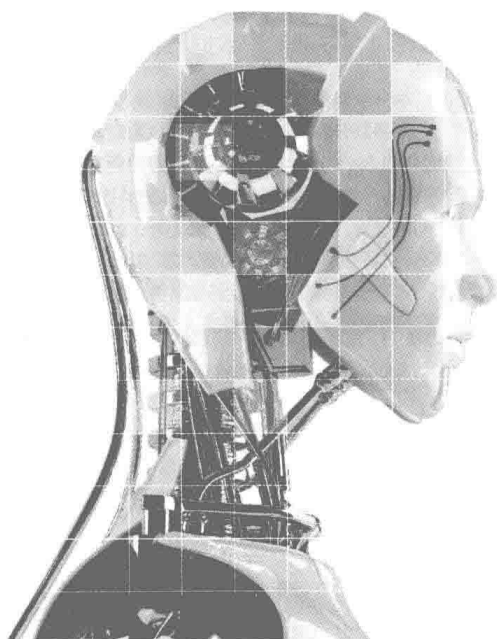
聚焦实战技能，剖析底层原理

尼恩 编著

解读高并发开发、架构、面试中的核心难题



机械工业出版社
China Machine Press



Netty、Redis、 ZooKeeper高并发实战

尼恩 编著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

Netty、Redis、Zookeeper高并发实战 / 尼恩编著. —北京: 机械工业出版社, 2019.8

ISBN 978-7-111-63290-0

I. ①N… II. ①尼… III. ①JAVA语言—程序设计 IV. ①TP312.8

中国版本图书馆CIP数据核字 (2019) 第153161号

本书从操作系统底层的 IO 原理入手, 同时提供高性能开发的实战案例, 是一本高并发 Java 编程应用基础图书。

本书共分为 12 章。第 1~5 章为高并发基础, 浅显易懂地剖析高并发 IO 的底层原理, 细致地讲解 Reactor 高性能模式, 图文并茂地介绍 Java 异步回调模式。这些原理方面的基础知识非常重要, 会为读者打下坚实的基础, 也是日常开发 Java 后台应用时解决实际问题的金钥匙。第 6~9 章为 Netty 原理和实战, 是本书的重中之重, 主要介绍高性能通信框架 Netty、Netty 的重要组件、单体 IM 的实战设计和模块实现。第 10~12 章对 ZooKeeper、Curator API、Redis、Jedis API 的使用进行详尽的说明, 以提升读者设计和开发高并发、可扩展系统的能力。

本书兼具基础知识和实战案例, 既可作为对 Java NIO、高性能 IO、高并发编程感兴趣的大专院校学生和初、中级 Java 工程师的学习参考书, 也可作为在生产项目中需要用到 Netty、Redis、ZooKeeper 三大框架的架构师或项目人员的使用参考书。

Netty、Redis、ZooKeeper 高并发实战

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: 夏非彼 迟振春

责任校对: 闫秀华

印刷: 中国电影出版社印刷厂

版次: 2019 年 8 月第 1 版第 1 次印刷

开本: 188mm × 260mm 1/16

印张: 23.5

书号: ISBN 978-7-111-63290-0

定价: 79.00 元

客服电话: (010) 88361066 88379833 68326294

投稿热线: (010) 88379604

华章网站: www.hzbook.com

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

前言

移动时代、5G 时代、物联网时代的大幕已经开启，它们对于高性能、高并发的开发知识和技术的要求，抬升了 Java 工程师的学习台阶和面试门槛。

大公司的面试题从某个侧面映射出生产场景中对专项技术的要求。高并发的面试题以前基本是 BAT 等大公司的专利，现在几乎蔓延至与 Java 项目相关的整个行业。例如，与 Java NIO、Reactor 模式、高性能通信、分布式锁、分布式 ID、分布式缓存、高并发架构等技术相关的面试题，从以前的加分题变成了现在的基础题，这也映射出开发 Java 项目所必需的技术栈：分布式 Java 框架、Redis 缓存、分布式搜索 Elasticsearch、分布式协调 ZooKeeper、消息队列 Kafka、高性能通信框架 Netty。

本书内容

本书的内容源于“疯狂创客圈社群”的博客，以及社群持续迭代的 CrazyIM 项目，虽然书中重在讲解 Netty、Redis、ZooKeeper 的使用方法，但是还有一个更大的价值，就是为大家打下 Java 高并发开发技术的坚实基础。

首先，本书从操作系统的底层原理开始讲解：浅显易懂地剖析高并发 IO 的底层原理，并介绍如何让单体 Java 应用支持百万级的高并发；从传统的阻塞式 OIO 开始，细致地解析 Reactor 高性能模式，介绍高性能网络开发的基础知识；从 Java 的线程 Join 和线程池开始，介绍 Java Future 和 Guava ListenableFuture 两种常用异步回调技术。这些原理方面的基础知识非常重要，是大家在日常开发 Java 后台应用时解决实际问题的金钥匙。

接着，重点讲解 Netty。这是目前当之无愧的高性能通信框架皇冠上的明珠，是支撑其他众多著名的高并发、分布式、大数据框架底层的框架。这里有两大大特色：一是从 Reactor 模式入手，以四两拨千斤的方式来学习 Netty 原理；二是通过 Netty 来解决网络编程中的重点难题，如 ProtoBuf 序列化问题、半包问题等。

然后，对 ZooKeeper 进行详细的介绍。除了全面地介绍使用 Curator API 操作 ZooKeeper 之外，还从实战的角度出发，介绍如何使用 ZooKeeper 来设计分布式 ID 生成器，并对重要的 Snowflake 算法进行详细的介绍。另外，还通过图文并茂和结合小故事的方式浅显易懂地介绍分布式锁的基本原理，并完成一个 ZooKeeper 分布式锁的小实践案例。

接下来，从实践开发层面对 Redis 进行说明，详细介绍 Redis 的 5 种数据类型、客户端操作指令、Jedis Java API。另外，还通过 spring-data-redis 来完成两种方式的数据分布式缓存，并详尽地介绍 Spring 的缓存注解以及涉及的 SpEL 表达式语言。

最后，通过 CrazyIM 项目介绍一个亿级流量的高并发 IM 系统模型。这个高并发架构的系统模型不仅仅限于 IM 系统，通过简单的调整和适配，就可以应用于当前主流的 Java 后台系统。

读者对象

- (1) 对 Java NIO、高性能 IO、高并发编程感兴趣的大专院校学生。
- (2) 需要学习 Java 高并发技术、高并发架构的初、中级 Java 工程师。
- (3) 生产项目中需要用到 Netty、Redis、ZooKeeper 三大框架的架构师或者项目人员。

本书源代码下载

本书的源代码可以从 https://gitee.com/sfasdfasdfsdf/netty_redis_zookeeper_source_code.git 下载。另外，还可以登录机械工业出版社华章公司网站（www.hzbook.com）下载，先搜索到本书，然后在页面上的“资料下载”模块下载即可。如果下载有问题，请发送电子邮件至 booksaga@126.com，邮件主题为“求 Netty、Redis、ZooKeeper 高并发实战下载资源”

勘误和支持

由于作者水平和能力有限，不妥之处在所难免，希望读者批评指正。本书的读者 QQ 群为 104131248，目前群中已经包含了不少高质量的面试题以及开发技术难题，欢迎读者入群进行交流。

致谢

写书，不仅仅是一项技术活，而且是一项工匠活，为了确保书中知识的全面性、系统性，我需要不断地思考与总结。为了保证书中的每一行程序都是正确的，我需要反复地编写 LLT 用例去进行验证。总之，一本优质的书，意味着需要牺牲陪伴家人的大量时间。感谢我的妻子、孩子们给我一贯的支持和帮助！

感谢卞诚君老师在我写书过程中给予的指导和帮助。没有他的提议，我不会想到将自己发布在“疯狂创客圈”社群中高并发方面的博客文章整理成一本书出版。另外，还让我感觉到写博客和写书不是一个层面的事情。博客里的很多内容是不全面、不严谨的，甚至是错误的。

感谢“疯狂创客圈”社群中的小伙伴们，虽然你们的很多技术难题，我不一定能给出最佳的解答方案，但正是因为一路同行，一直坦诚、纯粹的技术交流，大家相互启发了许多技术灵感，拓展了彼此的技术视野，最终提升了水平。欢迎大家来“砸”问题，也欢迎大家多多交流。

尼恩
中国武汉

目 录

前言	
第 1 章 高并发时代的必备技能	1
1.1 Netty 为何这么火	1
1.1.1 Netty 火热的程度	1
1.1.2 Netty 是面试的必杀器	2
1.2 高并发利器 Redis	2
1.2.1 什么是 Redis	2
1.2.2 Redis 成为缓存事实标准的原因	3
1.3 分布式利器 ZooKeeper	3
1.3.1 什么是 ZooKeeper	3
1.3.2 ZooKeeper 的优势	4
1.4 高并发 IM 的综合实践	4
1.4.1 高并发 IM 的学习价值	4
1.4.2 庞大的应用场景	5
1.5 Netty、Redis、ZooKeeper 实践计划	5
1.5.1 第 1 天: Java NIO 实践	5
1.5.2 第 2 天: Reactor 反应器模式实践	6
1.5.3 第 3 天: 异步回调模式实践	7
1.5.4 第 4 天: Netty 基础实践	8
1.5.5 第 5 天: 解码器 (Decoder) 与编码器 (Encoder) 实践	9
1.5.6 第 6 天: JSON 和 ProtoBuf 序列化实践	11
1.5.7 第 7~10 天: 基于 Netty 的单聊实战	12
1.5.8 第 11 天: ZooKeeper 实践计划	14
1.5.9 第 12 天: Redis 实践计划	14
1.6 本章小结	16
第 2 章 高并发 IO 的底层原理	17
2.1 IO 读写的基础原理	17
2.1.1 内核缓冲区与进程缓冲区	18
2.1.2 详解典型的系统调用流程	18
2.2 四种主要的 IO 模型	19

2.2.1	同步阻塞 IO (Blocking IO)	20
2.2.2	同步非阻塞 NIO (None Blocking IO)	21
2.2.3	IO 多路复用模型 (IO Multiplexing)	22
2.2.4	异步 IO 模型 (Asynchronous IO)	23
2.3	通过合理配置来支持百万级并发连接	24
2.4	本章小结	26
第 3 章	Java NIO 通信基础详解	27
3.1	Java NIO 简介	27
3.1.1	NIO 和 OIO 的对比	28
3.1.2	通道 (Channel)	28
3.1.3	Selector 选择器	28
3.1.4	缓冲区 (Buffer)	29
3.2	详解 NIO Buffer 类及其属性	29
3.2.1	Buffer 类	29
3.2.2	Buffer 类的重要属性	29
3.2.3	4 个属性的小结	31
3.3	详解 NIO Buffer 类的重要方法	31
3.3.1	allocate()创建缓冲区	31
3.3.2	put()写入到缓冲区	32
3.3.3	flip()翻转	33
3.3.4	get()从缓冲区读取	34
3.3.5	rewind()倒带	35
3.3.6	mark()和 reset()	37
3.3.7	clear()清空缓冲区	38
3.3.8	使用 Buffer 类的基本步骤	38
3.4	详解 NIO Channel (通道) 类	38
3.4.1	Channel (通道) 的主要类型	39
3.4.2	FileChannel 文件通道	39
3.4.3	使用 FileChannel 完成文件复制的实践案例	41
3.4.4	SocketChannel 套接字通道	42
3.4.5	使用 SocketChannel 发送文件的实践案例	44
3.4.6	DatagramChannel 数据报通道	46
3.4.7	使用 DatagramChannel 数据包通道发送数据的实践案例	47
3.5	详解 NIO Selector 选择器	49
3.5.1	选择器以及注册	49
3.5.2	SelectableChannel 可选择通道	50
3.5.3	SelectionKey 选择键	50
3.5.4	选择器使用流程	50

3.5.5 使用 NIO 实现 Discard 服务器的实践案例.....	52
3.5.6 使用 SocketChannel 在服务器端接收文件的实践案例.....	54
3.6 本章小结.....	57
第 4 章 鼎鼎大名的 Reactor 反应器模式.....	59
4.1 Reactor 反应器模式为何如此重要.....	59
4.1.1 为什么首先学习 Reactor 反应器模式.....	59
4.1.2 Reactor 反应器模式简介.....	60
4.1.3 多线程 OIO 的致命缺陷.....	60
4.2 单线程 Reactor 反应器模式.....	62
4.2.1 什么是单线程 Reactor 反应器.....	62
4.2.2 单线程 Reactor 反应器的参考代码.....	63
4.2.3 一个 Reactor 反应器版本的 EchoServer 实践案例.....	65
4.2.4 单线程 Reactor 反应器模式的缺点.....	67
4.3 多线程的 Reactor 反应器模式.....	68
4.3.1 多线程池 Reactor 反应器演进.....	68
4.3.2 多线程 Reactor 反应器的实践案例.....	68
4.3.3 多线程 Handler 处理器的实践案例.....	70
4.4 Reactor 反应器模式小结.....	72
4.5 本章小结.....	73
第 5 章 并发基础中的 Future 异步回调模式.....	74
5.1 从泡茶的案例说起.....	74
5.2 join 异步阻塞.....	75
5.2.1 线程的 join 合并流程.....	75
5.2.2 使用 join 实现异步泡茶喝的实践案例.....	75
5.2.3 详解 join 合并方法.....	77
5.3 FutureTask 异步回调之重武器.....	77
5.3.1 Callable 接口.....	77
5.3.2 初探 FutureTask 类.....	78
5.3.3 Future 接口.....	79
5.3.4 再探 FutureTask 类.....	79
5.3.5 使用 FutureTask 类实现异步泡茶喝的实践案例.....	80
5.4 Guava 的异步回调.....	82
5.4.1 详解 FutureCallback.....	82
5.4.2 详解 ListenableFuture.....	83
5.4.3 ListenableFuture 异步任务.....	84
5.4.4 使用 Guava 实现泡茶喝的实践案例.....	84
5.5 Netty 的异步回调模式.....	87

5.5.1	详解 GenericFutureListener 接口	87
5.5.2	详解 Netty 的 Future 接口	88
5.5.3	ChannelFuture 的使用	88
5.5.4	Netty 的出站和入站异步回调	89
5.6	本章小结	90
第 6 章	Netty 原理与基础	91
6.1	第一个 Netty 的实践案例 DiscardServer	91
6.1.1	创建第一个 Netty 项目	91
6.1.2	第一个 Netty 服务器端程序	92
6.1.3	业务处理器 NettyDiscardHandler	93
6.1.4	运行 NettyDiscardServer	94
6.2	解密 Netty 中的 Reactor 反应器模式	95
6.2.1	回顾 Reactor 反应器模式中 IO 事件的处理流程	95
6.2.2	Netty 中的 Channel 通道组件	96
6.2.3	Netty 中的 Reactor 反应器	96
6.2.4	Netty 中的 Handler 处理器	97
6.2.5	Netty 的流水线 (Pipeline)	98
6.3	详解 Bootstrap 启动器类	100
6.3.1	父子通道	100
6.3.2	EventLoopGroup 线程组	101
6.3.3	Bootstrap 的启动流程	101
6.3.4	ChannelOption 通道选项	104
6.4	详解 Channel 通道	105
6.4.1	Channel 通道的主要成员和方法	105
6.4.2	EmbeddedChannel 嵌入式通道	107
6.5	详解 Handler 业务处理器	108
6.5.1	ChannelInboundHandler 通道入站处理器	109
6.5.2	ChannelOutboundHandler 通道出站处理器	110
6.5.3	ChannelInitializer 通道初始化处理器	111
6.5.4	ChannelInboundHandler 的生命周期的实践案例	112
6.6	详解 Pipeline 流水线	115
6.6.1	Pipeline 入站处理流程	115
6.6.2	Pipeline 出站处理流程	116
6.6.3	ChannelHandlerContext 上下文	118
6.6.4	截断流水线的处理	118
6.6.5	Handler 业务处理器的热拔插	120
6.7	详解 ByteBuf 缓冲区	122
6.7.1	ByteBuf 的优势	122

6.7.2	ByteBuf 的逻辑部分	123
6.7.3	ByteBuf 的重要属性	123
6.7.4	ByteBuf 的三组方法	124
6.7.5	ByteBuf 基本使用的实践案例.....	125
6.7.6	ByteBuf 的引用计数	127
6.7.7	ByteBuf 的 Allocator 分配器	128
6.7.8	ByteBuf 缓冲区的类型	130
6.7.9	三类 ByteBuf 使用的实践案例.....	131
6.7.10	ByteBuf 的自动释放	133
6.8	ByteBuf 浅层复制的高级使用方式	136
6.8.1	slice 切片浅层复制.....	136
6.8.2	duplicate 整体浅层复制	137
6.8.3	浅层复制的问题.....	138
6.9	EchoServer 回显服务器的实践案例	138
6.9.1	NettyEchoServer 回显服务器的服务器端.....	138
6.9.2	共享 NettyEchoServerHandler 处理器.....	139
6.9.3	NettyEchoClient 客户端代码	140
6.9.4	NettyEchoClientHandler 处理器	142
6.10	本章小结	143
第 7 章	Decoder 与 Encoder 重要组件.....	144
7.1	Decoder 原理与实践	144
7.1.1	ByteToMessageDecoder 解码器.....	145
7.1.2	自定义 Byte2IntegerDecoder 整数解码器的实践案例.....	146
7.1.3	ReplayingDecoder 解码器	148
7.1.4	整数的分包解码器的实践案例	149
7.1.5	字符串的分包解码器的实践案例	152
7.1.6	MessageToMessageDecoder 解码器	156
7.2	开箱即用的 Netty 内置 Decoder	157
7.2.1	LineBasedFrameDecoder 解码器	157
7.2.2	DelimiterBasedFrameDecoder 解码器	158
7.2.3	LengthFieldBasedFrameDecoder 解码器	159
7.2.4	多字段 Head-Content 协议数据帧解析的实践案例.....	162
7.3	Encoder 原理与实践	164
7.3.1	MessageToByteEncoder 编码器.....	165
7.3.2	MessageToMessageEncoder 编码器	166
7.4	解码器和编码器的结合.....	167
7.4.1	ByteToMessageCodec 编解码器	168
7.4.2	CombinedChannelDuplexHandler 组合器.....	169

7.5 本章小结	169
第 8 章 JSON 和 ProtoBuf 序列化.....	171
8.1 详解粘包和拆包.....	172
8.1.1 半包问题的实践案例	172
8.1.2 什么是半包问题	174
8.1.3 半包现象的原理	174
8.2 JSON 协议通信	175
8.2.1 JSON 序列化的通用类	175
8.2.2 JSON 序列化与反序列化的实践案例.....	176
8.2.3 JSON 传输的编码器和解码器之原理.....	178
8.2.4 JSON 传输之服务器端的实践案例.....	179
8.2.5 JSON 传输之客户端的实践案例.....	180
8.3 Protobuf 协议通信.....	182
8.3.1 一个简单的 proto 文件的实践案例.....	182
8.3.2 控制台命令生成 POJO 和 Builder.....	183
8.3.3 Maven 插件生成 POJO 和 Builder	183
8.3.4 消息 POJO 和 Builder 的使用之实践案例.....	184
8.4 Protobuf 编解码的实践案例.....	187
8.4.1 Protobuf 编码器和解码器的原理	187
8.4.2 Protobuf 传输之服务器端的实践案例	188
8.4.3 Protobuf 传输之客户端的实践案例	189
8.5 详解 Protobuf 协议语法.....	191
8.5.1 proto 的头部声明.....	191
8.5.2 消息结构体与消息字段	192
8.5.3 字段的数据类型	193
8.5.4 其他的语法规范	194
8.6 本章小结	195
第 9 章 基于 Netty 的单体 IM 系统的开发实践	196
9.1 自定义 ProtoBuf 编解码器	196
9.1.1 自定义 Protobuf 编码器	197
9.1.2 自定义 Protobuf 解码器	198
9.1.3 IM 系统中 Protobuf 消息格式的设计	199
9.2 概述 IM 的登录流程.....	202
9.2.1 图解登录/响应流程的 9 个环节	203
9.2.2 客户端涉及的主要模块	203
9.2.3 服务器端涉及的主要模块	204
9.3 客户端的登录处理的实践案例.....	204

9.3.1	LoginConsoleCommand 和 User POJO.....	205
9.3.2	LoginSender 发送器	207
9.3.3	ClientSession 客户端会话	209
9.3.4	LoginResponseHandler 登录响应处理器	211
9.3.5	客户端流水线的装配	212
9.4	服务器端的登录响应的实践案例	213
9.4.1	服务器流水线的装配	214
9.4.2	LoginRequestHandler 登录请求处理器	215
9.4.3	LoginProcessor 用户验证逻辑	216
9.4.4	EventLoop 线程和业务线程相互隔离	217
9.5	详解 ServerSession 服务器会话	218
9.5.1	通道的容器属性	219
9.5.2	ServerSession 服务器端会话类	220
9.5.3	SessionMap 会话管理器	222
9.6	点对点单聊的实践案例	223
9.6.1	简述单聊的端到端流程	223
9.6.2	客户端的 ChatConsoleCommand 收集聊天内容	224
9.6.3	客户端的 CommandController 发送 POJO	224
9.6.4	服务器端的 ChatRedirectHandler 消息转发	225
9.6.5	服务器端的 ChatRedirectProcessor 异步处理	226
9.6.6	客户端的 ChatMsgHandler 接收 POJO	227
9.7	详解心跳检测	228
9.7.1	网络连接的假死现象	228
9.7.2	服务器端的空闲检测	229
9.7.3	客户端的心跳报文	230
9.8	本章小结	232
第 10 章	ZooKeeper 分布式协调	233
10.1	ZooKeeper 伪集群安装和配置	233
10.1.1	创建数据目录和日志目录:	234
10.1.2	创建 myid 文件	234
10.1.3	创建和修改配置文件	235
10.1.4	配置文件示例	237
10.1.5	启动 ZooKeeper 伪集群	238
10.2	使用 ZooKeeper 进行分布式存储	239
10.2.1	详解 ZooKeeper 存储模型	239
10.2.2	zkCli 客户端命令清单	240
10.3	ZooKeeper 应用开发的实践	241
10.3.1	ZkClient 开源客户端介绍	242

10.3.2	Curator 开源客户端介绍	242
10.3.3	Curator 开发的环境准备	243
10.3.4	Curator 客户端实例的创建	244
10.3.5	通过 Curator 创建 ZNode 节点	245
10.3.6	在 Curator 中读取节点	247
10.3.7	在 Curator 中更新节点	248
10.3.8	在 Curator 中删除节点	249
10.4	分布式命名服务的实践	251
10.4.1	ID 生成器	252
10.4.2	ZooKeeper 分布式 ID 生成器的实践案例	253
10.4.3	集群节点的命名服务之实践案例	254
10.4.4	使用 ZK 实现 SnowFlakeID 算法的实践案例	256
10.5	分布式事件监听的重点	261
10.5.1	Watcher 标准的事件处理器	261
10.5.2	NodeCache 节点缓存的监听	265
10.5.3	PathChildrenCache 子节点监听	267
10.5.4	Tree Cache 节点树缓存	272
10.6	分布式锁的原理与实践	276
10.6.1	公平锁和可重入锁的原理	276
10.6.2	ZooKeeper 分布式锁的原理	277
10.6.3	分布式锁的基本流程	279
10.6.4	加锁的实现	280
10.6.5	释放锁的实现	285
10.6.6	分布式锁的使用	287
10.6.7	Curator 的 InterProcessMutex 可重入锁	288
10.7	本章小结	289
第 11 章	分布式缓存 Redis	290
11.1	Redis 入门	290
11.1.1	Redis 安装和配置	290
11.1.2	Redis 客户端命令	292
11.1.3	Redis Key 的命名规范	294
11.2	Redis 数据类型	295
11.2.1	String 字符串	295
11.2.2	List 列表	296
11.2.3	Hash 哈希表	297
11.2.4	Set 集合	298
11.2.5	Zset 有序集合	299
11.3	Jedis 基础编程的实践案例	300

11.3.1	Jedis 操作 String 字符串	301
11.3.2	Jedis 操作 List 列表	303
11.3.3	Jedis 操作 Hash 哈希表	304
11.3.4	Jedis 操作 Set 集合	305
11.3.5	Jedis 操作 Zset 有序集合	306
11.4	JedisPool 连接池的实践案例	308
11.4.1	JedisPool 的配置	308
11.4.2	JedisPool 创建和预热	310
11.4.3	JedisPool 的使用	312
11.5	使用 spring-data-redis 完成 CRUD 的实践案例	313
11.5.1	CRUD 中应用缓存的场景	313
11.5.2	配置 spring-redis.xml	315
11.5.3	使用 RedisTemplate 模板 API	316
11.5.4	使用 RedisTemplate 模板 API 完成 CRUD 的实践案例	321
11.5.5	使用 RedisCallback 回调完成 CRUD 的实践案例	323
11.6	Spring 的 Redis 缓存注解	325
11.6.1	使用 Spring 缓存注解完成 CRUD 的实践案例	325
11.6.2	spring-redis.xml 中配置的调整	327
11.6.3	详解@CachePut 和 @Cacheable 注解	328
11.6.4	详解@CacheEvict 注解	329
11.6.5	详解@Caching 组合注解	330
11.7	详解 SpringEL (SpEL)	331
11.7.1	SpEL 运算符	332
11.7.2	缓存注解中的 SpringEL 表达式	334
11.8	本章小结	336
第 12 章	亿级高并发 IM 架构的开发实践	337
12.1	如何支撑亿级流量的高并发 IM 架构的理论基础	337
12.1.1	亿级流量的系统架构的开发实践	338
12.1.2	高并发架构的技术选型	338
12.1.3	详解 IM 消息的序列化协议选型	339
12.1.4	详解长连接和短连接	339
12.2	分布式 IM 的命名服务的实践案例	340
12.2.1	IM 节点的 POJO 类	341
12.2.2	IM 节点的 ImWorker 类	342
12.3	Worker 集群的负载均衡之实践案例	345
12.3.1	ImLoadBalance 负载均衡器	346
12.3.2	与 WebGate 的整合	348
12.4	即时通信消息的路由和转发的实践案例	349

12.4.1	IM 路由器 WorkerRouter	349
12.4.2	IM 转发器 WorkerReSender	352
12.5	Feign 短连接 RESTful 调用	354
12.5.1	短连接 API 的接口准备.....	355
12.5.2	声明远程接口的本地代理	355
12.5.3	远程 API 的本地调用.....	356
12.6	分布式的在线用户统计的实践案例.....	358
12.6.1	Curator 的分布式计数器.....	358
12.6.2	用户上线和下线的统计	360
12.7	本章小结	361

第 1 章

高并发时代的必备技能

高并发时代已然到来，Netty、Redis、ZooKeeper 是高并发时代的必备工具。

1.1 Netty 为何这么火

Netty 是 JBOSS 提供的一个 Java 开源框架，是基于 NIO 的客户端/服务器编程框架，它既能快速开发高并发、高可用、高可靠性的网络服务器程序，也能开发高可用、高可靠的客户端程序。

注：NOI 是指非阻塞输入输出（Non-Blocking IO），也称非阻塞 IO。另外，本书为了行文上的一致性，把输入输出的英文缩写统一为 IO，而不用 I/O。

1.1.1 Netty 火热的程度

Netty 已经有了成百上千的分布式中间件、各种开源项目以及各种商业项目的应用。例如火爆的 Kafka、RocketMQ 等消息中间件、火热的 Elasticsearch 开源搜索引擎、大数据处理 Hadoop 的 RPC 框架 Avro、主流的分布式通信框架 Dubbo，它们都使用了 Netty。总之，使用 Netty 开发的项目，已经有点数不过来了……

Netty 之所以受青睐，是因为 Netty 提供异步的、事件驱动的网络应用程序框架和工具。作为一个异步框架，Netty 的所有 IO 操作都是异步非阻塞的，通过 Future-Listener 机制，用户可以方便地主动获取或者通过通知机制获得 IO 操作结果。

与 JDK 原生 NIO 相比，Netty 提供了相对十分简单易用的 API，因而非常适合网络编程。Netty 主要是基于 NIO 来实现的，在 Netty 中也可以提供阻塞 IO 的服务。

Netty 之所以这么火，与它的巨大优点是密不可分的，大致可以总结如下：

- API 使用简单，开发门槛低。

- 功能强大，预置了多种编解码功能，支持多种主流协议。
- 定制能力强，可以通过 ChannelHandler 对通信框架进行灵活扩展。
- 性能高，与其他业界主流的 NIO 框架对比，Netty 的综合性能最优。
- 成熟、稳定，Netty 修复了已经发现的所有 JDK NIO 中的 BUG，业务开发人员不需要再为 NIO 的 BUG 而烦恼。
- 社区活跃，版本迭代周期短，发现的 BUG 可以被及时修复。

1.1.2 Netty 是面试的必杀器

Netty 是互联网中间件领域使用最广泛、最核心的网络通信框架之一。几乎所有互联网中间件或者大数据领域均离不开 Netty，掌握 Netty 是作为一名初中级工程师迈向高级工程师重要的技能之一。

目前来说，主要的互联网公司，例如阿里、腾讯、美团、新浪、淘宝等，在高级工程师的面试过程中，就经常会问一些高性能通信框架方面的问题，还会问一些“你有没有读过什么著名框架的源代码？”等类似的问题。

如果掌握了 Netty 相关的技术问题，更进一步说，如果你能全面地阅读和掌握 Netty 源代码，相信面试大公司时，一定底气十足，成功在握。

1.2 高并发利器 Redis

任何高并发的系统，不可或缺的就是缓存。Redis 缓存目前已经成为缓存的事实标准。

1.2.1 什么是 Redis

Redis 是 Remote Dictionary Server（远程字典服务器）的缩写，最初是作为数据库的工具来使用的。是目前使用广泛、高效的一款开源缓存。Redis 使用 C 语言开发，将数据保存在内存中，可以看成是一款纯内存的数据库，所以它的数据存取速度非常快。一些经常用并且创建时间较长的内容，可以缓存到 Redis 中，而应用程序能以极快的速度存取这些内容。举例来说，如果某个页面经常会被访问到，而创建页面时需要多次访问数据库、造成网页内容的生成时间较长，那么就可以使用 Redis 将这个页面缓存起来，从而减轻了网站的负担，降低了网站的延迟。

Redis 通过键-值对（Key-Value Pair）的形式来存储数据，类似于 Java 中的 Map 映射。Redis 的 Key 键，只能是 string 字符串类型。Redis 的 Value 值类型包括：string 字符串类型、map 映射类型、list 列表类型、set 集合类型、sortedset 有序集合类型。

Redis 的主要应用场景：缓存（数据查询、短连接、新闻内容、商品内容等）、分布式会话（Session）、聊天室的在线好友列表、任务队列（秒杀、抢购、12306 等）、应用排行榜、访问统计、数据过期处理（可以精确到毫秒）。