

39 | 案例分析（二）：高性能网络应用框架Netty

2019-05-28 王宝令

Java并发编程实战

[进入课程 >](#)



讲述：王宝令

时长 08:44 大小 8.01M



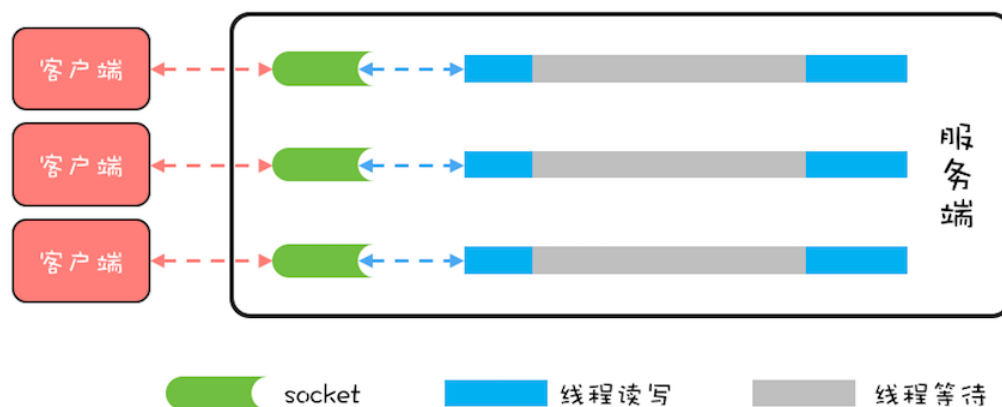
Netty 是一个高性能网络应用框架，应用非常普遍，目前在 Java 领域里，Netty 基本上成为网络程序的标配了。Netty 框架功能丰富，也非常复杂，今天我们主要分析 Netty 框架中的线程模型，而**线程模型直接影响着网络程序的性能**。

在介绍 Netty 的线程模型之前，我们首先要把问题搞清楚，了解网络编程性能的瓶颈在哪里，然后再看 Netty 的线程模型是如何解决这个问题的。

网络编程性能的瓶颈

在《[32 | Balking 模式：再谈线程安全的单例模式](#)》中，我们写过一个简单的网络程序 echo，采用的是阻塞式 I/O（BIO）。BIO 模型里，所有 read() 操作和 write() 操作都会阻塞当前线程的，如果客户端已经和服务端建立了一个连接，而迟迟不发送数据，那么服务端

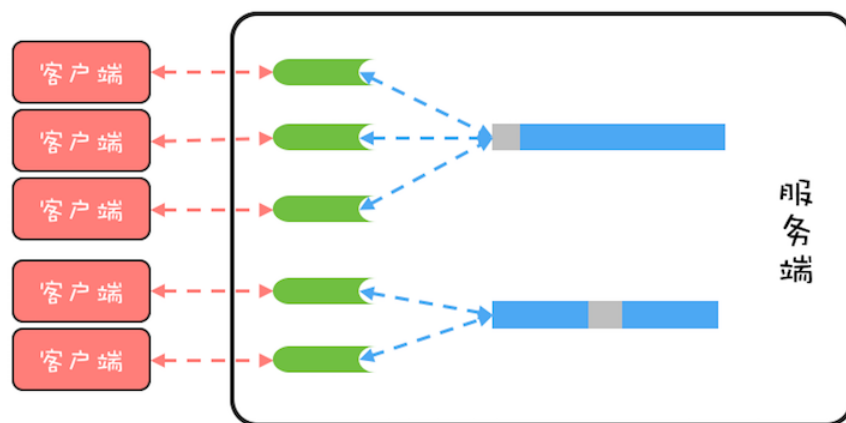
的 `read()` 操作会一直阻塞，所以使用 **BIO 模型**，一般都会为每个 **socket** 分配一个独立的**线程**，这样就不会因为线程阻塞在一个 **socket** 上而影响对其他 **socket** 的读写。BIO 的线程模型如下图所示，每一个 **socket** 都对应一个独立的线程；为了避免频繁创建、消耗线程，可以采用线程池，但是 **socket** 和线程之间的对应关系并不会变化。



BIO 的线程模型

BIO 这种线程模型适用于 **socket** 连接不是很多的场景；但是现在的互联网场景，往往需要服务器能够支撑十万甚至百万连接，而创建十万甚至上百万个线程显然并不现实，所以 BIO 线程模型无法解决百万连接的问题。如果仔细观察，你会发现互联网场景中，虽然连接多，但是每个连接上的请求并不频繁，所以线程大部分时间都在等待 I/O 就绪。也就是说线程大部分时间都阻塞在那里，这完全是浪费，如果我们能够解决这个问题，那就不需要这么多线程了。

顺着这个思路，我们可以将线程模型优化为下图这个样子，可以用一个线程来处理多个连接，这样线程的利用率就上来了，同时所需的线程数量也跟着降下来了。这个思路很好，可是使用 BIO 相关的 API 是无法实现的，这是为什么呢？因为 BIO 相关的 **socket** 读写操作都是阻塞式的，而一旦调用了阻塞式 API，在 I/O 就绪前，调用线程会一直阻塞，也就无法处理其他的 **socket** 连接了。

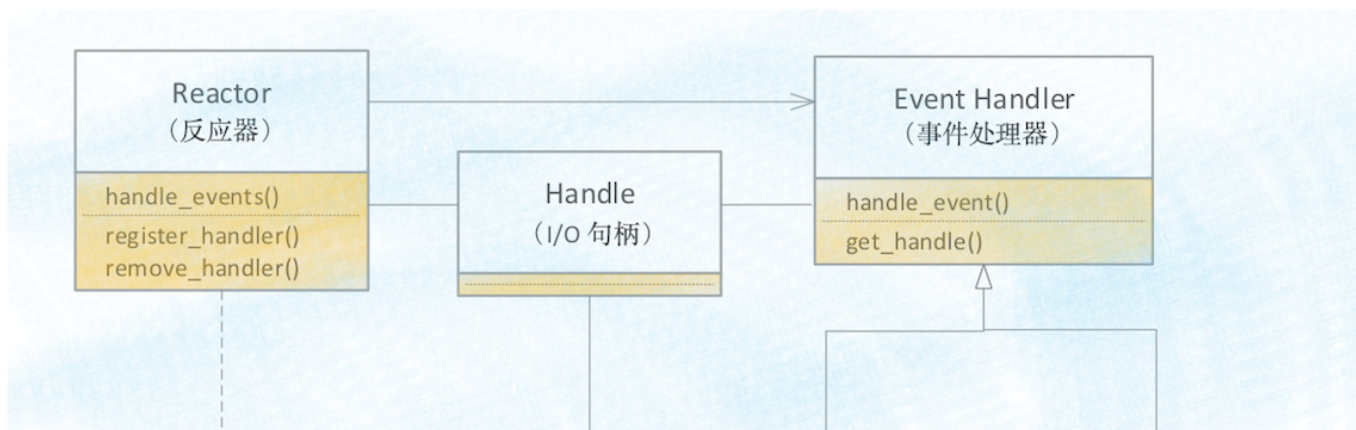


理想的线程模型图

好在 Java 里还提供了非阻塞式（NIO）API，**利用非阻塞式 API 就能够实现一个线程处理多个连接了**。那具体如何实现呢？现在普遍都是**采用 Reactor 模式**，包括 Netty 的实现。所以，要想理解 Netty 的实现，接下来我们就需要先了解一下 Reactor 模式。


Reactor 模式

下面是 Reactor 模式的类结构图，其中 Handle 指的是 I/O 句柄，在 Java 网络编程里，它本质上就是一个网络连接。Event Handler 很容易理解，就是一个事件处理器，其中 `handle_event()` 方法处理 I/O 事件，也就是每个 Event Handler 处理一个 I/O Handle；`get_handle()` 方法可以返回这个 I/O 的 Handle。Synchronous Event Demultiplexer 可以理解为操作系统提供的 I/O 多路复用 API，例如 POSIX 标准里的 `select()` 以及 Linux 里面的 `epoll()`。



Reactor 模式类结构图

Reactor 模式的核心自然是**Reactor 这个类**，其中 `register_handler()` 和 `remove_handler()` 这两个方法可以注册和删除一个事件处理器；**`handle_events()` 方式是核心**，也是 Reactor 模式的发动机，这个方法的核心逻辑如下：首先通过同步事件多路选择器提供的 `select()` 方法监听网络事件，当有网络事件就绪后，就遍历事件处理器来处理该网络事件。由于网络事件是源源不断的，所以在主程序中启动 Reactor 模式，需要以 `while(true){}` 的方式调用 `handle_events()` 方法。

 复制代码

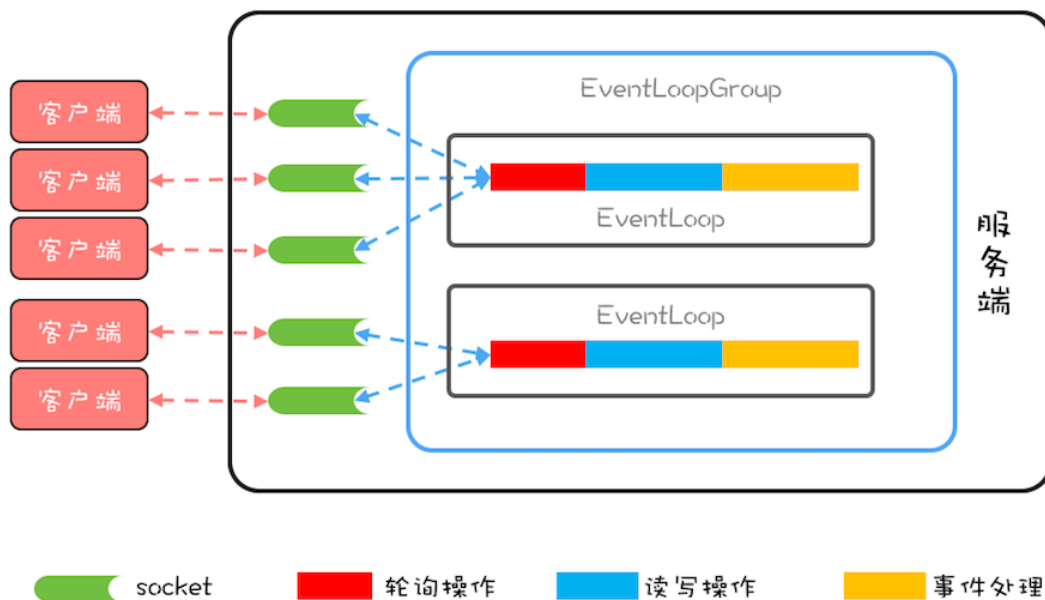
```
1 void Reactor::handle_events(){
2     // 通过同步事件多路选择器提供的
3     //select() 方法监听网络事件
4     select(handlers);
5     // 处理网络事件
6     for(h in handlers){
7         h.handle_event();
8     }
9 }
10 // 在主程序中启动事件循环
11 while (true) {
12     handle_events();
```

Netty 中的线程模型

Netty 的实现虽然参考了 Reactor 模式，但是并没有完全照搬，**Netty 中最核心的概念是事件循环（EventLoop）**，其实也就是 Reactor 模式中的 Reactor，**负责监听网络事件并调用事件处理器进行处理**。在 4.x 版本的 Netty 中，网络连接和 EventLoop 是稳定的多对 1 关系，而 EventLoop 和 Java 线程是 1 对 1 关系，这里的稳定指的是关系一旦确定就不再发生变化。也就是说一个网络连接只会对应唯一的一个 EventLoop，而一个 EventLoop 也只会对应到一个 Java 线程，所以**一个网络连接只会对应到一个 Java 线程**。

一个网络连接对应到一个 Java 线程上，有什么好处呢？最大的好处就是对于一个网络连接的事件处理是单线程的，这样就**避免了各种并发问题**。

Netty 中的线程模型可以参考下图，这个图和前面我们提到的理想的线程模型图非常相似，核心目标都是用一个线程处理多个网络连接。



Netty 中的线程模型

Netty 中还有一个核心概念是**EventLoopGroup**，顾名思义，一个 EventLoopGroup 由一组 EventLoop 组成。实际使用中，一般都会创建两个 EventLoopGroup，一个称为 bossGroup，一个称为 workerGroup。为什么会有两个 EventLoopGroup 呢？

这个和 socket 处理网络请求的机制有关，socket 处理 TCP 网络连接请求，是在一个独立的 socket 中，每当有一个 TCP 连接成功建立，都会创建一个新的 socket，之后对 TCP 连接的读写都是由新创建处理的 socket 完成的。也就是说**处理 TCP 连接请求和读写请求是**

通过两个不同的 socket 完成的。上面我们在讨论网络请求的时候，为了简化模型，只是讨论了读写请求，而没有讨论连接请求。

在 Netty 中，bossGroup 就用来处理连接请求的，而 workerGroup 是用来处理读写请求的。bossGroup 处理完连接请求后，会将这个连接提交给 workerGroup 来处理，workerGroup 里面有多个 EventLoop，那新的连接会交给哪个 EventLoop 来处理呢？这就需要有一个负载均衡算法，Netty 中目前使用的是**轮询算法**。


下面我们用 Netty 重新实现以下 echo 程序的服务端，近距离感受一下 Netty。

用 Netty 实现 Echo 程序服务端

下面的示例代码基于 Netty 实现了 echo 程序服务端：首先创建了一个事件处理器（等同于 Reactor 模式中的事件处理器），然后创建了 bossGroup 和 workerGroup，再之后创建并初始化了 ServerBootstrap，代码还是很简单的，不过有两个地方需要注意一下。

第一个，如果 NettybossGroup 只监听一个端口，那 bossGroup 只需要 1 个 EventLoop 就可以了，多了纯属浪费。

第二个，默认情况下，Netty 会创建 “2*CPU 核数” 个 EventLoop，由于网络连接与 EventLoop 有稳定的关系，所以事件处理器在处理网络事件的时候是不能有阻塞操作的，否则很容易导致请求大面积超时。如果实在无法避免使用阻塞操作，那可以通过线程池来异步处理。

 复制代码

```
1 // 事件处理器
2 final EchoServerHandler serverHandler
3     = new EchoServerHandler();
4 //boss 线程组
5 EventLoopGroup bossGroup
6     = new NioEventLoopGroup(1);
7 //worker 线程组
8 EventLoopGroup workerGroup
9     = new NioEventLoopGroup();
10 try {
11     ServerBootstrap b = new ServerBootstrap();
12     b.group(bossGroup, workerGroup)
13         .channel(NioServerSocketChannel.class)
14         .childHandler(new ChannelInitializer<SocketChannel>() {
15             @Override
16             public void initChannel(SocketChannel ch){
```

```
17     ch.pipeline().addLast(serverHandler);
18     }
19     });
20     //bind 服务端端口
21     ChannelFuture f = b.bind(9090).sync();
22     f.channel().closeFuture().sync();
23 } finally {
24     // 终止工作线程组
25     workerGroup.shutdownGracefully();
26     // 终止 boss 线程组
27     bossGroup.shutdownGracefully();
28 }
29
30 //socket 连接处理器
31 class EchoServerHandler extends
32     ChannelInboundHandlerAdapter {
33     // 处理读事件
34     @Override
35     public void channelRead(
36         ChannelHandlerContext ctx, Object msg){
37         ctx.write(msg);
38     }
39     // 处理读完成事件
40     @Override
41     public void channelReadComplete(
42         ChannelHandlerContext ctx){
43         ctx.flush();
44     }
45     // 处理异常事件
46     @Override
47     public void exceptionCaught(
48         ChannelHandlerContext ctx, Throwable cause) {
49         cause.printStackTrace();
50         ctx.close();
51     }
52 }
```

总结

Netty 是一款优秀的网络编程框架，性能非常好，为了实现高性能的目标，Netty 做了很多优化，例如优化了 ByteBuffer、支持零拷贝等等，和并发编程相关的就是它的线程模型了。Netty 的线程模型设计得很精巧，每个网络连接都关联到了一个线程上，这样做的好处是：对于一个网络连接，读写操作都是单线程执行的，从而避免了并发程序的各种问题。

你要想深入理解 Netty 的线程模型，还需要对网络相关知识有一定的理解，关于 Java IO 的演进过程，你可以参考[Scalable IO in Java](#)，至于 TCP/IP 网络编程的知识你可以参考韩国尹圣雨写的经典教程——《TCP/IP 网络编程》。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。



Java 并发编程实战

全面系统提升你的并发编程能力

王宝令
资深架构师



新版升级：点击「请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 38 | 案例分析（一）：高性能限流器Guava RateLimiter

下一篇 40 | 案例分析（三）：高性能队列Disruptor

精选留言 (12)

写留言



那只羊

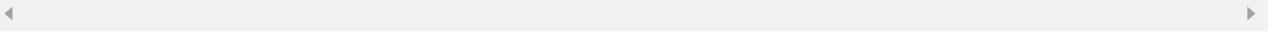
2019-05-28

4

QQ怪：Netty可以先从《Netty实战》开始，虽然翻译得一般，但是对于它的整体及各个组件你都能了解到；再就是调试源码来了解它了；最后应用到项目中去啦，比如实现一个简单的RPC，一个IM之类的

展开 ▾

作者回复: 🙏感谢回复!



QQ怪

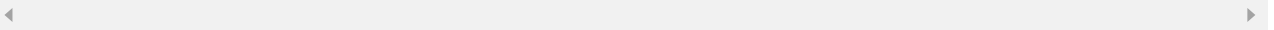
2019-05-28

👍 4

老师，学习netty除了学习老师的专栏还有什么从入门到专精的学习路线吗？

展开 ▾

作者回复: 热心同学回复了，我再加一点就是把网络编程的基础搞好



锦

2019-05-30

👍

问下老师零拷贝是怎么实现的呢？

展开 ▾



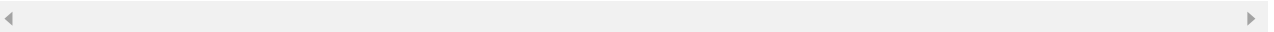
张德

2019-05-29

👍

谢谢老师讲这个reactor模式 我最近要优化的系统主体就是采用这个模式 今天看了一天都云里雾里的 看到这篇文章瞬间有了一种有章可循的感觉

作者回复: 对你有帮助就好 😊



王维

2019-05-29

👍

分享一下我之前学Netty的学习笔记，主要是源码分析：
<https://wangwei.one/tags/Netty/>



ban

👍

2019-05-28

文章说的echo那篇文章应该是
34 | Worker Thread模式：如何避免重复创建线程？
才对。

展开 ▾



张三

2019-05-28



打卡！了解皮毛是不够的。

展开 ▾



Sunqc

2019-05-28



我想知道老师后续有发布新的课程吗，喜欢你的课程

展开 ▾

作者回复: 感谢信任 😊 写不动了 😊 😊 😊



周治慧

2019-05-28



没太明白netty的线程模型，老师说一个socket对应一个Java线程，一个Java线程对应一个eventGroup，那图中不应该是一个socket对应一个eventgroup吗

展开 ▾



晓杰

2019-05-28



之前做的充电桩也是用的netty，但是只能单机部署，因为netty用的是长连接，但是在分布式框架中网络连接是随机的，请问老师这种情况怎么解决

作者回复: 没太明白你的痛点，你可以在客户端做负载均衡





苏志辉
2019-05-28



netty中eventloop是延迟创建的

展开 ▾



GeekAml
2019-05-28



netty可以开设另一门课啦

展开 ▾