链滴

Mina 架构与优化指南

作者：woyehua

原文链接：https://ld246.com/article/1542068931869

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

MINA架构

这里，我借用了一张Trustin Lee在Asia 2006的ppt里面的图片来介绍MINA的架构。

Remote Peer就是客户端，而下方的框是MINA的主要结构，各个框之间的箭头代表数据流向。

大家可以对比刚刚的例子来看这个架构图，IoService就是整个MINA的入口，负责底层的IO操作，客端发过来的消息就是由它处理。刚刚我们使用的IoAcceptor就是一个IoService，之所以抽象成IoServie，是因为MINA用同样的架构来处理服务器和客户端编程，IoService的另一个子类就是IoConnecto，用于客户端。不过根据笔者的使用经验，使用非阻塞的模型进行客户端编程非常的不方便，你最好求其他的阻塞通讯框架。

IoService把数据转化成一个一个的事件，传递给IoFilterChain。你可以加入一连串的IoFilter，进行种功能。笔者的建议是将一些功能性的，业务不相关的代码，用IoFilter来实现，使得整个应用结构更晰，也方便代码重用。

被IoFilter处理过的事件，发送给 IoHandler，然后我们在这里实现具体的业务逻辑。这个部分很简单如果你有Swing的使用经验的话，你会发现它跟Swing的事件非常相像，你要做的事情，仅仅是重载需要的方法，然后编写具体的业务功能。在这其中，最重要的一个方法就是messageReceived了。

值得留意的是一个IoSession的类，每一个IoSession实例代表这一个连接，我们需要对连接进行的任操作都通过这个类来实现。

从IoHandler通过调用IoSession.write等方法向客户端发送的消息，会通过跟输入数据相反的次序依传递，直至由IoService负责把数据发送给客户端。

这就已经是MINA的全部，是不是很简单。

接下来，我会详细介绍我们编写具体代码的时候主要涉及到的三个类，IoHandler、IoSession和IoFilter。

IoHandler

MINA的内部实现了一个事件模型，而IoHanlder则是所有事件最终产生响应的位置。每一个方法的字很明确表明该事件的含义。messageReceived是接收客户端消息的事件，我们应该在这里实现业逻辑。messageSent是服务器发送消息的事件，一般情况下我们不会使用它。sessionClosed是客户断开连接的事件，可以在这里进行一些资源回收等操作。值得留意的是，客户端连接有两个事件，sessonCreated和sessionOpened，两者稍有不同，sessionCreated是由I/O processor线程触发的，而ssionOpened在其后，由业务线程触发的，由于MINA的I/O processor线程非常少，因此如果我们的需要使用sessionCreated，也必须是耗时短的操作，一般情况下，我们应该把业务初始化的功能放sessionOpened事件中。

细心的读者可能会发现，我们刚刚的例子继承的是IoHandlerAdapter，IoHandlerAdapter其实就是个IoHanlder的空的实现，这样我们就可以不用重载不感兴趣的事件。

IoSession

IoSession是一个接口，MINA里很多的地方都使用接口，很好地体现了面向接口编程的思想。它提供对当前连接的操作功能，还有用户定义属性的存储功能，这点非常重要。IoSession是线程安全的，就是我们能够在多线程环境中随意操作IoSession，这点给开发带来很大的好处。我们来看看具体提的方法，笔者列举一些比较常用和重要的方法

在这里，笔者把IoSession的方法大致分成三类

第一类，连接操作功能。

最主要的方法有两个，向客户端发送消息和断开连接。可以看的出，write接受的变量是一个Object但是实际上应该传入什么类型呢？具体还得看你是否使用了ProtocolCodecFilter（下面会详细介绍，如果使用了ProtocolCodecFilter，那这个message将可能是一个String，或者是一个用户定义的JaaBean。默认的情况，message是一个ByteBuffer。ByteBuffer是MINA的一个类，跟java.nio.ByteBffer类同名，MINA 2.0将会将它改成IoBuffer，以避免讨论上的误会。

另一个值得留意的是Future类，MINA是一个非阻塞的通信框架，其中一个明显的体现就是调用IoSesson.write方法是不会阻塞的。用户调用了write方法之后，消息内容会发到底层等候发送，至于什么时发出，就不得而知了。当然，实际上调用了write之后，数据几乎是立刻发出的，这得益与NIO的高性。但是，如果我们必须确认了消息发出，然后进行某些处理，我们就需要使用Future类，以下是一个常见的代码。

通过调用future.join，程序就会阻塞，直至消息处理结束。我们还能通过future.isWritten得知消息否成功发送。

在这里，笔者顺便说一个实际使用的发现，消息发送是会自动合并的，简单来说，如果在很短的时间，对同一个IoSession进行了两次write操作，客户端有可能只收到一条消息，而这条消息就是服务器出的两条消息前后接起来。这样的设计可以在高并发的时候节省网络开销，而笔者的实际使用过程中效果也相当好。但是如果这样行为会导致客户端工作不正常，你也可以通过参数关闭它。

第二类，属性存储操作。

通常来说，我们的系统是有用户状态的，我们就需要在连接上存储用户属性，IoSession的Attribute是这样一个功能。例如两个连接同时连入服务器，一个连接是用户A，用户ID是13，另一个连接是用B，用户ID是14，我们就可以在用户登录成功之后，调用IoSession.setAttribute(“login_id”,13)，后在其后的操作中，通过IoSession.getAttribute(“login_id”)获得当前登录用户ID，并进行相应的作。简单来说，就是一个类似HttpSession的功能，当然具体的实现方法不一样。

第三类，连接状态。

这里就不多说了，从方法名上我们就能知道它具体的功能。

IoFilter

过滤器是MINA的一个很重要的功能。IoFilter也是一个接口，但是相对比较复杂，这里就不列举它的法了。简单来说IoFilter就像ServletFilter，在事件被IoHandler处理之前或之后进行一些特定的操作但是它比ServletFilter复杂，可以处理很多种事件，除了包括IoHandler的7个事件以外，还有一些内的事件可以进行操作。

MINA提供了一些常用的IoFilter实现，例如有LoggingFilter（日志功能）、BlacklistFilter（黑名单能）、CompressionFilter（压缩功能）、SSLFilter（SSL支持），这些过滤器比较简单，通过阅读们的源代码，能够更进一步理解过滤器的实现。笔者在这里要重点介绍两个过滤器，ProtocolCodecFiter和ExecutorFilter

ProtocolCodecFilter

网络传输的内容其实本质是一个二进制流，但是我们的业务功能不会，或者说不应该去直接操作二进流。MINA默认向IoHandler传入的message是一个ByteBuffer，如果我们直接在IoHandler操作Byteuffer，会导致大量协议分析的代码和实际的业务代码混杂在一起。最适合的做法，就是在IoFilter把ByeBuffer转换成String或者JavaBean，ProtocolCodecFilter正是这样的一个功能的过滤器。

使用ProtocolCodecFilter很简单，我们只要把ProtocolCodecFilter加入到FilterChain就可以了，但我们需要提供一个ProtocolCodecFactory。其实ProtocolCodecFilter仅仅是实现了过滤器部分的功，它会将最终的转换工作，交给从ProtocolCodecFactory获得的Encode和Decode。如果我们需要写自己的ProtocolCodec，就应该从ProtocolCodecFactory入手。MINA内置了几个ProtocolCodecactory，比较常用的就是ObjectSerializationCodecFactory和TextLineCodecFactory。

ObjectSerializationCodecFactory是Java Object序列化之后的内容直接跟ByteBuffer互相转化，比适合两端都是Java的情况使用。TextLineCodecFactory就是String跟ByteBuffer的转化，说白了就是本，例如你要实现一个SMTP服务器，或者POP服务器，就可以使用它。而笔者的实际使用，大多数况都是使用

TextLineCodecFactory

这里提及一下IoFilter的顺序问题，IoFilter是有加入顺序的，例如，先加入LoggingFilter再加入ProtoolCodecFilter，和先加入ProtocolCodecFilter再加入LoggingFilter的效果是不一样的，前者LogginFilter写入日志的内容是ByteBuffer，而后者写入日志的是转换后具体的