Mina 使用

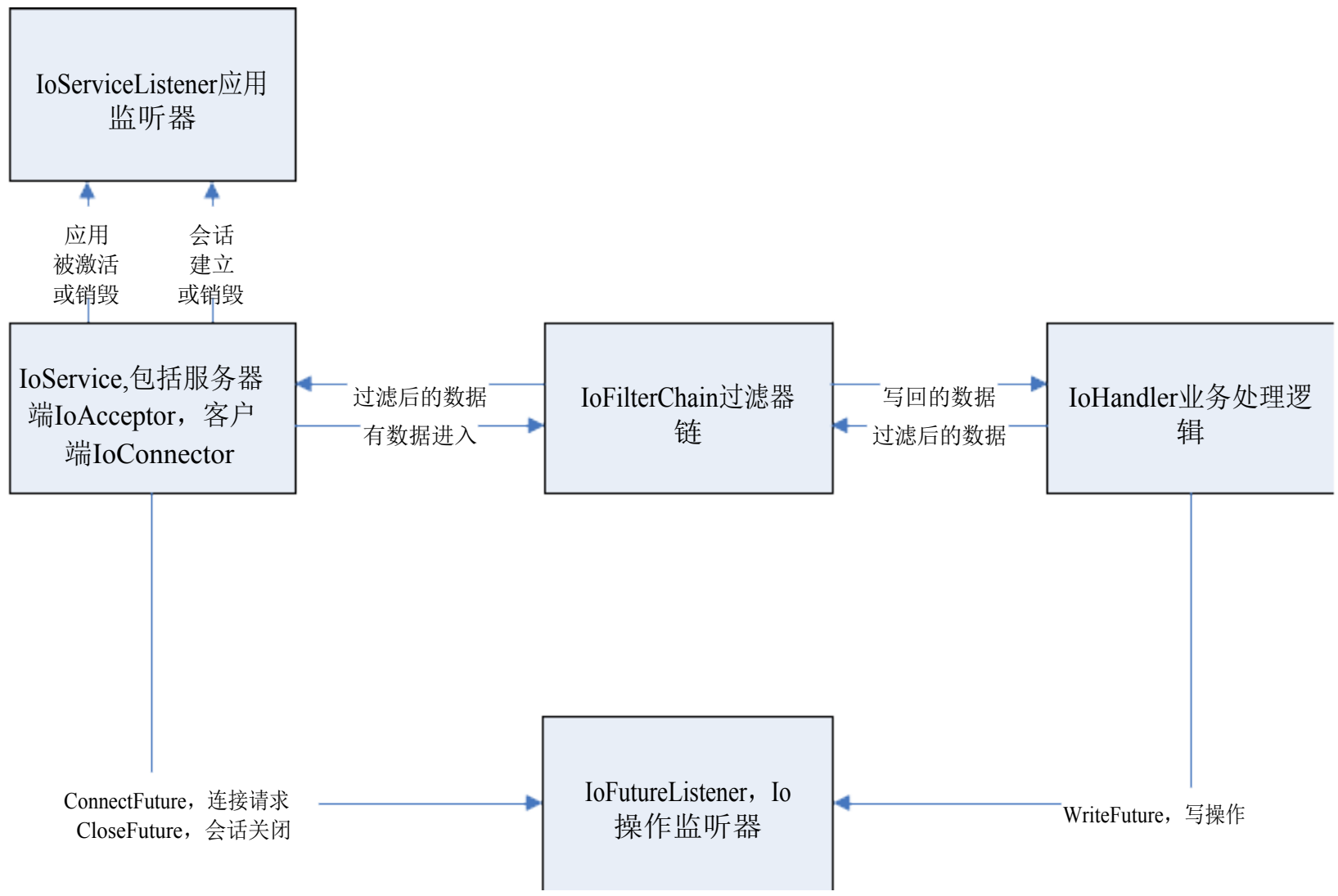
1Mina 简介：

Apache MINA(Multipurpose Infrastructure for Network Applications) 是 Apache 组织一个较新的项目，它为开发高性能和高可用性的网络应用程序提供了非常便利的框架。当前发行的 MINA 版本支持基于 Java NIO 技术的 TCP/UDP 应用程序开发、串口通讯程序（只在最新的预览版中提供），MINA 所支持的功能也在进一步的扩展中。。本文将通过官方网站上的快速入门程序 来介绍 MINA 的基础架构的同时演示如何使用 MINA 开发网络应用程序。

2 环境准备：

- 首先到官方网站下载最新的 MINA 版本，地址是：~~http://mina.apache.org/downloads.html~~。下载之前先介绍一下 MINA 的两个版本：1.0.x 适合运行环境为 JDK1.4，1.1.x 适合 JDK1.5 的版本，两者的编译环境都需要 JDK1.5。JDK1.5 已经是非常普遍了，本文中使用 1.1.5 版本的 MINA，编译和运行所需的文件是 mina-core-1.1.7.jar。
- 下载 MINA 的依赖包 slf4j。MINA 使用此项目作为日志信息的输出，而 MINA 本身并不附带此项目包，请到 ~~http://www.slf4j.org/download.html~~ 地址下载 slf4j 包，slf4j 项目解压后有很多的文件，本例中只需要其中的 slf4j-api-1.5.2.jar 和 slf4j-simple-1.5.2.jar 这两个 jar 文件。如果没有这两个文件就会导致启动例子程序的时候报 org/slf4j/LoggerFactory 类没找到的错误。当然要求机器上必须装有 1.5 或者更新版本的 JDK。
- 最好你应该选择一个顺手的 Java 开发环境例如 Eclipse 或者 NetBeans 之类的，可以更方便的编码和调试。

3MINA 基本类的描述：



3.1 IoService 应用程序入口

IoService 负责底层通讯接入，IoAcceptor（服务器端）和 IoConnector（客户端）是 IoService 的扩展接口。

备注：IoAcceptor（）可以同时启动多个端口，每个端口可以指定不同的 handler 和 filter， 但是一个服务端只有一个监听器。

IoService 中的方法	
void	**addListener**(IoServiceListener listener) Adds an IoServiceListener that listens any events related with this service. 添加监听器用来监听与这个 service 相关的所有事件。
IoServiceConfig	**getDefaultConfig**() Returns the default configuration which is used when you didn't specify any configuration. 返回这个 service 的默认配置。
DefaultIoFilterChainBuilder	**getFilterChain**() A shortcut for ((DefaultIoFilterChainBuilder) getFilterChainBuilder()).

	获取 DefaultIoFilterChainBuilder 的简便方法。
IoFilterChainBuilder	**getFilterChainBuilder**() Returns the global IoFilterChainBuilder which will modify the IoFilterChain of all IoSessions which is managed by this service. 获取一个全局的 IoFilterChainBuilder(过滤器链构建器，暂时这么翻译)，这个构建器可以修改与当前 service 相关的所有 session 的过滤器链。
Set<SocketAddress>	**getManagedServiceAddresses**() Returns all SocketAddresses this service is managing. 返回当前 service 所管理的所有的 Socket 地址
Set<IoSession>	**getManagedSessions**(SocketAddress serviceAddress) Returns all sessions with the specified remote or local address, which are currently managed by this service. 返回当前 service 管理的指定地址的所有 session 回话。
boolean	**isManaged**(SocketAddress serviceAddress) Returns true if this service is managing the specified serviceAddress. 给定一个 Socket地址，如果当前 service正在管理这个地址，则返回真。
void	**removeListener**(IoServiceListener listener) Removed an existing IoServiceListener that listens any events related with thisservice. 移除指定监听器。
void	**setFilterChainBuilder**(IoFilterChainBuilder builder) Sets the global IoFilterChainBuilder which will modify the IoFilterChain of all IoSessions which is managed by this service. 设置一个全局的 IoFilterChainBuilder(过滤器链构建器，暂时这么翻译)，这个构建器可以修改与当前 service 相关的所有 session 的过滤器链。

3.1 IoAcceptor 相当于网络应用程序中的服务器端

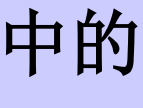
void	**bind**(SocketAddress address, IoHandler handler) 绑定服务器到指定的地址(address)，并指定处理器(handler)来处理接入的链接。
void	**bind**(SocketAddress address, IoHandler handler, IoServiceConfig config) 设置服务器的设置（config），绑定服务器到指定的地址(address)，并指定处理器(handler)来处理接入的链接。

IoSession **newSession** (SocketAddress remoteAddress, SocketAddress localAddress) (可选) 在本地(localAddress)与远程（remoteAddress）处于链接的情况下，获取一个建立在它们之间的新的会话。
void	**unbind** (SocketAddress address) 解除服务器与指定地址的绑定，并断开所有与此服务器链接的客户端。
void	**unbindAll** () 解除所有由当前 acceptor 绑定的所有地址。

程序见 4.1 附件。

3 3 IoConnector 相当于客户端

IoConnector 中的方法	
ConnectFuture **connect** (SocketAddress address, IoHandler handler) Connects to the specified address. 连接到指定地址的服务器。
ConnectFuture **connect** (SocketAddress address, IoHandler handler, IoServiceConfig config) Connects to the specified address. 连接到指定地址的服务器。
ConnectFuture **connect** (SocketAddress address, SocketAddress localAddress, IoHandler handler) Connects to the specified address. 连接到指定地址的服务器。
ConnectFuture **connect** (SocketAddress address, SocketAddress localAddress, IoHandler handler, IoServiceConfig config) Connects to the specified address. 连接到指定地址的服务器。



示例程序见附件：

3.4 IoSession 当前客户端到服务器端的一个连接实例

CloseFuture **close** () 关闭当前会话。
IoFilterChain **getFilterChain** ()

	获取当前会话的过滤器链。
IoHandler	**getHandler**() 获取当前会话的处理器。
SocketAddress	**getLocalAddress**() 获取与当前会话链接的本地地址。
SocketAddress	**getRemoteAddress**() 获取链接到当前会话的远程计算机地址。
WriteFuture	**write**(Object message) 发送指定的 message 到远程计算机。

3.4 IoHandler 业务处理逻辑

该接口有五个实现类 ChainedIoHandler, DemuxingIoHandler, IoHandlerAdapter, SingleSessionIoHandlerDelegate, StreamIoHandler。其中 ChainedIoHandler, DemuxingIoHandler, StreamIoHandler 实现了接口,并继承了 IoHandlerAdapter,IoHandlerAdapter 实现了接口的所有方法， 但是在方法中并没有做什么，我们可以继承它，根据需要有选择的重写其中的方法。

一个 IoHandler 接口具有如下一些方法：

void	**exceptionCaught**(IoSession session, Throwable cause) 当接口中其他方法抛出异常未被捕获时触发此方法
void	**messageReceived**(IoSession session, Object message) 当接收到客户端的请求信息后触发此方法.
void	**messageSent**(IoSession session, Object message) 当信息已经传送给客户端后触发此方法.
void	**sessionClosed**(IoSession session) 当连接被关闭时触发，例如客户端程序意外退出等等.
void	**sessionCreated**(IoSession session) 当一个新客户端连接后触发此方法.
void	**sessionIdle**(IoSession session, IdleStatus status) 当连接空闲时触发此方法.
void	**sessionOpened**(IoSession session) 当连接后打开时触发此方法，一般此方法与 sessionCreated 会被同时触发

StreamIoHandler 类提供了基于流的 I/O 支持，继承这个类并实现 processStreamIo(IoSession session, InputStream in, OutputStream out)方法来执行 I/O 操作：

protected abstract void	**processStreamIo**(IoSession session, InputStream in, OutputStream out) 实现这个方法来执行你的流 I/O 操作。

DemuxingIoHandler 将接收事件分离到指定的 MessageHandler 中。

<E> MessageHandler<? super E>	**addMessageHandler**(Class<E> type, 将接收事件引入到一个指定的 MessageHandler 中。
<E> MessageHandler<? super E>	**removeMessageHandler**(Class<E> type) 注销一个指定类型的 MessageHandler。

构造方法

ChainedIoHandler() Creates a new instance which contains an empty IoHandlerChain. 创建一个含有空的 IoHandlerChain 的实例。
ChainedIoHandler(IoHandlerChain chain) Creates a new instance which executes the specified IoHandlerChain on a messageReceived event. 创建一个含有指定 IoHandlerChain 的实例。

ChainedIoHandler

Method Summary

IoHandlerChain	**getChain**() Returns the IoHandlerCommand this handler will use to handle messageReceived events. 返回一个用来处理 messageReceived 事件的 IoHandlerChain
void	**messageReceived**(IoSession session, Object message)

	Handles the specified messageReceived event with the IoHandlerCommand or IoHandlerChain you specified in the constructor. 通过在构造方法中定义的 IoHandlerChain 或 IoHandlerCommand 来处理指定的 messageReceived 事件。

3.4 IoFilter 过滤器用于悬接通讯层接口与业务层接口。

IoFilter 是 MINA 核心构造之一，扮演非常重要的角色。它过滤所有的 I/O 事件和请求，这些事件和请求由 IoService 最终到达 IoHandler。

过滤器的生命周期： 一个过滤器只有当它处于过滤器链中时才会起过滤作用。

当添加一个过滤器到过滤器链时：

a. ReferenceCountingIoFilter 在第一时间调用 init()方法。

b. 调用 onPreAdd 方法来告知程序，一个过滤器将被添加到过滤器链中。

c. 当过滤器被添加到过滤器链后，所有的事件和 I/O 请求都会通过过滤器到达 IoHandler。

d. 调用 onPostAdd 方法来告知程序，一个过滤器已被添加到过滤器链中。

e. 当 onPostAdd 方法 抛出异常时，过滤器将会从过滤器链中被删除，如果这个过滤器是整个过滤器链中的最后一个，那么 ReferenceCountingIoFilter 将会调用 destory()销毁该过滤器。

当从过滤器链中移除过滤器时：

a. 调用 onPreRemove 方法来告知程序，一个过滤器将会从过滤器链中被移除。

b. 过滤器从过滤器链中被移除后后，所有的事件和 I/O 请求都不会通过该过滤器到达 IoHandler。

c. 调用 onPostRemove 方法来告知程序，一个过滤器已经从过滤器链中被移除。

d. 当这个过滤器是过滤器链中的最后一个过滤器时，那么 ReferenceCountingIoFilter 将会调用 destory()销毁该过滤器。

嵌套类	
static interface	**IoFilter.NextFilter** Represents the next IoFilter in IoFilterChain. 过滤器链中的下个过滤器。

Static class	**IoFilter.WriteRequest** Represents write request fired by IoSession.write(Object). 由 IoSession.write(Object)发出的写请求。

方法	
void	**destroy**() 销毁此过滤器。由 ReferenceCountingIoFilter 调用
void	**exceptionCaught**(IoFilter.NextFilter nextFilter, IoSession session, Throwable cause) 过滤 IoHandler 中的 exceptionCaught 事件。
void	**filterClose**(IoFilter.NextFilter nextFilter, IoSession session) 过滤 IoSession.close()方法。
void	**filterWrite**(IoFilter.NextFilter nextFilter, IoSession session, IoFilter.WriteRequest 过滤 IoSession.write(Object)方法。
void	**init**() 当过滤器被添加到过滤器链中后调用该方法，这样可以在第一时间为其分配资源。由 ReferenceCountingIoFilter 调用
void	**messageReceived**(IoFilter.NextFilter nextFilter, IoSession session, Object message) 过滤 Iohandler 中的 messageReceived 事件。
void	**messageSent**(IoFilter.NextFilter nextFilter, IoSession session, Object message) 过滤 Iohandler 中的 messageSent 事件。
void	**onPostAdd**(IoFilterChain parent, String name, IoFilter.NextFilter nextFilter) 在这个过滤器被添加到过滤器链之后调用此方法。
void	**onPostRemove**(IoFilterChain parent, String name, IoFilter.NextFilter nextFilter) 在这个过滤器从过滤器链中被删除之后调用此方法。
void	**onPreAdd**(IoFilterChain parent, String name, IoFilter.NextFilter nextFilter) 在这个过滤器被添加到过滤器链之前调用此方法。
void	**onPreRemove**(IoFilterChain parent, String name, IoFilter.NextFilter nextFilter) 在这个过滤器从过滤器链中被删除之前调用此方法。
void	**sessionClosed**(IoFilter.NextFilter nextFilter, IoSession session) 过滤 IoHandler 中的 **sessionClosed** 事件。
void	**sessionCreated**(IoFilter.NextFilter nextFilter, IoSession session) 过滤 IoHandler 中的 sessionCreated 事件。
void	**sessionIdle**(IoFilter.NextFilter nextFilter, IoSession session, IdleStatus status) 过滤 IoHandler 中的 sessionIdle 事件。
void	**sessionOpened**(IoFilter.NextFilter nextFilter, IoSession session) 过滤 IoHandler 中的 sessionOpened 事件。

MINA 自身带有一些常用的过滤器，例如 codec（字符编号）、LoggingFilter（日志记录）、BlackListFilter（黑名单过滤）、CompressionFilter（压缩）、SSLFilter（SSL 加密）等

- **IoFilterAdapter(过滤器适配器)，这个类只实现了 IoFilter 接口中的方法，但是方法中并没有做任何事。**
- **BlackListFilter(黑名单过滤器)是 IoFileter 的一个实现类，作用是将远程客户端添加到黑名单中后，该客户端就会访问不到服务器。**

Method Summary	
void	**block**(InetAddress address) 将指定地址的计算机添加黑名单中。
void	**block**(InetAddress address, String error_string) 将指定地址的计算机添加黑名单中。
void	**setBlacklist**(Collection<InetAddress> addresses) 将多个指定地址添加到黑名单中。
void	**setBlacklist**(InetAddress addresses) 将多个指定地址添加到黑名单中。
void	**unblock**(InetAddress address) 将指定地址的计算机添加黑名单中

程序见附件：

- **LoggingFilter(日志过滤器)，将此过滤器加到过滤器链中后就可以实现 MINA 的日志功能。**程序见附件：

- **ProtocolCodecFilter(协议编解码过滤器)，通过 ProtocolCodecFactory,ProtocolEncoder, 或 ProtocolDecoder 该过滤器可以实现普通的二进制或特殊的协议数据与 POJO 之间的相互转换。**程序见附件：

构造方法
ProtocolCodecFilter(Class<? extends ProtocolEncoder> encoderClass, Class<?

extends ProtocolDecoder> decoderClass)
ProtocolCodecFilter(ProtocolCodecFactory factory)
ProtocolCodecFilter(ProtocolEncoder encoder, ProtocolDecoder decoder)

- **协议编码器（ProtocolEncoder）将高级的信息对象编码成二进制或特殊的协议数据。MINA 会对位于 IoSession 写队列中的所有消息调用 encode 函数，然后编码器会将经过编码处理过的消息放到 ByteBuffer 中，通过 ProtocolEncoderOutput 函数送出，到达过滤器链中的下个过滤器。**

ProtocolEncoder 中的方法

void	**dispose**(IoSession session) 释放所有与这个编码器相关的资源。
void	**encode**(IoSession session, Object message, ProtocolEncoderOutput out) 将高级的消息对象编码成二进制或特殊的协议数据

ProtocolEncoder 的实现类

NettyEncoder	该编码器可以将 Netty2 信息编码成字节 buffers
ObjectSerializationEncoder	该编码器通过 ByteBuffer.putObject(Object)可以编码普通的 java 类

新增方法

int	**getMaxObjectSize**() Returns the allowed maximum size of the encoded object. 返回允许被编码的 Object 的最大限制
void	**setMaxObjectSize**(int axObjectSize) Sets the allowed maximum size of the encoded object. 设置允许被编码的 Object 的最大限制

ProtocolEncoderAdapter	ProtoclEncoder 适配器。
SynchronizedProtocolEncoder,	线程方面的。
TextLineEncoder	该编码器可以将字符串按分隔符编码成一行一行的文本。

构造方法

TextLineEncoder()