MINA v2.0 Quick Start Guide

Introduction

This tutorial will walk you through the process of building a MINA based program. This tutorial will walk through building a time server. The following prerequisites are required for this tutorial:

- MINA 2.x Core
- JDK 1.5 or greater
- SLF4J 1.3.0 or greater
    - **Log4J 1.2** users: `slf4j-api.jar`, `slf4j-log4j12.jar`, and Log4J 1.2.x
    - **Log4J 1.3** users: `slf4j-api.jar`, `slf4j-log4j13.jar`, and Log4J 1.3.x
    - **`java.util.logging`** users: `slf4j-api.jar` and `slf4j-jdk14.jar`
    - **IMPORTANT**: Please make sure you are using the right `slf4j-*.jar` that matches to your logging framework. For instance, `slf4j-log4j12.jar` and `log4j-1.3.x.jar` can not be used together, and will malfunction.

I have tested this program on both Windows© 2000 professional and linux. If you have any problems getting this program to work, please do not hesitate to contact us in order to talk to the MINA developers. Also, this tutorial has tried to remain independent of development environments (IDE, editors.. etc). This tutorial will work with any environment that you are comfortable with. Compilation commands and steps to execute the program have been removed for brevity. If you need help learning how to either compile or execute java programs, please consult the Java tutorial.

Writing the MINA time server

We will begin by creating a file called MinaTimeServer.java. The initial code can be found below:

```
public class MinaTimeServer {

    public static void main(String[] args) {
            // code will go here next
    }
}
```

This code should be straightforward to all. We are simply defining a main method that will be used to kick off the program. At this point, we will begin to add the code that will make up our server. First off, we need an object that will be used to listen for incoming connections. Since this program will be TCP /IP based, we will add a `SocketAcceptor` to our program.

```
import org.apache.mina.core.service.IoAcceptor;
import org.apache.mina.transport.socket.nio.NioSocketAcceptor;

public class MinaTimeServer
{

    public static void main( String[] args )
    {
        IoAcceptor acceptor = new NioSocketAcceptor();
    }

}
```

With the `NioSocketAcceptor` class in place, we can go ahead and define the handler class and bind the `NioSocketAcceptor` to a port.

Next we add a filter to the configuration. This filter will log all information such as newly created sessions, messages received, messages sent, session closed. The next filter is a `ProtocolCodecFilter`. This filter will translate binary or protocol specific data into message object and vice versa. We use an existing TextLine factory because it will handle text base message for you (you don't have to write the codec part)

```
import java.nio.charset.Charset;

import org.apache.mina.core.service.IoAcceptor;
import org.apache.mina.filter.codec.ProtocolCodecFilter;
import org.apache.mina.filter.codec.textline.TextLineCodecFactory;
import org.apache.mina.filter.logging.LoggingFilter;
import org.apache.mina.transport.socket.nio.NioSocketAcceptor;

public class MinaTimeServer
{
    public static void main( String[] args )
    {
        IoAcceptor acceptor = new NioSocketAcceptor();

        acceptor.getFilterChain().addLast( "logger", new LoggingFilter() );
        acceptor.getFilterChain().addLast( "codec", new ProtocolCodecFilter( new TextLineCodecFactory( Charset.
forName( "UTF-8" ))));
    }
}
```

At this point, we will define the handler that will be used to service client connections and the requests for the current time. The handler class is a class that must implement the interface `IoHandler`. For almost all programs that use MINA, this becomes the workhorse of the program, as it services all incoming requests from the clients. For this tutorial, we will extend the class `IoHandlerAdapter`. This is a class that follows the adapter design pattern which simplifies the amount of code that needs to be written in order to satisfy the requirement of passing in a class that implements the `IoHandler` interface.

```
import java.io.IOException;
import java.nio.charset.Charset;

import org.apache.mina.core.service.IoAcceptor;
import org.apache.mina.filter.codec.ProtocolCodecFilter;
import org.apache.mina.filter.codec.textline.TextLineCodecFactory;
import org.apache.mina.filter.logging.LoggingFilter;
import org.apache.mina.transport.socket.nio.NioSocketAcceptor;

public class MinaTimeServer
{
    public static void main( String[] args ) throws IOException
    {
        IoAcceptor acceptor = new NioSocketAcceptor();

        acceptor.getFilterChain().addLast( "logger", new LoggingFilter() );
        acceptor.getFilterChain().addLast( "codec", new ProtocolCodecFilter( new TextLineCodecFactory( Charset.
forName( "UTF-8" ))));

        acceptor.setHandler(  new TimeServerHandler() );
    }
}
```

We will now add in the `NioSocketAcceptor` configuration. This will allow us to make socket-specific settings for the socket that will be used to accept connections from clients.

```
import java.io.IOException;
import java.nio.charset.Charset;

import org.apache.mina.core.session.IdleStatus;
import org.apache.mina.core.service.IoAcceptor;
import org.apache.mina.filter.codec.ProtocolCodecFilter;
import org.apache.mina.filter.codec.textline.TextLineCodecFactory;
import org.apache.mina.filter.logging.LoggingFilter;
import org.apache.mina.transport.socket.nio.NioSocketAcceptor;

public class MinaTimeServer
{
    public static void main( String[] args ) throws IOException
    {
        IoAcceptor acceptor = new NioSocketAcceptor();

        acceptor.getFilterChain().addLast( "logger", new LoggingFilter() );
        acceptor.getFilterChain().addLast( "codec", new ProtocolCodecFilter( new TextLineCodecFactory( Charset.
forName( "UTF-8" ))));

        acceptor.setHandler(  new TimeServerHandler() );

        acceptor.getSessionConfig().setReadBufferSize( 2048 );
        acceptor.getSessionConfig().setIdleTime( IdleStatus.BOTH_IDLE, 10 );
    }
}
```

There are 2 new lines in the MinaTimeServer class. These methods set the set the `IoHandler`, input buffer size and the idle property for the sessions. The buffer size will be specified in order to tell the underlying operating system how much room to allocate for incoming data. The second line will specify when to check for idle sessions. In the call to setIdleTime, the first parameter defines what actions to check for when determining if a session is idle, the second parameter defines the length of time in seconds that must occur before a session is deemed to be idle.

The code for the handler is shown below:

```
import java.util.Date;

import org.apache.mina.core.session.IdleStatus;
import org.apache.mina.core.service.IoHandlerAdapter;
import org.apache.mina.core.session.IoSession;

public class TimeServerHandler extends IoHandlerAdapter
{
    @Override
    public void exceptionCaught( IoSession session, Throwable cause ) throws Exception
    {
        cause.printStackTrace();
    }

    @Override
    public void messageReceived( IoSession session, Object message ) throws Exception
    {
        String str = message.toString();
        if( str.trim().equalsIgnoreCase("quit") ) {
            session.close();
            return;
        }

        Date date = new Date();
        session.write( date.toString() );
        System.out.println("Message written...");
    }

    @Override
    public void sessionIdle( IoSession session, IdleStatus status ) throws Exception
    {
        System.out.println( "IDLE " + session.getIdleCount( status ));
    }
}
```

The methods used in this class are **exceptionCaught**, **messageReceived** and **sessionIdle**. **exceptionCaught** should always be defined in a handler to process and exceptions that are raised in the normal course of handling remote connections. If this method is not defined, exceptions may not get properly reported.

The **exceptionCaught** method will simply print the stack trace of the error and close the session. For most programs, this will be standard practice unless the handler can recover from the exception condition.

The **messageReceived** method will receive the data from the client and write back to the client the current time. If the message received from the client is the word "quit", then the session will be closed. This method will also print out the current time to the client. Depending on the protocol codec that you use, the object (second parameter) that gets passed in to this method will be different, as well as the object that you pass in to the session.write(Object) method. If you do not specify a protocol codec, you will most likely receive a `IoBuffer` object, and be required to write out a `IoBuffer` object.

The **sessionIdle** method will be called once a session has remained idle for the amount of time specified in the call *acceptor.getSessionConfig(). setIdleTime(IdleStatus.BOTH_IDLE, 10);*.

All that is left to do is define the socket address that the server will listen on, and actually make the call that will start the server. That code is shown below:

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.charset.Charset;

import org.apache.mina.core.service.IoAcceptor;
import org.apache.mina.core.session.IdleStatus;
import org.apache.mina.filter.codec.ProtocolCodecFilter;
import org.apache.mina.filter.codec.textline.TextLineCodecFactory;
import org.apache.mina.filter.logging.LoggingFilter;
import org.apache.mina.transport.socket.nio.NioSocketAcceptor;

public class MinaTimeServer
{
    private static final int PORT = 9123;

    public static void main( String[] args ) throws IOException
    {
        IoAcceptor acceptor = new NioSocketAcceptor();

        acceptor.getFilterChain().addLast( "logger", new LoggingFilter() );
        acceptor.getFilterChain().addLast( "codec", new ProtocolCodecFilter( new TextLineCodecFactory( Charset.
forName( "UTF-8" ))));

        acceptor.setHandler( new TimeServerHandler() );
        acceptor.getSessionConfig().setReadBufferSize( 2048 );
        acceptor.getSessionConfig().setIdleTime( IdleStatus.BOTH_IDLE, 10 );
        acceptor.bind( new InetSocketAddress(PORT) );
    }
}
```

As you see, there is a call to *acceptor.setLocalAddress(new InetSocketAddress(PORT));*. This method defines what host and port this server will listen on. The final method is a call to `IoAcceptor.bind()`. This method will bind to the specified port and start processing of remote clients.

Try out the Time server

At this point, we can go ahead and compile the program. Once you have compiled the program you can run the program in order to test out what happens. The easiest way to test the program is to start the program, and then telnet in to the program:

Client Output	Server Output
user@myhost:~> telnet 127.0.0.1 9123 Trying 127.0.0.1... Connected to 127.0.0.1. Escape character is '^]'. hello Wed Oct 17 23:23:36 EDT 2007 quit Connection closed by foreign host. user@myhost:~>	MINA Time server started. Message written...

What's Next?

Please visit our Documentation page to find out more resources. You can also keep reading other tutorials.