Integrating with Spring

Integrating with Spring

This article demonstrates integrating MINA application with Spring. I wrote this article on my blog, and though to put it here, where this information actually belongs to. Can find the original copy at Integrating Apache MINA with Spring.

Application Structure

We shall take a standard MINA application which has following construct

- One Handler
- Two Filter - Logging Filter and a ProtocolCodec Filter
- NioDatagram Socket

Initialization Code

Lets see the code first. For simplicity we have omitted the glue code.

```
public void initialize() throws IOException {

        // Create an Acceptor
        NioDatagramAcceptor acceptor = new NioDatagramAcceptor();

        // Add Handler
        acceptor.setHandler(new ServerHandler());

        acceptor.getFilterChain().addLast("logging",
                                new LoggingFilter());
        acceptor.getFilterChain().addLast("codec",
                                new ProtocolCodecFilter(new SNMPCodecFactory()));

        // Create Session Configuration
        DatagramSessionConfig dcfg = acceptor.getSessionConfig();
        dcfg.setReuseAddress(true);
        logger.debug("Starting Server......");
        // Bind and be ready to listen
        acceptor.bind(new InetSocketAddress(DEFAULT_PORT));
        logger.debug("Server listening on "+DEFAULT_PORT);
}
```

Integration Process

To integrate with Spring, we need to do following:

- Set the IO handler
- Create the Filters and add to the chain
- Create the Socket and set Socket Parameters

NOTE: The latest MINA releases doesn't have the package specific to Spring, like its earlier versions. The package is now named Integration Beans, to make the implementation work for all DI frameworks.

Lets see the Spring xml file. Please see that I have removed generic part from xml and have put only the specific things needed to pull up the implementation.
This example has been derived from Chat example shipped with MINA release. Please refer the xml shipped with chat example.

Now lets pull things together

Lets set the IO Handler in the spring context file

```
<!-- The IoHandler implementation -->
  <bean id="trapHandler" class="com.ashishpaliwal.udp.mina.server.ServerHandler" />
```

Lets create the Filter chain

```
<bean id="snmpCodecFilter" class="org.apache.mina.filter.codec.ProtocolCodecFilter">
    <constructor-arg>
      <bean class="com.ashishpaliwal.udp.mina.snmp.SNMPCodecFactory" />
    </constructor-arg>
  </bean>

  <bean id="loggingFilter" class="org.apache.mina.filter.logging.LoggingFilter" />

  <!-- The filter chain. -->
  <bean id="filterChainBuilder" class="org.apache.mina.core.filterchain.DefaultIoFilterChainBuilder">
    <property name="filters">
      <map>
        <entry key="loggingFilter" value-ref="loggingFilter"/>
        <entry key="codecFilter" value-ref="snmpCodecFilter"/>
      </map>
    </property>
  </bean>
```

Here, we create instance of our IoFilter. See that for the ProtocolCodec factory, we have used Constructor injection. Logging Filter creation is straight forward. Once we have defined the beans for the filters to be used, we now create the Filter Chain to be used for the implementation. We define a bean with id "FilterChainBuidler" and add the defined filters to it. We are almost ready, and we just need to create the Socket and call bind

Lets complete the last part of creating the Socket and completing the chain

```
<bean class="org.springframework.beans.factory.config.CustomEditorConfigurer">
    <property name="customEditors">
      <map>
        <entry key="java.net.SocketAddress">
          <bean class="org.apache.mina.integration.beans.InetSocketAddressEditor" />
        </entry>
      </map>
    </property>
 </bean>

<!-- The IoAcceptor which binds to port 161 -->
<bean id="ioAcceptor" class="org.apache.mina.transport.socket.nio.NioDatagramAcceptor" init-method="bind"
destroy-method="unbind">
    <property name="defaultLocalAddress" value=":161" />
    <property name="handler" ref="trapHandler" />
    <property name="filterChainBuilder" ref="filterChainBuilder" />
</bean>
```

Now we create our ioAcceptor, set IO handler and Filter Chain. Now we have to write a function to read this file using Spring and start our application. Here's the code

```
public void initializeViaSpring() throws Exception {
    new ClassPathXmlApplicationContext("trapReceiverContext.xml");
}
```

We just call this method from main, and this shall initialize our MINA application.