2009

MINA 框架新人指南

支付宝（中国）网络技术有限公司

蓝秋鹏(仲景)

基本信息及修订记录				
创建时间：2009-08-05				
版本	时间	人物	事件	备注
1.0	2009-08-05	仲景	创建	
1.1	2009-10-27	仲景	新增“进阶”一节	

目 录

1. 前 言

在资金线混有两个技术框架是一定要懂得如何使用的，它们就是 MINA 和 HTTPCLIENT（还没听过的，就先百度一下）。支付宝和银行前置机之间的通讯基本都是使用者两种框架，即使你不懂 SOCKET 或者 HTTP 很底层的协议也没多大关系。

对于 HTTPCLIENT 有过很多实践了，从我刚进支付宝的第一个项目个人诚信通，到最近的建行境外收单直连改造，都是使用 HTTP 通讯。

突然间要做建行银行卡代扣，需要使用 SOCKET 跟建行通讯，对 SOCKET 一点都不懂，只是听大禹和香石说使用 MINA 很简单，闭关修炼了两天确实很简单， 哈哈。先找到 MINA 的主页，肯定有很多资料可以学习的， 然后再编写 DEMO，跑起来，改改 DEMO，添加一些个性化的功能，两天时间基本上就算入门啦，可以开始进入项目编码啦。这些时间对于开发工程师来说，就是项目的技术准备阶段，在 BRD 或者 PRD 阶段投入的资源，让开发人员在开发阶段不再担心技术问题，可以快速编码，甚至提前完成开发任务。

跟 MINA 类似的框架，还有著名的 Jboss Netty，代码框架非常类似，使用方法也大同小异，可以说 Jboss Netty 是 MINA 的改良版，但是目前我们系统还没有 Jboss Netty 的实际应用，不考虑引入到实际项目中。

本文档作为 MINA 框架的入门手册，浅显易懂，没有高级应用，比如配置线程模型，就可以满足日常开发。

2.实 践

跟银行的通讯，首先我们要定位自己的角色，我们扮演的是客户端，还是服务端，大部分场景我们都是扮演客户端的角色，如果银行有通知接口，在业务做完时银行会主动通知我们，推进我们的业务，这时我们就是服务端。

2.1. 服务端演示

通过 DEMO 实现一个简单的服务端，MINA 框架已经做了很好的封装，我们只需要编写几行代码，通过 main 启动 MinaServer，监听端口 1314：

```
public static void main(String[] args) throws IOException {
    IoAcceptor acceptor = new SocketAcceptor();
    IoFilter filter = new ProtocolCodecFilter(new TextLineCodecFactory());
    acceptor.getFilterChain().addLast("ZhongJing", filter);
    acceptor.bind(new InetSocketAddress(1314), new ServerHandler());
}
```

SOCKET 接口是 TCP/IP 网络的 API，不说这些很抽象的概念，我们可以直接使用 MINA 封装的 SocketAcceptor 。 main 方法中定义了整型的端口 1314，在实际应用中一般通过 antx.properties 配置端口，这样就可以把配置的权限交给运维部门。

每一个信息都会通过在 IoAcceptor 中定义的过滤器链的所有过滤器，完成个性化的日志记录和解码工作。日志过滤器用 SL4J 库记录信息，而编码过滤器则解码所有收到的信息.

这个简单的例子，我们使用 new TextLineCodecFactory() 发送的信息进行编码，这是 MINA 自带的，功能有限，只能处理文本或者 String 类型。

还有一个参数 new ServerHandler()，传的是实现 IoHandler 接口的类，对客户端的请求要进行什么业务处理，都在这里实现。

主要参考服务端处理器的实现：

```
public class ServerHandler extends IoHandlerAdapter {
    public void messageReceived(IoSession session, Object message) {
        System.out.println("收到客户端消息: " + message.toString());
        String str = "客户端IP: " + session.getRemoteAddress();
        session.write(str).join();
    }
    public void exceptionCaught(IoSession session, Throwable cause) {
        System.out.println("我是服务器，系统出现异常\n" + cause);
    }
}
```

messageReceived(...)，对接收到的消息（已经解码）进行下一步处理，DEMO 中演示的就是收到客户端的请求消息之后，都返回客户端的 IP 地址。

exceptionCaught(...)，自定义异常处理， 要不然异常会被“吃掉”。

这是服务端处理器比较重要的两个接口，还有其他一些辅助的接口，对于需要比较详细的日志记录时可以使用。

命令行：**telnet 127.0.0.1 1314** （这种方法经常用来测试我们跟银行建立的专线是否畅通，而不是使用 ping 命令）

建立连接以后，输入 hello，服务端就会有相应。测试演示图：

2.2. 客户端演示

我们使用的是 telnet 做客户端发包给服务器， 现在我们使用 MINA 框架编写一个简单的客户端，你会发现客户端和服务器的代码非常类似。

通过 main 方法启动客户端：

```
public static void main(String[] args) {
    SocketConnector connector = new SocketConnector();
    IoFilter filter = new ProtocolCodecFilter(new TextLineCodecFactory());
    connector.getFilterChain().addLast("ZhongJing", filter);
    SocketAddress soketAddress = new InetSocketAddress("127.0.0.1", 1314);
    ConnectFuture future
                    = connector.connect(soketAddress, new ClientHandler());
    future.join();
    if (!future.isConnected()) {
        System.out.println("连接失败");
        return;
    }
    future.getSession().write("客户端到此一游").join();
}
```

客户端演示的是直接发送 String 消息，可以使用 MINA 自带的编码协议，客户端处理器和服务端处理器是很像的，它们都是在收到数据包之后的处理。

主要参考客户端处理器的实现：

```
public class ClientHandler extends IoHandlerAdapter {
    public void messageReceived(IoSession session, Object message) {
        System.out.println("我是客户端，收到响应：" + message.toString());
    }
    public void messageSent(IoSession session, Object message) {
        System.out.println("我是客户端，我发送的消息：" + message);
    }
    public void exceptionCaught(IoSession session, Throwable cause) {
        System.out.println("我是客户端，系统出现异常\n" + cause);
    }
}
```

这里比服务端处理器新增一个接口 `messageSent(...)`，在实际应用中我们经常需要记录我们发送给银行的日志，可以作为一种交易凭证，方便后期问题追踪。日志的记录需要有唯一性的标记，最好的办法就是把报文的标记在发送之前保存在 `IoSession`，使用 `session.getAttribute("KEY")` 取出标记。

客户端效果演示（因为没有配置 `log4j`，会抛出红色警告）：

服务端效果演示：

3. 进 阶

在项目开发的实际应用中，我们跟银行的通讯报文一般都不会使用 String 传输，因为系统的运行环境或者编程语言一般都是不同的，最常用的就是以字节传输，报文的格式需要双方约定规范，比如 xml 格式，或者定长格式，或者 8583 格式，最终都会转换成字节传输给银行。那么这里就需要我们编写自定义的编码和解码方案。

编写编码方案工厂类，注册自定义的编码方案：

```
public class ProtocolCodecFactory extends DemuxingProtocolCodecFactory {
    public ProtocolCodecFactory(ResponseDecoder decoder) {
        super.register(decoder);
    }
    public ProtocolCodecFactory() {
        super.register(ResponseDecoder.class);
    }
}
```

这里提供了两种构造函数，对于启动 main 方式的测试，选择空构造行数就可以了；带参构造函数是为了兼容在实际应用中 Spring 框架，在 bean 声明中采用构造函数的方式注入。

```
<bean id="protocolCodecFactory"class="test.ProtocolCodecFactory">
    <constructor-arg index="0">
        <ref bean="responseDecoder" />
    </constructor-arg>
</bean>
<bean id="responseDecoder" class="test.ResponseDecoder" />
```

工厂类的使用方法，在客户端和服务器的 main 启动程序，修改 MINA 框架自带的 String 编码方案工厂类：

```
IoFilter filter = new ProtocolCodecFilter(new ProtocolCodecFactory());
```

自定义的编码方案实现类：

```
public class ResponseDecoder extends MessageDecoderAdapter {
    public MessageDecoderResult decodable(IoSession session,
                                                    ByteBuffer in) {
        return MessageDecoderResult.OK;
    }

    public MessageDecoderResult decode(IoSession session, ByteBuffer in,
                            ProtocolDecoderOutput out) throws Exception {
        CharsetDecoder decoder = Charset.defaultCharset().newDecoder();
        String responseStr = in.getString(decoder);
        if (StringUtil.isBlank(responseStr)) {
           return MessageDecoderResult.NEED_DATA;
        }
        out.write(responseStr);
        return MessageDecoderResult.OK;
    }
}
```

这里处理很简单，就是把字节转化成 String，那么 messageReceived(IoSession session, Object message)接口获取 message 就是 String 类型的，已经经过解码。在这里可以进行很多方式的解码， 比如转化成 String 以后，还可以进一步转化成 POJO，那么获取的 message 就是 POJO。