**一、独立工程使用原始的ActiveMQ**

**1、新建普通Java工程**，引入包activemq-all-5.5-SNAPSHOT.jar，和其它日志相关包log4j-1.2.14.jar，slf4j-api-1.5.11.jar，slf4j-log4j12-1.5.11.jar

**2、新建消息接收类**

注意引用了一些JMS的接口类，类似于JDBC的概念。

import javax.jms.Connection;

import javax.jms.ConnectionFactory;

import javax.jms.Destination;

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.MessageConsumer;

import javax.jms.MessageListener;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnection;

import org.apache.activemq.ActiveMQConnectionFactory;

public class Receiver {

public static void main(String[] args) {

// ConnectionFactory ：连接工厂，JMS 用它创建连接

ConnectionFactory connectionFactory;

// Connection ：JMS 客户端到JMS Provider 的连接

Connection connection = null;

// Session： 一个发送或接收消息的线程

Session session;

// Destination ：消息的目的地;消息发送给谁.

Destination destination;

// 消费者，消息接收者

MessageConsumer consumer;

**connectionFactory** = new ActiveMQConnectionFactory(

ActiveMQConnection.DEFAULT\_USER,

ActiveMQConnection.DEFAULT\_PASSWORD,

"tcp://localhost:61616");

try {

// 构造从工厂得到连接对象

**connection** = connectionFactory.createConnection();

// 启动

connection.start();

// 获取操作连接

**session** = connection.createSession(Boolean.FALSE,

Session.AUTO\_ACKNOWLEDGE);

// 获取session注意参数值xingbo.xu-queue是一个服务器的queue，须在在ActiveMq的console配置

**destination** = session.createQueue("FirstQueue");

**consumer** = session.createConsumer(destination);

// 第一种方式

// while (true) {

// //设置接收者接收消息的时间，为了便于测试，这里谁定为100s

// //TextMessage message = (TextMessage) consumer.receive(100000);

// TextMessage message = (TextMessage) consumer.receive(500000);

// if (null != message) {

// System.out.println("收到消息" + message.getText());

// } else {

// break;

// }

// }

//第二种方式

consumer.setMessageListener(new MessageListener() {

public void **onMessage**(Message message) {

TextMessage m = (TextMessage)message;

try {

m.getText();

System.out.println(m.getText());

} catch (JMSException e) {

e.printStackTrace();

}

}

});

Thread.sleep(30000);

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

if (null != connection)

connection.close();

} catch (Throwable ignore) {

}

}

}

}

**3、新建消息发送类**

注意流程与消息接收类类似，只是MessageConsumer 变成了MessageProducer

import javax.jms.Connection;

import javax.jms.ConnectionFactory;

import javax.jms.DeliveryMode;

import javax.jms.Destination;

import javax.jms.MessageProducer;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnection;

import org.apache.activemq.ActiveMQConnectionFactory;

public class Sender {

private static final int SEND\_NUMBER = 5;

public static void main(String[] args) {

// ConnectionFactory ：连接工厂，JMS 用它创建连接

ConnectionFactory connectionFactory;

// Connection ：JMS 客户端到JMS Provider 的连接

Connection connection = null;

// Session： 一个发送或接收消息的线程

Session session;

// Destination ：消息的目的地;消息发送给谁.

Destination destination;

// MessageProducer：消息发送者

MessageProducer producer;

// TextMessage message;

// 构造ConnectionFactory实例对象，此处采用ActiveMq的实现jar

connectionFactory = new ActiveMQConnectionFactory(

ActiveMQConnection.DEFAULT\_USER,

ActiveMQConnection.DEFAULT\_PASSWORD,

"tcp://localhost:61616");

try {

// 构造从工厂得到连接对象

connection = connectionFactory.createConnection();

// 启动

connection.start();

// 获取操作连接

session = connection.createSession(Boolean.TRUE,

Session.AUTO\_ACKNOWLEDGE);

// 获取session注意参数值xingbo.xu-queue是一个服务器的queue，须在在ActiveMq的console配置

destination = session.createQueue("FirstQueue");

// 得到消息生成者【发送者】

producer = session.createProducer(destination);

// 设置不持久化，此处学习，实际根据项目决定

producer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);

// 构造消息，此处写死，项目就是参数，或者方法获取

sendMessage(session, producer);

session.commit();

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

if (null != connection)

connection.close();

} catch (Throwable ignore) {

}

}

}

public static void sendMessage(Session session, MessageProducer producer)

throws Exception {

for (int i = 1; i <= SEND\_NUMBER; i++) {

TextMessage message = session

.createTextMessage("ActiveMq 发送的消息" + i);

// 发送消息到目的地方

System.out.println("发送消息：" + "ActiveMq 发送的消息" + i);

producer.send(message);

}

}

}

**4、ActiveMQ控制台查看消息信息**

（1）apache-activemq-5.5-SNAPSHOT-bin\bin 启动activemq.bat

（2）进入控制台

http://localhost:8161/admin/ 用户名/密码：admin/admin

**二、gspmg**

**1、配置XML**

**（1）配置MQ的全局配置**

<!-- gspmq的全局配置，配置MQ连接地址、用户名、密码及默认消息处理模式等 -->

<bean id="gspmq-config" class="hk.guosen.ibp.commu.Config">

<property name="connInfo">

<bean class="hk.guosen.ibp.commu.ConnInfo">

<property name="url" value="failover:(tcp://localhost:61616)" />

<property name="userName" value="wtm"/>

<property name="password" value="Guosen1507"/>

</bean>

</property>

<property name="sysInfo">

<bean class="hk.guosen.ibp.commu.SysInfo">

<property name="milliRequestTimeout" value="20000"/>

</bean>

</property>

</bean>

**（2）配置一个自定义ID的bean，实现类统一为hk.guosen.ibp.accessor.spi.impl.ServiceImpl**

<bean id="informationSvc" class="hk.guosen.ibp.accessor.spi.impl.ServiceImpl" init-method="start" destroy-method="stop">

<!-- 资讯管理服务名称为INFO -->

<constructor-arg value="INFO" />

<!-- 引用前面定义的MQ全局变量-->

<constructor-arg ref="gspmq-config" />

<constructor-arg>

<!-- 在此配置跟资讯管理相关的服务功能处理类 -->

<list>

<!-- 此为配置服务功能处理类的实例，每个服务功能处理类需要在构造函数里配置此服务对应的功能号（如果不配置，则默认为去掉Func后的类名）及其处理服务需要的其他类 -->

<bean class="hk.guosen.wtm.accessor.information.cls.GetNewsClsFunc">

<constructor-arg value="GET\_NEWS\_CLS"/>

<property name="clsService" ref="NewsClsService" />

</bean>

<bean class="hk.guosen.wtm.accessor.information.info.GetNewsListFunc">

<constructor-arg value="GET\_NEWS\_LIST"/>

<property name="newsService" ref="NewsTitleService" />

<property name="camService" ref="ICamService" />

</bean>

</list>

</constructor-arg>

</bean>

**2、定义实现类**

**（1）定义请求类**

public class GetNewsContReq {

private Integer newsId;

private Long userId;

public Long getUserId() {

return userId;

}

public void setUserId(Long userId) {

this.userId = userId;

}

public Integer getNewsId() {

return newsId;

}

public void setNewsId(Integer newsId) {

this.newsId = newsId;

}

}

**（2）定义响应类，做为FuncRet类的retObj属性**

public class GetNewsClsResp {

private int total;

private int recCount;

private List<NewsClsVO> recs;

public int getTotal() {

return total;

}

public void setTotal(int total) {

this.total = total;

}

public int getRecCount() {

return recCount;

}

public void setRecCount(int recCount) {

this.recCount = recCount;

}

public List<NewsClsVO> getRecs() {

return recs;

}

public void setRecs(List<NewsClsVO> recs) {

this.recs = recs;

}

}

附框架定义的服务执行的返回结果类FuncRet

//返回码，请参考ApplicationMsg中返回码的规定。

private String retCode;

//返回消息

private String retMsg;

//返回消息的处理方式。为空采用全局默认配置。

private MsgInfo msgInfo;

//返回内容。如果为列表，则建议使用数组形式返回。

private Object retObj;

**（3）定义服务实现类**

public class GetNewsClsFunc extends BaseFunc<String>{

private INewsClsService clsService;

//XML配置的构造函数中，已传入服务名称INFO

public GetNewsClsFunc(String name){

super(name);

}

//实现服务的具体实现逻辑

public FuncRet execute(String input) throws Exception {

Object[] result = clsService.getAll(-1);

GetNewsClsResp resp = new GetNewsClsResp();

List list = (List)result[0];

resp.setRecCount(list.size());

resp.setTotal(list.size());

resp.setRecs(list);

return FuncRet.newOKMsg(resp);

}

public void setClsService(INewsClsService clsService) {

this.clsService = clsService;

}

}

**3、定义客户端调用的测试类**

测试端的代码比较流程化，主要是2点。

1、服务的IP，用户名和密码

2、服务的名称，即消息通道名称

public class ClientTestGetNewsClsFunc extends BaseClientTest{

public static void main(String[] args) throws Exception {

Config config = new Config();

config.getDefaultMsgInfo().setEncrypted(false);

config.getDefaultMsgInfo().setLogLv(MsgInfo.LOG\_MARSHAL);

config.getConnInfo().setUrl("tcp://192.168.230.66:61616");

config.getConnInfo().setUserName("wtmtest");

config.getConnInfo().setPassword("Guosen1507");

//构造一个客户端接入对象，一般 一个应用对应一个客户端接入。客户端接入支持多线程并发调用。

ClientAccessorImpl clt = new ClientAccessorImpl(config);

//启动客户端接入。

clt.start(null);

//GetNewsContReq request = new GetNewsContReq();

//request.setNewsId(920883746);

try

{

//构造一个应用消息，传入的参数分别为服务名称，任务名称和请求参数，跟服务端的定义对应。

//请求参数根据不同的任务而不同，请参考跟服务相关的消息定义。

ApplicationMsg req = new ApplicationMsg("INFO", "GET\_NEWS\_CLS", new Object());

//使用客户端接入发送请求到服务端并返回应答。

ApplicationMsg resp =clt.request(req);

if(resp == null)

{//返回空，代表应答消息超时。

System.out.println("消息处理超时");

return;

}

if(resp.success())

{//处理成功返回，则可以取得应答消息体。

//从应答消息中取得消息体，处理结果放在消息体中。处理结果根据不同的任务而有不同的返回类定义。请参考跟服务相关的消息定义

GetNewsClsResp listResp = resp.getBody();

//对应答消息进行处理。此处直接打印输出。

}

else

{//处理过程中出错了。打印错误消息。

System.out.println("调用服务出错。");

String errMsg = resp.getHead().getRetMsg();

System.out.println(errMsg);

System.out.println(resp);

}

}

finally

{//确保在finally里面关闭客户接入，否则处理过程抛出异常，会导致客户接入不正常关闭，则消息处理线程还会继续运行。

clt.stop();

}

}

}

三、包问题

芝商所需求中，我在jetty中执行不报错，但在Tomcat中发布时报错。

经过排查，加入MQ的包后，在Tomcat中发布会报错。

activemq-all-5.4.0.jar（新加入的包）

activemq-pool-5.4.0.jar（新加入的包）

gsp-mq-2.0.jar（原来fut工程中有）

附Tomcat中的报错信息

Caused by: org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'txManager' defined in class path resource [config/persistence/transaction.xml]: Initialization of bean failed; nested exception is org.springframework.beans.TypeMismatchException: Failed to convert property value of type [org.objectweb.jotm.Current] to required type [javax.transaction.UserTransaction] for property 'userTransaction'; nested exception is java.lang.IllegalArgumentException: Cannot convert value of type [org.objectweb.jotm.Current] to required type [javax.transaction.UserTransaction] for property 'userTransaction': no matching editors or conversion strategy found

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:480)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$1.run(AbstractAutowireCapableBeanFactory.java:409)

at java.security.AccessController.doPrivileged(Native Method)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:380)

at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:264)

at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:221)

at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:261)

at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:185)

at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:164)

at org.springframework.beans.factory.support.BeanDefinitionValueResolver.resolveReference(BeanDefinitionValueResolver.java:269)

... 58 more

Caused by: org.springframework.beans.TypeMismatchException: Failed to convert property value of type [org.objectweb.jotm.Current] to required type [javax.transaction.UserTransaction] for property 'userTransaction'; nested exception is java.lang.IllegalArgumentException: Cannot convert value of type [org.objectweb.jotm.Current] to required type [javax.transaction.UserTransaction] for property 'userTransaction': no matching editors or conversion strategy found

at org.springframework.beans.BeanWrapperImpl.convertForProperty(BeanWrapperImpl.java:391)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.convertForProperty(AbstractAutowireCapableBeanFactory.java:1288)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.applyPropertyValues(AbstractAutowireCapableBeanFactory.java:1249)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapableBeanFactory.java:1010)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:472)

... 67 more

Caused by: java.lang.IllegalArgumentException: Cannot convert value of type [org.objectweb.jotm.Current] to required type [javax.transaction.UserTransaction] for property 'userTransaction': no matching editors or conversion strategy found

at org.springframework.beans.TypeConverterDelegate.convertIfNecessary(TypeConverterDelegate.java:219)

at org.springframework.beans.TypeConverterDelegate.convertIfNecessary(TypeConverterDelegate.java:138)

at org.springframework.beans.BeanWrapperImpl.convertForProperty(BeanWrapperImpl.java:386)

解决方案为：

将activemq-all-5.4.0.jar包中的javax\ transaction包删除即可。

**怎么实现？**

1、消息类

ApplicationMsg

2、客户端接入的实现类

ClientAccessorImpl