

JMS

JMS 的概念

JMS(Java Message Service), 即 Java 消息服务, 是一组 Java 应用程序接口, 用以提供创建、发接、接收、读取消息的服务。SUN 只提供接口, 由不同的厂商根据该接口提供自己的实现。

JMS 的中间件

- IBM MQ 系列产品提供的服务使得应用程序可以使用消息队列进行交流, 通过一系列基于 Java 的 API 提供了 MQseries 在 Java 中应用开发的方法。它支持点到点和发布/订阅两种消息模式。
- WebLogic 是 BEA 公司实现的基于工业标准的 J2EE 应用服务器, 支持大多数企业级 Java API, 它完全兼容 JMS 规范, 支持点到点和发布/订阅消息模式, 它支持消息的多点广播、持久消息存储的文件和数据库、XML 消息, 以及动态创建持久队列和主题。
- JBoss 是 JBoss 公司开发的一个免费开源的应用服务器, 它提供了 EJB 运行的环境, 并能够结合 EJB 进行 JMS 消息的收取, 支持点到点模型和发布/订阅模型。
- ActiveMQ 是一个基于 Apache 2.0 Licenced 发布的开放源代码的 JMS 产品, 它能够提供点到点消息模式和发布/订阅消息模式, 支持 JBoss、Geronimo 等开源应用服务器, 支持 Spring 框架的消息驱动, 新增了一个 P2P 传输层, 可以用于创建可靠的 P2P JMS 网络连接, 拥有消息持久化、事务、集群支持等 JMS 基础设施服务。
- OpenJMS 是一个开源的 JMS 规范的实现, 它支持点对点模型和发布/订阅模型, 支持同步与异步消息发送、可视化管理界面, 支持 Applet, 能够与 Jakarta Tomcat 这样的 Servlet 容器结合, 支持 RMI、TCP、HTTP 与 SSL 协议。

在这里我们使用 ActiveMQ 中间件来做演示

一. ActiveMQ 演示

首先创建一个消息提供者-----JmsProducerDemo

```
package com.sszd.jms;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.command.ActiveMQQueue;

/**
 * Title:
 * Description: 消息产生类
 * Copyright: Copyright (c)2010
 * Company: YeePay
 * @author qi.zhang
```

```

*
*/
public class JmsProducerDemo {

    public static void main(String[] args) {

        // 产生连接工厂
        ConnectionFactory cf = new ActiveMQConnectionFactory("tcp://localhost:61616");

        Connection conn = null;

        Session session = null;

        try
        {
            // 产生连接
            conn = cf.createConnection();

            Session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

            // 设置消息目的地类型（ActiveMQTopic为订阅目的地）
            Destination destination = new ActiveMQQueue("myQueue");

            MessageProducer producer = session.createProducer(destination);

            TextMessage message = session.createTextMessage();

            message.setText("Just a simaple JMS Testing.");

            producer.send(message);

        }
        catch(JMSEException e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                if(session!=null)
                {
                    session.close();
                }
            }
        }
    }
}

```

```

        if(conn!=null)
        {
            conn.close();
        }
    }
    catch(JMSEException ex)
    {
        ex.printStackTrace();
    }
}
}
}

```

接下来就是编写消息使用者-----JmsReceiverDemo

```

package com.sszd.jms;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSEException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.Session;
import javax.jms.TextMessage;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.command.ActiveMQQueue;
import com.sszd.springjms.StudentMessageConverter;
/**
 * Title:
 * Description: 消息接收类
 * Copyright: Copyright (c)2010
 * Company: YeePay
 * @author qi.zhang
 *
 */
public class JmsReceiverDemo {

    public static void main(String[] args) {

        // 产生连接工厂
        ConnectionFactory cf = new ActiveMQConnectionFactory("tcp://localhost:61616");

        Connection conn = null;
    }
}

```

```
Session session = null;

try
{
    // 产生连接
    conn = cf.createConnection();

    session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

    // 设置消息目的地类型
    Destination destination = new ActiveMQQueue("myQueue");

    MessageConsumer consumer = session.createConsumer(destination);

    conn.start();

    Message message = consumer.receive();

    TextMessage textMessage = (TextMessage)message;

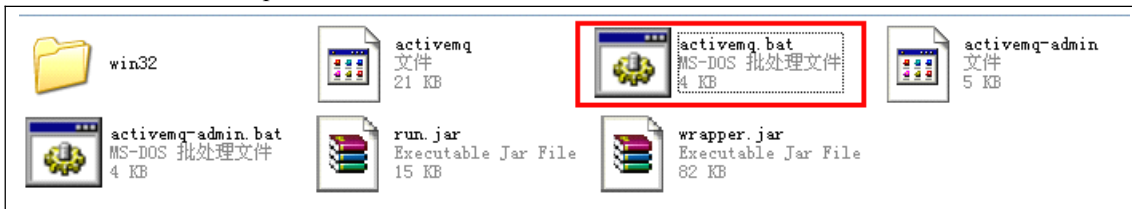
    System.out.println(textMessage.getText());
}
catch(JMSEException e)
{
    e.printStackTrace();
}
finally
{
    try
    {
        if(session!=null)
        {
            session.close();
        }
        if(conn!=null)
        {
            conn.close();
        }
    }
    catch(JMSEException ex)
    {
        ex.printStackTrace();
    }
}
```

```

    }
}
}

```

启动本地的 activemq.bat



分别运行 JmsProducerDemo 和 JmsReceiverDemo



这就是一个最简单的 JMS 演示，仅供入门。

二. 在 spring 中使用 ActiveMQ

首先就是我们要在 spring 中来定义我们的消息中介，也就是我们的 *connectionFactory*，以及消息类型是队列还是订阅方式。

```

<!-- 消息中介 -->
<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
    <property name="brokerURL" value="tcp://localhost:61616"/>
</bean>
<!-- 队列 目的地-->
<bean id="myQueue" class="org.apache.activemq.command.ActiveMQQueue">
    <constructor-arg index="0" value="notifyQueue"/>
</bean>
<!-- 订阅 目的地-->
<bean id="myTopic" class="org.apache.activemq.command.ActiveMQTopic">
    <constructor-arg index="0" value="notifyTopic"/>
</bean>

```

在 Spring 中，Spring 提供了 JMS 的模板--*jmsTemplate*，直接使用 JMS 就可以对消息进行操作。同样，在 Spring 中也要定义它如下

```

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory" ref="connectionFactory"/>
    <!-- 设置默认的消息目的地 -->
    <property name="defaultDestination" ref="myQueue"/>
    <!-- 由于receiver方法是同步的，所以我们在这里对接收设置超时时间 -->
    <property name="receiveTimeout" value="60000"/>

```

```
</bean>
```

编写消息提供者

```
package com.sszd.springjms;

import javax.jms.JMSException;
import javax.jms.MapMessage;
import javax.jms.Message;
import javax.jms.Session;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;

/**
 * Title:
 * Description: 利用Spring中的JmsTemplate产生消息
 * Copyright: Copyright (c)2010
 * Company: YeePay
 * @author qi.zhang
 *
 */

public class JMSProducer {

    private JmsTemplate jmsTemplate;

    public JmsTemplate getJmsTemplate() {
        return jmsTemplate;
    }

    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    // 在这个示例中我们传送一个 Student 对象（重写了 toString()方法）
    public void send(final Student student)
    {
        this.getJmsTemplate().send(
            new MessageCreator(){
                public Message createMessage(Session session)
                    throws JMSException
                {
                    MapMessage message = session.createMapMessage();
                    message.setString("key1", student.getName());
                    message.setString("key2", student.getAge());
                    return message;
                }
            }
        );
    }
}
```

```

        }
    });
}
}

```

编写消息消费者

```

package com.sszd.springjms;

import javax.jms.JMSException;
import javax.jms.MapMessage;
import org.springframework.jms.core.JmsTemplate;
/**
 * Title:
 * Description:  利用Spring中的JmsTemplate消费消息
 * Copyright: Copyright (c)2010
 * Company: YeePay
 * @author qi.zhang
 *
 */

public class JMSReceiver {

    private JmsTemplate jmsTemplate;

    public JmsTemplate getJmsTemplate() {
        return jmsTemplate;
    }

    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    public void receive()
    {
        MapMessage message = (MapMessage)this.getJmsTemplate().receive();
        try {
            if(message!=null)
            {
                System.out.println(message.getString("key1")+message.getString("key2"));
            }
            else
            {
                System.out.println("未接收到内容");
            }
        }
    }
}

```

```

        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}

```

在 Spring 配置文件中配置提供者和消费者

```

<!-- 消息发送者 -->
<bean id="consumer" class="com.sszd.springjms.JMSProducer">
    <property name="jmsTemplate" ref="jmsTemplate"/>
    <!-- 消息目的地，因为jmsTemplate有默认的了，所以这里可以省略 -->
    <property name="destination" ref="myQueue"/>
    -->
</bean>
<!-- 接收消息，但它是同步的，会发生消息阻塞 -->
<bean id="receiver" class="com.sszd.springjms.JMSReceiver">
    <property name="jmsTemplate" ref="jmsTemplate"/>
</bean>

```

启动 ActiveMQ，运行我们的测试程序

SpringJMSDemo.java

```

package com.sszd.springjms;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
/**
 * Title:
 * Description: 消息产生测试类
 * Copyright: Copyright (c)2010
 * Company: YeePay
 * @author qi.zhang
 *
 */
public class SpringJMSDemo {

    public static void main(String[] args) {

        ApplicationContext context
            = new ClassPathXmlApplicationContext("classpath:applicationContext.xml");

        JMSProducer jmsProducer =(JMSProducer)context.getBean("consumer");

        Student student = new Student();
    }
}

```



```
        student.setName("zhangqi");
        student.setAge("25");

        jmsProducer.send(student);

        System.out.println("消息已经发送.....");    }
    }
```

```
package com.sszd.springjms;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
/**
 * Title:
 * Description: 消息消费测试类
 * Copyright: Copyright (c)2010
 * Company: YeePay
 * @author qi.zhang
 *
 */

public class SpringJMSReceiverDemo {

    public static void main(String[] args) {

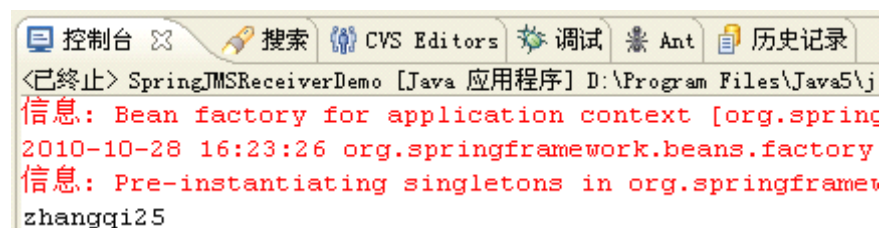
        ApplicationContext context
            = new ClassPathXmlApplicationContext("classpath:applicationContext.xml");

        JMSReceiver jmsReceiver =(JMSReceiver)context.getBean("receiver");

        jmsReceiver.receive();
    }

}
```

结果



```
<已终止> SpringJMSReceiverDemo [Java 应用程序] D:\Program Files\Java5\j
信息: Bean factory for application context [org.spring
2010-10-28 16:23:26 org.springframework.beans.factory
信息: Pre-instantiating singletons in org.springframework
zhangqi25
```

三. 消息转换

在上一个例子中，我们传递了一个 `Student` 对象，其内容分别是姓名和年龄。我们会发现一个问题，我们需要将 `student` 中的各个属性都摘出来，然后根据各个属性组成一个 `MapMessage`，并且在消息接收端，需要根据 `map` 中 `Key`，将 `MapMessage` 对象中的数据逐个取出后，再重新组装成一个新的 `student`。

这样我们在发送和接收的时候就要不断的拆装对象。

为了减少这个重复的拆装对象的操作，我们可以利用消息转换机制来实现每次发送和接收消息时自动拆装对象。

这时，我们可以利用 `Spring` 中的 `MessageConverter` 接口做到消息转换
接口中有两个方法

```
//接收消息的方法
Object fromMessage(Message message) throws JMSEException,MessageConversionException;
//发送消息的方法
Message toMessage(Object object,Session session) throws JMSEException
                                                ,MessageConversionException;
```

下面是一个基于 `Student` 对象的消息转换的实现类

```
package com.sszd.springjms;

import javax.jms.JMSEException;
import javax.jms.MapMessage;
import javax.jms.Message;
import javax.jms.Session;
import org.springframework.jms.support.converter.MessageConversionException;
import org.springframework.jms.support.converter.MessageConverter;

/**
 * Title:
 * Description: 消息转换类
 * Copyright: Copyright (c)2010
 * Company: YeePay
 * @author qi.zhang
 *
 */
public class StudentMessageConverter implements MessageConverter {

    public Object fromMessage(Message message) throws JMSEException,
                                                MessageConversionException {
        if(!(message instanceof MapMessage))
        {
            throw new MessageConversionException("Message is not a MapMessage");
        }
    }
}
```

```

        MapMessage mapMessage = (MapMessage)message;

        Student student = new Student();

        student.setName(mapMessage.getString("key1"));

        student.setAge(mapMessage.getString("key2"));

        return student;
    }

    public Message toMessage(Object object, Session session) throws JMSEException,
                                                                    MessageConversionException {
        if(!(object instanceof Student))
        {
            throw new MessageConversionException("Object is not a student");
        }

        Student student = (Student) object;

        MapMessage message = session.createMapMessage();

        message.setString("key1", student.getName());

        message.setString("key2", student.getAge());

        return message;
    }
}

```

修改我们第一个例子中的代码中的生产者（JmsProducerDemo.java）

```

package com.sszd.jms;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSEException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.command.ActiveMQQueue;
import com.sszd.springjms.Student;
import com.sszd.springjms.StudentMessageConverter;

```

```

/**
 * Title:
 * Description: 消息产生类
 * Copyright: Copyright (c)2010
 * Company: YeePay
 * @author qi.zhang
 *
 */
public class JmsProducerDemo {

    public static void main(String[] args) {

        // 产生连接工厂
        ConnectionFactory cf = new ActiveMQConnectionFactory("tcp://localhost:61616");

        Connection conn = null;

        Session session = null;

        try
        {
            // 产生连接
            conn = cf.createConnection();

            session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

            // 设置消息目的地类型
            Destination destination = new ActiveMQQueue("myQueue");
            MessageProducer producer = session.createProducer(destination);

            Student student = new Student();
            student.setName("zq");
            student.setAge("25");

            // 使用消息转换
            StudentMessageConverter studentConverter = new StudentMessageConverter();
            producer.send(studentConverter.toMessage(student, session));

        }
        catch(JMSEException e)
        {
            e.printStackTrace();
        }
        finally
    }
}

```

```

        {
            try
            {
                if(session!=null)
                {
                    session.close();
                }
                if(conn!=null)
                {
                    conn.close();
                }
            }
            catch(JMSEException ex)
            {
            }
        }
    }
}
}

```

修改我们第一个例子中的代码中的消费者（JmsReceiverDemo.java）

```

package com.sszd.jms;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSEException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.Session;
import javax.jms.TextMessage;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.apache.activemq.command.ActiveMQQueue;
import com.sszd.springjms.StudentMessageConverter;
/**
 * Title:
 * Description: 消息产生类
 * Copyright: Copyright (c)2010
 * Company: YeePay
 * @author qi.zhang
 *
 */

```

```

public class JmsReceiverDemo {

    public static void main(String[] args) {

        // 产生连接工厂
        ConnectionFactory cf = new ActiveMQConnectionFactory("tcp://localhost:61616");

        Connection conn = null;

        Session session = null;

        try
        {
            // 产生连接
            conn = cf.createConnection();

            session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

            // 设置消息目的地类型
            Destination destination = new ActiveMQQueue("myQueue");

            MessageConsumer consumer = session.createConsumer(destination);

            conn.start();

            Message message = consumer.receive();

            // 使用消息转换
            StudentMessageConverter studentConverter = new StudentMessageConverter();

            System.out.println(studentConverter.fromMessage(message));

        }
        catch(JMSEException e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                if(session!=null)
                {
                    session.close();
                }
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

```

        }
        if(conn!=null)
        {
            conn.close();
        }
    }
    catch(JMSEException ex)
    {

    }
}
}
}

```

如果是在 Spring 中使用消息转换，那么我们还要在 *jmsTemplate* 配置中加上消息转换的类。

```

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate" >
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="defaultDestination" ref="myQueue"/>
    <!--消息转换 -->
    <property name="messageConverter" ref="studentMessageConverter"/>
    <!-- 接收超时时间 receiver方法是同步的 -->
    <property name="receiveTimeout" value="50000"/>
</bean>

<!-- 转换消息 -->
<bean id="studentMessageConverter" class="com.sszd.springjms.StudentMessageConverter"/>

```

然后我们把发送和接收的方法分别修改为

```
this.getJmsTemplate().convertAndSend(student);
```

和

```
this.getJmsTemplate().receiveAndConvert();
```

就要以得到和修改之前一样的结果了。

四. 使用 **Spring** 监听实现异步接收消息

我们之前所举的例子都是同步接收消息，这样就会形成消息堵塞。

首先我们需要创建一个监听方法实现消息监听接口（这个类就叫消息驱动 POJO）

```

package com.sszd.springjms;

import javax.jms.JMSEException;
import javax.jms.MapMessage;

```

```

import javax.jms.Message;
import javax.jms.MessageListener;
/**
 * Title:
 * Description: 消息监听类
 * Copyright: Copyright (c) 2010
 * Company: YeePay
 * @author qi.zhang
 *
 */
public class StudentMDP implements MessageListener {

    public void onMessage(Message message) {

        MapMessage mapMessage = (MapMessage)message;

        Student student = new Student();

        try {
            student.setName(mapMessage.getString("key1"));

            student.setAge(mapMessage.getString("key2"));

            System.out.println(student);

        } catch (JMSEException e) {
            e.printStackTrace();
        }
    }
}

```

在 Spring 中配置我们的监听

```

<!-- 消息接收监听器 用于异步接收消息 ， 用这个就不能配 receiver了-->
<bean class="org.springframework.jms.listener.SimpleMessageListenerContainer">
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="destination" ref="myQueue"/>
    <property name="messageListener" ref="studentMDP"/>
</bean>
<!-- 消息驱动pojo -->
<bean id="studentMDP" class="com.sszd.springjms.StudentMDP"/>

```

还原我们的创建消息方法（仍然是发送一个 MapMessage）

```

public void send(final Student student)
{

```



```

        this.getJmsTemplate().send(
            new MessageCreator(){
                public Message createMessage(Session session)
                    throws JMSException
                {
                    MapMessage message = session.createMapMessage();
                    message.setString("key1", student.getName());
                    message.setString("key2", student.getAge());
                    return message;
                }
            });
    }
}

```

在发送测试用例中我们增加一行用于打印发送结束信息

```

public class SpringJMSDemo {

    public static void main(String[] args) {

        ApplicationContext context
            = new ClassPathXmlApplicationContext("classpath:applicationContext.xml");
        JMSProducer jmsProducer =(JMSProducer)context.getBean("consumer");
        Student student = new Student();
        student.setName("zhangqi");
        student.setAge("25");
        jmsProducer.send(student);
        System.out.println("消息已经发送.....");
    }
}

```

启动 ActiveMQ，运行 SpringJMSDemo，我们会看到结果被监听拦截到，接收打印了出来。并且在 console 中显示的应用程序仍然为消息创造者。



```

SpringJMSDemo [Java 应用程序] D:\Program Files\Java5\jdk1
信息: Bean factory for application context
2010-10-28 17:34:20 org.springframework.bes
信息: Pre-instantiating singletons in org.s
消息已经发送.....
zhangqi 25

```

在这个例子中，onMessage() 方法相当于一个管道代码，StudentMDP 类实现了 MessageListener 接口，但是这样我们的 StudentMDP 类就不是一个纯的 POJO 类了。

五. 使用纯 POJO 实现 Spring 中的导步消息监听机制

```

package com.sszd.springjms;

```

```

import java.util.Map;
/**
 * Title:
 * Description: 纯POJO实现消息接收
 * Copyright: Copyright (c) 2010
 * Company: YeePay
 * @author qi.zhang
 *
 */
public class PureStudentMDP {

    public void process(Map map)
    {
        Student student = new Student();
        student.setName((String)map.get("key1"));
        student.setAge((String)map.get("key2"));
        System.out.println("PureStudentMDP:");
        System.out.println(student);
    }
}

```

我们可以看到上面的 PureStudentMDP 类，它没有借助任何其它的类和接口。
接下来就是配置 Spring 文件了

首先是 PureStudentMDP 本身

```
<bean id="pureStudentMDP" class="com.sszd.springjms.PureStudentMDP">
```

接下来有一些特殊的地方，就是纯的 POJO 来实现消息异步监听要借助适配器

```

<bean id="pureMDPAdapter"
      class="org.springframework.jms.listener.adapter.MessageListenerAdapter">
    <property name="delegate" ref="pureStudentMDP"/>
    <property name="defaultListenerMethod" value="process"/>
</bean>

```

修改监听配置

```

<bean class="org.springframework.jms.listener.SimpleMessageListenerContainer">
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="destination" ref="myQueue"/>
    <!-- <property name="messageListener" ref="studentMDP"/>-->
    <!--纯 POJO，中间要用适配器 -->
    <property name="messageListener" ref="pureMDPAdapter"/>
</bean>

```

运动消息创建测试，结果如下

```
2010-10-30 16:08:23 org.spr
信息: Pre-instantiating sing
PureStudentMDP:
zhangqi 25
消息已经发送.....
```

此外, 在纯 POJO 编写的例子中同样可以通知之前提到的[消息转换类](#)
修改我们的 PureStudentMDP (POJO)的 process 方法

```
public void process(Student student)
{
    System.out.println("PureStudentMDP:process(Student student)");
    System.out.println(student);
}
```

在 Spring 中配置我们的消息转换

```
<bean id="pureMDPAdapter"
    class="org.springframework.jms.listener.adapter.MessageListenerAdapter">
    <property name="delegate" ref="pureStudentMDP"/>
    <property name="defaultListenerMethod" value="process"/>
    <!-- 消息转换.注意发送的时候也要用消息转换的方式发送 -->
    <property name="messageConverter" ref="studentMessageConverter"/>
</bean>
```

这里要注意, 我们的发送类也要相应修改在以消息转换方式主送, 以及 JmsTemplate

```
//利用转换消费
this.getJmsTemplate().convertAndSend(student);
```

```
<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate" >
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="defaultDestination" ref="myQueue"/>
    <property name="messageConverter" ref="studentMessageConverter"/> <!-- 消息转换 -->
    <!-- 接收超时时间 receiver 方法是同步的 -->
    <property name="receiveTimeout" value="50000"/>
</bean>
```

运行测试

```
2010-10-30 16:20:17 org.springframework.x
信息: Pre-instantiating singletons in org
PureStudentMDP:process(Student student)
zhangqi 25
消息已经发送.....
```

以上的全部介绍就是一些基本的 JMS 在 Spring 中使用的方法。