Chapter 14 SPRING과 JMS 연동

: 게시판 시스템에 새로운 게시글이 등록되었을 때 색인을 위해 검색 엔진 시스템에 게시글 정보를 전송해야 하는 경우를 생각해 보자. 이 경우, 게시판 시스템과

검색 엔진 시스템은 약속된 규칙에 따라 게시글 정보를 담고 있는 Message를 주고 받아야 하는데, 이렇게 서로 다른 시스템 간에 Message를 주고 받아야 할 때

사용되는 기술중의 하나가 JMS(Java Message Service)이다.

SPRING은 JMS API를 이용하여 Message를 발생하고 수신할 때 사용할 수 있는 JmsTemplate Class를 제공하고 있어서, JdbcTemplate을 사용할 때와 마찬가지로

중첩되는 Code를 줄일 수 있도록 도와 준다. 또한, POJO Class를 이용하여 EJB의 MDB(Message Driven Bean)과 동일한 기능을 제공하는 MDP(Message Driven

Pojo) 빈을 지원하고 있어서 블로킹 없이 비동기로 Message를 수신할 수 있다.

본 절에서는 먼저 JdbcTemplate을 이용하여 JMS를 이용한 기본적인 Message 송수신 방법을 살펴보고, 그 다음으로 MDP를 이용해서 블로킹 없이 Message를 수신하는

방법을 살펴 볼 것이다.

1. JMS 연동을 위한 ConnectionFactory 및 Message 목적지 설정

: SPRING이 제공하는 JmsTemplate Class를 사용하려면 다음의 두 가지를 설정해 주어야 한다.

- JMS 서버와의 연결을 제공하는 ConnectionFactory

- Message 목적지 설정(Queue 또는 Topic)

Database 연동을 하기 위해 가장 먼저 DataSaource와 SessionFactory 등을 설정했던 것처럼, JMS 연동을 하려면 javax.jms.ConnectionFactory를 설정해

주어야 한다. JEE Container를 사용한다면 JNDI로부터 가져올 것이다.

ActiveMQ를 Message 브로커로 사용하고 있다면 ActiveMQConnectionFactory Class를 사용해서 ConnectionFactory를 설정할 수 있다. 아래 Code는 ActiveMQ를

사용하는 경우의 설정 예이다.

<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory"

brokerURL="tcp://localhost:61616"/>

ConnectionFactory를 설정했다면, 그 다음으로 할 작업은 Message 목적지(destination)를 설정하는 것이다. ActiveMQ를 사용할 경우 다음과 같이 큐와 토픽 타입의

목적지를 설정할 수 있다.

<bean id="queueDestination" class="org.apache.activemq.command.ActiveMQQueue">

<constructor-arg value="bbs.article.queue" />

</bean>

<bean id="topicDestination" class="org.apache.activemq.command.ActiveMQTopic">

<constructor-arg value="bbs.article.topic" />

</bean>

2. JmsTemplate을 이용한 Message 송수신

: JmsTemplate을 사용함으로써 Resource 처리를 위한 try~catch~finally Code의 반복을 제거했던 것처럼, JmsTemplate Class를 사용하면 JMS를 이용할 때 Resource 처리를

위해 반복되는 Code를 제거할 수 있다.

또한, JmsTemplate Class는 JMS 연동 과정에서 발생하는 javax.jms.JMSException 타입 및 하위 타입의 예외를 SPRING이 제공하는 org.springframework.jms.JmsExcpetion

타입 및 하위 타입의 예외로 알맞게 변환해 주는 기능도 제공하고 있다. Javax.jms.JMSException과 달리 SPRING의 JmsException은 RuntimeException이기 때문에

불필요한 try~catch Code를 입력하지 않아도 된다.

JmsTemplate Class를 사용하려면 먼저 설정 파일에 JmsTemplate을 빈으로 등록해 주어야 한다. connectionFactory 프로퍼티에는 앞서 설명한 ConnectionFactory

빈 객체를 설정해 준다.

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"

connectionFactory-ref="connectionFactory" />

<bean class="madvirus.spring.chap14.domain.ArticleMessageReceiver"

p:jmsTemplate-ref="jmsTemplate"

p:destination-ref="queueDestination" />

JMS를 이용해서 Message를 송신하거나 수신할 빈은 JmsTemplate과 큐나 토픽 목적지를 설정한 빈을 전달 받아 사용하면 된다.

2.1 JMS Message 보내기

: JmsTemplate Class와 send() 메서드를 사용하면 JMS Message를 전송할 수 있다. send() 메서드는 Message를 전송할 목적지를 의미하는 Destination 객체와

MessageCreator 구현 객체를 전달 받는다. MessageCreatoer Interface를 구현한 객체는 createMessage() 메서드를 이용해서 전송할 Message 객체를 생성한다.

아래 Code는 JmsTemplate Class의 send() 메서드의 사용 예를 보여 주고 있다.

import javax.jms.Destination;

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;

public class ArticleMessageSender {

private JmsTemplate jmsTemplate;

private Destination destination;

public void setJmsTemplate(JmsTemplate jmsTemplate) {

this.jmsTemplate = jmsTemplate;

}

public void setDestination(Destination destination) {

this.destination = destination;

}

public void sendMessage(final Article article) {

jmsTemplate.send(destination, new MessageCreator() {

@Override

public Message createMessage(Session session) throws JMSException {

MapMessage mapMessage = session.createMapMessage();

mapMessage.setString("subject", article.getSubject());

return mapMessage;

}

});

// jmsTemplate.convertAndSend(destination, article);

}

}

JmsTemplate.send() 메서드는 createMessage() 메서드가 리턴한 Message 객체를 목적지에 전송한다. 따라서, MessageCreator Interface 구현 객체의

createMessage() 메서드는 Message를 전송하기 위한 알맞은 타입의 javax.jms.Message 객체를 리턴하면 된다.

2.2 JMS Message 받기

: JmsTemplate Class의 receive() 메서드를 사용하면 JMS Message를 수신할 수 있다. 다음과 같이 receive() 메서드에 Message를 전달 받을 목적지를 지정하면,

해당 목적지로부터 Message 객체를 가져와 리턴한다.

import javax.jms.Destination;

import javax.jms.JMSException;

import javax.jms.MapMessage;

import org.springframework.jms.core.JmsTemplate;

import org.springframework.jms.support.JmsUtils;

public class ArticleMessageReceiver {

private JmsTemplate jmsTemplate;

private Destination destination;

public void setJmsTemplate(JmsTemplate jmsTemplate) {

this.jmsTemplate = jmsTemplate;

}

public void setDestination(Destination destination) {

this.destination = destination;

}

public Article receive() {

MapMessage mapMessage = (MapMessage) jmsTemplate.receive(destination);

Article article = new Article();

try {

article.setSubject(mapMessage.getString("subject"));

return article;

} catch (JMSException e) {

throw JmsUtils.convertJmsAccessException(e);

}

// Article article = (Article)jmsTemplate.receiveAndConvert(destination);

// return article;

}

}

Message를 수신하는 Code는 JmsTemplate.receive() 메서드가 리턴한 객체를 알맞은 타입의 Message로 변환한 뒤에, Message로부터 필요한 정보를 읽어와

사용하게 된다.

2.3 JmsTemplate의 기본 목적지 설정

: JmsTemplate Class의 send() 메서드와 receive() 메서드를 호출할 때 목적지를 지정하지 않으면 defaultDestination 프로퍼티로 설정한 Destination을

Message 송 수신 목적지로 사용한다. 아래 Code는 defaultDestination 프로퍼티의 설정 예이다.

<bean id="articleQueueDestination" class="org.apache.activemq.command.ActiveMQQueue">

<constructor-arg value="bbs.article.queue" />

</bean>

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"

connectionFactory-ref="connectionFactory"

p:defaultDestination-ref="articleQueueDestination" />

위와 같이 설정했다면, 목적지를 지정하지 않고 send() 메서드나 receive() 메서드를 호출할 수 있다.

jmsTemplate.send(new MessageCreator(){

@Override

public Message createMessage(Session session) throws JMSException{

...

}

});

Message message = jmsTemplate.receive();

3. MessageConverter를 이용한 Message와 객체 Mapping

: 앞서 Message를 전송하는 Code를 보면 다음과 같이 MessageCreator 구현 객체에서 Article 객체를 알맞은 타입의 Message 객체로 변환해서 생성해 주었다.

public void sendMessage(final Article article) {

jmsTemplate.send(destination, new MessageCreator() {

@Override

public Message createMessage(Session session) throws JMSException {

MapMessage mapMessage = session.createMapMessage();

mapMessage.setString("subject", article.getSubject());

return mapMessage;

}

});

}

Message를 수신하는 Code에서는 반대로 Message로부터 정보를 읽어와 알맞은 객체를 생성하였다.

public Article receive() {

MapMessage mapMessage = (MapMessage) jmsTemplate.receive(destination);

Article article = new Article();

try {

article.setSubject(mapMessage.getString("subject"));

return article;

} catch (JMSException e) {

throw JmsUtils.convertJmsAccessException(e);

}

}

JMS Message를 송수신하는 모든 Code에서 위와 같은 변환 작업을 해 주어야 한다면, 이는 꽤나 성가실 뿐만 아니라 변환하는 Code도 여러 Code에서 중복될

것이다. SPRING은 MessageConverter Interface를 제공하고 있는데, 이 Interface를 사용하면 중복 작업과 같은 문제를 해소할 수 있다.

MessageConverter Interface는 다음과 같이 정의되어 있다.

package org.springframework.jms.support.converter;

import javax.jms.JMSEception;

import javax.jms.Message;

import javax.jms.Session;

public interface MessageConverter{

Message toMessage(Object object, Session session) throws JMSException, MessageConcersionException;

Object fromMessage(Message message) throws JMSException, MessageConcersionException;

}

toMessage() 메서드는 객체를 JMS Message로 변환할 때 사용되며, fromMessage() 메서드는 반대로 Message를 원하는 객체로 변환할 때 사용된다. 따라서,

toMessage() 메서드는 객체를 Message로 전송할 때, 반대로 fromMessage() 메서드는 Message를 수신할 때 사용된다. 아래 Code는 MessageConverter Interface의

구현 예이다.

package madvirus.spring.chap14.domain.converter;

import javax.jms.JMSException;

import javax.jms.MapMessage;

import javax.jms.Message;

import javax.jms.Session;

import madvirus.spring.chap14.domain.Article;

import org.springframework.jms.support.converter.MessageConversionException;

import org.springframework.jms.support.converter.MessageConverter;

public class ArticleMessageConverter implements MessageConverter {

@Override

public Object fromMessage(Message message) throws JMSException,

MessageConversionException {

MapMessage mapMessage = (MapMessage) message;

Article article = new Article();

article.setSubject(mapMessage.getString("subject"));

return article;

}

@Override

public Message toMessage(Object object, Session session)

throws JMSException, MessageConversionException {

Article article = (Article) object;

MapMessage mapMessage = session.createMapMessage();

mapMessage.setString("subject", article.getSubject());

return mapMessage;

}

}

MessageConverter Interface를 구현한 Class를 작성했다면, 다음과 같이 JmsTemplate의 messageConverter 프로퍼티를 사용해서 설정해 줄 수 있다.

<bean id="articleMessageConverter"

class="madvirus.spring.chap14.domain.converter.ArticleMessageConverter" />

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"

p:connectionFactory-ref="connectionFactory"

p:messageConverter-ref="articleMessageConverter" />

JmsTemplate에 MessageConverter를 설정한 뒤에는 convertAndSend() 메서드를 사용해서 Message를 전송하고, receiveAndConvert() 메서드를 사용해서

Message를 수신할 수 있다.

jmsTemplate.convertAndSend(destination, articel);

Article article = (Article)jmsTemplate.receiveAndConvert(destination);

Message와 객체사이의 변환 처리를 MessageConverter가 처리해 주기 때문에, Code에서는 Message 타입이 아닌 Application에 알맞은 타입을 사용해서

Message 송수신 부분의 Code를 작성할 수 있는 장점이 있다.

4. JmsGatewaySupport Class를 이용한 JMS 연동 Class 구현

: JmsTemplate을 필요로 하는 Class는 다음과 같이 JmsTemplate을 전달받기 위한 메서드와 필드를 선언하게 된다.

public class ArticleMessageReceiver {

private JmsTemplate jmsTemplate;

public void setJmsTemplate(JmsTemplate jmsTemplate) {

this.jmsTemplate = jmsTemplate;

}

...

}

JmsGatewaySupport Class를 상속받아 구현하면 JmsTemplate을 전달받기 위한 Code를 제거할 수 있다. JmsGatewaySupport Class는 JmsTemplate 객체를

리턴하는 getJmsTemplate() 메서드를 제공하고 있으며, 하위 Class에서는 getJmsTemplate() 메서드를 사용해서 JmsTemplate 객체를 구한 뒤 알맞은 작업을

수행하면 된다. 아래 Code는 JmsGatewaySupport를 상속받는 Class의 구현 예이다.

import org.springframework.jms.core.support.JmsGatewaySupport;

public class ArticleMessageReceiver extends JmsGatewaySupport{

public Article receive(){

Article article = (Article)getJmsTemplate().receiveAndConvert();

return article;

}

}

JmsGatewaySupport를 상속받은 Class가 올바르게 동작하려면 jmsTemplate 프로퍼티를 이용해서 JmsTemplate 객체를 설정하거나 connectionFactory 프로퍼티를

이용해서 ConnectionFactory 객체를 설정해 주어야 한다. 아래 Code는 jmsTemplate 프로퍼티를 사용해서 설정하는 예를 보여 주고 있다.

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"

p:connectionFactory-ref="connectionFactory"

p:messageConverter-ref="articleMessageConverter"

p:defaultDestination-ref="articleQueueDestination" />

<bean id="articleReceiver"

class="madvirus.spring.chap14.domain.ArticleMessageReceiver"

p:jmsTemplate-ref="jmsTemplate" />

5. POJO 기반의 Message-Driven 빈

: JmsTemplate.receive() 메서드는 메서드가 도착하기 전까지 블로킹 되어 있다가, Message가 도착하면 비로서 Message 객체를 리턴한다.

//Message가 도착할 때 블로킹

MapMessage mapMessage = (MapMessage)jmsTemplate.receive(destination);

Message가 도착할 때까지 블로킹 되기 때문에, 큐나 토픽으로부터 계속해서 Message를 가져와서 알맞은 작업을 처리해야 하는 Code를 작성해야 할 경우 다음과 같이

조건에 따라 무한히 실행되는 while 블록을 사용해야 한다.

while(!isClose()){

MapMessage mapMessage = (MapMessage)jmsTemplate.receive(destination);

...

}

EJB 2부터 새롭게 추가된 MDB(Message Driven Bean)는 Message가 도착할 때까지 블로킹 되는 방식을 사용하지 않고, 반대로 Message가 도착하면 지정한 메서드가

호출되는 방식을 사용한다. 즉 비동기로 Message를 수신할 수 있도록 지원한다.

SPRING은 EJB Container가 아니기 때문에 MDB를 직접적으로 지원하지는 않지만, 자체적으로 MDB와 동일하게 비동기로 Message를 처리할 수 있는 방법을 제공하고

있다. SPRING은 javax.jms.MessageListener Interface를 구현한 빈 객체를 사용하는 방법과 JMS Interface를 구현하지 않은 POJO 빈 객체를 사용하는 방법을

제공하고 있는데, 본 절에서는 이 두 가지 방법을 이용해서 SPRING에서 비동기로 Message를 처리하는 방법을 살펴보도록 하자.

5.1 MessageListener와 MessageListener Container를 이용한 Message-Driven 구현

: javax.jms.MessageListener Interface는 Message가 도착할 때 호출 될 onMessage() 메서드를 정의하고 있다.

public interface MessageListener{

void onMessage(Message message);

}

다음 Code는 MessageListener Interface의 구현 예를 보여 주고 있다.

package madvirus.spring.chap14.domain.listener;

import javax.jms.JMSException;

import javax.jms.MapMessage;

import javax.jms.Message;

import javax.jms.MessageListener;

import madvirus.spring.chap14.domain.Article;

import org.springframework.jms.support.JmsUtils;

public class ArticleMessageListener implements MessageListener {

@Override

public void onMessage(Message message) {

MapMessage mapMessage = (MapMessage) message;

try {

Article article = new Article();

article.setSubject(mapMessage.getString("subject"));

processReceiveArtice(article);

} catch (JMSException e) {

throw JmsUtils.convertJmsAccessException(e);

}

}

private void processReceiveArtice(Article article) {

System.out.println("Article 색인 처리 요청:" + article.getSubject());

}

}

MessageListener Interface를 구현한 Class를 작성했다고 해서 자동으로 Message가 onMessage() 메서드에 전달되는 것은 아니다. MessageListener에

Message가 전달되려면 추가적으로 MessageListenerContainer를 설정해 주어야 한다. EJB Container가 Message를 읽어와 MDB에 전달하듯이,

MessageListenerContainer는 큐나 토픽으로부터 Message를 읽어와 MessageListener에 Message를 전달한다.

Message 큐 / 토픽 =======> MessageListenerContainer =======> MessageListener

SPRING이 제공하는 MessageListenerContainer 목록은 아래와 같다.

Container Class 설 명

========================================================================================

SimpleMessageListenerContainer 단순한 Container로서 정해진 개수의 JMS Session을 사용하고 Transaction을 지원하지 않는다.

========================================================================================

DefaultMessageListenerContainer SimpleMessageListenerContainer와 동일하며 추가적으로 Transaction을 지원한다.

========================================================================================

serversession. Transaction 뿐만 아니라 동적으로 JMS Session을 관리한다.

ServerSessionMessageListenerContainer

========================================================================================

아래 Code는 DefaultMessageListenerContainer의 설정 예를 보여 주고 있다.

<bean id="messageListenerContainer"

class="org.springframework.jms.listener.DefaultMessageListenerContainer"

p:connectionFactory-ref="connectionFactory" p:destination-ref="articleQueueDestination"

p:messageListener-ref="messageListenerAdapter"

p:transactionManager-ref="jmsTxManager" />

<bean id="articleMessageListener"

class="madvirus.spring.chap14.domain.listener.ArticleMessagePojoListener" />

connectionFactory 프로퍼티와 destination 프로퍼티는 JmsTemplate을 생성할 때 사용되는 프로퍼티와 동일하다. messageListener 프로퍼티는

Container로부터 Message를 전달받을 MessageListener를 설정한다.

SPRING이 제공하는 MessageListenerContainer는 비동기 처리를 위해 쓰레드를 내부적으로 사용한다. 따라서 SPRING Container를 종료해 주지 않으면

JVM 프로세스가 종료되지 않으므로, 사용이 끝나면 알맞게 SPRING Container를 종료해 주어야 한다.

5.2 MessageListenerAdapter를 이용한 POJO 기반의 MD 구현

: MessageListenerAdapter를 사용하면 MessageListener Interface를 구현하지 않은 객체에 Message를 전달할 수 있다. MessageListenerAdapter는

MessageListenerContainer로부터 전달받은 Message를 MessageConverter를 이용해서 변환한 뒤에 지정한 객체에 전달하는 기능을 제공하고 있다.

Message 큐 / 토픽 ====> MessageListenerContainer ====> MessageListenerAdapter ====> MessagePojoListener

따라서, MessageListenerAdapter를 사용하면 큐나 토픽으로부터 전달받은 Message 객체를 알맞은 타입으로 변환해서 MessageListener Interface를

구현하지 않은 POJO 객체에 전달할 수 있게 된다. 아래 Code는 MessageListenerAdapter의 설정 예이다.

<bean id="messageListenerContainer"

class="org.springframework.jms.listener.DefaultMessageListenerContainer"

p:connectionFactory-ref="connectionFactory" p:destination-ref="articleQueueDestination"

p:messageListener-ref="messageListenerAdapter"

p:transactionManager-ref="jmsTxManager" />

<bean id="messageListenerAdapter"

class="org.springframework.jms.listener.adapter.MessageListenerAdapter"

p:delegate-ref="articleMessageListener" p:defaultListenerMethod="received"

p:messageConverter-ref="articleMessageConverter" />

<bean id="articleMessageListener"

class="madvirus.spring.chap14.domain.listener.ArticleMessagePojoListener" />

Message Container의 messageListener 프로퍼티에는 MessageListenerAdapter 빈 객체를 값으로 전달한다. 이렇게 함으로써 Message Container는

MessageListenerAdapter에 Message 객체를 전달하게 된다.

Message Container로부터 Message를 전달받은 MessageListenerAdapter는 delegate 프로퍼티에 지정한 객체의 메서드를 호출한다. 호출되는 메서드 이름은

defaultListenerMethod 프로퍼티를 사용해서 설정한다. 위 Code의 경우 MessageListenerAdapter는 articleMessageListener 빈 객체의 received() 메서드를 호출한다.

delegate 프로퍼티로 지정한 객체의 메서드를 호출할 때에는 Message 객체를 직접 전달하기 보다는 MessageConverter 사용해서 Message를 알맞게 변환한 객체를

전달한다. MessageConverter는 messageConverter 프로퍼티를 사용해서 설정하며, MessageConverter를 설정하지 않을 경우 기본적으로 SimpleMessageConverter가

사용된다. SimpleMessageConverter는 아래의 규칙에 따라 Message를 객체로 변환한다.

Message 타입 변환 규칙

============================================================

TextMessage String 타입으로 변환(getText() 리턴)

============================================================

ByteMessage byte[] 타입으로 변환(message.readBytes()로 읽어온 byte 배열 리턴)

============================================================

MapMessage Map 타입으로 변환

============================================================

ObjectMessage Serializable 타입의 객체로 변환(getObject() 리턴)

============================================================

이 이외의 Message 타입 Message 객체를 그대로 사용

============================================================

예를 들어, MapMessage 타입의 Message가 전달될 경우 MessageListenerAdapter로 부터 Message를 전달받은 POJO Class는 다음과 같이 메서드의 파라미터 타입으로

Map을 사용하면 된다.

public void rececived(Map map){

Article article = new Article();

article.setSubject((String)map.get("subject"));

precessReceiveArticle(article(;

}

SimpleMessageConverter가 변환해 주는 타입이 아닌 원하는 타입을 사용하고 싶다면 알맞은 MessageConverter 객체를 messageConverter 프로퍼티로 설정해

주어야 한다. 아래 Code는 설정 예이다.

<bean id="messageListenerAdapter"

class="org.springframework.jms.listener.adapter.MessageListenerAdapter"

p:delegate-ref="articleMessageListener" p:defaultListenerMethod="received"

p:defaultListenerMethod="received"

p:messageConverter-ref="articleMessageConverter" />

<bean id="articleMessageConverter"

class="madvirus.spring.chap14.domain.converter.ArticleMessageConverter" />

<bean id="articleMessageListener"

class="madvirus.spring.chap14.domain.listener.ArticleMessagePojoListener" />

ArticleMessageConverter가 Message 객체를 Article 객체로 변환해 준다고 할 경우, Message를 전달받을 ArticlePojoListener의 received() 메서드는 다음과 같이

파라미터로 Article 객체를 전달받을 수 있게 된다.

public class ArticlePojoListener{

...

public void received(Article article){

processReceiveArticle(article);

}

...

}

6. JMS Transaction 처리

: Message를 전송한 이후에 논리적으로 오류가 발생했다고 하자. 이 경우 Message 전송이 취소되지 않는다면 잘못된 Message가 전달될 것이고, 최종적으로 Message를

사용하는 Code는 정상적이지 못한 Message를 사용해서 문제가 발생하게 될 것이다. 반대로 Message를 수신하는 Code에서 Message를 받은 이후에 예외가 발생했다고

하자. 이 경우 Message를 받은 것 자체를 취소하지 않으면 해당 Message는 사라져 버리게 된다.

이 얘기는 JMS API를 이용해서 Message를 송수신 할 때에도 Transaction이 필요하다는 것을 의미한다. SPRING은 JMS API를 위한 Transaction 관리자인

JmsTransactionManager를 제공하고 있으며, JmsTransactionManager를 사용함으로써 SPRING이 제공하는 Transaction 관리 기능을 사용할 수 있게 된다.

6.1 JmsTransactionManager을 이용한 Transaction 설정

: JmsTransactionManager를 이용해서 Transaction을 설정하는 방법은 앞서 '9장 SPRING의 Transaction 관리'에서 살펴봤던 방법과 동일하다.

즉, TransactionTemplate이나 설정 파일 및 Annotation을 이용해서 JmsTemplate Class를 이용하는 Class를 Transaction 범위 내에서 실행할 수 있다.

JmsTransactionManager를 이용해서 Transaction을 처리하려면 아래 Code와 같이 JmsTransactionManager의 connectionFactory 프로퍼티에

connectionFactory 빈을 전달하면 된다.

<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">

<property name="brokerURL">

<value>tcp://localhost:61616</value>

</property>

</bean>

<bean id="jmsTxManager"

class="org.springframework.jms.connection.JmsTransactionManager">

<property name="connectionFactory" ref="connectionFactory" />

</bean>

<tx:advice id="txAdvice" transaction-manager="jmsTxManager">

<tx:attributes>

<tx:method name="sendMessage" />

</tx:attributes>

</tx:advice>

<aop:config>

<aop:advisor advice-ref="txAdvice"

pointcut="execution(public void sendMessage(..))" />

</aop:config>

위 Code에서 JmsTemplate과 JmsTransactionManager는 동일한 ConnectionFactory를 사용하므로, JmsTemplate은 JmsTransactionManager가 관리하는

Transaction 범위내에서 실행된다. 따라서, JmsTemplate.send() 메서드가 실행되더라도 Transaction이 커밋 되기 전까지는 Message가 목적지 큐나 토픽에

저장되지 않는다. 반대로 JmsTemplate.receive() 메서드는 Transaction이 커밋 된 이후에 큐나 토픽에서 Message를 완전히 제거한다. Transaction이 롤백 될 경우

큐나 토픽에서 Message가 제거되지 않는다.

DefaultMessageListenerContainer를 사용할 경우 transactionManager 프로퍼티에 Transaction 관리자를 전달해 주면 된다.

<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">

p:brokerURL="tcp://localhost:61616"/>

<bean id="messageListenerContainer"

class="org.springframework.jms.listener.DefaultMessageListenerContainer"

p:connectionFactory-ref="connectionFactory"

p:destination-ref="deliveryCompleteQueue"

p:messageListener-ref="articleMessageAdapter"

p:transactionManager-ref="jmsTxManager" />

<bean id="jmsTxManager"

class="org.springframework.jms.connection.JmsTransactionManager">

p:connectionFactory-ref="connectionFactory" />

DefaultMessageListenerContainer에 Transaction 관리자를 설정하면, Message가 도착했을때 실행되는 MessageListener 메서드가 Transaction 범위 내에서 실행된다.

6.2 글로벌 Transaction 설정

: DBMS 처리와 JMS 처리를 하나의 Transaction으로 묶어서 처리하려면 JtaTransactionManager를 이용해 주어야 한다. 본 책의 9장에서는 TransactionsEssentials를

이용해서 글로벌 Transaction을 설정하는 방법을 살펴봤는데, TransactionsEssentials는 JMS ConnectionFactory를 글로벌 Transaction과 연동하기 위한

AtimikosConnectionFactoryBean Class를 제공하고 있으며, 이 Class를 이용해서 JMS ConnectionFactory를 제공할 수 있다.

아래 Code는 TransactionsEssentials를 이용해서 글러벌 Transaction을 설정하는 Code 예를 보여주고 있다.

<bean id="userTransactionService" class="com.atomikos.icatch.config.UserTransactionServiceImp"

init-method="init" destroy-method="shutdownForce">

...

</bean>

<!-- Atomikos UserTransactionManager 생성. SPRING 설정에 필요함 -->

<bean id="atomikosTransactionManager" class="com.atomikos.icatch.jta.UserTransactionManager"

init-method="init" destroy-method="close" depends-on="userTransactionService">

<property name="forceShutdown" value="false" />

</bean>

<!-- Atomikos UserTransactionImp 생성. SPRING 설정에 필요함 -->

<bean id="atomikosUserTransaction" class="com.atomikos.icatch.jta.UserTransactionImp"

depends-on="userTransactionService">

<property name="transactionTimeout" value="300" />

</bean>

<!-- Atomikos를 사용하는 SPRING JtaTransactionManager 설정 -->

<bean id="transactionManager"

class="org.springframework.transaction.jta.JtaTransactionManager"

depends-on="userTransactionService">

<property name="transactionManager" ref="atomikosTransactionManager" />

<property name="userTransaction" ref="atomikosUserTransaction" />

</bean>

<bean id="xaConnectionFactory" class="org.apache.activemq.ActiveMQXAConnectionFactory">

<property name="brokerURL">

<value>tcp://localhost:61616</value>

</property>

</bean>

<bean id="connectionFactory" class="com.atomikos.jms.AtomikosConnectionFactoryBean"

init-method="init" destroy-method="close">

<property name="uniqueResourceName" value="amq1" />

<property name="xaConnectionFactory" ref="xaConnectionFactory" />

</bean>

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">

<property name="connectionFactory" ref="connectionFactory" />

<property name="messageConverter" ref="articleMessageConverter" />

<property name="sessionTransacted" value="true" />

</bean>

AtomikosConnectionFactoryBean Class를 사용해서 ConnectionFactory 객체를 생성하는데, AtomikosConnectionFactoryBean Class는 xaConnectionFactory 프로퍼티를

이용해서 XAConnectionFactory 객체를 전달받는다. 위 Code에서는 ActiveMQ의 ActiveMQXAConnectionFactory Class를 이용해서 XAConnectionFactory 객체를

생성하였다.

JmsTemplate Class는 AtomikosConnectionFactoryBean을 이용해서 생성한 ConnectionFactory를 사용하면 된다. 또한, sessionTransacted 속성을 true로

지정해서 JmsTemplate이 사용할 Session을 생성할 때 Transaction 여부를 true로 지정할 수 있도록 해 주어야 한다.

DefaultMessageListenerContainer를 사용하는 경우에도 다음과 같이 sessionTransacted 속성의 값을 true로 설정해 주어야 한다.

<bean id="messageListenerContainer"

class="org.springframework.jms.listener.DefaultMessageListenerContainer"

p:connectionFactory-ref="connectionFactory"

p:destination-ref="articleQueueDestination"

p:messageListener-ref="articleMessageListener"

p:transactionManager-ref="transactionManager"

p:sessionTransacted="true" />

<bean id="transactionManager"

class="org.springframework.transaction.jta.JtaTransactionManager"

depends-on="userTransactionService">

<property name="transactionManager" ref="atomikosTransactionManager" />

<property name="userTransaction" ref="atomikosUserTransaction" />

</bean>

<bean id="xaConnectionFactory" class="org.apache.activemq.ActiveMQXAConnectionFactory">

<property name="brokerURL">

<value>tcp://localhost:61616</value>

</property>

</bean>

<bean id="connectionFactory" class="com.atomikos.jms.AtomikosConnectionFactoryBean"

init-method="init" destroy-method="close">

<property name="uniqueResourceName" value="amq1" />

<property name="xaConnectionFactory" ref="xaConnectionFactory" />

</bean>

7. <jms:listener-container>를 이용한 설정

: <jms:listener-container> 태그를 이용해도 MessageListenerContainer를 설정할 수 있다. 아래 Code는 설정 예이다.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:jms="http://www.springframework.org/schema/jms" xmlns:p="http://www.springframework.org/schema/p"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/jms

http://www.springframework.org/schema/jms/spring-jms-3.0.xsd">

<jms:listener-container transaction-manager="jmsTxManager">

<jms:listener destination="bbs.article.queue" ref="articleMessageListener" />

</jms:listener-container>

<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">

<property name="brokerURL">

<value>tcp://localhost:61616</value>

</property>

</bean>

<bean id="jmsTxManager"

class="org.springframework.jms.connection.JmsTransactionManager">

<property name="connectionFactory" ref="connectionFactory" />

</bean>

<bean id="articleMessageListener"

class="madvirus.spring.chap14.domain.listener.ArticleMessageListener" />

</beans>

<jms:listener-container> 태그는 MessageListenerContainer를 생성한다. <jms:listener-container> 태그의 주요 속성은 다음과 같다.

속 성 설 명

=========================================================================

connection-factory ConecctionFactory 빈 객체 이름을 지정한다. 기본 값은 "connectionFactory"이다.

=========================================================================

transaction-manager Transaction 관리자 빈 객체 이름을 지정한다.

destination-type 목적지 타입을 지정한다. "queue","topic","durableTopic" 중 한 가지 값을 갖는다.

기본 값은 "queue"이다.

=========================================================================

message-converter JMS Message를 Java 객체로 변환할 때 사용할 MessageConverter 빈을 지정한다.

지정하지 않을 경우 SimpleMessageConverter가 사용된다.

=========================================================================

acknowledge JMS 인식모드를 설정한다. "auto","client","dups-ok","transacted" 중 한가지 값을 갖는다.

기본 값은 "auto"이다.

=========================================================================

<jms:listener-container> 태그는 한 개 이상의 <jms:listener> 태그를 갖는다. <jms:listener> 태그는 destination 속성을 이용해서 Message를 수신할

목적지를 지정하며 ref 속성을 이용해서 수신한 Message를 전달받을 빈 객체를 지정한다. 만약 Message를 수신할 객체가 MessageListener 타입이 아닌

POJO 타입이라면 method 속성을 이용해서 Message를 수신할 메서드의 이름을 지정하면 된다.

<jms:listener-container transaction-manager="jmsTxManager">

<jms:listener destination="bbs.article.queue" ref="articleMessageListener" />

<jms:listener destination="bbs.common.queue" ref="commonMessageListener" method="received"/>

</jms:listener-container>