
Welcome to
Spring Forward

Introduction to Spring JMS

Mark Pollack

Principal
CodeStreet LLC

Speaker's Qualifications

- Mark Pollack is a founding partner at CodeStreet LLC, a software and consulting firm in the financial services industry.
- .NET and J2EE architect and developer specializing in financial front office solutions, EAI, and message based middleware
- TIBCO
- Spring Developer (2003)
  - JmsTemplate
- Contributing author
  - Java Development with the Spring Framework
    - JMS Section
- Founder and Co-lead of Spring.NET (2004)

Outline

- What is Messaging Middleware?
- Introduction to JMS
- Spring's JMS goals
- Sending Messages
- Consuming Messages
- .NET/Java interoperability

Messaging Middleware

- Communication styles
  - Synchronous: “request/reply”
  - Asynchronous
- Messaging styles: “domains”
  - Point-to-Point: “queues”
  - Publish-Subscribe: “topics”
- Important characteristics
  - Broker responsible for delivery
  - QOS – persistence, priority and time to live.
  - Producer decoupled from Subscriber

Messaging Middleware

- Popular before Java/J2EE
  - IBM – MQ-Series
  - TIBCO
  - Microsoft – MSMQ
  - Others…
- Heterogeneous
  - Platform and Language

What is JMS?

- JCP specification for messaging
  - 1.0.2 (2000), part of J2EE 1.3
  - 1.1 (2002), part of J2EE 1.4
- Common API and behavior across vendors
- Vendor differentiate on primarily non-functional features
  - Administration
  - Security
  - Failover
  - Language bindings
  - Alternative Ack modes and Message types

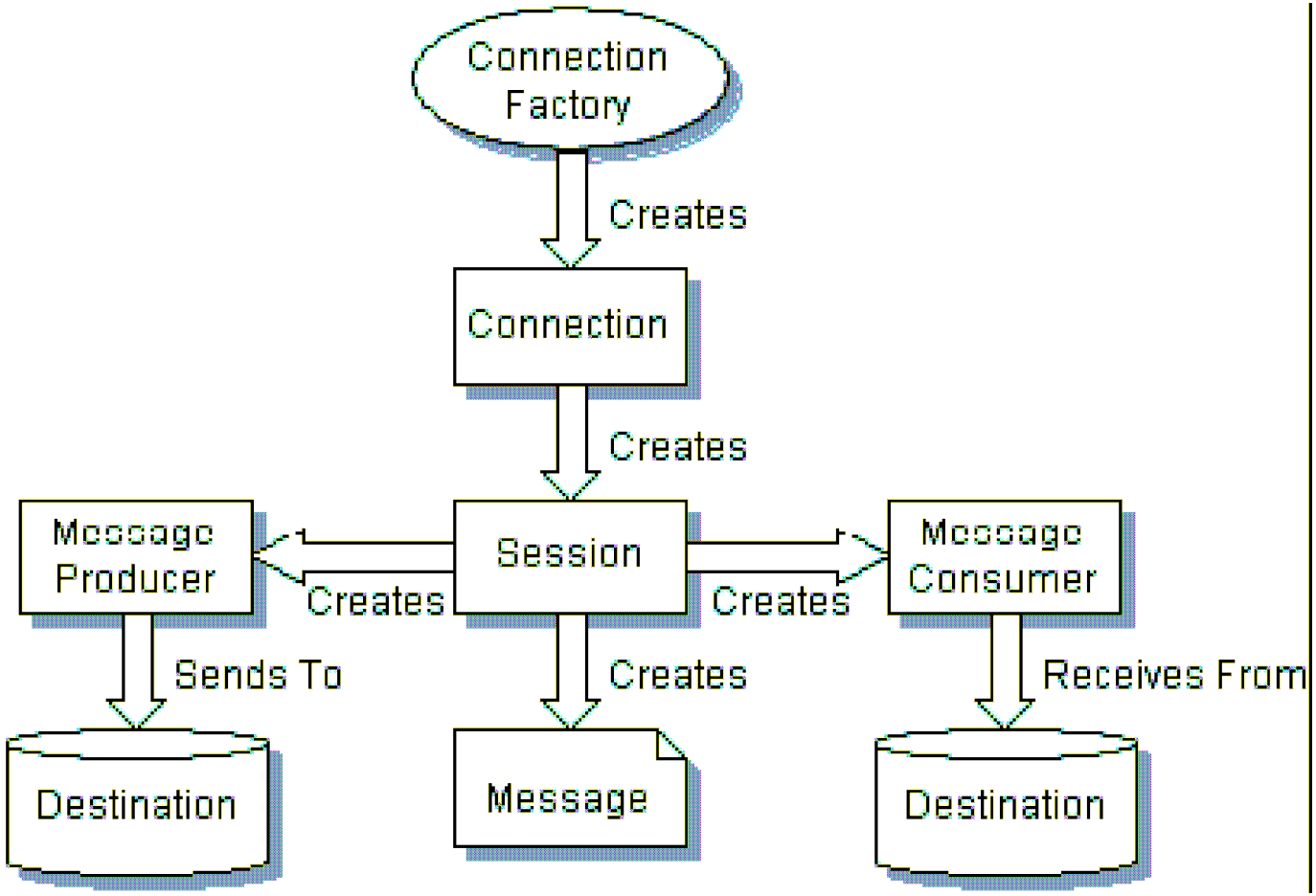
What is JMS

- Specs distinguishes between
  - Non-J2EE “stand alone” environment
    - Typically a custom server
      - You define lifecycle of objects
      - Fair amount of boiler-plate code
    - Access to full JMS functionality/API
    - Typically no connection/session/producer pooling
  - J2EE/EJB 2.0 app server environment
    - Lifecycle of EJB component model
    - Restricts functionality available
    - Configuration based async consumers

Whirlwind tour of the JMS API

- Important JMS classes
  - **ConnectionFactory**
    - creates connections
    - *Administered JNDI object*
  - **Connection**
    - Socket that communicates to broker
  - **Session**
    - Create messages, producers, and consumer
  - **MessageProducer**
    - Sends message - ‘async’ to consumer - ‘sync’ to broker.
  - **MessageConsumer**
    - ‘sync’ receive
  - Register **MessageListener** with MessageConsumer
    - ‘async’ receive
  - **Destination**
    - Where to send the message (topics/queues)
    - *Administered JNDI object*

JMS Object Relationships



Plain Jane JMS Hello World!

- Not showing JNDI lookup of ConnectionFactory and Destination

```
Connection connection = connectionFactory.createConnection();

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

MessageProducter producer = session.createProducer(null);

Message message = session.createTextMessage("Hello world!");

producer.send(destination, message);
```

- Not showing try/finally code to close all intermediate objects (resources)

Spring's JMS Goals

- 0th order similarity to Spring's JDBC support
- Remove API noise
  - One-liner send/sync-receive
  - Few line multithreaded asynchronous consumers
- Perform Resource Management
  - Clean up done in case of exceptions
- Shield differences between JMS 1.0.2/1.1 APIs
  - Works with both APIs
    - 1.0.2/1.1 'flavored' implementations provided
- Destination Management
  - In practice destinations are often created at runtime.
- Translation from checked to unchecked exceptions

Spring's JMS Goals

- Integration with Spring's transaction management services
  - JMS local transactions
    - Group multiple send/receive operations
  - XA distributed transactions
    - Group message consumption and database access
- Facilitate Best Practices
  - POJO programming model
  - Object/Message conversion
  - Direct support for Gateway design pattern

JmsTemplate

- Common send/sync-receive tasks are one liners
- Familiar callback design for advanced scenarios
- Message conversion
- Programmatic Example
  - Sending a TextMessage to Queue

```
String serverUrl = "tcp://localhost:7222";
ConnectionFactory connFactory =
              new TibjmsQueueConnectionFactory(serverUrl);

JmsTemplate jmsTemplate = new JmsTemplate(connFactory);
jmsTemplate.convertAndSend("partyline.queue",
                           "Hello World! @ " + new Date());
```

JmsTemplate

- Synchronous receiving from a queue

```
String reply =
  (String) jmsTemplate.receiveAndConvert("anotherqueue");
```

- Using callback to get at JMS objects

```
jmsTemplate.send(newMessageCreator() {

  public Message createMessage(Session session)
      throws JMSException {

    TextMessage msg = session.createTextMessage("myMessage");
    msg.setIntProperty("myCounter", 2);
    return msg;

  }
});
```

Destination Management

- DestinationResolver
  - Convert string to JMS Destination object
  - Default used in DynamicDestinationResolver
    - Session.createQueue/createTopic
  - JndiDesintationResolver
    - Does JNDI lookup – fallback to dynamic behavior

Resource Pooling

- Connection factory is entry point
  - Responsible for pooling of all created objects
- Pooling is default case in J2EE container
- Stand-alone behavior depends on vendor
- Spring provides SingleConnectionFactory adapter
  - Reuses same connection – thread safe
- ActiveMQ provides adapter that pools sessions/producers.

MessageConverters

- Convert to/from objects in business domain to JMS messages.
- Default – SimpleMessageConverter
  - `String` ←→ `TextMessage`
  - `Byte[]` ←→ `BytesMessages`
  - `Map` ←→ `MapMessage`
  - `Serializable` ←→ `ObjectMessage`
- Serializable is a brittle choice in messaging environment.

MessageConverter Interface

- Extension point for more robust converters

```
public interface MessageConverter {

  Message toMessage(Object object, Session session)

       throws JMSException, MessageConversionException;

  Object fromMessage(Message message)

       throws JMSException, MessageConversionException;

}
```

- Example – “PersonConverter”

JmsGateway

- Gateway design pattern
  - Wrapper that provides domain specific API in order to simplify access to external resources that have complex resource-specific API
  - Client code insulated from particularities of the resource's API that would have otherwise obscured the business processing code
- `JmsGateway` akin to `JdbcDaoSupport`
  - Set ConnectionFactory – create JmsTemplate
  - Set JmsTemplate explicity

JmsGateway Example

- Client code refers to InvoiceService

```
public interface InvoiceService {

  public void reissue(String accountId, Date billingEndDate);

}
```

```
public class JmsInvoiceService extends JmsGatewaySupport
                                    implements InvoiceService {

  public void reissue(String accountId, Date billingEndDate){
    Map m = new HashMap();
    m.put("accountid", accountId);
    m.put("date", billingEndDate.toGMTString());
    getJmsTemplate().convertAndSend(m);
  }
}
```

Asynchronous Message Consumption

- Multithreaded processing of messages
- JMS MessageListener
  - callback interface asynchronously notified upon receipt of message

```
public interface MessageListener {

  void onMessage(Messagemessage);

}
```

- Stand-alone environment
  - Code it up: MessageConsumer.setMessageListener()
- J2EE environment
  - Configure Message Driven Beans

Spring's Message Listener Containers

- New in Spring 2.0
- Easy creation of multithreaded asynchronous message consumers
  - Based on standard JMS MessageListener
  - Alternatively Spring's SessionAwareMessageListener
    - Expose active JMS Session
    - Useful in some providers for sending reply messages.
- Functional equivalent to MDBs
  - Supported in both container and stand-alone environments
  - Support for XA message reception
- Variety of container implementations
- Spring raises level of abstraction
  - Message driven POJOs

JMS Container Implementations

- SimpleMessageListenerContainer
  - For stand-alone environment
  - Fixed number of JMS MessageConsumers
  - No XA support
  - Worker threads are those of JMS provider
- DefaultMessageListenerContainer
  - Stand-alone or J2EE environment
  - XA support
  - Fixed number of concurrent consumers – sync receive in a loop.
  - Worker threads are taken from TaskExecutor implementation
- ServerSessionMessageListenerContainer
  - For stand-alone environment
  - Uses ServerSessionPool SPI
    - Optional part of JMS spec.
  - No XA support
  - Dynamic number of concurrent consumers

SimpleMessageListenerContainer

```
String serverUrl = "tcp://localhost:7222";
ConnectionFactory connFactory =
                new TibjmsQueueConnectionFactory(serverUrl);

SimpleMessageListenerContainer container =
               new SimpleMessageListenerContainer();

container.setConnectionFactory(connFactory);
container.setDestinationName("partyline.queue");
container.setConcurrentConsumers(10);
container.setMessageListener(new SimpleListener());

container.afterPropertiesSet();
```

- Standard JMS listener

```
public class SimpleListener implements MessageListener {
  public void onMessage(Message msg) {
       System.out.println("Received message = " + msg);
  }
}
```

DEMO

MessageListenerAdapter

- Delegates JMS callback method to JMS-agnostic method
  - Default method name is ‘handleMethod’
- Listener method arguments
  - Use result of MessageConverter if configured, otherwise javax.jms.Message
- Listener method return value
  - Use result of MessageConverter if configured, otherwise must be javax.jms.Message
  - Sent on JMS-ReplyTo address
- Subclass and override methods to customize behavior
  - Duplicate detection

DEMO

JMS based .NET/Java interop

- Reflection based MessageConverter
- Objects are converted via reflection to map messages
  - Vendor support for nested map messages required
  - Preserves ability to consume from other language bindings.
- Java objects automatically translated to .NET via Microsoft's Java Language Conversion Assistant

Related Projects/Resources

- O'Reilly JMS book
- Lingo
  - Transparent JMS "Remoting"
- Jenks
  - "MessageListenerContainer" via JCA
- Hermes
  - GUI for JMS
- ActiveMQ
  - Open source JMS provider – rich feature set
- CodeStreet ReplayService for JMS
  - Record/Replay/Reinject JMS messages

Summary

- Spring raises level of abstraction when dealing with JMS
- Familiar “Springy” design and configuration
- JmsTemplate for send and sync receive
- MessageContainers for asynchronous consumption
- Extend out of the box behavior
  - MessageConverter, JmsGateway, MessageListenerAdapter
- Stay tuned for Spring.NET JMS support…

Q&A