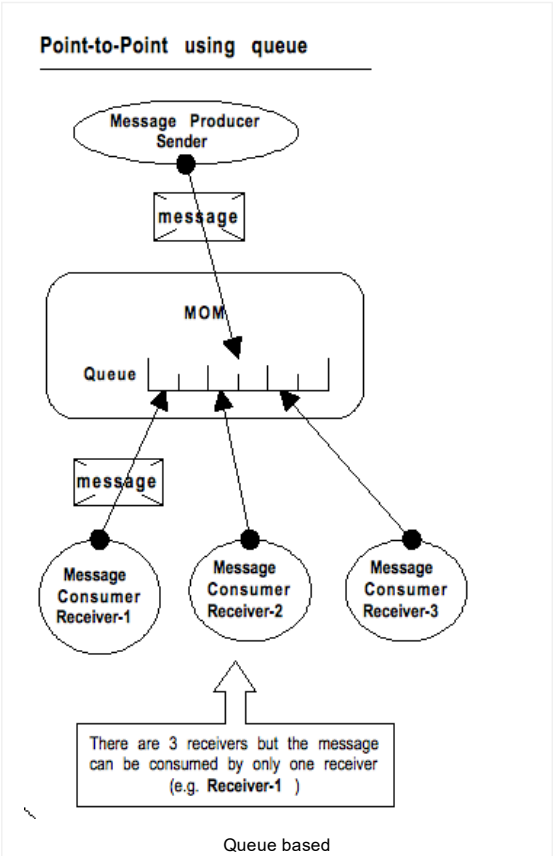
♦ 16 FAQ JMS interview questions & answers

Posted on September 4, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. What types of messaging paradigms are provided by JMS?

A1. **Point-to-Point**: provides a traditional queue based mechanism where the client application sends a message through a queue to typically one receiving client that receives messages sequentially. A JMS message queue is an administered object that represents the message destination for the sender and the message source for the receiver. A Point-to-Point application has the following characteristics:

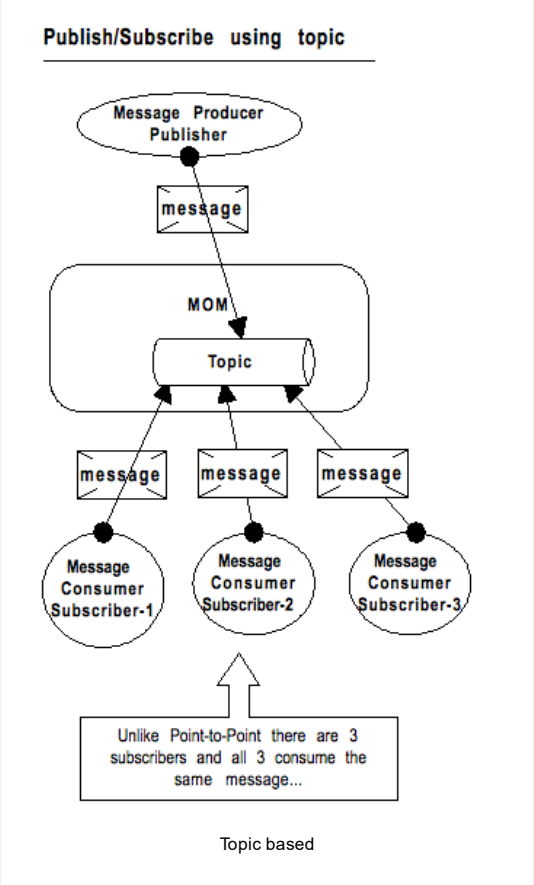


Queue based

– A Point-to-Point producer is a sender (i.e. QueueSender).
– A Point-to-Point consumer is a receiver (i.e. QueueReceiver).
– A Point-to-Point destination is a queue (i.e. Queue).
– **A message can only be consumed by one receiver**.

Example: A call center application may use a queue based Point-to-Point domain to process all the calls where all the phone calls do not go to all the operators, but only one.

Publish/Subscribe: is a **one-to-many publishing model** where client applications publish messages to topics, which are in turn subscribed by other interested clients. All subscribed clients will receive each message. A Publish/Subscribe application has the following characteristics:



Topic based

– A Publish/Subscribe producer is a publisher (i.e. TopicPublisher).
– A Publish/Subscribe consumer is a subscriber (i.e. TopicSubscriber).
– A Publish/Subscribe destination is a topic (i.e. Topic).
– **A message can be consumed by multiple subscribers**.

If a message publisher is also a subscriber, then a publisher can receive its own message sent to the destination. This behavior is only applicable to publish/subscribe model. This behavior can be controlled by setting the “noLocal” attribute to true when creating the publisher or the subscriber.

Example: A bulletin board application may use a topic based publish/subscribe model where everyone who is interested in particular news becomes a subscriber and when a message is published, it is sent to all its subscribers.

SQL, XML, UML, JSON, Regex I
Hadoop & BigData Interview Q&A
Java Architecture Interview Q&A
Scala Interview Q&As (13)
Spring, Hibernate, & Maven Inter
Testing & Profiling/Sampling Java
Other Interview Q&A for Java De
▶Free Java Interview Videos (4)

16 Technical Key Areas

open all | close all
Best Practice (6)
Coding (26)
Concurrency (6)
Design Concepts (7)
Design Patterns (11)
Exception Handling (3)
Java Debugging (21)
Judging Experience Interview Q&
Low Latency (7)
Memory Management (7)
Performance (13)
QoS (8)
Scalability (4)
SDLC (6)
Security (13)
Transaction Management (5)

80+ step by step Java Tutorials

open all | close all
Setting up Tutorial (6)
Tutorial - Diagnosis (2)
Akka Tutorial (9)
Core Java Tutorials (20)
Hadoop & Spark Tutorials (29)
JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tutorials (26)
Tools Tutorials (19)
Other Tutorials (45)

100+ Java pre-interview coding tests

open all | close all
Can you write code? (22)
♦ Complete the given code (13)

Q2. How do you determine whether it would be better to use a Topic or Queue?
A2. You must choose to use a Topic if one of the following conditions applies:

1) Same message must be replicated to multiple consumers (With Queue a message can only be consumed by one receiver).

2) A message should be dropped if there are no active consumers that would select it.

3) There are many subscribers each with a unique selector.

Q3. What is Message Oriented Middleware (MOM)?
A3. Message Oriented Middleware (MOM) is generally defined as a software infrastructure that asynchronously communicates with other disparate systems (e.g. Mainframe system, C++ System, etc) through the production and consumption of messages. A message may be a request, a report, or an event sent from one part of an enterprise application to another.

Q4. Why use a messaging system as opposed to using Data Transfer Objects (aka DTOs, Value Objects) or Web services with JSON on XML data?
A4. Firstly, messaging enables loosely coupled distributed communication. A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate and also they are not aware of each other. In fact, the sender does not need to know anything about the receiver; nor does the receiver need to know anything about the sender. The sender and the receiver need to know only what message format and what destination to use. In this respect, messaging differs from tightly coupled technologies, such as Remote Method Invocation (RMI) and Web services.

Q5. Why use JMS with MOM?
A5. Message Oriented Middleware (MOM) systems like WebsphereMQ, SonicMQ, WebMethods, etc are proprietary systems. Java Message Service (JMS) is a Java API that allows applications to create, send, receive, and read messages in a standard way. The JMS API defines a common set of interfaces and associated semantics that allow programs written in the Java programming language to communicate with other messaging implementations (e.g. SonicMQ, TIBCO etc). The JMS API minimizes the set of concepts a programmer must learn to use messaging products but provides enough features to support sophisticated messaging applications. It also strives to maximize the portability of JMS applications across JMS providers. Similar to JDBC, a good example of the **adapter design pattern** in action.

Many companies have spent decades developing their legacy systems. So, XML can be used in a non-proprietary way to move data from legacy systems to distributed systems like JEE over the wire using MOM (i.e. Implementation) and JMS (i.e. Interface).

Q6. What are the components of the JMS architecture?
A6. JMS 2.0 (year 2013) released in Java EE 7 makes the JMS architecture much easier and simpler to work with. The following components are based on JMS 1.1 (i.e. from year 2002).

Message producers: A component that is responsible for creating a message. E.g. QueueSender, and TopicPublisher. An application can have several message

Converting from A to B (6)
Designing your classes & interfac
Java Data Structures & Algorithm
Passing the unit tests (5)
What is wrong with this code? (6)
Writing Code Home Assignement
Written Test Core Java (3)
Written Test JEE (1)

producers. Each producer might be responsible for creating different types of messages and sending them to different destinations (i.e. Topic or Queue). A message producer will send messages to a destination regardless of whether or not a consumer is there to consume it.

Message consumers: A component which resides on the receiving end of a messaging application. Its responsibility is to listen for messages on a destination (i.e. Topic or Queue) . E.g. QueueReceiver, TopicSubscriber, MessageDrivenBean (MDB). A MDB is simply a JMS message consumer. A client cannot access a MDB directly as you would do with Session or Entity beans. You can only interface with a MDB by sending a JMS message to a destination (i.e. Topic or Queue) on which the MDB is listening.

Message destinations: A component which a client uses to specify the target of messages it sends/receives. E.g. Topic (publish/Subscribe domain) and Queue (Point-to-Point domain). Message destinations typically live on a MOM, which is remote to the clients. Message destinations are administered objects that need to be configured.

JMS messages: A message is a component that contains the information (aka payload) that must be communicated to another application or component. E.g. TextMessage (XML file), ObjectMessage (serialized object) etc.

JMS Administered objects: JMS administered objects are objects containing configuration information that are set up during application deployment or configuration and later used by JMS clients. They make it practical to administer the JMS API in the enterprise. These administered objects are initialized when the application server starts. When a producer or a consumer needs to get a connection to receive or send a JMS message, then you need to locate the configured administered objects QueueConnectionFactory or TopicConnectionFactory. Message destinations are administered objects that need to be configured as well. These administered objects hide provider-specific details from JMS clients.

JNDI naming service: For a producer and consumer to be able to use the administered objects to send and receive messages, they must know how to locate things such as the destination and connection factories.

Example: To publish a message to a topic: (Note: exception handling etc are omitted for brevity)

```
1  String factoryJndiName = "WSMQTopicConnectionFactory";
2  String destinationJndiName = "wsmq/topic/ProductManagerTopic";
3
4  //JNDI lookup of administered ConnectionFactory object
5  Context iniCtx = new InitialContext();
6  TopicConnectionFactory topicCF = (TopicConnectionFactory) iniCtx.loo
7
8  //JNDI lookup of administered destination (i.e. Topic)
9  Topic topicDestination = (Topic) iniCtx.lookup(destinationJndiName);
10
11 //get a connection from the TopicConnectionFactory
12 TopicConnection publishConnection = topicCF.createTopicConnection();
13
14 //get a session from the connection. Session should be accessed by o
15 TopicSession publishSession =
16                  publishConnection.createTopicSession(false,TopicSes
17
18 //create a publisher from the session
19 TopicPublisher publisher = publishSession.createPublisher(reqDestina
20
21 //create a JMS message to send
22 TextMessage message = publishSession.createTextMessage();
23 message.setText("JMS test message");
```

```
24
25 //send the message
26 publisher.publish(message, DeliveryMode.NON_PERSISTENT, 4, 0);
27
```

To **consume a message**

```
1  String factoryJndiName = "WSMQTopicConnectionFactory";
2  String destinationJndiName = "wsmq/topic/ProductManagerTopic";
3
4  //JNDI lookup of administered ConnectionFactory object
5  Context iniCtx = new InitialContext();
6  TopicConnectionFactory topicCF = (TopicConnectionFactory) iniCtx.loo
7
8  //JNDI lookup of administered destination (i.e. Topic)
9  Topic topicDestination = (Topic) iniCtx.lookup(destinationJndiName);
10
11 //get a connection from the TopicConnectionFactory
12 TopicConnection subscribeConnection = topicCF.createTopicConnection(
13
14 //get a session from the connection
15 TopicSession subscribeSession =
16                   subscribeConnection.createTopicSession(false,Topic
17
18 //create a subscriber from the session
19 TopicSubscriber subscriber = subscribeSession.createsubscriber(reqDe
20
21 //look for messages every 1 second
22 while (true) {
23     Message response = subscriber.receive();
24
25     if (response != null &amp;amp;&amp;amp; response instanceof TextMessage)
26         System.out.println (((TextMessage) response).getText());
27     }
28
29     Thread.sleep(1000);
30 }
31
```

If you use JEE container with a **Message Driven Bean (MDB)** or Spring Container with a message listener, the container will provide the infrastructure for receiving messages, and invokes the onMessage(Message message) to process the message.

```
1 public void onMessage(Message message) {
2
3     String text = null;
4     if (message instanceof TextMessage) {
5        text = ((TextMessage)message).getText();
6     }
7
8     log.info(text);
9 }
```

Q7. What are the components of the JMS 2.0 architecture introduced in JEE 7?
A7. JMS 2.0 (year 2013) released in Java EE 7 makes the JMS architecture much easier and simpler to work with. The new simplified API require fewer objects like ***JMSContext***, ***JMSProducer***, and ***JMSConsumer***. In JEE, JMSContext can be injected and managed by the container.

The JMS1.1 annotation driven way

```
1  @Resource(lookup = "java:global/jms/myAppConnectionFactory")
2  ConnectionFactory connectionFactory;
3
4  @Resource(lookup = "java:global/jms/myAppQueue")
5  Queue myAppQueue;
6
7  public void sendMessage(String payload) {
8     try {
9        Connection connection = connectionFactory.createConnection();
10       try {
11          Session session = connection.createSession(false,Session.AU
12          MessageProducer messageProducer = session.createProducer(my
13          TextMessage textMessage = session.createTextMessage(payload
```

```
            messageProducer.send(textMessage);
        } finally {
            connection.close();
        }
    } catch (JMSException ex) {
        ex.printStackTrace();
    }
}
```

Drawbacks with JMS 1.1 code:

1) You have to create many intermediary objects like session, messageProducer, and textMessage.
2) Redundant and misleading arguments are used like Session.AUTO_ACKNOWLEDGE
3) Connections must be explicitly closed in the finally block.
4) Exceptions thrown are checked exceptions and need to be handled.

JMS 2.0 is still backward compatible, but has new methods to make things simpler and easier. JMS objects implement java.jang.AutoCloseable, hence requires Java 7.0. It has objects such as **Connection**, **Session**, **MessageProducer**, **MessageConsumer**, and **QueueBrowser**.

```
@@Resource(lookup = "java:global/jms/myAppConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "java:global/jms/myAppQueue")
Queue myAppQueue;

public void sendMessageNew(String payload) {
    try (JMSContext context = connectionFactory.createContext();){
        context.createProducer().send(myAppQueue, payload);
    } catch (JMSRuntimeException ex) {
        Logger.getLogger(getClass().getName()).log(Level.SEVERE, null,
    }
}
```

The pluses for JMS 2.0:

1) The JMSContext combines Connection and Session.
2) Payload can be sent directly without wrapping it in a TextMessage.
3) Runtime exception is thrown and connections are closed automatically with the Java 7.0 AutoCloseable interface.

Q8. What are some of the key message characteristics defined in a message header?
A8.

Characteristic	Explanation
JMSCorrelationID	Used in request/response situations where a JMS client can use the JMSCorrelationID header to associate one message with another. **For example**: a client request can be matched with a response from a server based on the JMSCorrelationID.
JMSMessageID	Uniquely identifies a message in the MOM environment.
JMSDeliveryMode	This header field contains the delivery modes: PERSISTENT or NON_PERSISTENT.
JMSExpiration	This contains the time-to-live value for a message. If it is set to zero, then a message will never expire.
JMSPriority	Sets the message priority but the actual meaning of prioritization is MOM vendor dependent.

JMS Header

Q9. What are the different body types (aka payload types) supported for messages?
A9. All JMS messages are read-only once posted to a queue or a topic.

- **Text message**: body consists of java.lang.String (e.g. XML).

- **Map message**: body consists of key-value pairs.
- **Stream message**: body consists of streams of Java primitive values, which are accessed sequentially.
- **Object message**: body consists of a Serializable Java object.
- **Byte message**: body consists of arbitrary stream of bytes.

Q10. What is a message broker?
A10. A message broker acts as a server in a MOM. A message broker performs the following operations on a message it receives:

- Processes message header information.
- Performs security checks and encryption/decryption of a received message.
- Handles errors and exceptions.
- Routes message header and the payload (aka message body).
- Invokes a method with the payload contained in the incoming message (e.g. calling onMessage(..) method on a Message Driven Bean (MDB)).
- Transforms the message to some other format. For example XML payload can be converted to other formats like HTML etc with XSLT.

Q11. Discuss some of the design decisions you need to make regarding your message delivery and transaction management?
A11. **Acknowledgement mode** and **transaction modes** are used to determine if a message will be lost or re-delivered on failure during message processing by the target application. Acknowledgement modes are set when creating a JMS session.

```
1 InitialContext ic = new InitialContext(…);
2 QueueConnectionFactory qcf =
3                   (QueueConnectionFactory)ic.lookup(“AccountConnectionF
4 QueueConnection qc = qcf.createQueueConnection();
5 QueueSession session = qc.createQueueSession(false, Session.AUTO_ACKN
6
```

In the above code sample, the transaction mode is set to false and acknowledgement mode is set to auto mode. Let us look at acknowledgement modes:

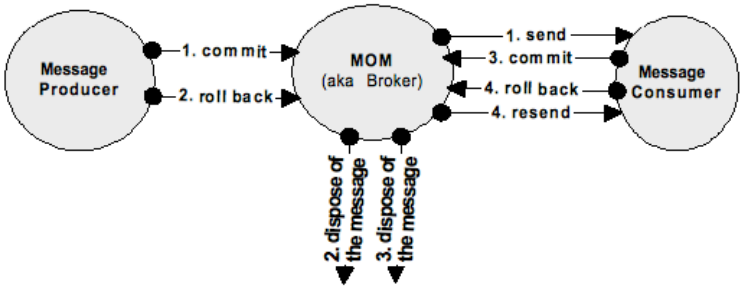
AUTO_ACKNOWLEDGE: The messages sent or received from the session are automatically acknowledged. This mode also guarantees once only delivery. If a failure occurs while executing onMessage() method of the destination MDB, then the message is re-delivered. A message is automatically acknowledged when it successfully returns from the onMessage(…) method.

DUPS_OK_ACKNOWLEDGE: This is just like AUTO_ACKNOWLEDGE mode, but under rare circumstances like during failure recovery messages might be delivered more than once. If a failure occurs then the message is re-delivered. This mode has fewer overheads than AUTO_ACKNOWLEDGE mode.

CLIENT_ACKNOWLEDGE: The messages sent or received from sessions are not automatically acknowledged. The destination application must acknowledge the message receipt. This mode gives an application full control over message acknowledgement at the cost of increased complexity. This can be acknowledged by invoking the acknowledge() method on javax.jms.Message class.

Transactional behavior is controlled at the session level. When a session is transacted, the message oriented middleware (MOM) stages the message until the client either commits or rolls back the transaction. The completion of a session’s current transaction automatically begins a new transaction.

The use of transactions in messaging affects both the producers and consumers of the messages as shown below:



JMS Transaction

Producers [As per the above diagram]:
1's Commit: The MOM send the group of messages that have been staged.
2's Rollback: The MOM disposes of the group of messages that have been staged.

Consumers [As per the above diagram] :
3's Commit: The MOM disposes of the group of messages that have been staged.
4's Rollback: The MOM resends the group of messages that have been staged.

In JMS, a **transaction organizes a message or a group of messages into an atomic processing unit**. So, if a message delivery is failed, then the failed message may be re-delivered. Calling the commit() method commits all the messages the session receives and calling the rollback method rejects all the messages.

```
1 InitialContext ic = new InitialContext(…);
2 QueueConnectionFactory qcf =
3                 (QueueConnectionFactory)ic.lookup(“AccountConnectionFa
4 QueueConnection qc = qcf.createQueueConnection();
5 QueueSession session = qc.createQueueSession(true, -1);
6
```

In the above code sample, the transaction mode is set to true and acknowledgement mode is set to -1, which means acknowledgement mode has no use in this mode. Let us look at transaction modes:

Message Driven Bean (MDB) with container managed transaction demarcation:
A MDB participates in a container transaction by specifying the transaction attributes in its deployment descriptor. A transaction automatically starts when the JMS provider removes the message from the destination and delivers it to the MDB's onMessage(…) method. Transaction is committed on successful completion of the onMessage() method. A MDB can notify the container that a transaction should be rolled back by setting the MessageDrivenContext to setRollBackOnly(). When a transaction is rolled back, the message is re-delivered.

```
1  public void onMessage(Message aMessage) {
2    …
3    if(someCondtionIsTrue) {
4       mdbContext.setRollbackOnly();
5    }
6    else{
7      //everything is good. Transaction will be committed automaticall
8       //completion of onMessage(..) method.
9    }
10 }
11
```

Message Driven Bean (MDB) with bean managed transaction demarcation: If a MDB chooses not to participate in a container managed transaction then the MDB programmer has to design and code programmatic transactions. This is achieved by creating a UserTransaction object from the MDB's MessageDrivenContext as shown below and then invoking the commit() and rollback() methods on this UserTransaction object.

```
1  public void onMessage(Message aMessage) {
2
3       UserTransaction uT = mdbContext.getUserTransaction();
4       uT.begin();
5       ….
6       if(someCondtionIsTrue) {
7          uT.rollback();
8       }
9       else{
10         uT.commit();
11      }
12 }
13
```

Transacted session: An application completely controls the message delivery by either committing or rolling back the session. An application indicates successful message processing by invoking Session class's commit() method. Also it can reject a message by invoking Session class's rollback() method. This committing or rollback is applicable to all the messages received by the session.

```
1  public void process(Message aMessage, QueueSession qs) {
2      ….
3     if(someCondtionIsTrue) {
4         qs.rollback();
5     }
6     else{
7         qs.commit();
8     }
9     …
10 }
11
```

Q. What happens to rolled-back messages?
A. Rolled back messages are re-delivered based on the re-delivery count parameter set on the JMS provider. The re-delivery count parameter is very important because some messages can never be successful and this can eventually crash the system. When a message reaches its re-delivery count, the JMS provider can either log the message or forward the message to an error destination. Usually it is not advisable to retry delivering the message soon after it has been rolled-back because the target application might still not be ready. So we can specify a time to re-deliver parameter to delay the re-delivery process by certain amount of time. This time delay allows the JMS provider and the target application to recover to a stable operational condition.

Care should be taken not to make use of a single transaction when using the JMS request/response paradigm where a JMS message is sent, followed by the synchronous receipt of a reply to that message. This is because a JMS message is not delivered to its destination until the transaction commits, and the receipt of the reply will never take place within the same transaction.

Q12. How does XML over HTTP compare with XML using JMS?
A12.

XML over HTTP: Simpler to implement, widely compatible and has less performance overhead but HTTP does not provide reliability in terms of guaranteed delivery because there is no message persistence, no inherent reporting facility for failed

message delivery and no guaranteed once only delivery. The application programmer must build these services into the application logic to provide reliability & persistence, which is not an easy task.

XML over JMS is reliable, scalable, loosely couples systems and robust. The main disadvantage of this approach is that the JMS providers (i.e. Message Oriented Middleware) use a **proprietary protocol** between producer and consumer. So, to communicate, you and your partners need to have the same MOM software (E.g. WebMethods). JMS allows you to toss one MOM software and plug-in another but you cannot mix providers without having to buy or build some sort of a bridge.

Q13. Why favor using XML with JMS?
A13.

#1. Organizations can leverage years or even decades of investment in Business-to-Business (B2B) Electronic Data Interchange (EDI) by using JMS with XML. **XML is an open standard and it represents the data in a non-proprietary way**.

#2. Sending XML messages as text reduces coupling even more compared to sending serializable objects. XML also solves the data representation differences with XML based technologies such as XSLT. For example, the way “Enterprise X” defines a purchase order will be different from the way “Enterprise Y” defines it. So the representation of XML message by “Enterprise X” can be transformed into the format understood by “Enterprise Y” using XSLT.

#3. Both enterprises may be using different applications to run their business. For example Enterprise “X” may be using Java/JEE, while “Enterprise Y” may be using SAP. XML can solve the data formatting problems since it is an open standard with a self describing data format, which allows the design of business specific markup languages and standards like FIXML (Financial Information eXchange Markup Language), FpML (Financial products Markup Language – derivative products), WML (Wireles Markup Language – for wireless devices), SAML (Security Assertion Markup Language), etc. The structure of an XML document is similar to that of business objects with various attributes. This allows for the natural conversion of application-specific objects to XML documents and vice versa.

#4. XML digital signature technology can be used to provide authentication, data integrity (tamper proofing) and non-repudiation. Unlike SSL, XML encryption can be used to encrypt and decrypt a section of a data. For example encrypt only the credit card information in a purchase order XML document.

You also need to consider sending messages across each organization’s corporate firewall. Not every organization will open a port in the firewall other than the well-known port 80 for HTTP traffic. The solution is to make use of HTTP tunneling, which involves sending the data as HTTP traffic through well-known port number 80 for HTTP and then, once inside the firewall, convert this data into messages. For example JProxy is a JEE based HTTP tunnel with SSL and JAAS with support for EJB, RMI, JNDI, JMS and CORBA.

Q14. Why do you need AMQP when there is JMS?
A14. AMQP stands for Advanced Message Queuing Protocol, and was developed to address the problem of interoperability by creating a standard for how messages should be structured and transmitted between platforms the same way as SMTP, HTTP, FTP, etc. have created interoperable systems. This standard binary wire level

protocol for messaging would therefore allow hetrogeneous disparate systems between and within companies to exchange messages regrdless of the message broker vendor or platform.

RabbitMQ, Apache Qpid, StormMQ, etc are open source message broker softwares (i.e. MOM – message-oriented middlewares) that implements the Advanced Message Queuing Protocol (AMQP).

Q15. How does AMQP differ from JMS?
A15. JMS is a standard messaging API for the Java platform. It provides a level of abstraction that frees developers from having to worry about specific implementation and wire protocols. This is similar to the JDBC API that allows you to easily switch databases. With JMS, you can switch from one JMS complian message broker (e.g. Web Methods) with another one (e.g. MQSeries or WebspehreMQ) with little or no changes to your source code. It also provides interoperability between other JVM based languages like Scala and Groovy. Altough JMS brokers can be used in .NET applications, the whole JMS specification does not guarantee interoperability, and integration between Java to .NET or Java to Ruby, is proprietary and can be quite tricky. In scenarios where you want to send a message from a Java based message producer to a .NET based message consumer, then you need a message based cross platform interoperability that is what AMQP does. With AMQP, you can use any AMQP compliant client library, and AMQP compliant message broker.

Q16. Can you explain the following messaging terms?

1) Poison messages.
2) Queue depth.
3) Message correlation id
4) Dead letter queue

A16.

1. **Poison messages**, are messages the application can never successfully process due to being badly-formatted. Such a message might make the receiving application fail and back out the receipt of the message. In this situation, such a message might get into an enless llop where get received, and then returned to the queue, repeatedly. These messages are known as poison messages. The ***ConnectionConsumer*** must be able to detect poison messages and reroute them to an alternative destination.

2. **Queue Depth** means the number of messages in a Queue. The JMS API does not provide an explicit method to pro-grammatically determine the queue depth. You need to rely on vendor specific plugins to pro-grammatically determine this. Queue depth need to be monitored in production systems to raise alerts if they get closer to their capacity. MOM provider specific tools can monitor queue depths.

3. There are different JMS messaging patterns like send only messages, receive only messages, request-response messages, etc. When you have JMS request-response messages, you need to correlate a response message with a request message. Messaging is highly distributed, hence **correlation ids** are handy to trace messages. So, MessageID and CorrelID fields are very handy for trace-ability.

4. **Failed messages are recorded in a dead-letter queue**. The failed delivery can be caused by reasons such as network issues, poison messages, queue not found, queue is full, etc

Differences between JMS 1.1 and JMS 2.0

JMS 1.1	JMS 2.0

`MessageProducer producer = session.createProducer();`
`producer.send(destination,message);`
	`JMSProducer producer = context.createProducer();`
	`producer.send(destination,message);`
	JMSProducer is lighter weight and supports method chaining.

`MessageProducer producer = session.createProducer();`
`producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);`
`producer.setPriority(1);`
`producer.setTimeToLive(2000);`
`producer.send(destination,message);`
Verbose with repeated “producer”.
	`context.createProducer()`
	`.setDeliveryMode(DeliveryMode.NON_PERSISTENT)`
	`.setPriority(1)`
	`.setTimeToLive(1000)`
	`.send(destination,message);`
	Less verbose with the “Builder design pattern”.

Extracting message body from the payload
`Message message = consumer.receive(2000);`
`TextMessage textMessage = (TextMessage) message;`
`String body = textMessage.getText();`
	Extracting message body from the payload
	`Message message = consumer.receive(1000);`
	`String body = message.getBody(String.class);`

In JEE 7, The **JMSContext** objects can be injected into Web or EJB containers with the @Inject annotation.

```
@Inject
@JMSConnectionFactory("jms/myAppConnectionFactory")
@JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
private JMSContext context;

@Resource(mappedName = "jms/myAppQueue")
private Queue myAppQueue;

public void sendMessage (String payload) {
   context.createProducer().send(myAppQueue, payload);
}
```

JMS 2.0 has made the message property “JMSXDeliveryCount” to mandatory from being optional in JMS 1.0 to handle poisonous messages.

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

766 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ ♦ 12 FAQ JDBC interview questions and answers

8 Java Annotations interview Questions and Answers ›

Posted in FAQ JEE Job Interview Q&A Essentials, JMS, member-paid

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

↑