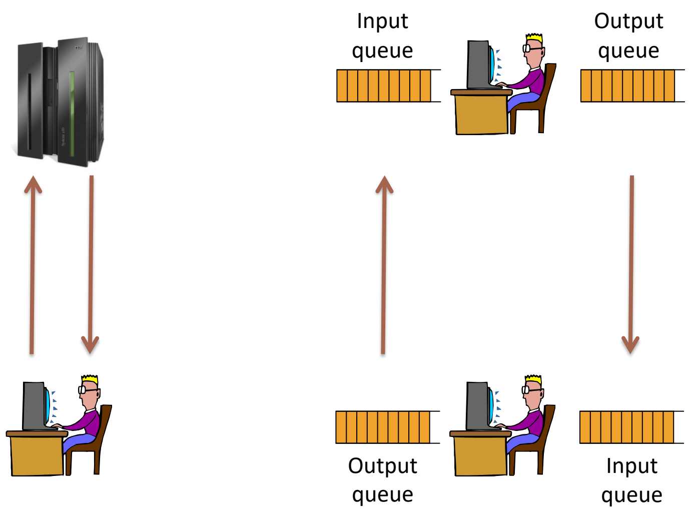
MESSAGE-ORIENTED MIDDLEWARE

10

10

RPC vs Queues

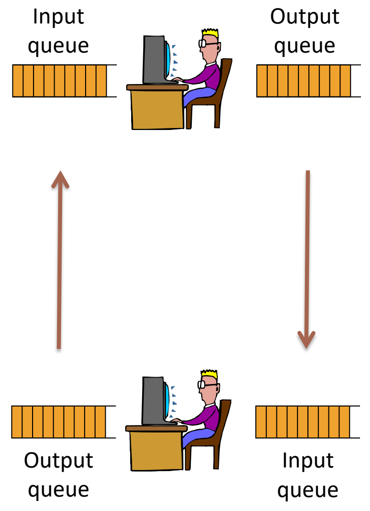


11

11

RPC vs Queues

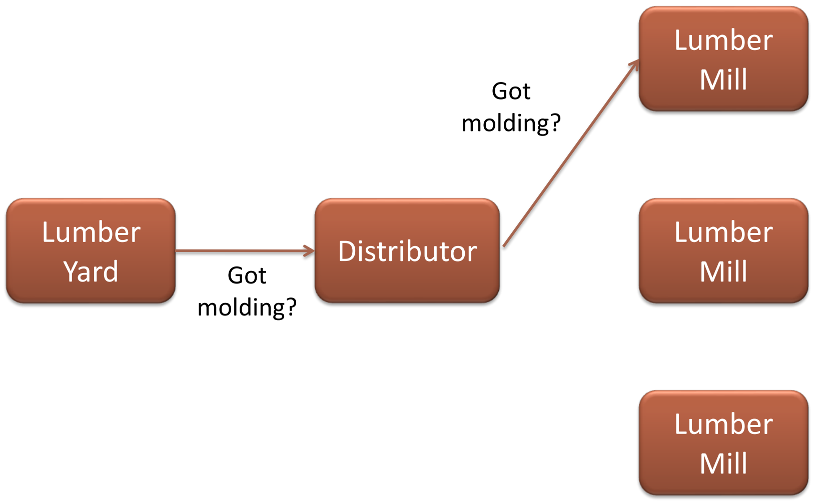
- Reliable queues
  - Persistent
  - Transactional
- Loose coupling
- Ex: IBM MQSeries, Tuxedo/Q



12

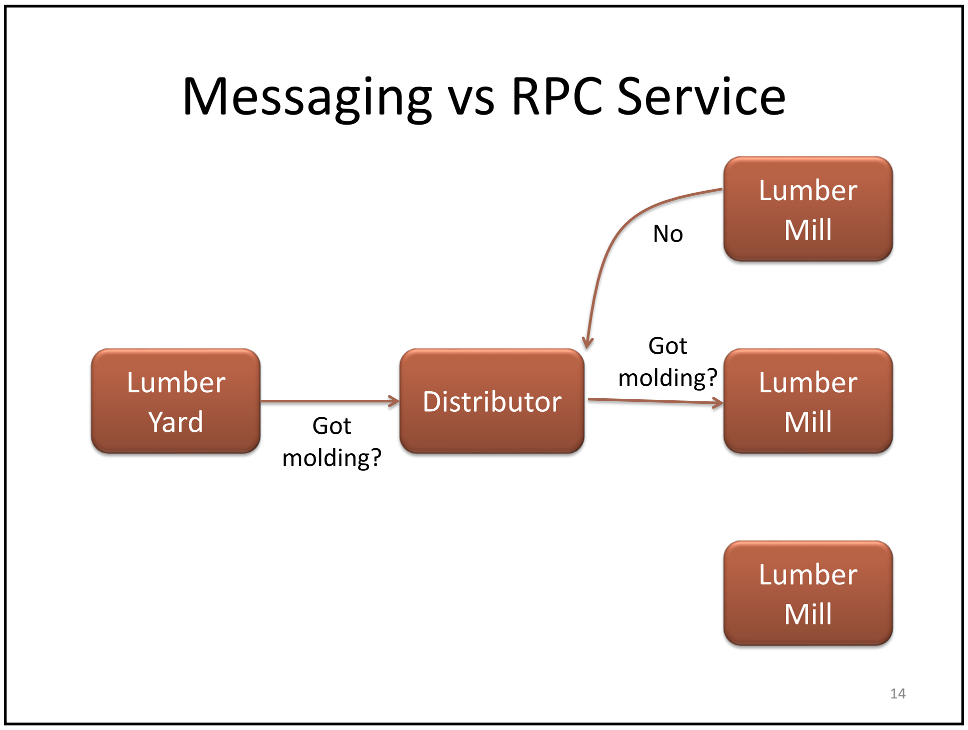
12

Messaging vs RPC Service



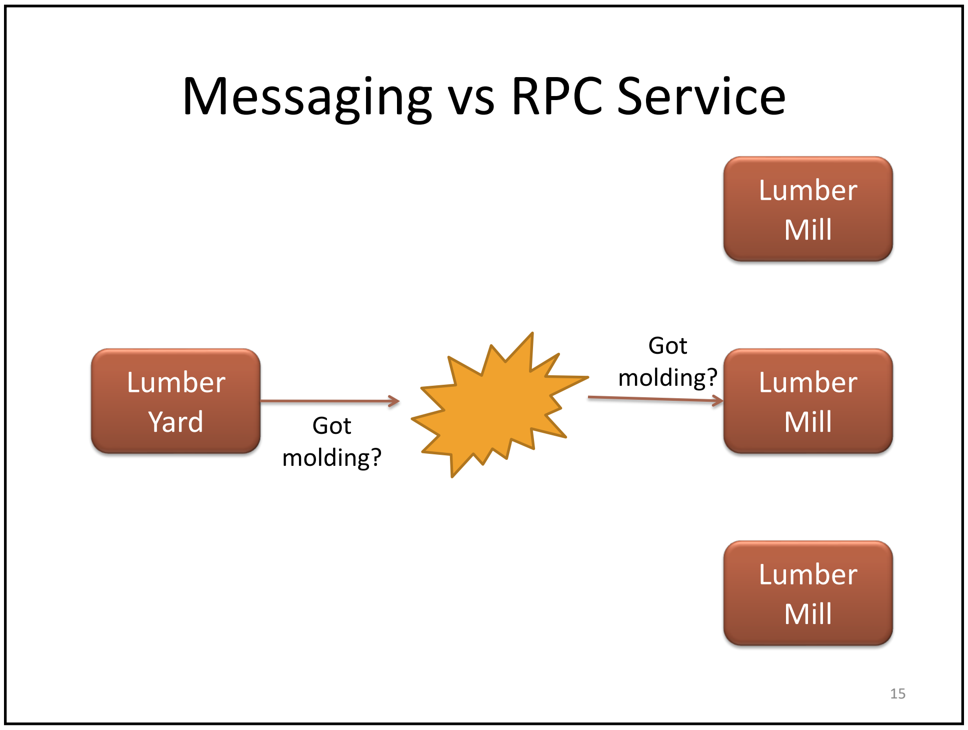
13

13



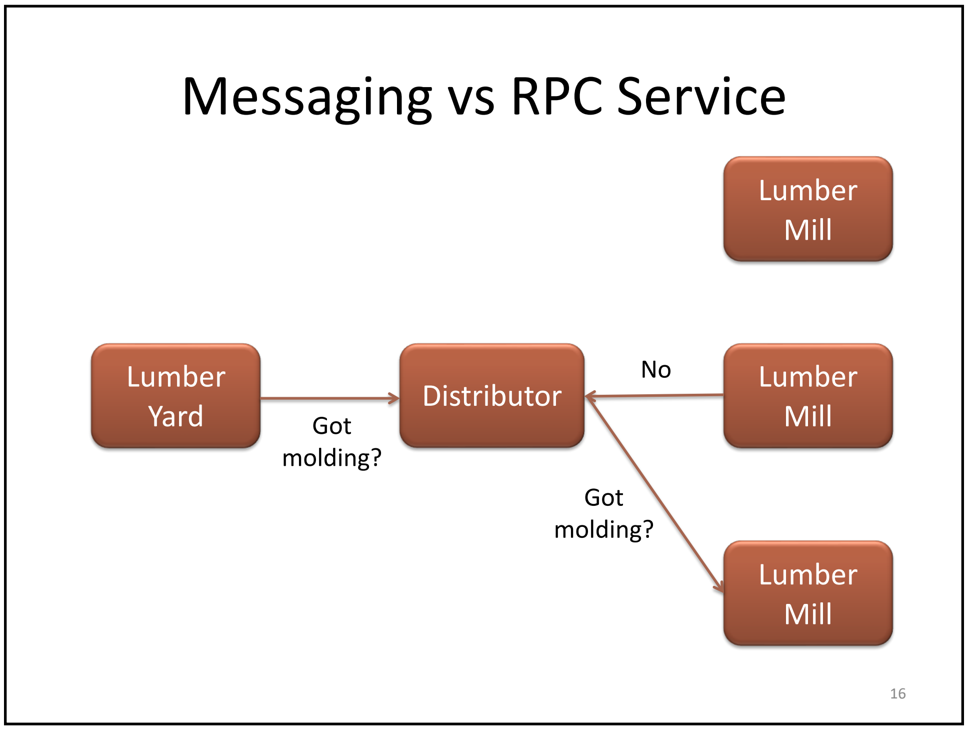
Messaging vs RPC Service

14



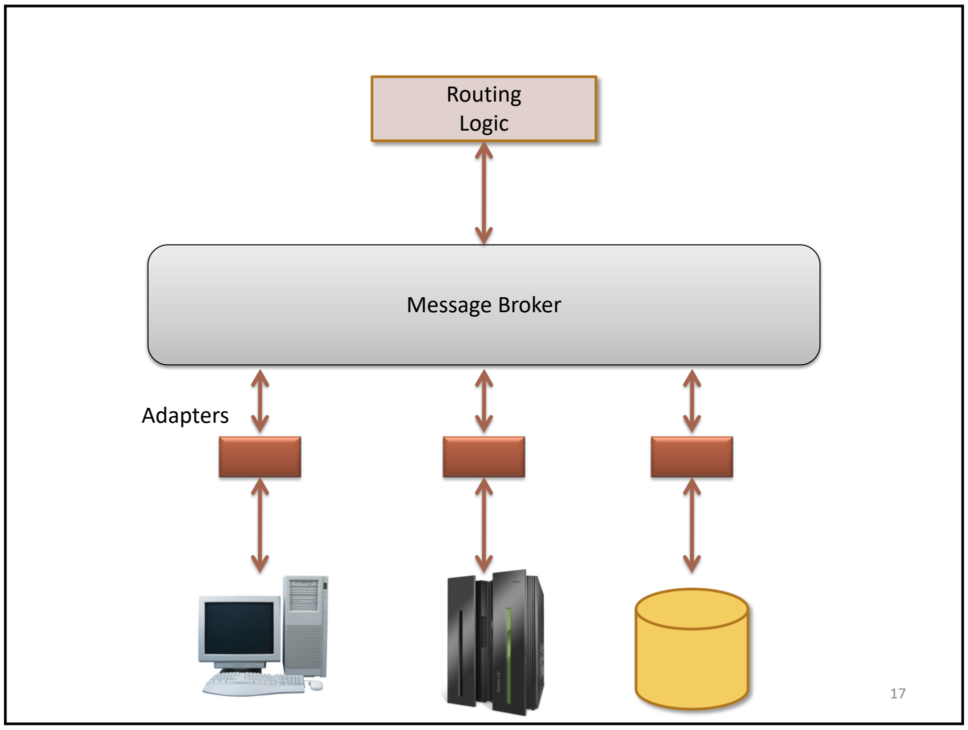
Messaging vs RPC Service

15



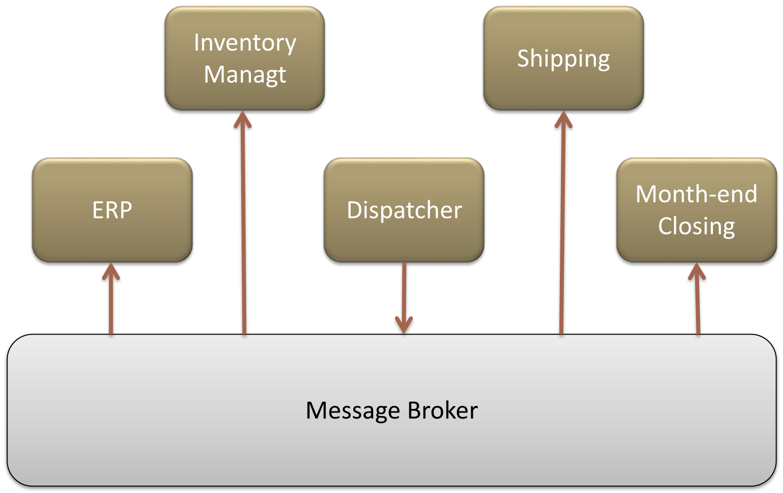
Messaging vs RPC Service

16



17

Publish-Subscribe

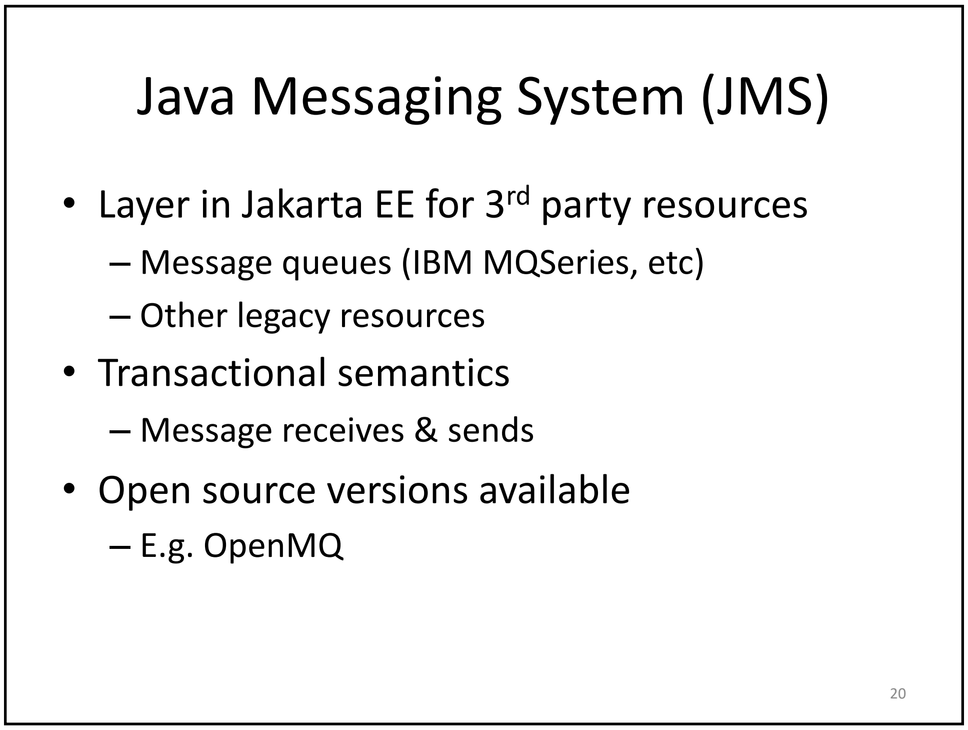


18

18

JAVA MESSAGE SYSTEM (JMS)

19



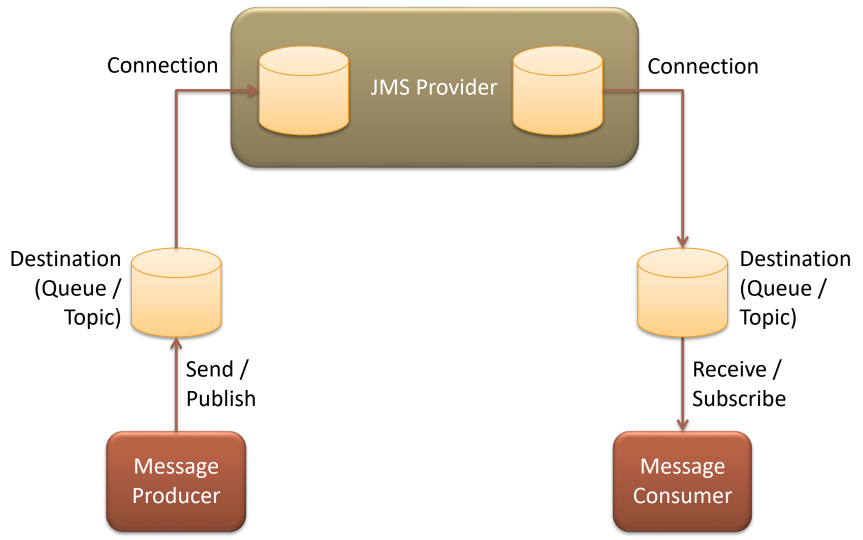
Java Messaging System (JMS)

- Layer in Jakarta EE for 3rd party resources
  - Message queues (IBM MQSeries, etc)
  - Other legacy resources
- Transactional semantics
  - Message receives & sends
- Open source versions available
  - E.g. OpenMQ

20

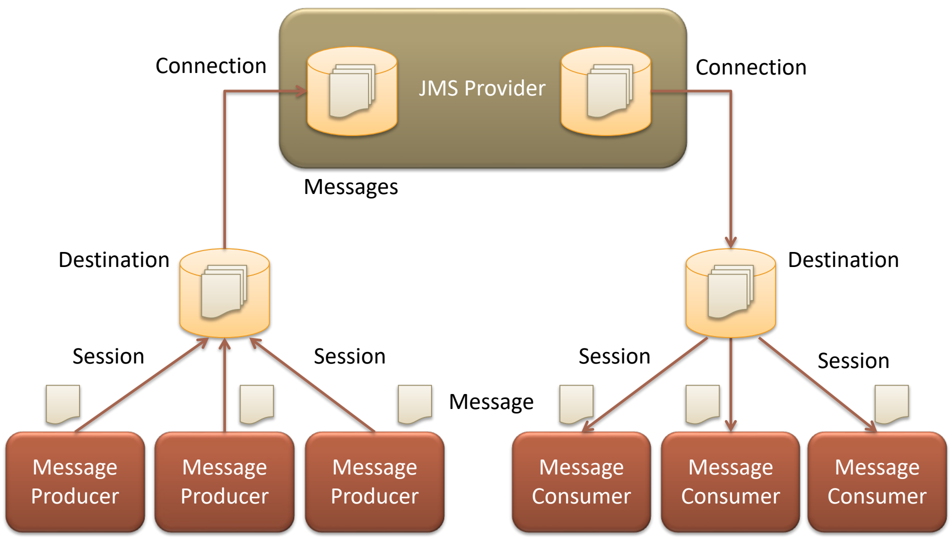
20

JMS Architecture



21

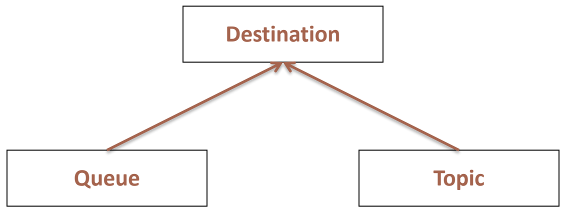
JMS Architecture



22

JMS API: Destinations

- Destinations:
  - Queue: point-to-point messaging domain destination
  - Topic: publish-subscribe messaging domain destination
- Destinations are accessed using JNDI

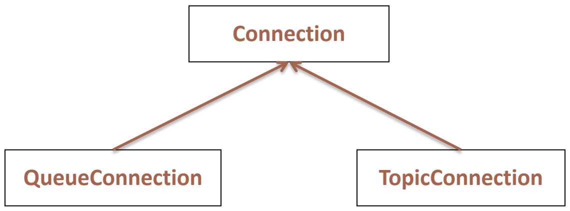


23

23

JMS API: Connections

- Connections:
  - From JMS client to JMS provider
  - Connection has to be started before use
- Connection Factory
  - Accessed via JNDI

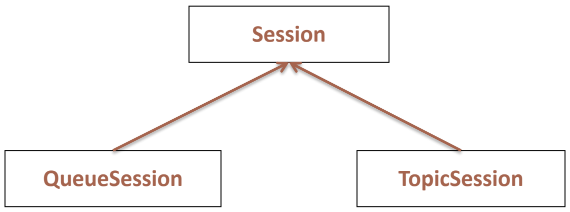


24

24

JMS API: Sessions

- Single-threaded context for sending & receiving
- Transactional: combine send and receive operations
- Created from connections

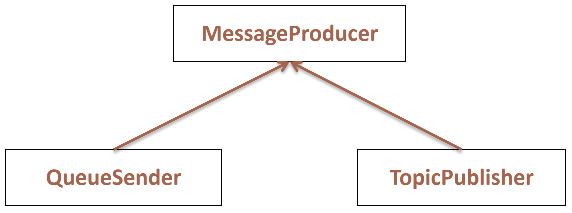


25

25

JMS API: Message Producers

- Used to send or publish message
- Create from session object (`createProducer`)
- Producer associated with destination (queue or topic)



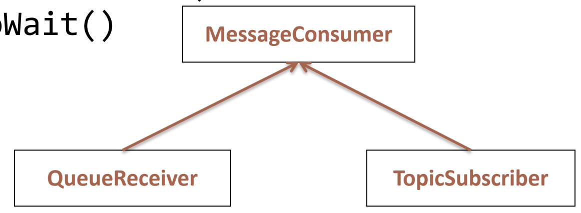
26

26

JMS API: Message Consumers

- Used to receive messages
- Create from session object (`createConsumer`)
- Consumer associated with destination (queue or topic)
- Receipt of message with:

```
receive()
receive(int timeout)
receiveNoWait()
```



27

27

JMS API: Message Listeners

- Asynchronous receipt of messages: Interface for objects that listen for arrival of messages

```
interface MessageListener {
  void onMessage(Message msg);
}
```

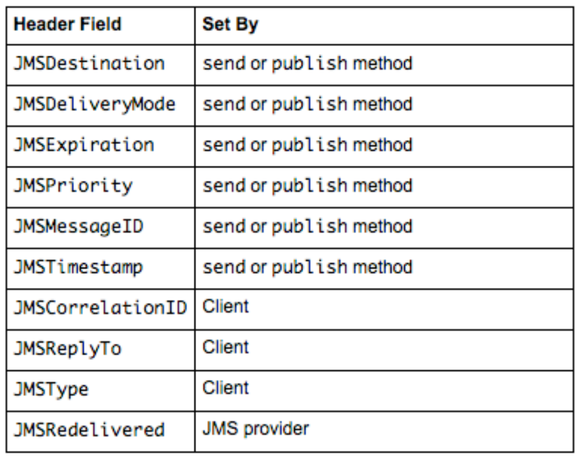
- Can be registered both in a QueueReceiver and a TopicSubscriber
- Per session only one listener is active (single threaded sessions)

28

28

Message Headers

Table 31-1 How JMS Message Header Field Values Are Set



Header Field	Set By
JMSDestination	send or publish method
JMSDeliveryMode	send or publish method
JMSExpiration	send or publish method
JMSPriority	send or publish method
JMSMessageID	send or publish method
JMSTimestamp	send or publish method
JMSCorrelationID	Client
JMSReplyTo	Client
JMSType	Client
JMSRedelivered	JMS provider

29

29

JMS Message Types

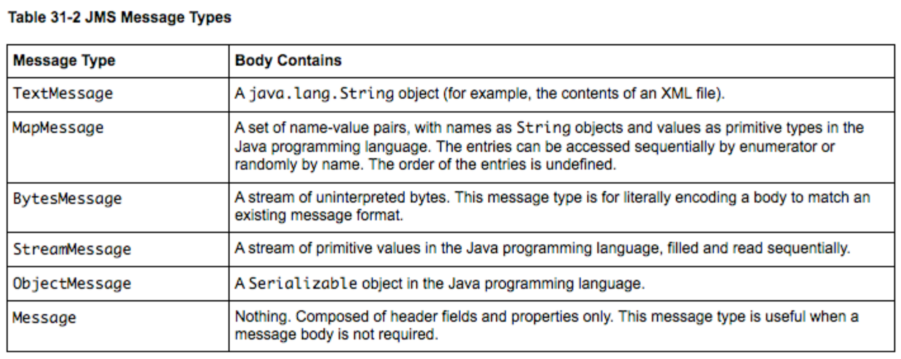


Table 31-2 JMS Message Types

Message Type	**Body Contains**
`TextMessage`	A `java.lang.String` object (for example, the contents of an XML file).
`MapMessage`	A set of name-value pairs, with names as `String` objects and values as primitive types in the Java programming language. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.
`BytesMessage`	A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.
`StreamMessage`	A stream of primitive values in the Java programming language, filled and read sequentially.
`ObjectMessage`	A `Serializable` object in the Java programming language.
`Message`	Nothing. Composed of header fields and properties only. This message type is useful when a message body is not required.

30

30

Acknowledgements

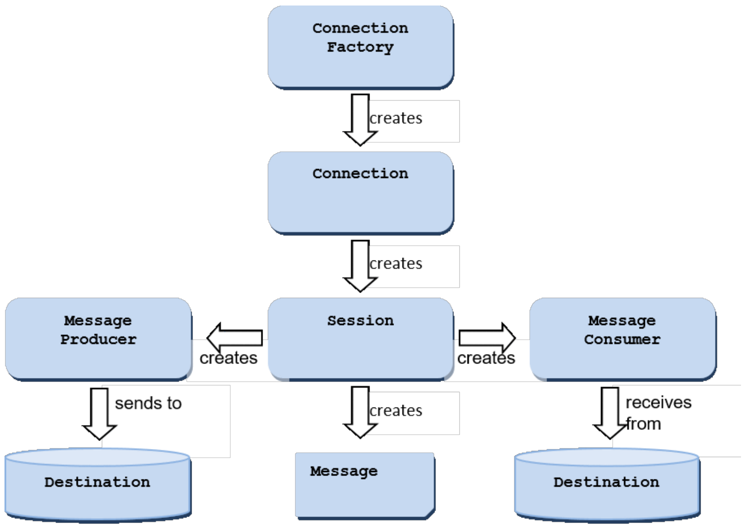
- Transactional session
  - Acknowledged automatically on txn commit

- Non-transactional session
  - AUTO_ACKNOWLEDGE mode
  - CLIENT_ACKNOWLEDGE mode
  - DUPS_OK_ACKNOWLEDGE mode

31

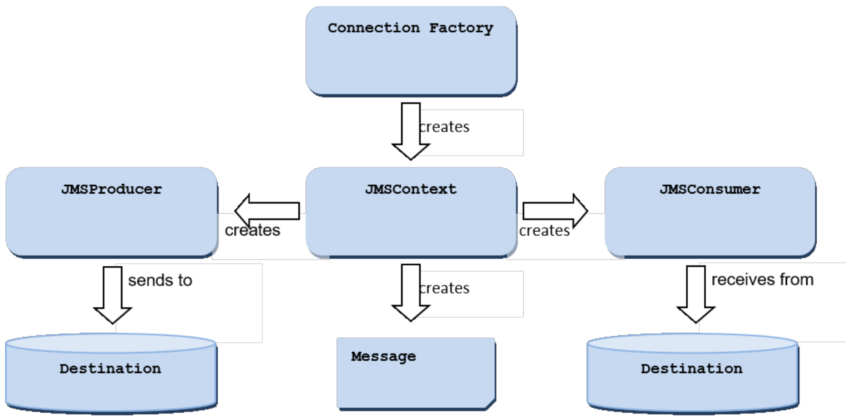
31

Classic JMS API



32

Simplified JMS API



33

JMS EXAMPLES

34

34

Message Producer

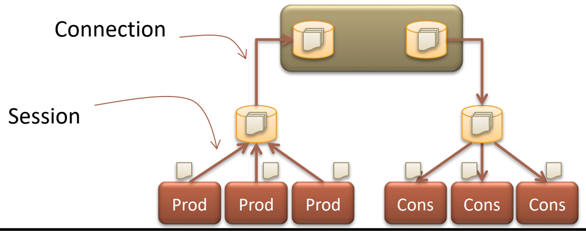
```
@Resource(mappedName = "jms/clinic/ConnectionFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/clinic/Treatment")
private static Topic topic;
```

35

35

Message Producer

```
@Resource(mappedName = "jms/clinic/ConnectionFactory")
private ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/clinic/Treatment")
private Topic topic;

// Creates the needed artifacts to connect to the queue
Connection connection = connectionFactory.createConnection();
Session session =
       connection.createSession(false,
                                Session.AUTO_ACKNOWLEDGE);
MessageProducer producer = session.createProducer(topic);
```



36

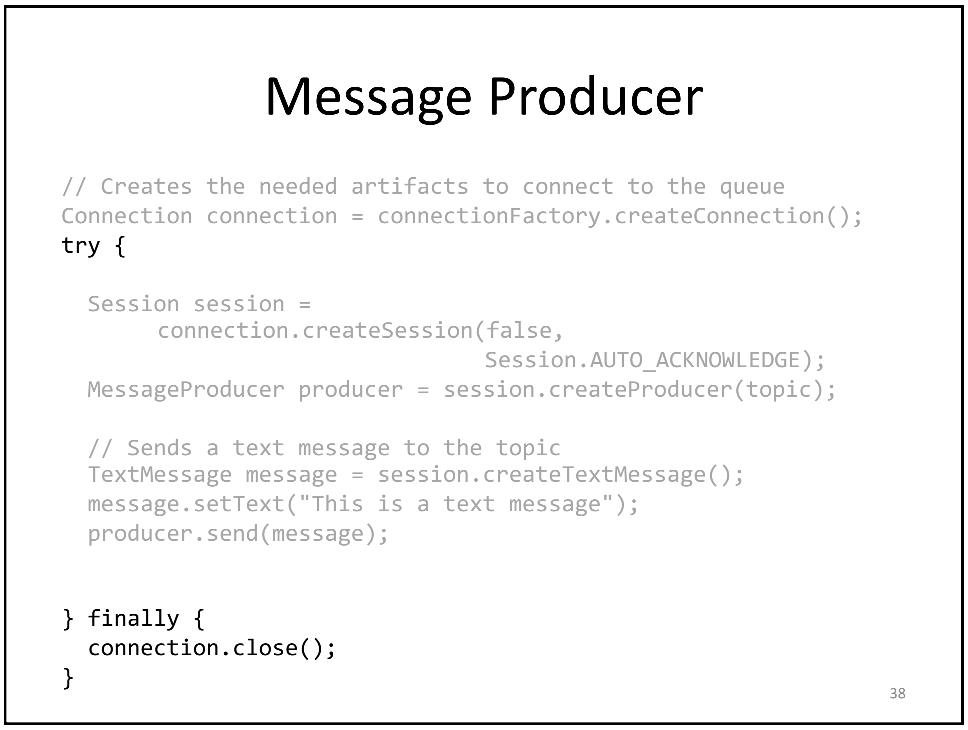
36

Message Producer

```
@Resource(mappedName = "jms/clinic/ConnectionFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/clinic/Treatment")
private static Topic topic;

// Creates the needed artifacts to connect to the queue
Connection connection = connectionFactory.createConnection();
Session session =
       connection.createSession(false,
                                Session.AUTO_ACKNOWLEDGE);
MessageProducer producer = session.createProducer(topic);

// Sends a text message to the topic
TextMessage message = session.createTextMessage();
message.setText("This is a text message");
producer.send(message);
connection.close();
```

37

37

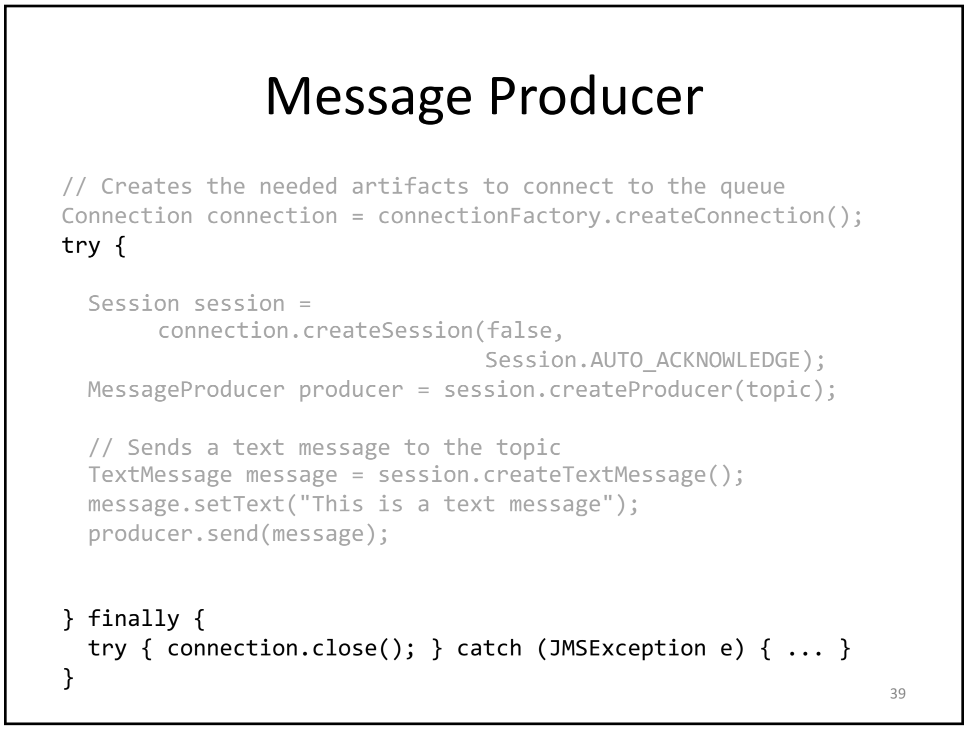


Message Producer

```
// Creates the needed artifacts to connect to the queue
Connection connection = connectionFactory.createConnection();
try {

  Session session =
       connection.createSession(false,
                                Session.AUTO_ACKNOWLEDGE);
  MessageProducer producer = session.createProducer(topic);

  // Sends a text message to the topic
  TextMessage message = session.createTextMessage();
  message.setText("This is a text message");
  producer.send(message);


} finally {
  connection.close();
}
```

38

38

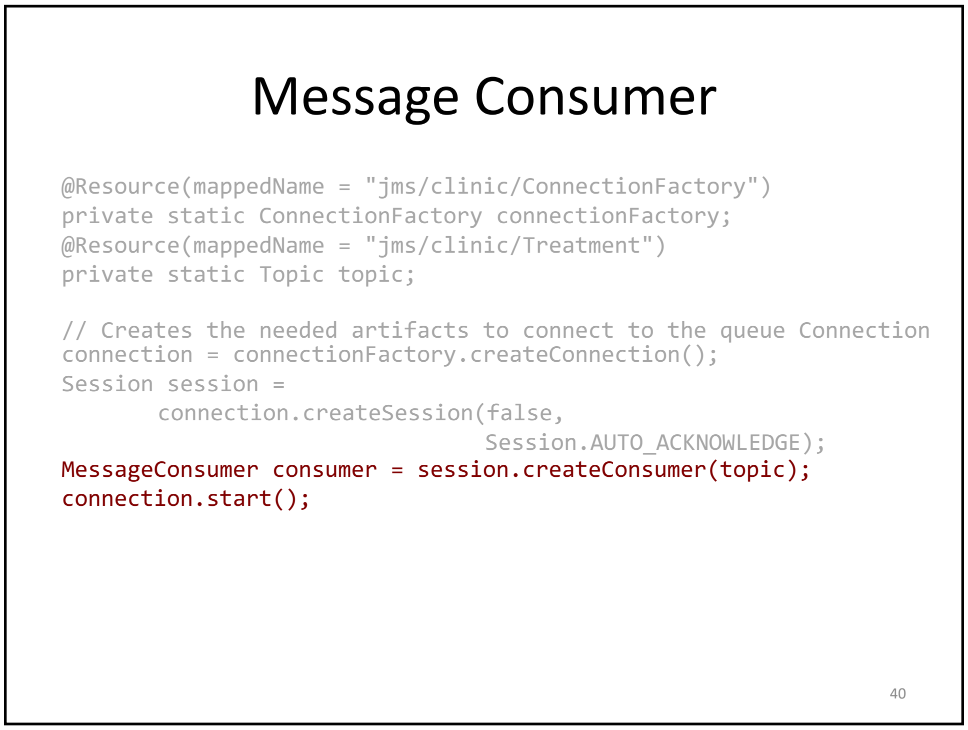


Message Producer

```
// Creates the needed artifacts to connect to the queue
Connection connection = connectionFactory.createConnection();
try {

  Session session =
       connection.createSession(false,
                                Session.AUTO_ACKNOWLEDGE);
  MessageProducer producer = session.createProducer(topic);

  // Sends a text message to the topic
  TextMessage message = session.createTextMessage();
  message.setText("This is a text message");
  producer.send(message);


} finally {
  try { connection.close(); } catch (JMSException e) { ... }
}
```

39

39

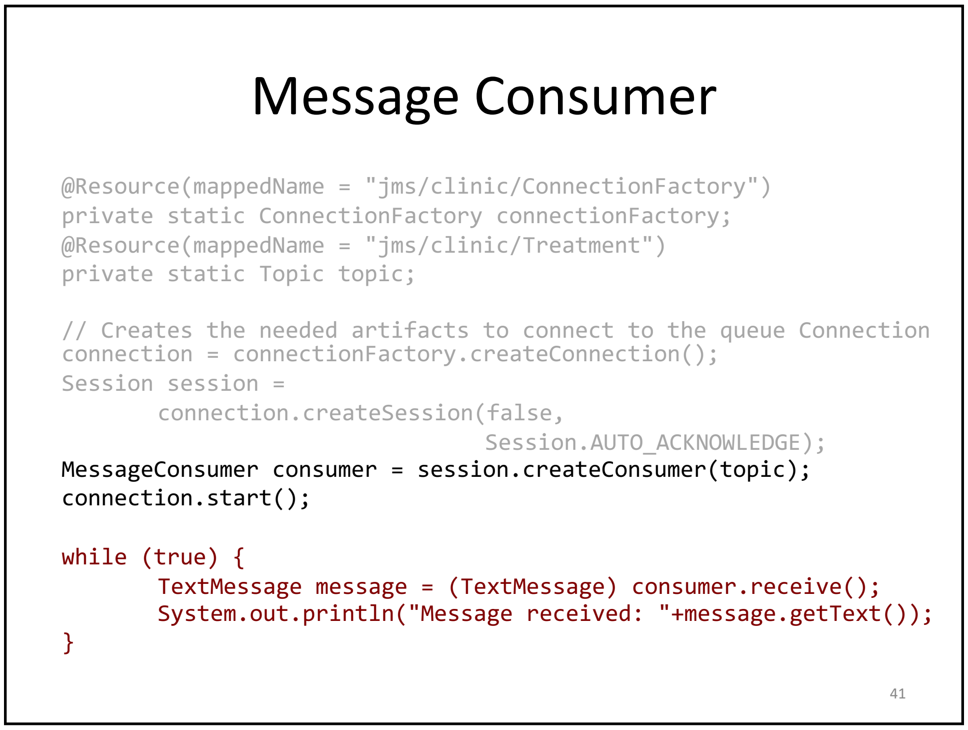


Message Consumer

```
@Resource(mappedName = "jms/clinic/ConnectionFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/clinic/Treatment")
private static Topic topic;

// Creates the needed artifacts to connect to the queue Connection
connection = connectionFactory.createConnection();
Session session =
       connection.createSession(false,
                                Session.AUTO_ACKNOWLEDGE);
MessageConsumer consumer = session.createConsumer(topic);
connection.start();
```

40

40



Message Consumer

```
@Resource(mappedName = "jms/clinic/ConnectionFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/clinic/Treatment")
private static Topic topic;

// Creates the needed artifacts to connect to the queue Connection
connection = connectionFactory.createConnection();
Session session =
       connection.createSession(false,
                                Session.AUTO_ACKNOWLEDGE);
MessageConsumer consumer = session.createConsumer(topic);
connection.start();

while (true) {
       TextMessage message = (TextMessage) consumer.receive();
       System.out.println("Message received: "+message.getText());
}
```

41

41

Message Consumer

```
@Resource(mappedName = "jms/clinic/ConnectionFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/clinic/Treatment")
private static Topic topic;

// Creates the needed artifacts to connect to the queue Connection
connection = connectionFactory.createConnection();
Session session =
       connection.createSession(false,
                                Session.AUTO_ACKNOWLEDGE);
MessageConsumer consumer = session.createConsumer(topic);
Consumer.setMessageListener(new TreatmentListener());
connection.start();
```

42

42

Message Consumer

```
@Resource(mappedName = "jms/clinic/ConnectionFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/clinic/Treatment")
private static Topic topic;

// Creates the needed artifacts to connect to the queue Connection
connection = connectionFactory.createConnection();
Session session =
       connection.createSession(false,
                                Session.AUTO_ACKNOWLEDGE);
MessageConsumer consumer = session.createConsumer(topic);
Consumer.setMessageListener(new TreatmentListener());
connection.start();

public class TreatmentListener implements MessageListener {
  public void onMessage(Message message) {
    System.out.println("Message received: " + ...;
  }
}
```

43

43

Selectors

- In receiver:

```
session.createConsumer(“JMSPriority < 3”);
session.createConsumer(“treatmentType=`Drug’”);
session.createConsumer
   (“JMSPriority<3 AND “treatmentType=`Drug’”);
```

- In producer:

```
message.setJMSPriority(2);
message.setStringProperty(“treatmentType”,
                          “Drug”);
```

44

44

MESSAGE-DRIVEN BEANS

45

45

Message-Driven Bean

- Producer:

```
@Resource(mappedName = "jms/clinic/Treatment")
private static Topic topic;
...
MessageProducer producer = session.createProducer(topic);
...
producer.send(message);
```

- Consumer

```
@MessageDriven(mappedName = "jms/clinic/Treatment")
public class TreatmentListener implements MessageListener {
  public void onMessage(Message message) {
       TextMessage msg = (TextMessage)message;
       System.out.println("Message received: "+msg.getText());
  }
}
```

46

46

Message-Driven Bean

```
@MessageDriven(mappedName = "jms/clinic/Treatment",
  activationConfig = {
    @ActivationConfigProperty(
       propertyName = "messageSelector",
       propertyValue = "treatmentType=`Drug’")
)
public class TreatmentListener implements MessageListener {
  public void onMessage(Message message) {
       TextMessage msg = (TextMessage)message;
       System.out.println("Message received: "+msg.getText());
  }
}
```

47

47

Resource Injection in MDB

- Persistence Context

  `@PersistenceContext EntityManager em;`
- Service Bean

  `@EJB ProviderBean provider;`
- Resources e.g. connection factory

  `@Resource ConnectionFactory cf;`
- MDB context

  `@Resource MessageDrivenContext context;`

48

48

Resource Injection in MDB

- Persistence Context

  `@PersistenceContext EntityManager em;`
- Service Bean

  `@EJB ProviderBean provider;`
- Resources e.g. connection factory

  `@Resource ConnectionFactory cf;`
- **MDB context**

  `@Resource MessageDrivenContext context;`

49

49

MDB Context

- Methods in `MessageDrivenContext`:
  - `getRollbackOnly()`
  - `getUserTransaction()`
  - `setRollbackOnly()`
  - `Lookup()`
  - `getCallerPrincipal()`
  - `isCallerInRole()`

50

50

Transactional Queues

- Container starts a new transaction when starting onMessage() execution
  - Transaction commits when method returns
  - Method body can roll back txn:

```
@Resource MessageDrivenContext context;
context.setRollbackOnly();
```

- Transactional context by default propagates to called EJB methods
- Any messages produced are not “visible” to clients until the txn commits

51

51