



Lập trình phân tán theo chủ đề - JMS

Cao Tuấn Dũng
Trịnh Tuấn Đạt
Vũ Hương Giang



Nội dung

- 1. Truyền thông điệp là gì
 - Các mô hình, tính tin dùng, giao dịch, truyền thông điệp phân tán, an toàn.
- 2. Tại sao cần truyền thông điệp
- 3. JMS là gì
- 4. Kiến trúc của JMS
- 5. APIs Lập trình JMS
- 6. Các bước viết JMS clients (bộ gửi và nhận)
- 7. Các đặc tính không có trong JMS



1. Truyền thông điệp là gì?



Truyền thông điệp (messaging)

- Mô hình trong đó các ứng dụng liên kết lỏng với nhau thông qua việc chuyển giao các thông điệp *tự - mô tả*.



Hệ thống truyền thông điệp

- Truyền thông kết nối lỏng (Loose coupled)
- Truyền thông không đồng bộ
- Thông điệp là phương tiện giao tiếp giữa các ứng dụng
- Các thuật ngữ gần nghĩa:
 - MOM (Message Oriented Middleware)
 - Messaging system
 - Messaging server
 - Messaging provider
 - JMS provider



Thực thể truyền thông

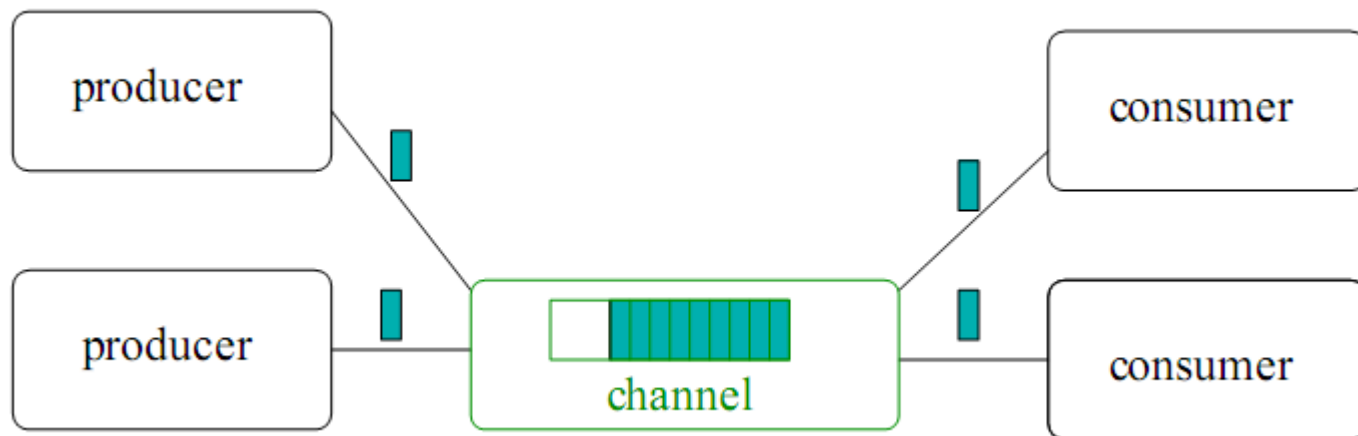
- Các bộ sinh thông điệp – producers
- Các bộ nhận thông điệp – consumers
- Sự khác nhau giữa Producer/Consumer và client/server
 - Truyền thông 1 - n, n - 1, hoặc n – n
 - Phía sinh không cần quan tâm tới phía nhận



Liên kết lỏng

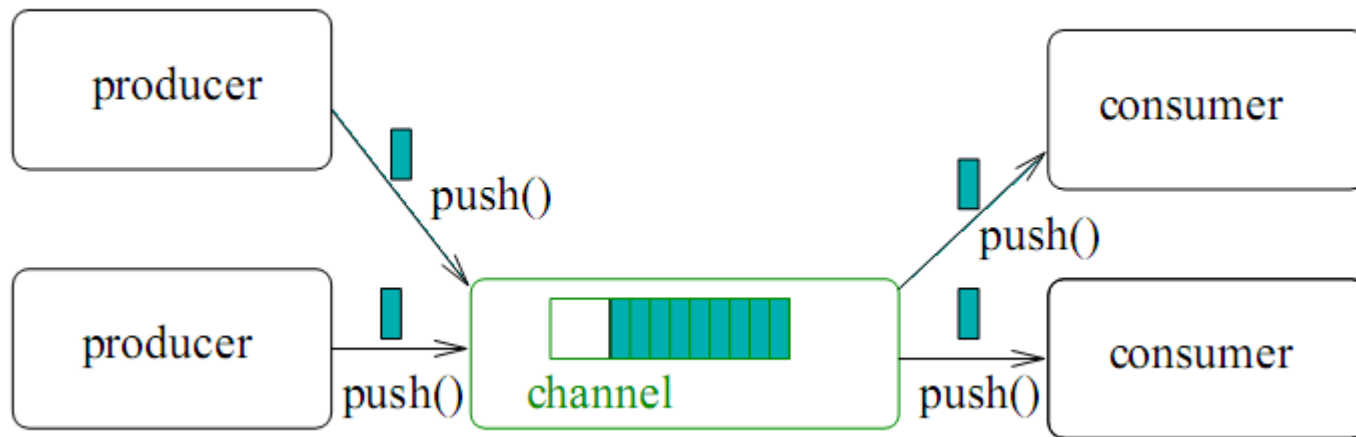
- Về không gian: Bộ sinh và nhận không kết nối trực tiếp qua một kênh truyền thông.
 - Không cần biết đến danh tính của nhau.
- Về thời gian:
 - Thời gian truyền thông điệp là không xác định
 - Bộ sinh /nhận không phải có mặt tại cùng một thời điểm
- Về cú pháp:
 - S/N không liên kết qua một API chung (khác với RMI)

Kênh truyền thông

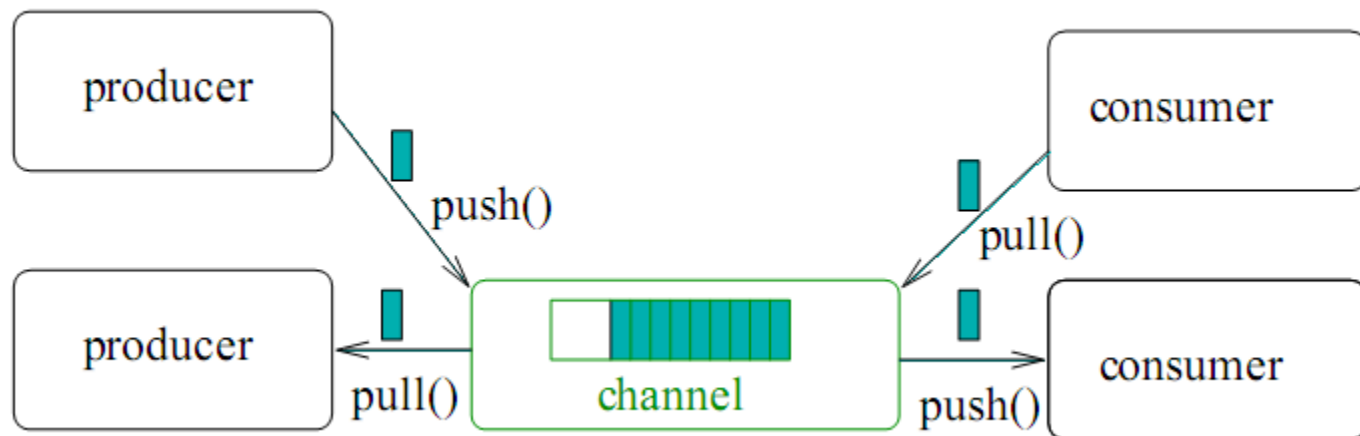


Mô hình Đẩy

- Sinh/Kênh – chủ động

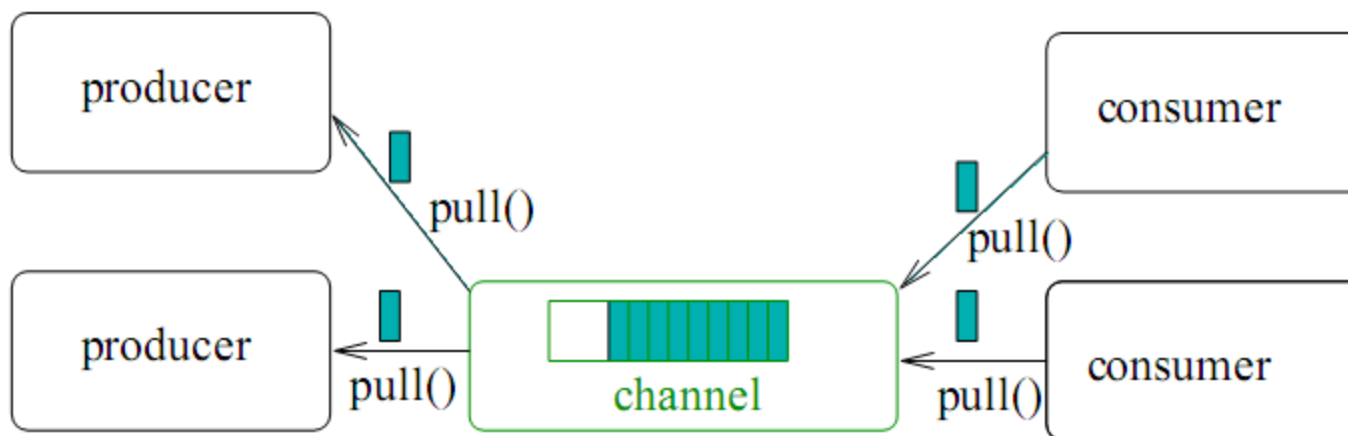


Mô hình lai

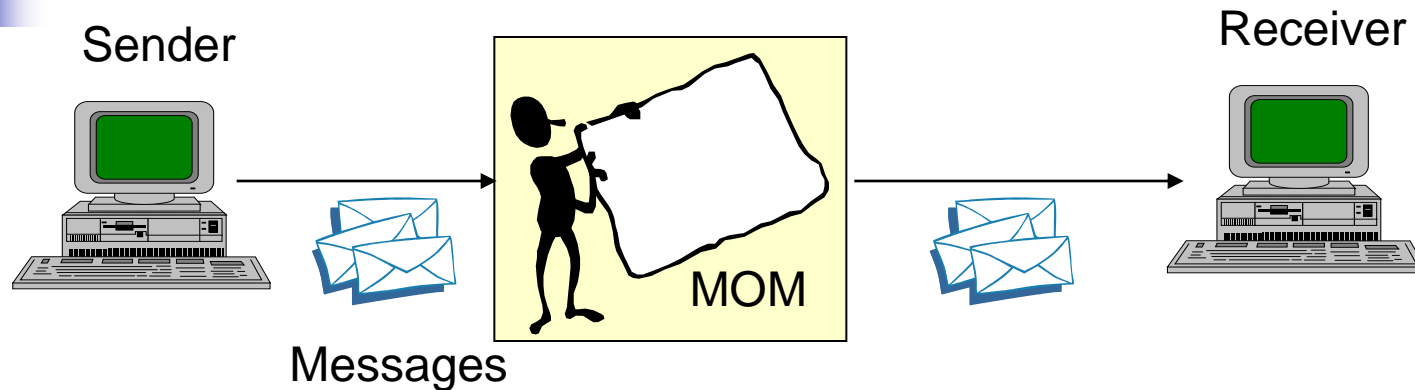


Mô hình kéo

- Kênh/Nhận – chủ động

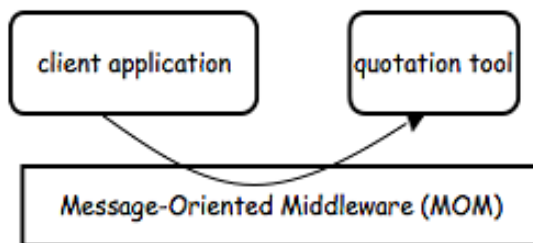


Phần dẻ hướng thông điệp

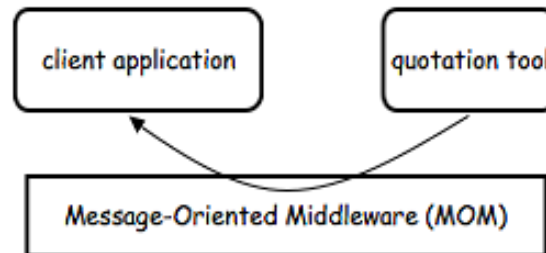


```
Message : quoteRequest {  
  QuoteReferenceNumber: 325  
  Customer: Acme,INC  
  Item:#115 (Ball-point pen, blue)  
  Quantity: 1200  
  RequestedDeliveryDate: Mar 16,2003  
  DeliveryAddress: Palo Alto, CA  
}
```

```
Message: quote {  
  QuoteReferenceNumber: 325  
  ExpectedDeliveryDate: Mar 12, 2003  
  Price:1200$  
}
```



(a)



(b)



Các tính năng của phần dẻo hướng thông điệp

- Hỗ trợ 2 mô hình truyền thông điệp:
 - Point-to-point: điểm-điểm
 - Publish-Subscribe
- Độ tin cậy
- Các thao tác giao dịch
- Truyền thông điệp phân tán
- An ninh



Các đặc tính thêm

- Truyền thông điệp thời gian thực
- Đảm bảo an ninh cho các giao dịch
- Chứng thực
- Cân bằng tải



Một số hệ thống MOM

- TIBCO
- JMS (Java Message Service)
- WebsphereMQ,
- ActiveMQ (apache)
- IBM MQ (Message Queues)
- MSMQ (Microsoft Message Queue)

1.1. Mô hình truyền thông điện



Mô hình truyền thông điệp

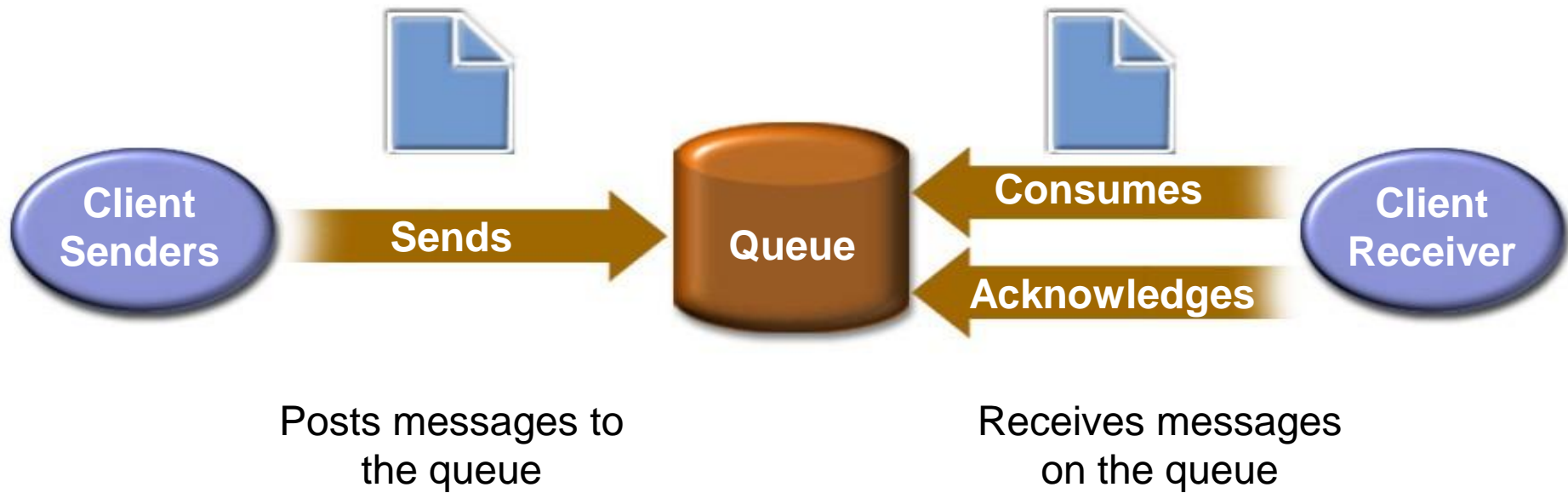
- Point to point
 - Một thông điệp được xử lý bởi 1 bên nhận duy nhất
- Publish/Subscribe
 - Một thông điệp được xử lý bởi nhiều bên nhận



Point-to-point

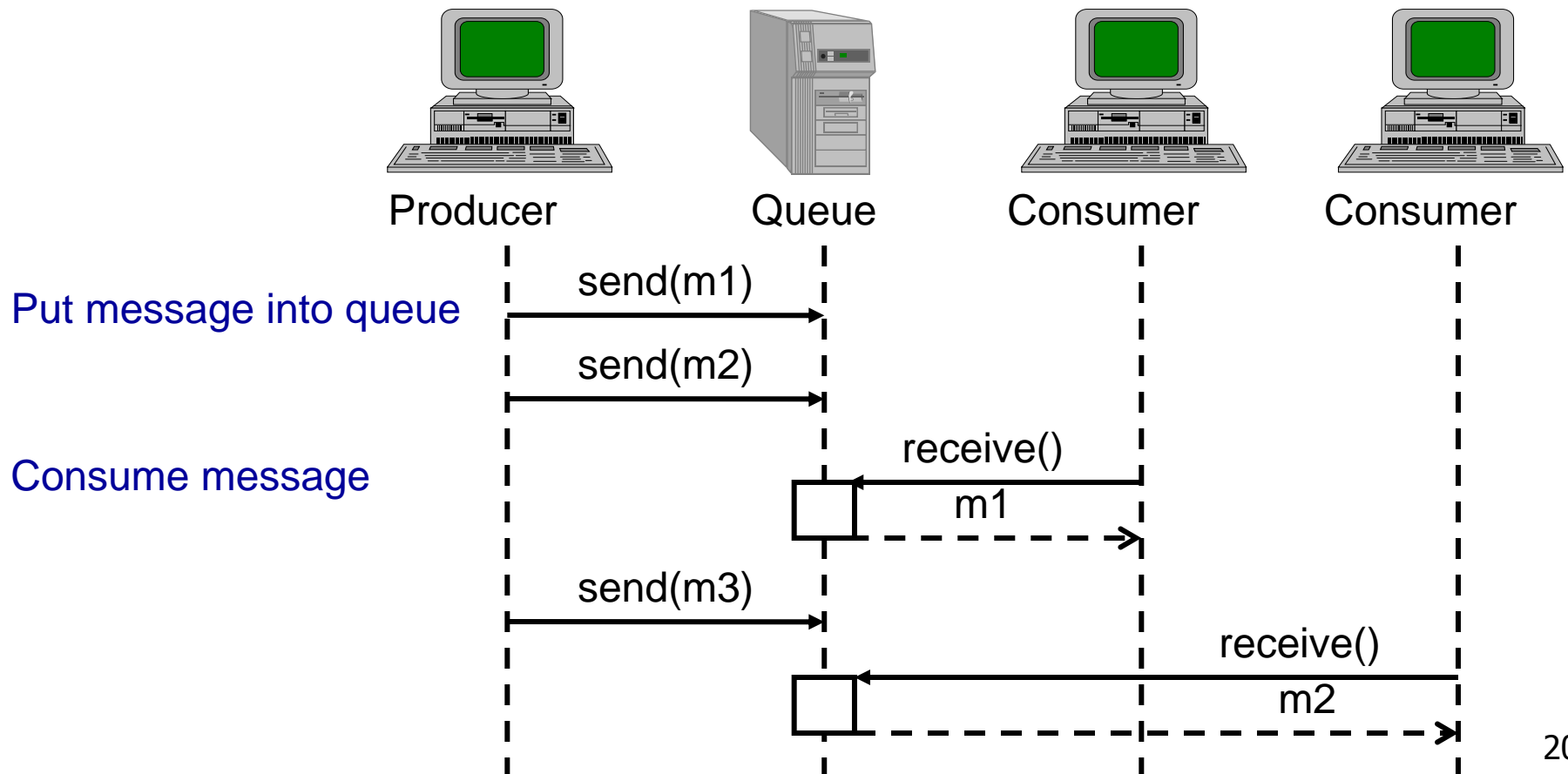
- Một thông điệp được xử lý bởi **1 bên nhận duy nhất**
- Có thể có nhiều bên gửi
- “Đích” của 1 thông điệp được gọi là **queue**
- First in, first out (nếu cùng mức ưu tiên)
- Bên gửi (senders/producers): gửi 1 thông điệp vào 1 queue (cùng mức ưu tiên)
- Bên nhận (receiver/consumer): Lấy ra 1 thông điệp từ queue

Point-to-point



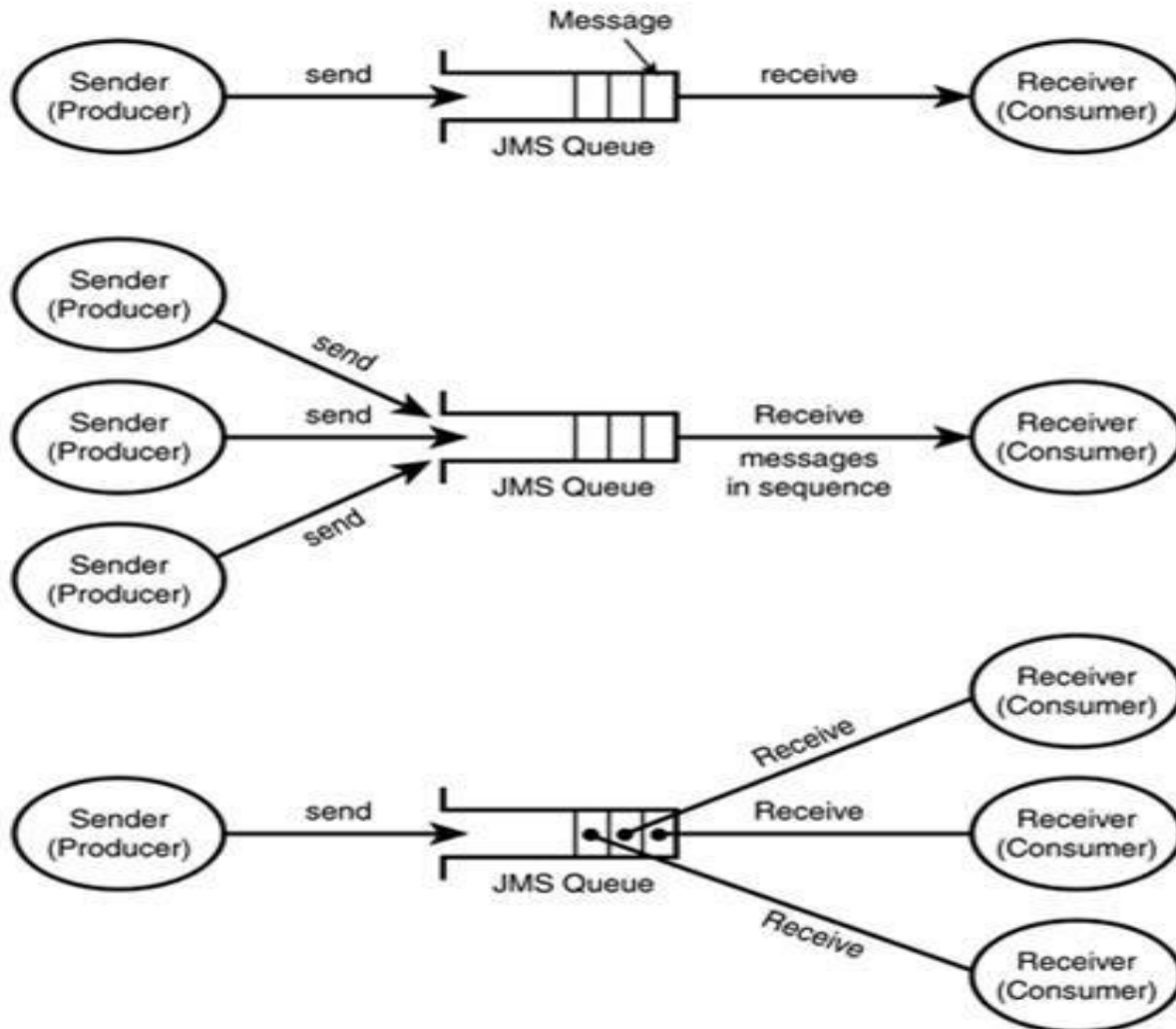
Hàng đợi thông điệp (MQ)

- Thông điệp gửi bởi client được đưa vào hàng đợi
- Khi thực thể nhận sẵn sàng, nó sẽ nhận thông điệp và xử lý



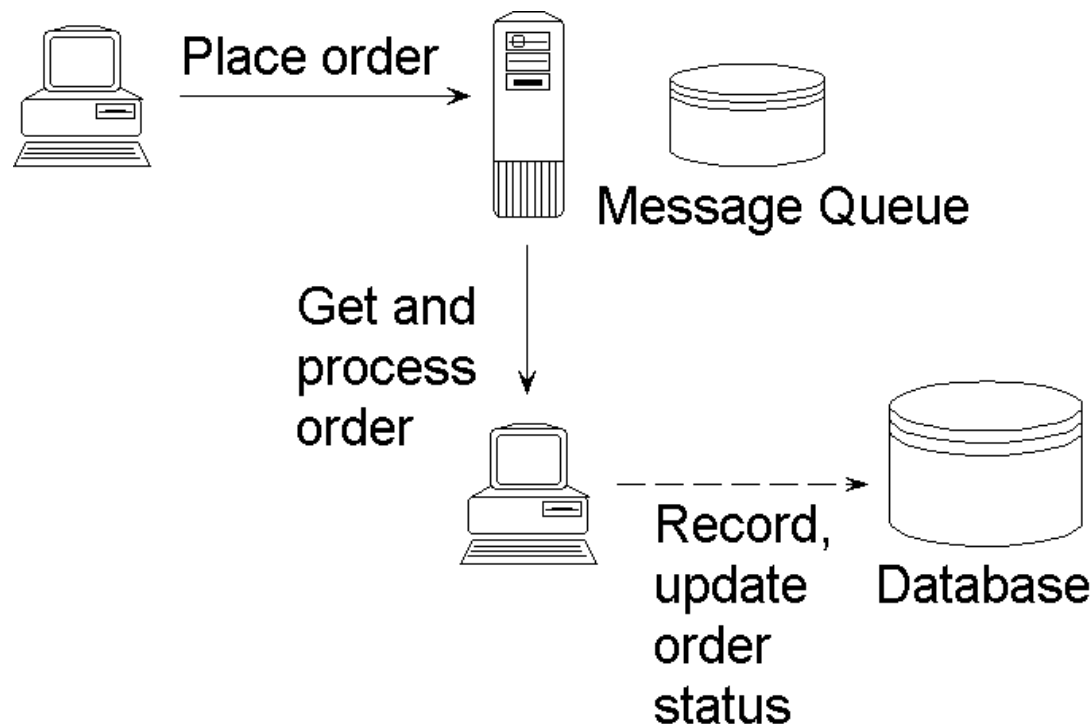
Một vài trường hợp

Point-2-Point Message Model



Khi nào sử dụng point-to-point?

- Sử dụng khi tất cả các thông điệp truyền đi phải được xử lý thành công bởi 1 consumer

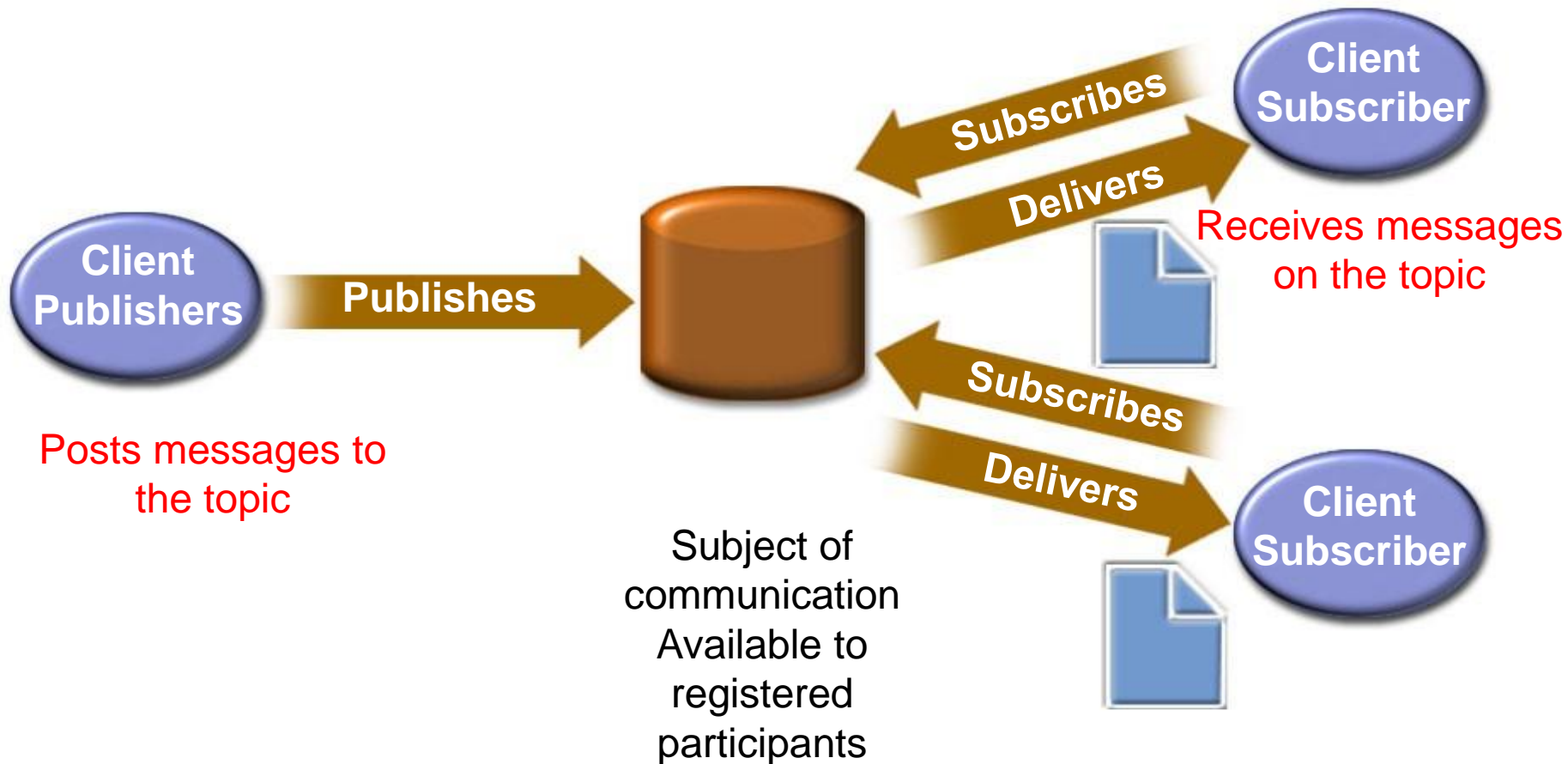




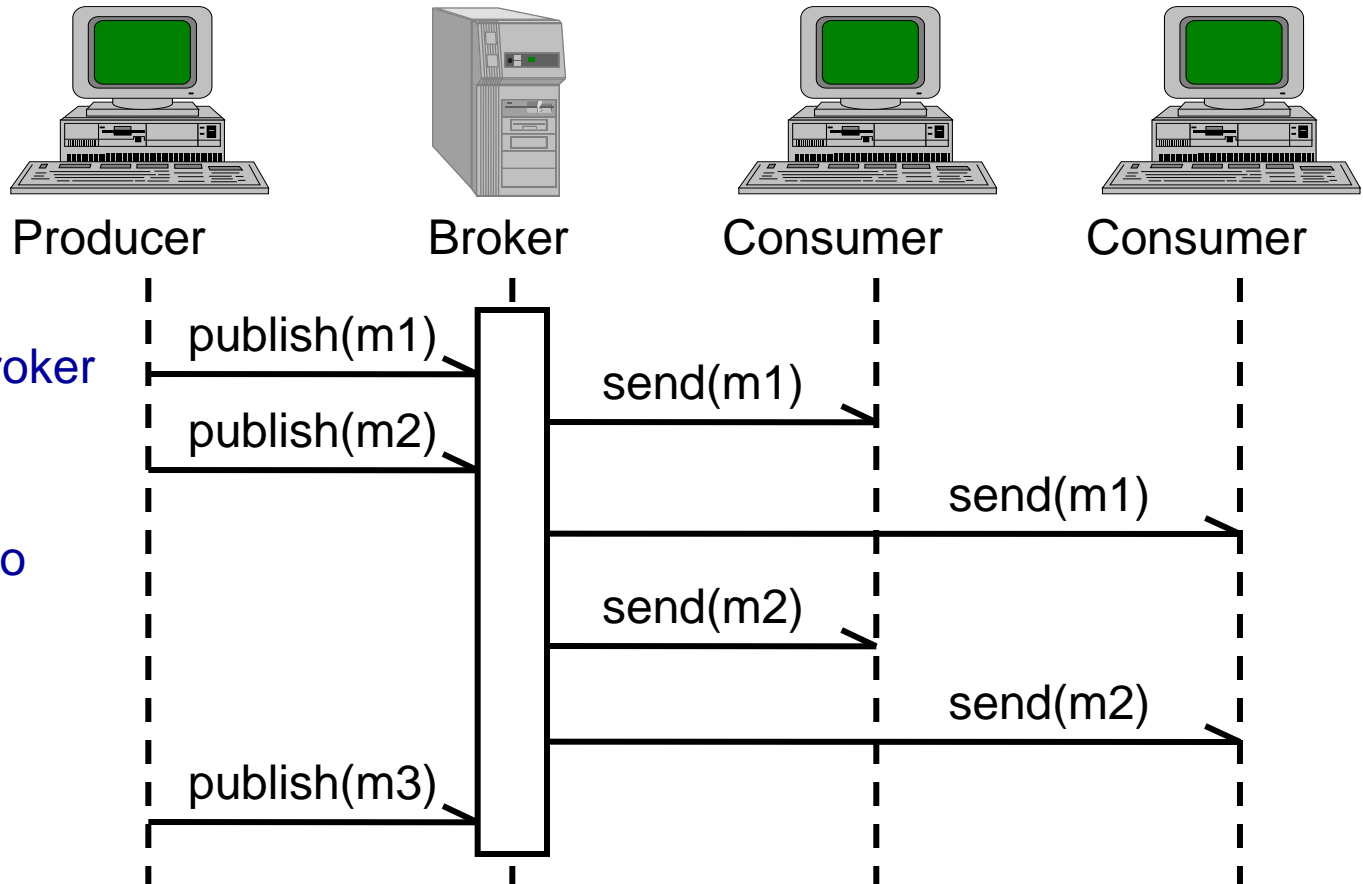
Publish/Subscribe (Pub/Sub)

- Một message được xử lý **bởi nhiều consumers**
- “Destination” của 1 message gọi là **topic**
 - Không phải là 1 queue
- Producers “**publish**” đến topic
- Consumers “**subscribe**” đến topic

Publish-and-Subscribe



Publish-Subscribe



Khi nào sử dụng Pub/Sub?

- Sử dụng khi 1 thông điệp gửi đi cần xử lý bởi nhiều consumers
- Ví dụ: ứng dụng nhân sự
 - Tạo "new hire" topic
 - Rất nhiều ứng dụng ("facilities", "payroll", ...) subscribe "new hire" topic

publish ("AAPL", 29.2);
publish("AAPL", 29.3);

publish ("SUN", 43.0);
publish("SUN", 42.7);

subscribe ("AAPL");
subscribe ("SUN");

subscribe ("AAPL");
subscribe ("SUN");



1.2. Tính tin cậy



Tính tin cậy (Reliability)

- Đôi khi phải đảm bảo việc truyền thông điệp:
 - Có nhiều mức độ tin cậy khác nhau
 - Sender có thể chỉ ra mức độ tin cậy cụ thể
 - Độ tin cậy cao thì lượng công việc càng ít hơn
- Đặc biệt sử dụng persistent storage để lưu trữ các thông điệp



1.3. Các thao tác giao dịch



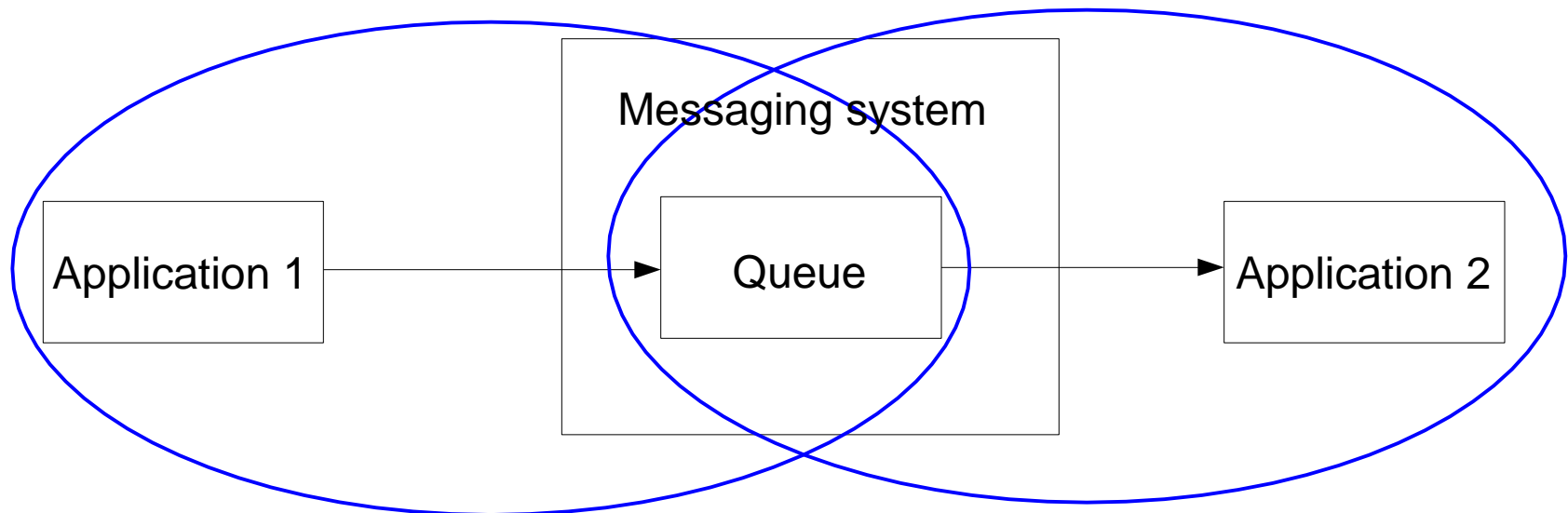
Các thao tác giao dịch

- Giao dịch sinh thông điệp
 - Sender nhóm 1 loạt các thông điệp vào 1 transaction
 - Hoặc tất cả hoặc không thông điệp nào được cho vào queue thành công
- Giao dịch nhận thông điệp
 - Consumer nhận 1 nhóm các thông điệp như 1 transaction
 - Trừ khi tất cả các thông điệp được nhận, nếu không chúng vẫn còn lại ở queue hoặc topic

Phạm vi giao dịch

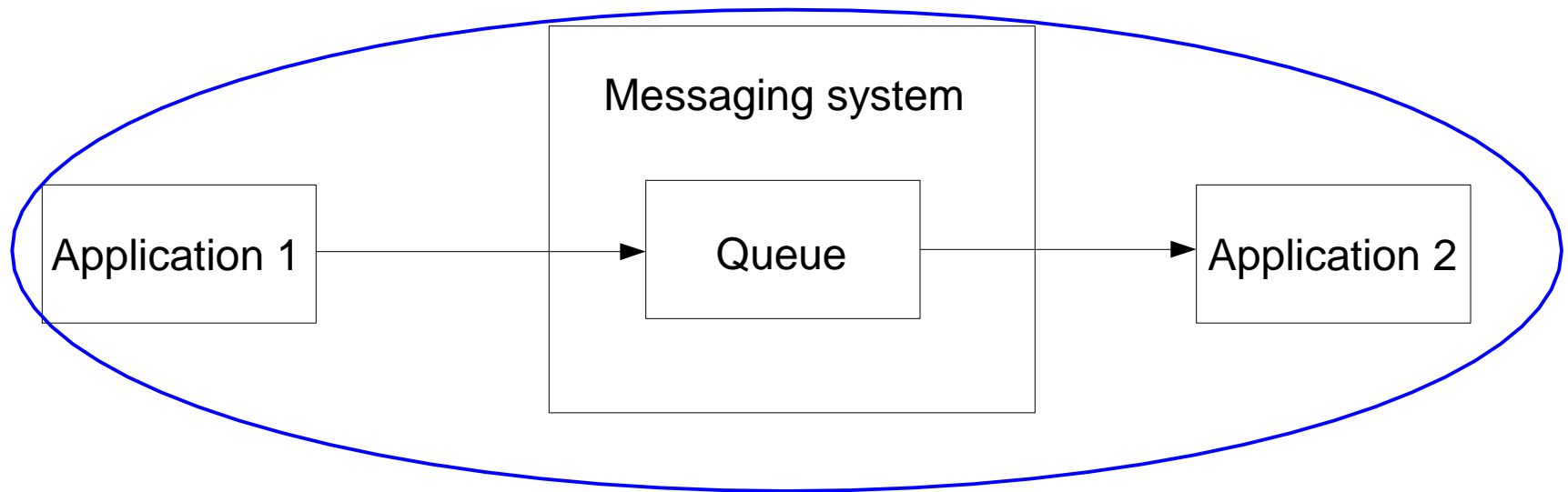
■ Phạm vi Client-to-Messaging

- Transaction bao phủ mỗi lần tương tác nhận/gửi thông điệp
- JMS hỗ trợ loại này



Phạm vi Client-to-Client

- Phạm vi Client-to-Client
 - Transaction bao phủ cả 2 client
 - JMS không hỗ trợ phạm vi này



1.4. Truyền thông điệp phân tán

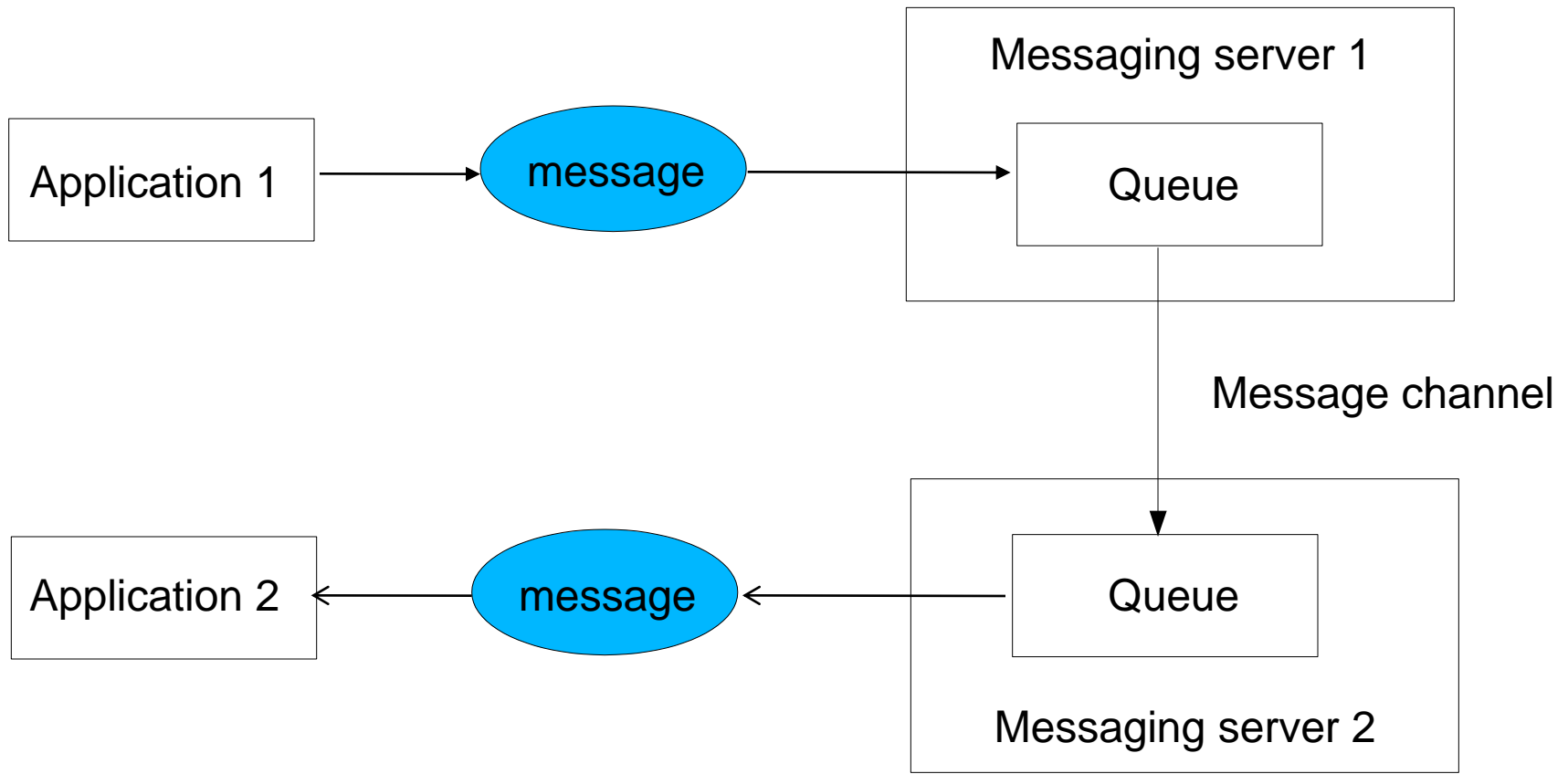




Truyền thông điệp phân tán

- Các hệ thống truyền thông điệp phân tán (Enterprise MS) có thể cung cấp hạ tầng trong đó, thông điệp được forward giữa các servers:
 - Được gọi là “message channel”

Truyền thông điệp phân tán





1.5. An ninh



Các vấn đề an ninh

- Chứng thực - Authentication
 - Hệ thống thông điệp yêu cầu client trình ra chứng chỉ được ký (signed certificates)
- Tính bí mật của các thông điệp
 - Hệ thống thông điệp cung cấp cơ chế mã hóa
- Tính toàn vẹn dữ liệu của các thông điệp
 - Hệ thống thông điệp cung cấp cơ chế đảm bảo toàn vẹn dữ liệu khi thông điệp được xử lý
- Được xử lý tùy theo nhà cung cấp



2. Tại sao cần truyền thông điệp?



Tại sao cần truyền thông điệp?

- Độc lập Platform
- Độc lập theo network location
- Làm việc tốt với các hệ thống không đồng nhất



Tại sao cần truyền thông điệp?

- Anonymity
 - Who doesn't matter
 - Where doesn't matter
 - When doesn't matter
- Ngược với hệ thống RPC
 - Corba
 - RMI



Tại sao cần truyền thông điệp?

■ Scalability

- Xử lý được với nhiều clients mà
 - Không cần thay đổi ở application
 - Không cần thay đổi ở architecture
 - Không làm hỏng công việc của hệ thống
- Tăng công suất phần cứng nếu yêu cầu mức scalability cao hơn



Tại sao cần truyền thông điệp?

- Khả năng chịu lỗi (robustness)
 - Thực thể nhận có thể hỏng.
 - Thực thể gửi có thể hỏng.
 - Mạng có thể hỏng
 - Hệ thống thông điệp vẫn tiếp tục hoạt động được bình thường



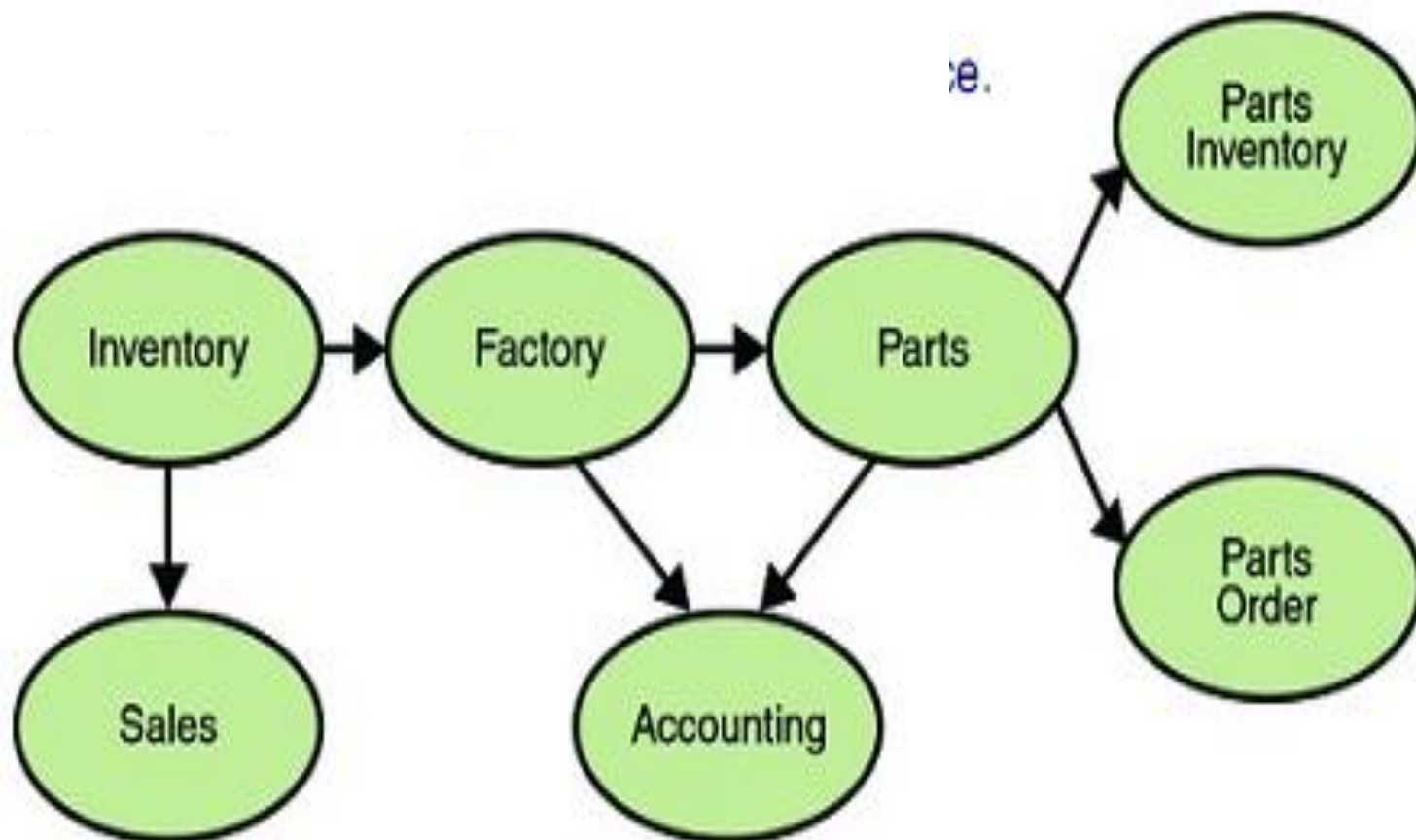
Ví dụ về các ứng dụng truyền thông điệp



Các ứng dụng truyền thông điệp

- Giao dịch thẻ tín dụng
- Báo cáo thời tiết
- Workflow
- Quản trị mạng
- Quản lý dây chuyền sản xuất
- Chăm sóc khách hàng
- Truyền thông (Voice Over IP, Paging Systems, etc.)
- ...

Ví dụ





Bài tập đọc hiểu

- <http://www.eaipatterns.com/BondTradingCaseStudy.html>



3. JMS (Java Message Service)



JMS là gì?

- JMS là một tập các Java interfaces kết hợp với các APIs định nghĩa cách thức một JMS client truy cập tới các chức năng của một hệ thống thông điệp
- Hỗ trợ các cơ chế truyền thông điệp
 - Synchronous or Asynchronous
 - transacted
 - Guaranteed
 - Durable



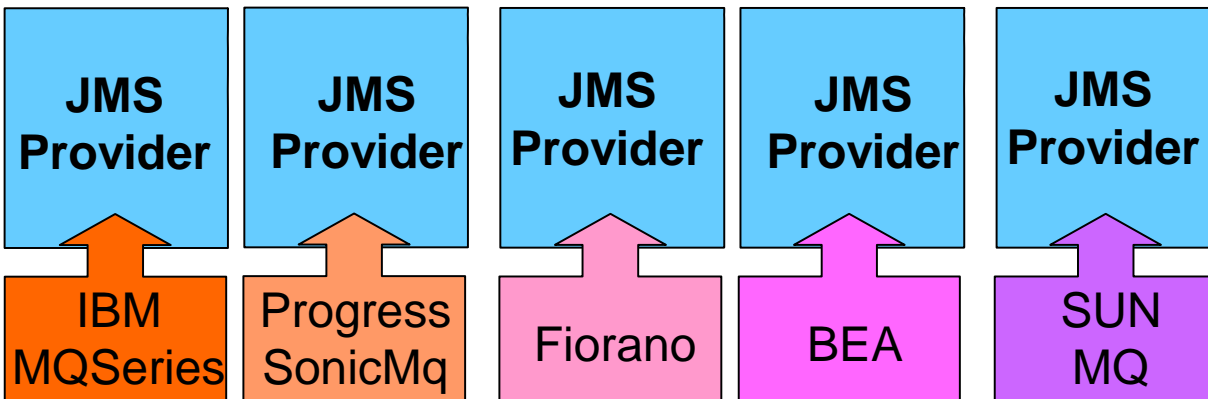
JMS là gì? (tiếp)

- Hỗ trợ các mô hình truyền thông điệp
 - Point-to-point (hàng đợi tin cậy)
 - Publish/Subscribe
- Message selectors (phía nhận)
- 5 loại message

JMS là một API

Java™ Application 1

JMS API



JMS và J2EE

- Cho phép Java developers truy cập tới hệ thống truyền thông điệp
- Là 1 phần của bộ J2EE Enterprise



3. Kiến trúc của 1 ứng dụng



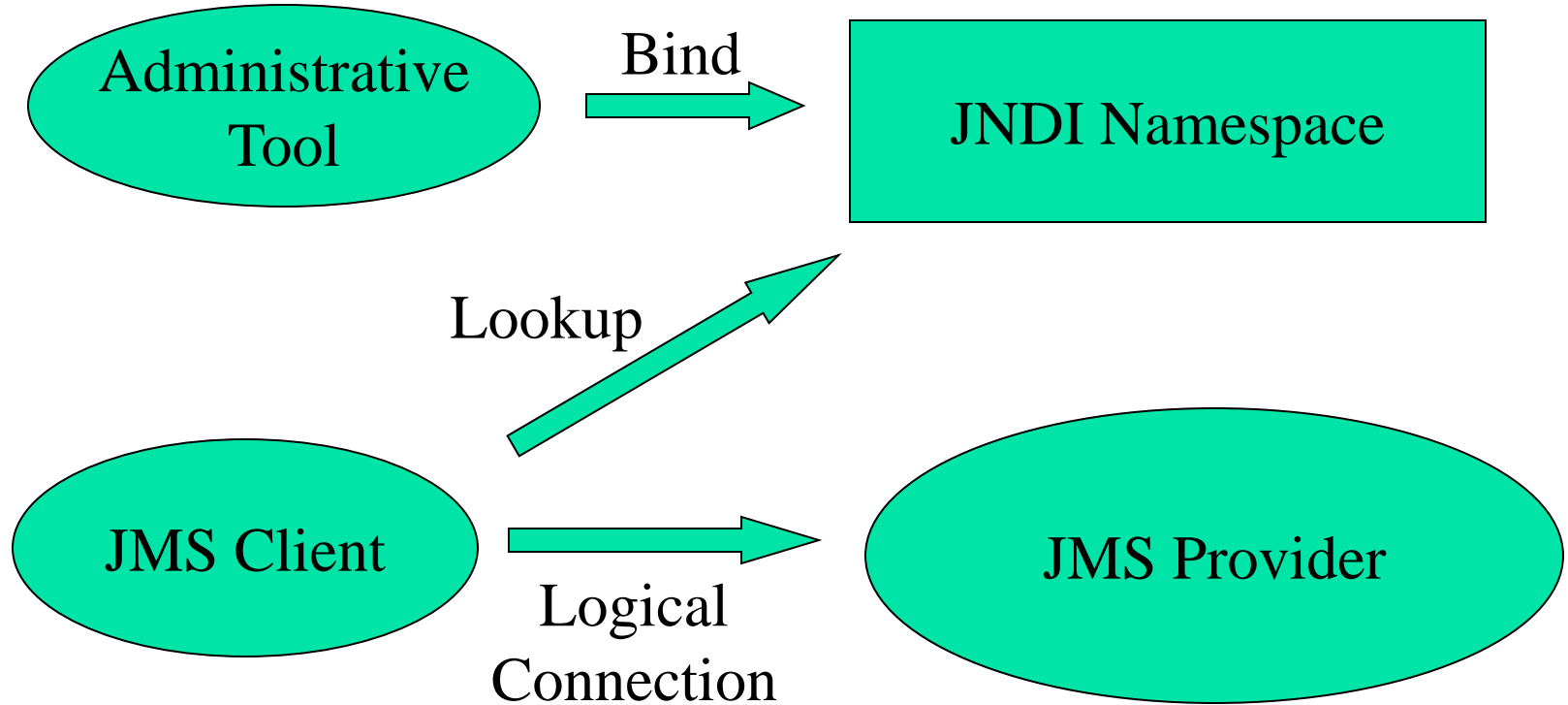
JMS



Các thành phần kiến trúc JMS

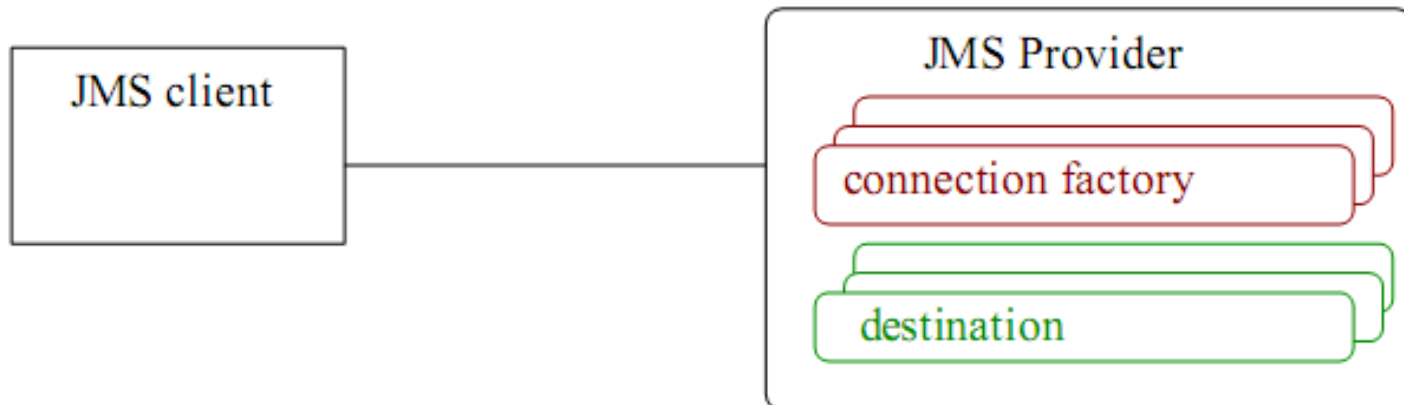
- JMS clients
- Messages
- JMS Provider (Messaging systems)
- JNDI administered objects
 - Destination
 - ConnectionFactory

Các thành phần kiến trúc JMS



Kiến trúc ứng dụng JMS

- JMS clients: Bộ sinh/ nhận
- Các thông điệp: Đối tượng truyền thông giữa các client.
- Các đối tượng quản lý: Connection factories và destinations





Các thuật ngữ trong JMS

- Domain (Mô hình truyền thông điệp)
 - Point-to-point, publish/subscribe
- Session
- Connection
- Destination
- Produce, send, publish
- Consume, receive, subscribe



JMS Domains (Mô hình truyền thông điệp)



JMS Domains (Mô hình truyền thông điệp)

- JMS Point-to-Point

- Các thông điệp trong 1 queue có thể là persistent hoặc non-persistent

- JMS Pub/Sub

- Non-durable
- Durable



JMS Pub/Sub Non-durable và JMS Pub/Sub Durable

■ Non-durable

- Các thông điệp là sẵn có chỉ trong thời gian subscribers là active
- Nếu subscriber không active (không được connected), nó sẽ thiếu các thông điệp được cung cấp trong suốt thời gian vắng mặt

■ Durable

- Các thông điệp được lưu giữ tại đại diện của subscriber không có mặt lúc thông điệp được sinh ra



JMS Messages



JMS Messages

- Thông điệp là các thức truyền tin giữa các applications
- Các message được truyền thực sự biến đổi tùy theo hệ thống truyền thông điệp
 - Một hệ thống có thể chỉ giao tiếp được với duy nhất 1 loại hệ thống thông điệp khác



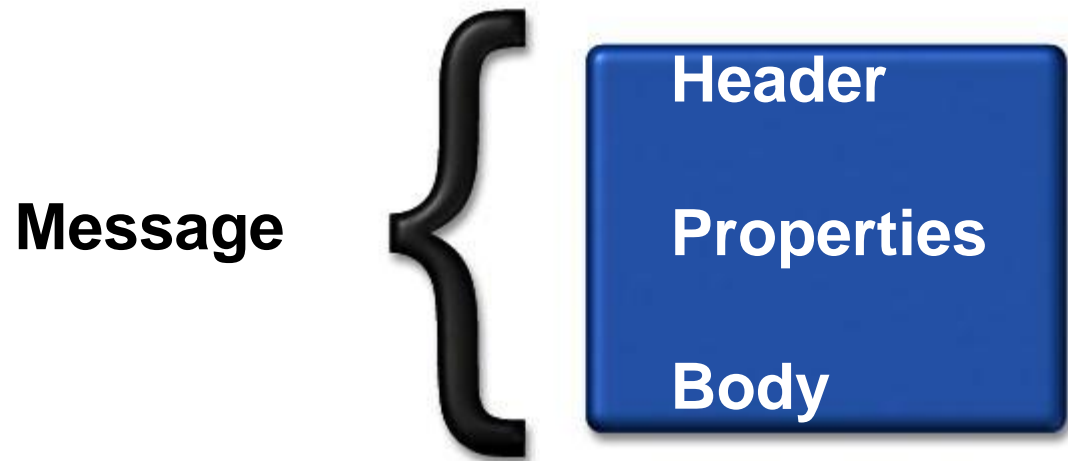
Message Java Interface

- JMS cung cấp 1 mô hình “unified” và “abstract” qua Interface này
- Đối tượng thực thi thực sự của interface này tùy theo provider



Message Components

- Header
- Properties
- Body





Message header

- Được sử dụng để định danh message và routing
- Chứa **Destination**
- Có thể bao gồm cả dữ liệu sau:
 - delivery mode (persistent, nonpersistent)
 - message ID
 - timestamp
 - priority
 - ReplyTo



Các trường của Message header

- JMSDestination
- JMSDeliveryMode
 - persistent or nonpersistent
- JMSMessageID
- JMSTimeStamp
- JMSRedelivered
- JMSExpiration



Message header fields

- JMSPriority
- JMSCorrelationID `replyMsg.setJMSCorrelationID(requestMsg.getJMSCorrelationID());`
- JMSReplyTo
 - đối tượng Destination của một client; nơi gửi thông điệp trả lời
- JMSType
 - Type of message body

```
Queue replyQ = (Queue)ctx.lookup(jmsReplyQ);  
requestMsg.setJMSReplyTo(replyQ);
```



Message Properties

- Các fields tùy theo Application
- Các fields do provider chỉ định
- Optional fields
- Các properties là các cặp Name/value
- Values có thể là byte, int, String, ...



Message body

- Lưu trữ nội dung thông điệp
- Hỗ trợ một số types
- Mỗi type được định nghĩa bởi 1 message interface:
 - StreamMessage
 - MapMessage
 - TextMessage
 - ObjectMessage
 - BytesMessage



Message Body Interfaces

- **StreamMessage**
 - Chứa các giá trị Java primitive
 - Được đọc tuần tự
- **MapMessage**
 - Lưu trữ các cặp name/value
 - Được đọc tuần tự hoặc theo name
- **BytesMessage**
 - Uninterpreted bytes
 - Được sử dụng để match 1 định dạng thông điệp có trước



Ví dụ: tạo 1 thông điệp text

- Để tạo 1 TextMessage đơn giản:

```
TextMessagemessage =  
    session.createTextMessage();  
  
message.setText("greetings");
```



Ví dụ: tạo 1 object message

- Để tạo một ObjectMessage đơn giản:

```
ObjectMessage message =
```

```
    session.createObjectMessage ();
```

```
message.setObject(myObject) ;
```

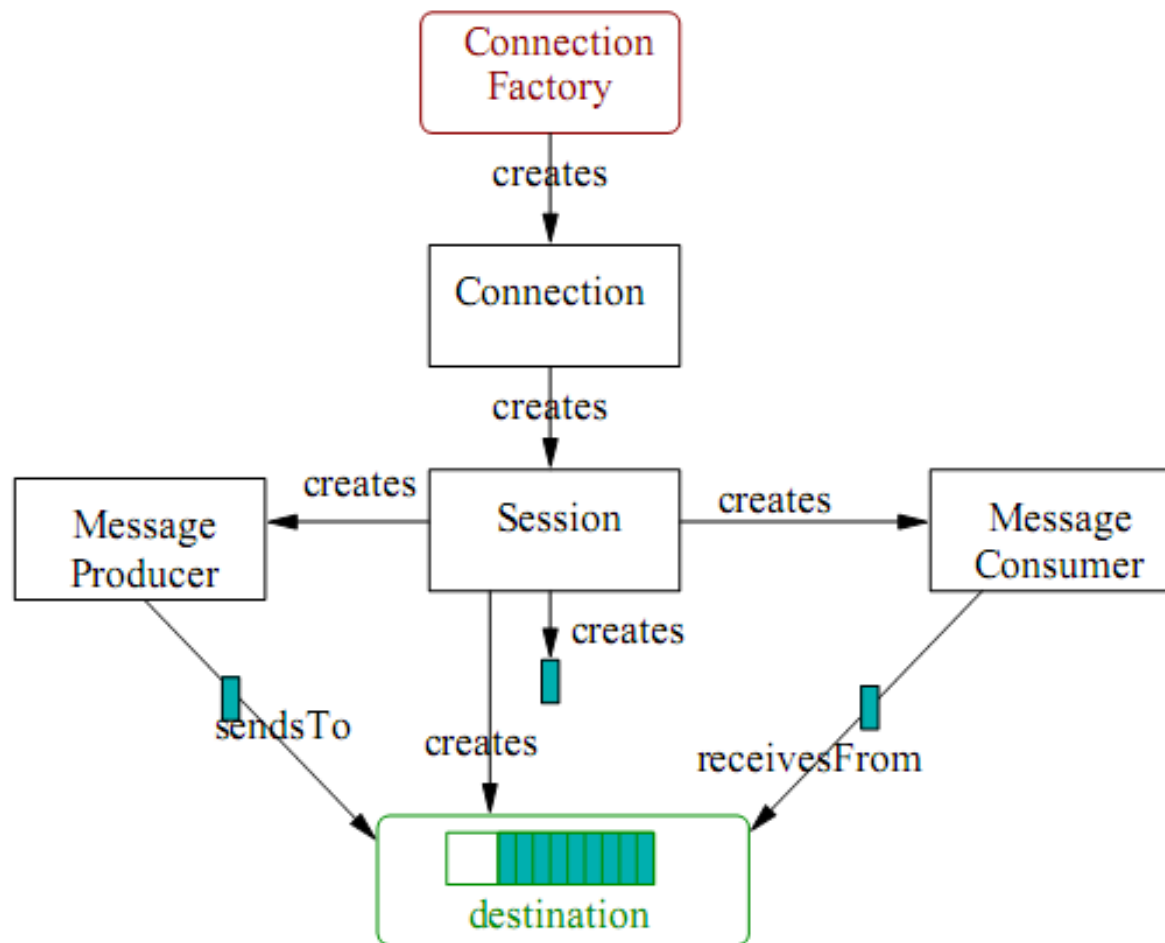
- Lưu ý:

- myObject phải implement
java.io.Serializable



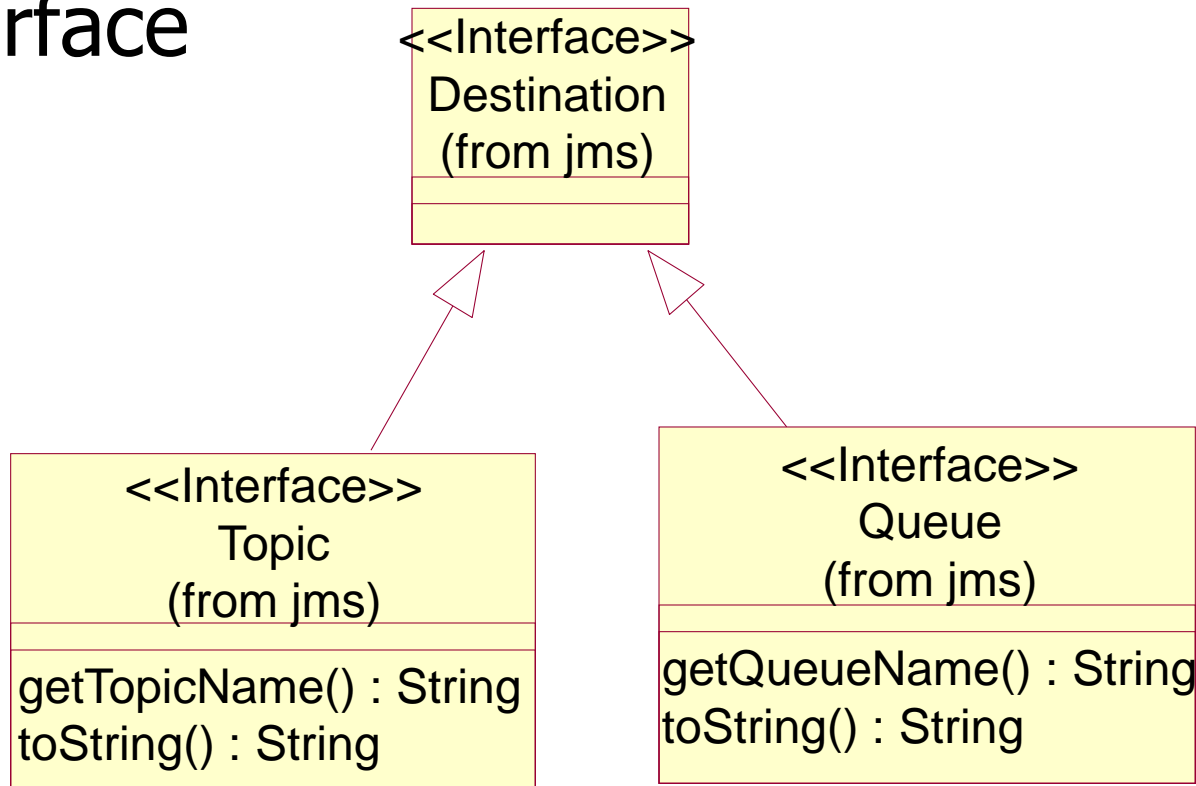
4. JMS Programming APIs

Các đối tượng JMS



Destination Java Interface

- Là trừu tượng hóa của topic & queue
- Là interface cha của Queue và Topic interface





Destination

- Destination là đối tượng được quản lý (là đối tượng phân tán)
- Là kênh truyền thông

```
String subject;
```

```
javax.jms.Session session;
```

```
javax.jms.Destination destination;
```

```
destination = session.createTopic(subject);
```

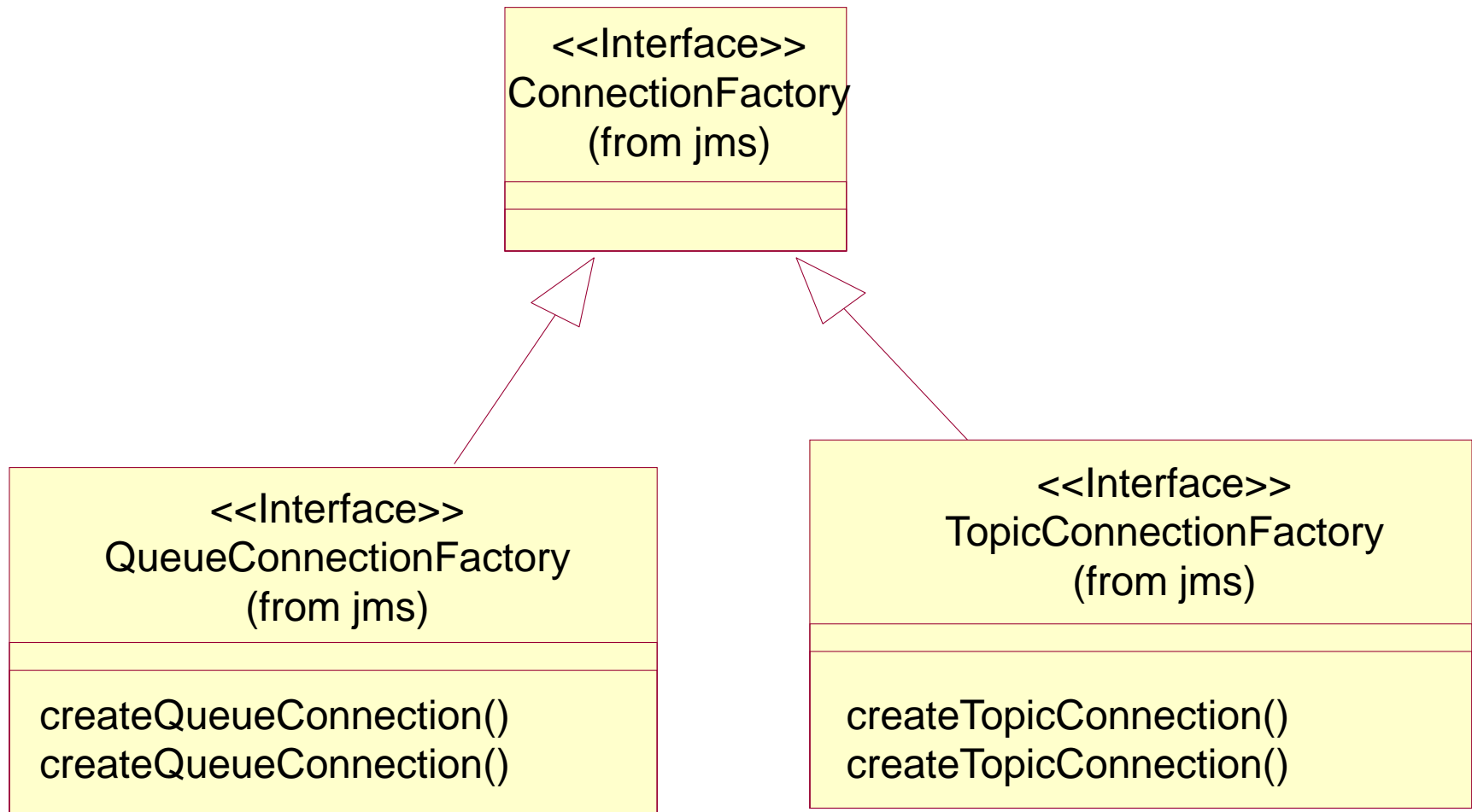
```
destination = session.createQueue(subject);
```



ConnectionFactory Java Interface

- Factory class để tạo 1 connection từ provider đến JMS server
- Tương tự như driver manager (`java.sql.DriverManager`) trong JDBC
- Là interface cha của:
 - `QueueConnectionFactory` interface
 - `TopicConnectionFactory` interface

ConnectionFactory Java Interface





Connection Factory

- Connection Factory là đối tượng được quản lý (phân tán).
 - Điểm truy nhập tới JMS server
 - Cần thiết để kết nối tới một JMS server

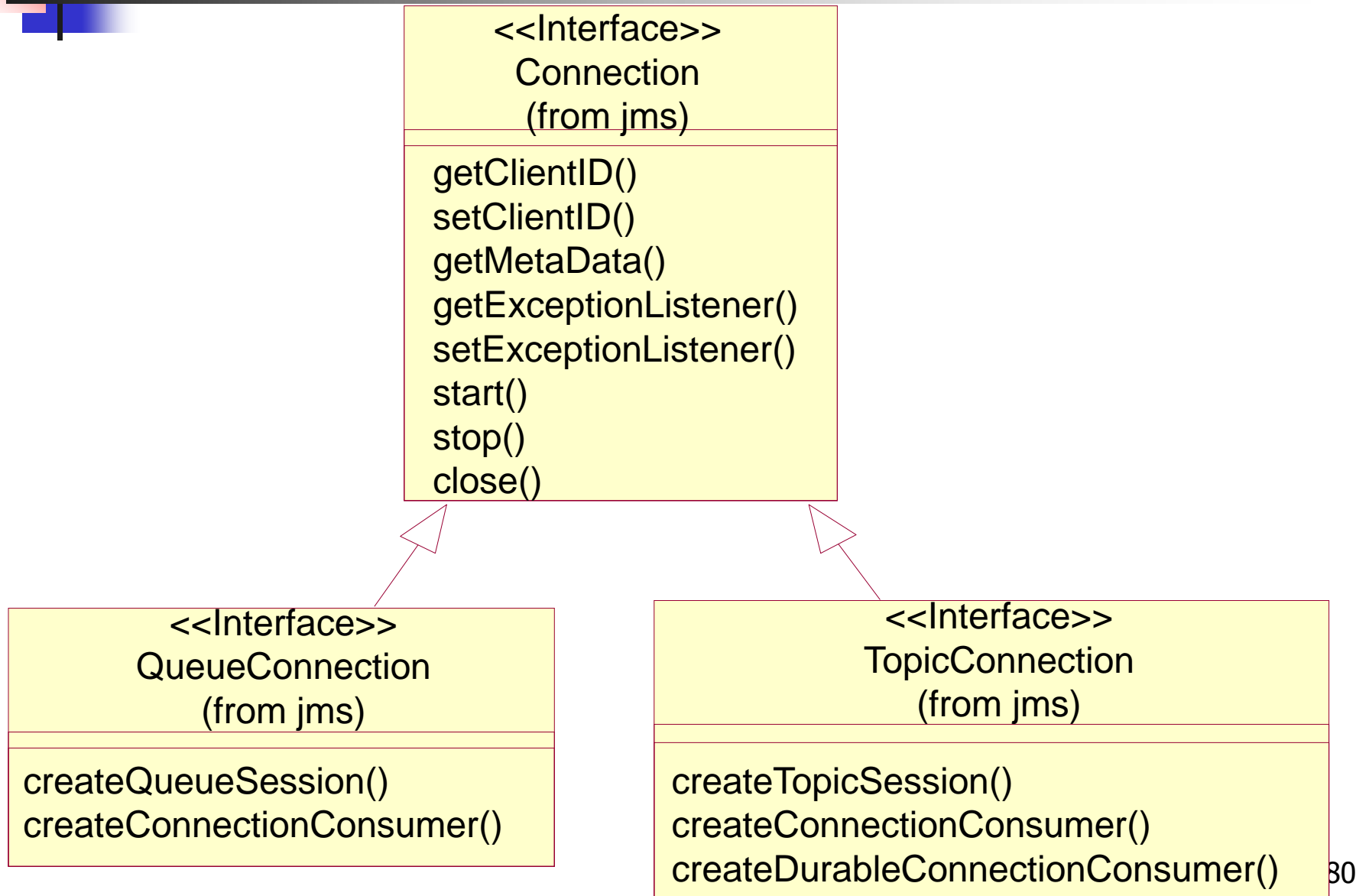
```
ConnectionFactory connectionFactory =  
new XXXXXConnectionFactory(user, password,  
    url) ;  
// url: e.g. tcp://localhost:61616
```



Connection Java Interface

- Là trừu tượng hóa, biểu diễn một kênh giao tiếp tới JMS provider
- Được tạo từ 1 đối tượng ConnectionFactory
- Một connection nên được đóng khi chương trình sử dụng xong nó

Connection Java Interface





Đối tượng Connection

- Đóng gói socket TCP/IP giữa một client và một JMS server
- Tạo ra session giữa bộ nhận/sinh và server

```
javax.jms.Connection connection;  
connection = connectionFactory.createConnection();  
connection.start();  
...  
connection.stop();//temporary stop message delivery  
...  
connection.start();  
...  
connection.close();
```



Session Java Interface

- Được tạo từ 1 đối tượng Connection
- Một khi được kết nối tới provider thông qua 1 Connection, tất cả các công việc được thực hiện dưới ngữ cảnh của 1 Session
- Một session ứng với một thread, tức là mọi thông điệp gửi và nhận tuần tự, tiếp nối nhau

Session Java Interface

<<Interface>> Session (from jms)

\$ AUTO_ACKNOWLEDGE : int = 1
\$ CLIENT_ACKNOWLEDGE : int = 2
\$ DUPS_OK_ACKNOWLEDGE : int = 3

createBytesMessage()
createMapMessage()
createMessage()
createObjectMessage()
createObjectMessage()
createStreamMessage()
createTextMessage()
getTransacted()
commit()
rollback()
close()
recover()
getMessageListener()
run()

<<Interface>> QueueSession (from jms)

createQueue()
createReceiver()
createSender()
createBrowser()
createTemporaryQueue()

<<Interface>> TopicSession (from jms)

createTopic()
createSubscriber() c
createDurableSubscriber()
createPublisher()
createTemporaryTopic()
unsubscribe()



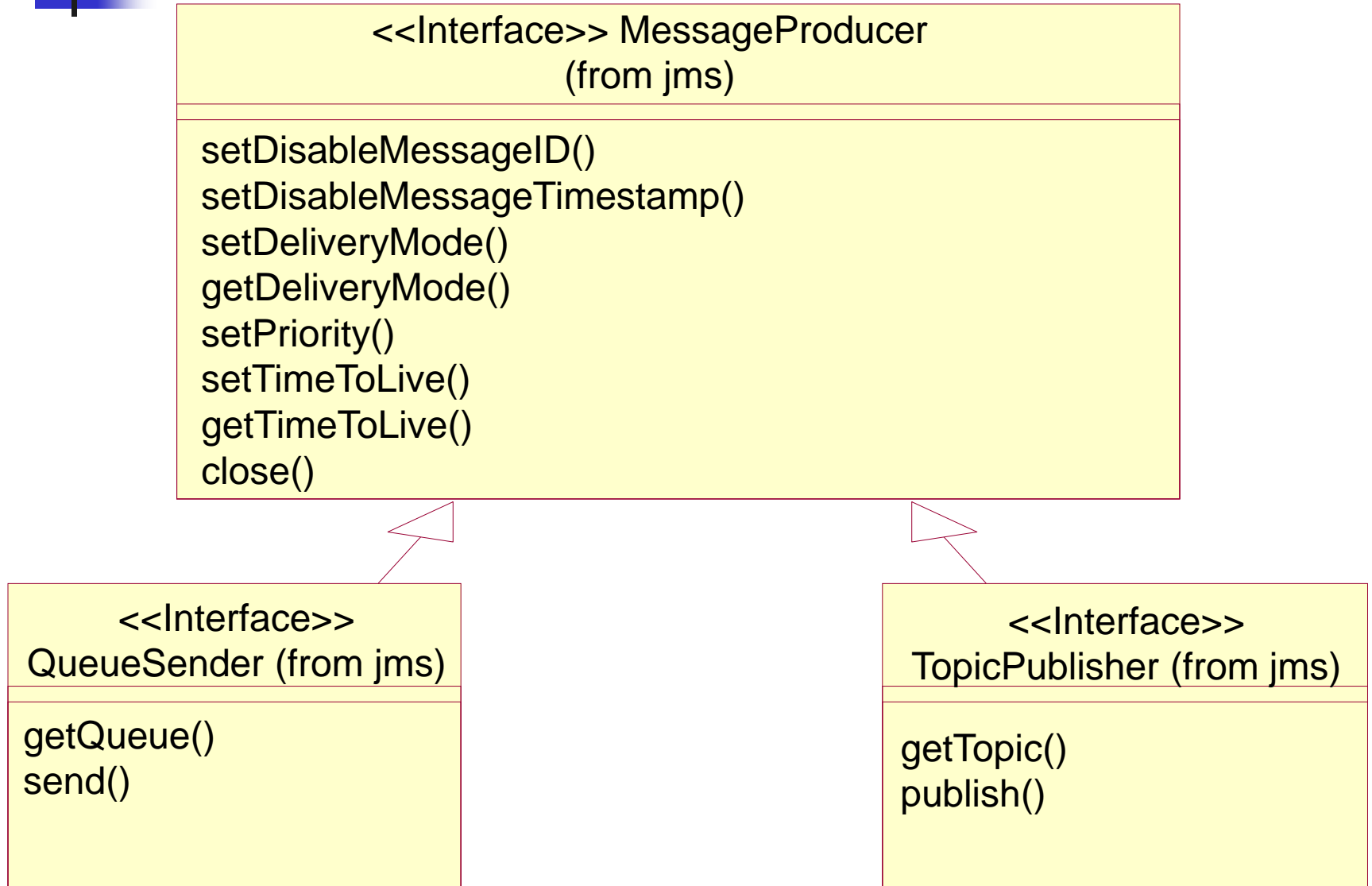
Session

- Tạo ra MessageProducer và MessageConsumer
- Đảm bảo thứ tự các thông điệp (xác định bởi số hiệu của chúng và destination trong phiên)
- Nằm trong 1 thread

```
javax.jms.Session session;  
session = connection.createSession(transactionMode,  
    ackMode);
```



MessageProducer Java Interface





MessageProducer Java Interface

- Client muốn nhận thông điệp tạo đối tượng MessageConsumer thông qua đối tượng Session
- Đối tượng MessageConsumer được gắn với 1 đối tượng Destination
- Client có thể nhận thông điệp theo 1 trong 2 chế độ:
 - Blocking
 - Non-blocking



Message Producer

- Tạo ra bởi 1 session
- Gửi các thông điệp tới destination

```
javax.jms.MessageProducer producer;  
javax.jms.TextMessage message;  
producer =  
    session.createProducer(destination) ;  
...  
message = session.createTextMessage(texte) ;  
producer.send(message) ;
```



Nhận thông điệp trong chế độ **Blocking**

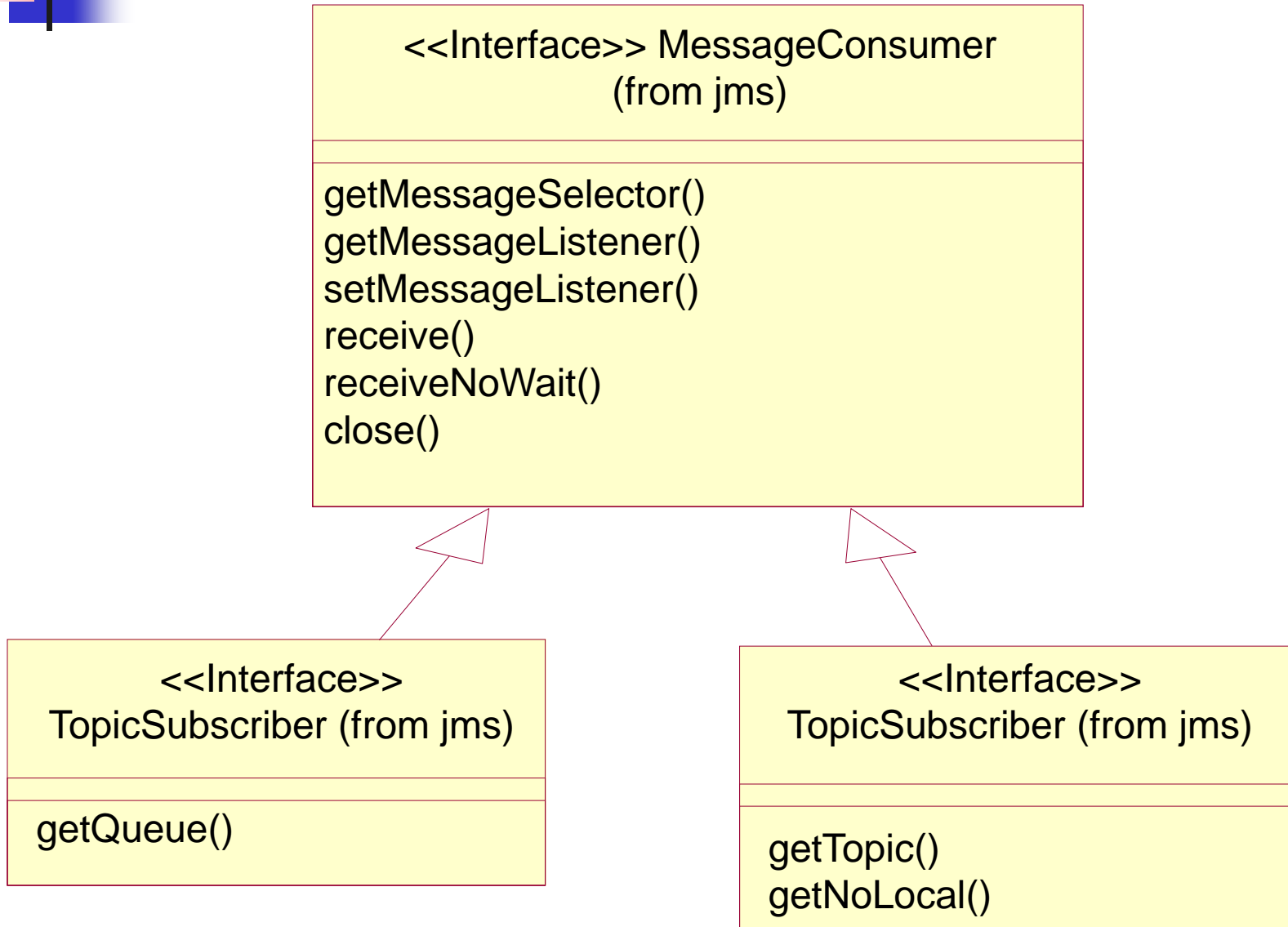
- Client gọi phương thức `receive()` của đối tượng `MessageConsumer`
- Client blocks cho đến khi có được thông điệp



Nhận thông điệp ở chế độ **Non-blocking**

- Client đăng ký 1 đối tượng **MessageListener**
- Client không bị block
- Khi một message đến, JMS provider gọi phương thức **onMessage()** của đối tượng **MessageListener**

MessageConsumer Java Interface





Message Consumer

- Tạo ra bởi 1 session
- nhận thông điệp từ destination

```
javax.jms.MessageConsumer consumer;  
javax.jms.Message message;  
consumer =  
    session.createConsumer(destination);  
message = consumer.receive(); // synchronous  
receive
```

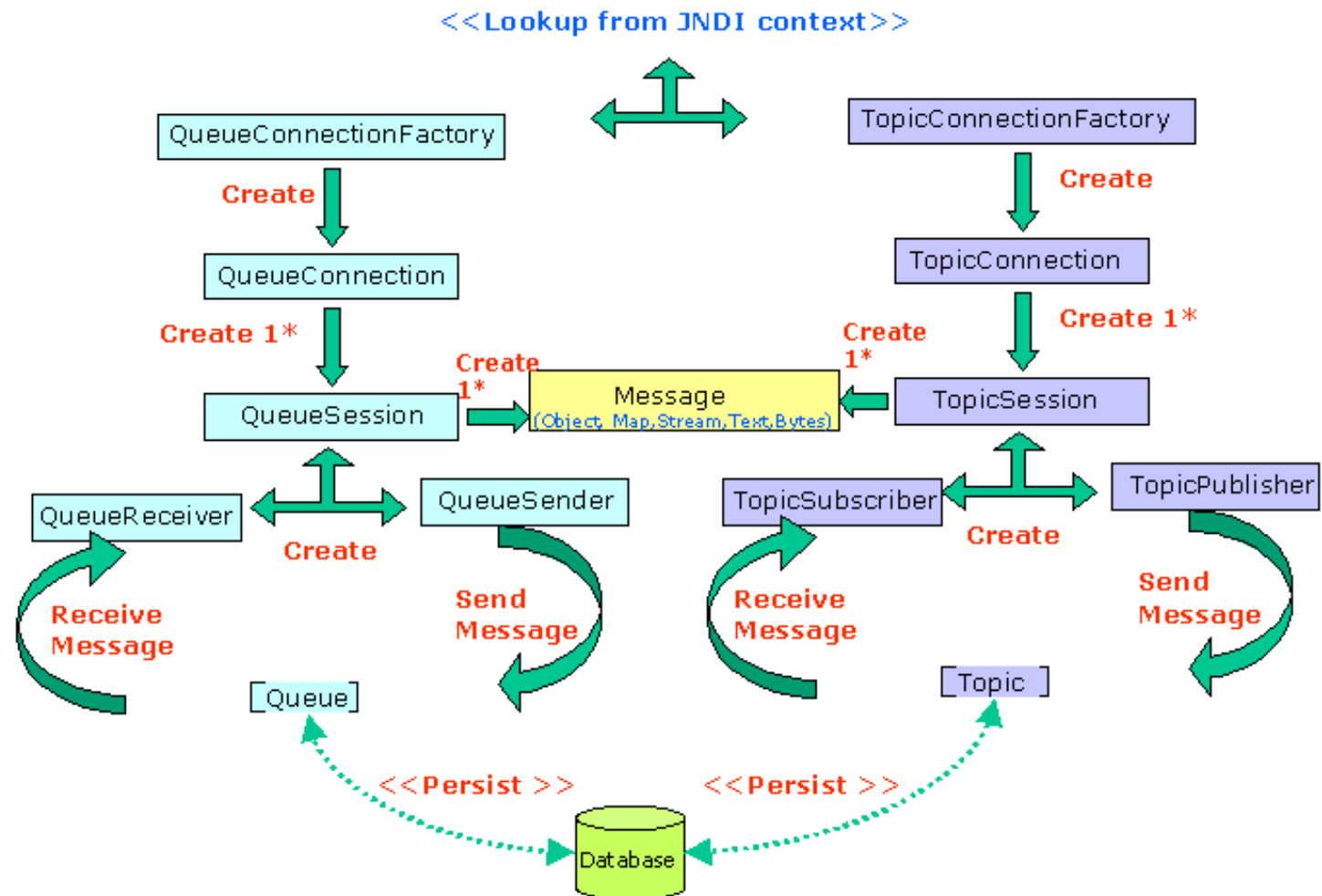


Message Listener

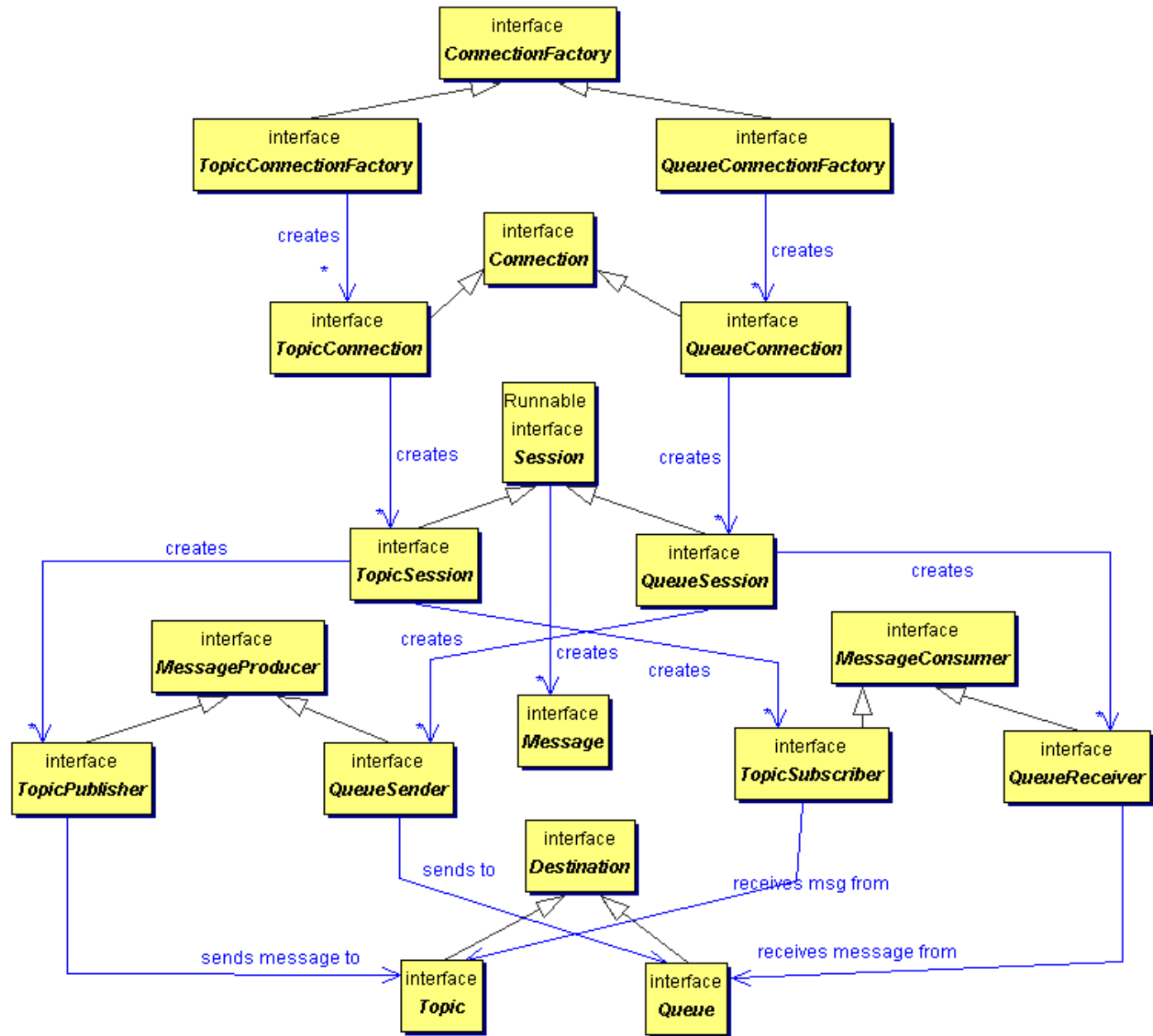
- Bộ quản lý sự kiện không đồng bộ
- Cài đặt giao diện MessageListener với phương thức onMessage
- Một message listener được đăng ký với MessageConsumer

```
public class MyListener implements MessageListener {  
    public void onMessage(Message message) {  
        ...  
    }  
}  
  
MyListener messageListener;  
consumer.setMessageListener(messageListener);
```

JMS APIs



JMS API





6. Các bước viết JMS clients

6.1. Các bước viết JMS Sender Application





Các bước viết JMS Sender Application

- 1. Lấy đối tượng ConnectionFactory và Destination (Topic hoặc Queue) qua JNDI
- 2. Tạo 1 Connection
- 3. Tạo 1 Session để send/receive thông điệp
- 4. Tạo 1 MessageProducer (TopicPublisher hoặc QueueSender)
- 5. Start connection
- 6. Send (publish) thông điệp
- 7. Đóng Session và Connection



(1) Lấy đối tượng ConnectionFactory và Destination (Topic hoặc Queue) qua JNDI

```
// Get JNDI InitialContext object
Context jndiContext = new InitialContext();

// Locate ConnectionFactory object via JNDI
TopicConnectionFactory factory =
(TopicConnectionFactory) jndiContext.lookup(
"MyTopicConnectionFactory");

// Locate Destination object (Topic or Queue)
// through JNDI
Topic weatherTopic =
(Topic) jndiContext.lookup("WeatherData");
```



(2) Tạo Connection Object

```
// Create a Connection object from  
// ConnectionFactory object
```

```
TopicConnection topicConnection =  
factory.createTopicConnection();
```



(3) Tạo 1 Session

```
// Create a Session from Connection object.  
//      1st parameter controls transaction  
//      2nd parameter specifies acknowledgment type
```

```
TopicSession session =  
topicConnection.createTopicSession (false,  
Session.CLIENT_ACKNOWLEDGE);
```



(4) Tạo Message Producer

```
// Create MessageProducer from Session object  
//     TopicPublisher for Pub/Sub  
//     QueueSender for Point-to-Point
```

```
TopicPublisher publisher =  
session.createPublisher(weatherTopic) ;
```



(5) Start Connection

```
// Until Connection gets started, message flow  
// is inhibited: Connection must be started before  
// messages will be transmitted.
```

```
topicConnection.start();
```



(6) Publish thông điệp

```
// Create a Message
```

```
TextMessage message =  
    session.createMessage();  
message.setText("text:35 degrees");
```

```
// Publish the message
```

```
publisher.publish(message);
```

6.2.1. Các bước viết JMS Application Receiver theo chế độ Non-blocking





Các bước viết JMS Application Receiver theo chế độ Non-blocking

- 1. Lấy đối tượng ConnectionFactory và Destination (Topic hoặc Queue) qua JNDI
- 2. Tạo 1 Connection
- 3. Tạo 1 Session để send/receive thông điệp
- 4. Tạo 1 MessageConsumer (TopicSubscriber hoặc QueueReceiver)
- 5. Đăng ký MessageListener cho chế độ non-blocking
- 6. Start Connection
- 7. Đóng Session và Connection



(4) Tạo Message Subscriber

```
// Create Subscriber from Session object  
TopicSubscriber subscriber =  
session.createSubscriber(weatherTopic) ;
```



(5) Đăng ký MessageListener cho chế độ non-blocking

```
// Create MessageListener object
WeatherListener myListener
= new WeatherListener();

// Register MessageListener with
// TopicSubscriber object
subscriber.setMessageListener(myListener);
```

6.2.2. Các bước viết JMS Application Receiver theo chế độ blocking





Các bước viết JMS Application Receiver theo chế độ blocking

- 1. Lấy đối tượng ConnectionFactory và Destination (Topic hoặc Queue) qua JNDI
- 2. Tạo 1 Connection
- 3. Tạo 1 Session để send/receive thông điệp
- 4. Tạo 1 MessageConsumer
- 5. Start Connection
- 6. Nhận message
- 7. Đóng Session và Connection



6.3. Cách thức tạo một ứng dụng JMS có tính chịu lỗi



Theo cách thức tin cậy nhất

- Cách tin cậy nhất để **produce** một thông điệp là gửi một PERSISTENT message trong một giao dịch
- Cách tin cậy nhất để **consume** một thông điệp là thực hiện trong một giao dịch, hoặc từ một queue, hoặc từ một subscription “bền lâu” tới một topic



Độ tin cậy cơ bản

- Điều khiển việc thừa nhận thông điệp
- Chỉ định truyền thông điệp Persistent
- Thiết lập các mức ưu tiên thông điệp
- Cho phép thông điệp hết hạn
- Tạo các destination tạm thời



Các cơ chế với độ tin cậy nâng cao

- Tạo các durable subscriptions
- Sử dụng các giao dịch cục bộ (local transactions)



6.3.1. Điều khiển việc thừa nhận thông điệp



Các pha xử lý thông điệp nhận được

- Client nhận thông điệp
- Client xử lý thông điệp
- Thông điệp được acknowledged
 - Việc thừa nhận được khởi tạo hoặc bởi JMS provider, hoặc bởi client, tùy theo chế độ thừa nhận của Session
 - Session.AUTO_ACKNOWLEDGE
 - Session.CLIENT_ACKNOWLEDGE: Bên nhận gọi `message.acknowledge()`
 - Session.DUPS_OK_ACKNOWLEDGE: Ví lý do hiệu năng, JMS server cho phép acknowledge mà không lưu lại ngưỡng cảnh, cho phép trùng lặp thông điệp.



Transaction và Acknowledgment

- Trong các sessions giao dịch hóa:
 - Việc thừa nhận (ACK) xảy ra tự động khi một transaction được commit
 - Nếu 1 transaction được roll back, tất cả các thông điệp đã được “tiêu thụ” sẽ được truyền lại
- Trong các sessions giao dịch hóa
 - Khi nào & cách thức 1 thông điệp được thừa nhận phụ thuộc vào giá trị được chỉ định (slide tiếp) làm tham số thứ 2 của phương thức `createSession`



Acknowledgment Types

- Auto acknowledgment (AUTO_ACKNOWLEDGE)
 - Thông điệp được xem như đã được thừa nhận khi trả về thành công trong `MessageConsumer.receive()` và `MessageListener.onMessage()`
- Client acknowledgment (CLIENT_ACKNOWLEDGE)
 - Client phải gọi phương thức `acknowledge()` của đối tượng `Message`
- Lazy acknowledgment (DUPS_OK_ACKNOWLEDGE)
 - Thông điệp được thừa nhận ngay khi chúng đến với consumers




Cách thức thiết lập Acknowledgment Type trong JMS

- Một Acknowledgment Type được thiết lập khi tạo Session bằng cách thiết lập cờ thích hợp
 - `QueueConnection.createQueueSession(.., <flag>)`
 - `TopicConnection.createTopicSession(.., <flag>)`
- Ví dụ:

```
TopicSession session =  
topicConnection.createTopicSession  
(false, Session.CLIENT_ACKNOWLEDGE) ;
```

6.3.2. Chỉ định truyền thông điệp Persistent (Các chế độ truyền)





2 chế độ truyền

- Chế độ truyền PERSISTENT
 - Mặc định
 - Hướng dẫn JMS provider thực hiện các hành động cần thiết để 1 thông điệp không bị mất khi truyền trong trường hợp gặp sự cố
- Chế độ truyền NON_PERSISTENT
 - Không yêu cầu JMS provider lưu trữ thông điệp
 - Hiệu năng tốt hơn



Cách thức thiết lập chế độ truyền

- Phương thức `setDeliveryMode` của interface `MessageProducer`
 - `producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);`
- Sử dụng kiểu sau của phương thức `send` hoặc `publish`
 - `producer.send(message, DeliveryMode.NON_PERSISTENT, 3, 10000);`

6.3.3. Thiết lập các mức ưu tiên thông điệp



Cách thức thiết lập mức ưu tiên khi truyền

- Có 10 mức ưu tiên
 - Từ 0 (thấp nhất)-9 (cao nhất)
 - Mặc định là 4
- Sử dụng phương thức `setPriority` của interface `MessageProducer`
 - `producer.setPriority(7);`
- Sử dụng kiểu sau của phương thức `send` hoặc `publish`
 - `producer.send(message, DeliveryMode.NON_PERSISTENT, 7, 10000);`

6.3.4. Cho phép thông điệp hết hạn



Cách thức thiết lập thông số hết hạn cho thông điệp

- Sử dụng phương thức `setTimeToLive` của interface `MessageProducer`
 - `producer.setTimeToLive(60000);`
- Sử dụng kiểu sau của phương thức `send` hoặc `publish`
 - `producer.send(message, DeliveryMode.NON_PERSISTENT, 3, 60000);`

6.3.5. Tạo các durable subscriptions





Độ tin cậy cực đại

- Để đảm bảo 1 ứng dụng pub/sub nhận được tất cả các published messages
 - Sử dụng chế độ truyền PERSISTENT cho publishers
 - Thêm vào đó, sử dụng các durable subscriptions cho các subscribers
 - Sử dụng phương thức `Session.createDurableSubscriber` để tạo 1 durable subscriber

`String clientId;`

`connection.setClientID(clientId);`

`consumer = session.createDurableSubscriber((Topic) destination,
consumerName);`



Cách thức durable subscription làm việc

- Một durable subscription có thể có chỉ 1 subscriber active tại 1 thời điểm
- Một durable subscriber đăng ký 1 durable subscription bằng việc chỉ ra 1 định danh duy nhất lưu trữ bởi JMS provider
- Các đối tượng subscriber sau có cùng định danh sẽ bắt đầu lại subscription ở trạng thái của các subscriber trước đó
- Nếu 1 durable subscription không có subscriber active nào, JMS provider sẽ lưu các thông điệp của subscription cho đến khi chúng được nhận bởi subscription hoặc cho đến khi nó hết hạn



6.3.6. Giao dịch trong JMS



Giao dịch trong JMS

- Phạm vi giao dịch là chỉ giữa client và hệ thống thông điệp, không phải giữa các client
 - 1 nhóm các thông điệp được phát ra như 1 đơn vị (từ phía gửi)
 - 1 nhóm các thông điệp được nhận lại như 1 đơn vị (từ phía nhận)
- Các giao dịch “Local” và “Distributed”



Giao dịch cục bộ trong JMS

- Các giao dịch cục bộ được điều khiển bởi đối tượng Session
- Giao dịch bắt đầu khi 1 session được tạo
 - Không có phương thức “bắt đầu giao dịch” tường minh
- Giao dịch kết thúc khi gọi `Session.commit()` hoặc `Session.abort()`
- Transactional session được tạo ra bằng cách chỉ định cờ thích hợp khi tạo 1 session
 - `QueueConnection.createQueueSession(true, ..)`
 - `TopicConnection.createTopicSession(true, ..)`



Giao dịch phân tán trong JMS

- Được điều hành bởi 1 transactional manager
- Các ứng dụng sẽ điều khiển giao dịch qua các phương thức JTA
 - Cấm sử dụng `Session.commit()` và `Session.rollback()`



6.3.7. Message Selector



JMS Message Selector

- (Receiver) JMS application sử dụng 1 bộ chọn để lấy ra chỉ những thông điệp quan tâm
- Một bộ chọn cơ bản nhất là 1 xâu SQL92, chỉ rõ luật chọn (hay lọc)



Ví dụ: JMS message selectors


- Bộ chọn không thể tham chiếu đến nội dung của 1 thông điệp
- Có thể truy cập tới properties và header
- Ví dụ
 - JMSType=='wasp'
 - phone LIKE '223'
 - price BETWEEN 100 AND 200
 - name IN('sameer','tyagi')
 - JMSType IS NOT NULL



Xử lý ngoại lệ

- exception cơ sở: JMSException
- Các lớp kế thừa: IllegalStateException, InvalidClientIDException, InvalidDestinationException, InvalidSelectorException, JMSSecurityException, MessageEOFException, MessageFormatException, MessageNotReadableException, MessageNotWritableException, ResourceAllocationException, TransactionInProgressException, TransactionRolledBackException

7. Các đặc tính thông điệp không được định nghĩa trong JMS





Các đặc tính thông điệp không được định nghĩa trong JMS

- Encryption
 - JMS giả thiết hệ thống thông điệp đã xử lý rồi
- Điều khiển truy cập
 - JMS giả thiết hệ thống thông điệp đã xử lý rồi
- Load balancing
- Quản trị các queue và topic



8. Xử lý giao dịch

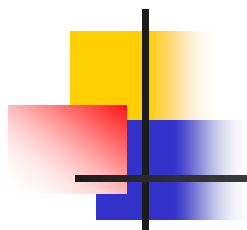
```
boolean transactionMode=true;
javax.jms.Session session;
session=connection.createSession(transaction
    Mode, ackMode);

....

session.commit();
session.rollback();
```



Phụ lục



Kiến trúc của ứng dụng JMS

