JMS技术

Next Generation Integration Platform

Allen Long

Email: allen@huihoo.com

http://www.huihoo.com

2004-04

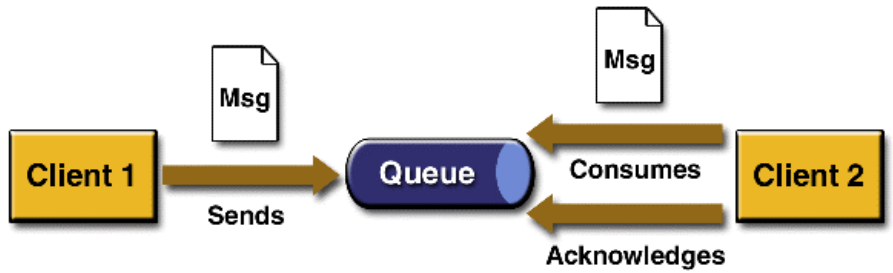
JMS specification

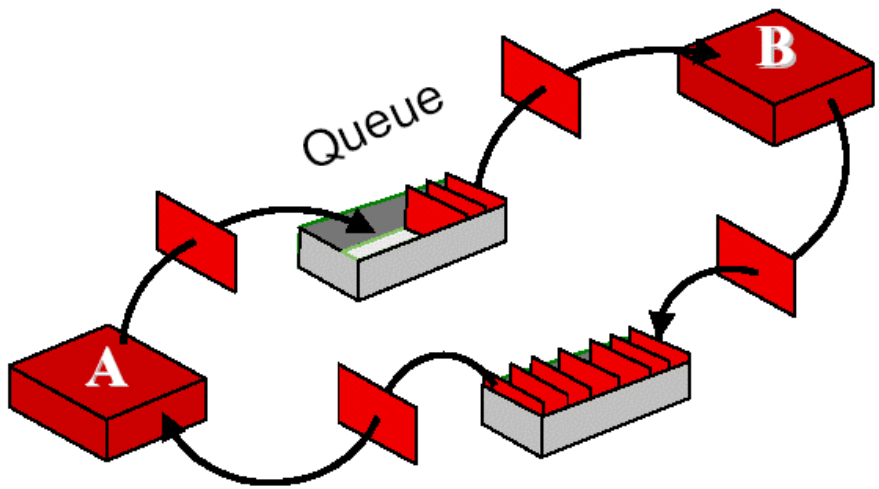



- Two message Models queues, topics
- Persistence and no-persistence message delivery
- Five message types – text, byte, object, map and stream
- Synchronous and Asynchronous message consumption
- Selector, priority, queueBrowser
- Local and distributed transaction

PTP Messaging Domain

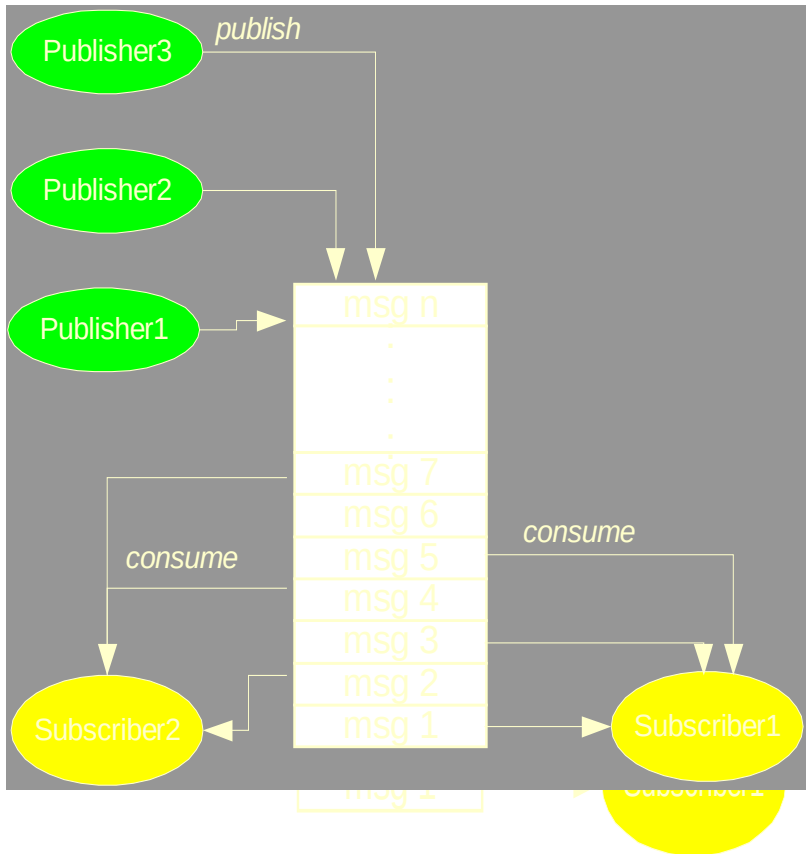
- Each message has only one consumer
- A sender and a receiver of a message have no timing dependencies. The receiver can fetch the message whether or not it was running when the client sent the message.
- The receiver acknowledges the successful processing of a message





JMS Specifications - Queues

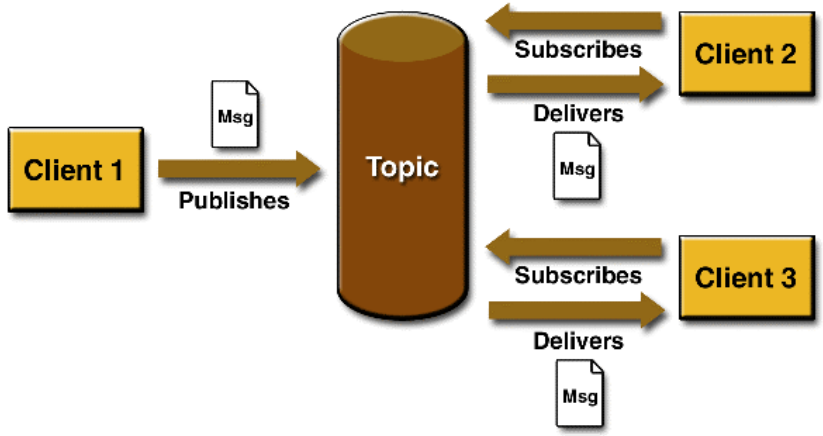
- Sender publishers to the Queue and Receiver consumers
- Supports multiple senders/receivers per queue
- Message Ordering
- Undefined Delivery semantics for > 1 receiver
- Non-persistent messages **delivered at most once**
- Persistent messages **delivered exactly once**

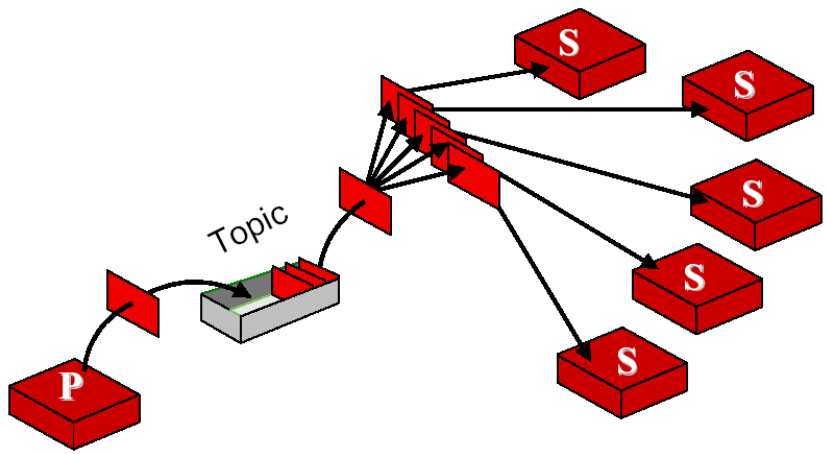


P/S Messaging Domain



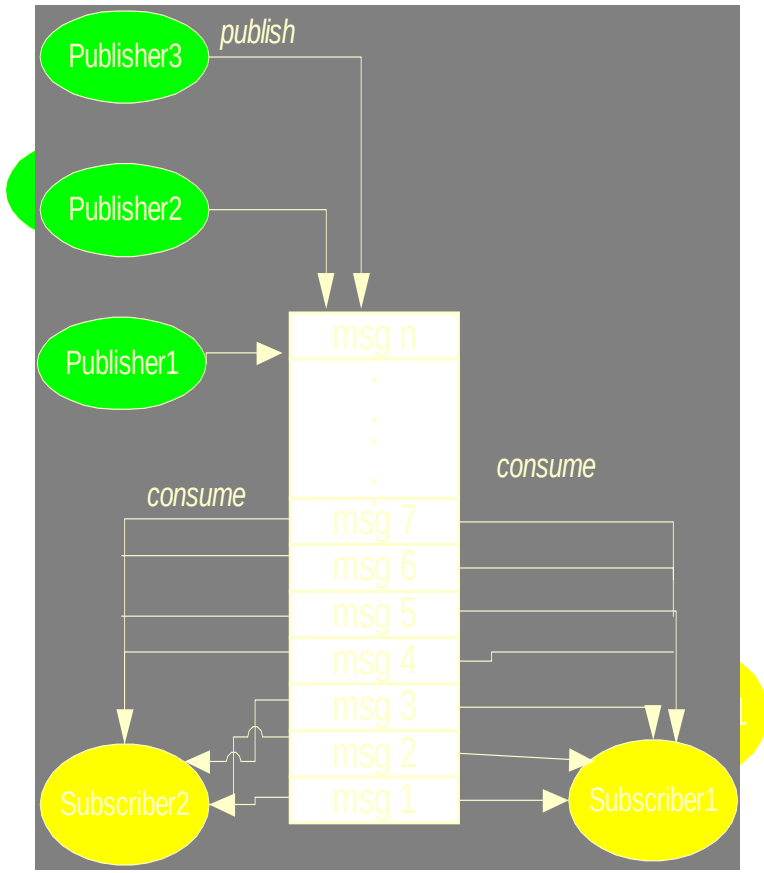

- Each message may have multiple consumers.
- Publishers and subscribers have a timing dependency A client that subscribes to a topic can consume only messages published after the client has created a subscription, and the subscriber must continue to be active in order for it to consume messages.



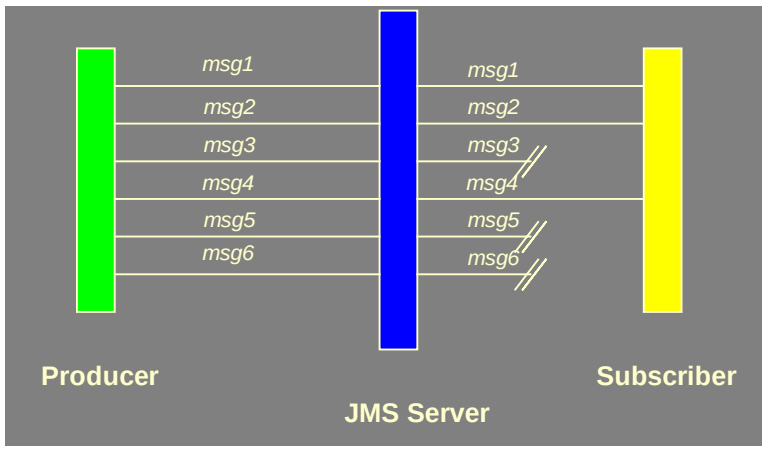


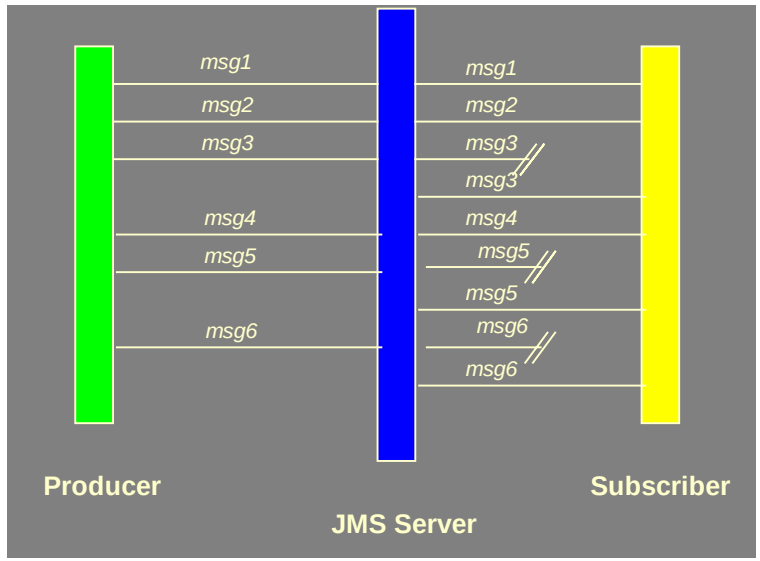
JMS Specifications - Topics

- Publisher publishes and Subscribers consumes
- Multiple publishers/subscribers per topic
- Subscriber each receives own copy of messages
- Priority effects message delivery order
- Persistent and non-persistent delivery semantics



Persistence and no-persistence message delivery





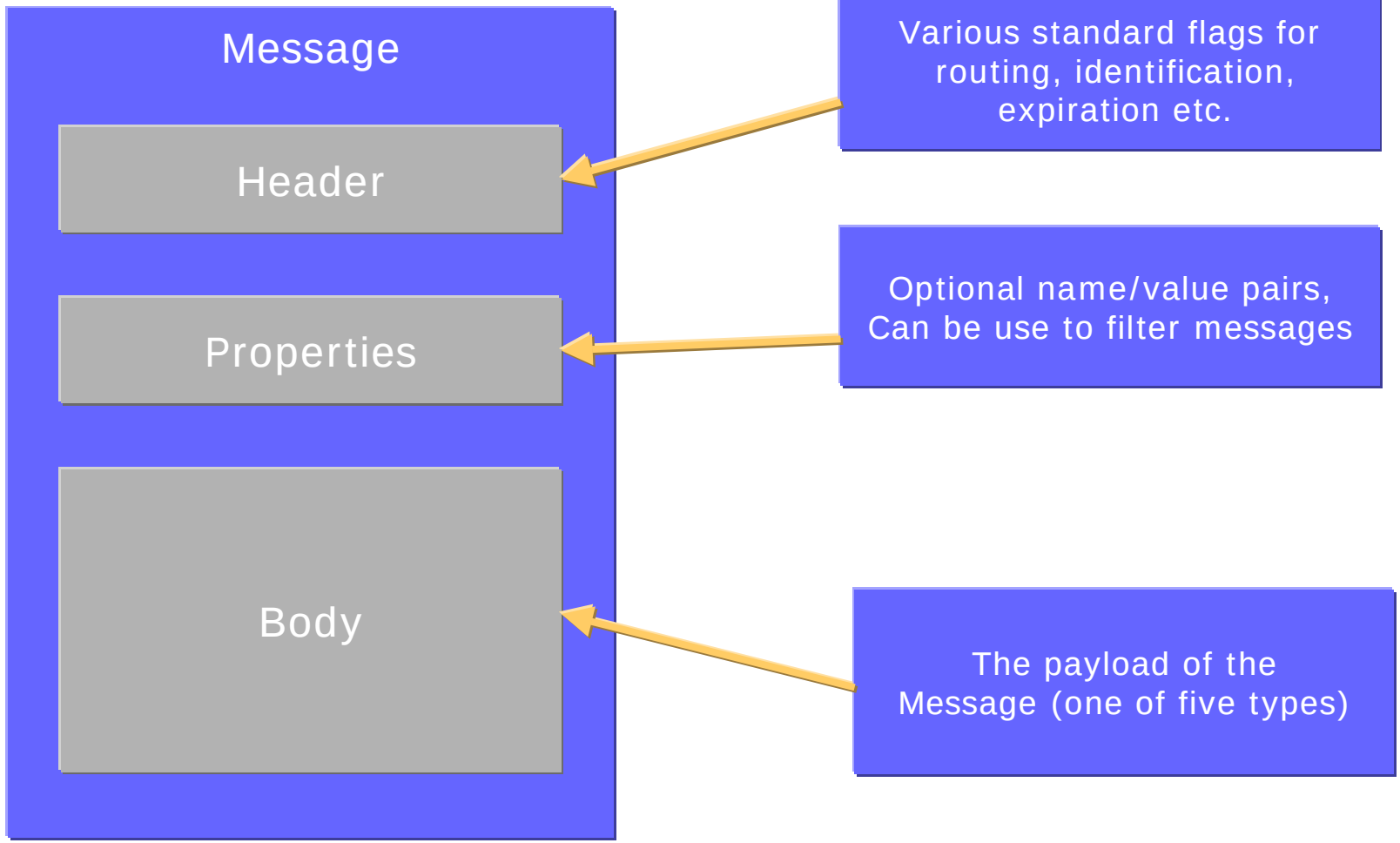
JMS Message

A JMS message has three parts:

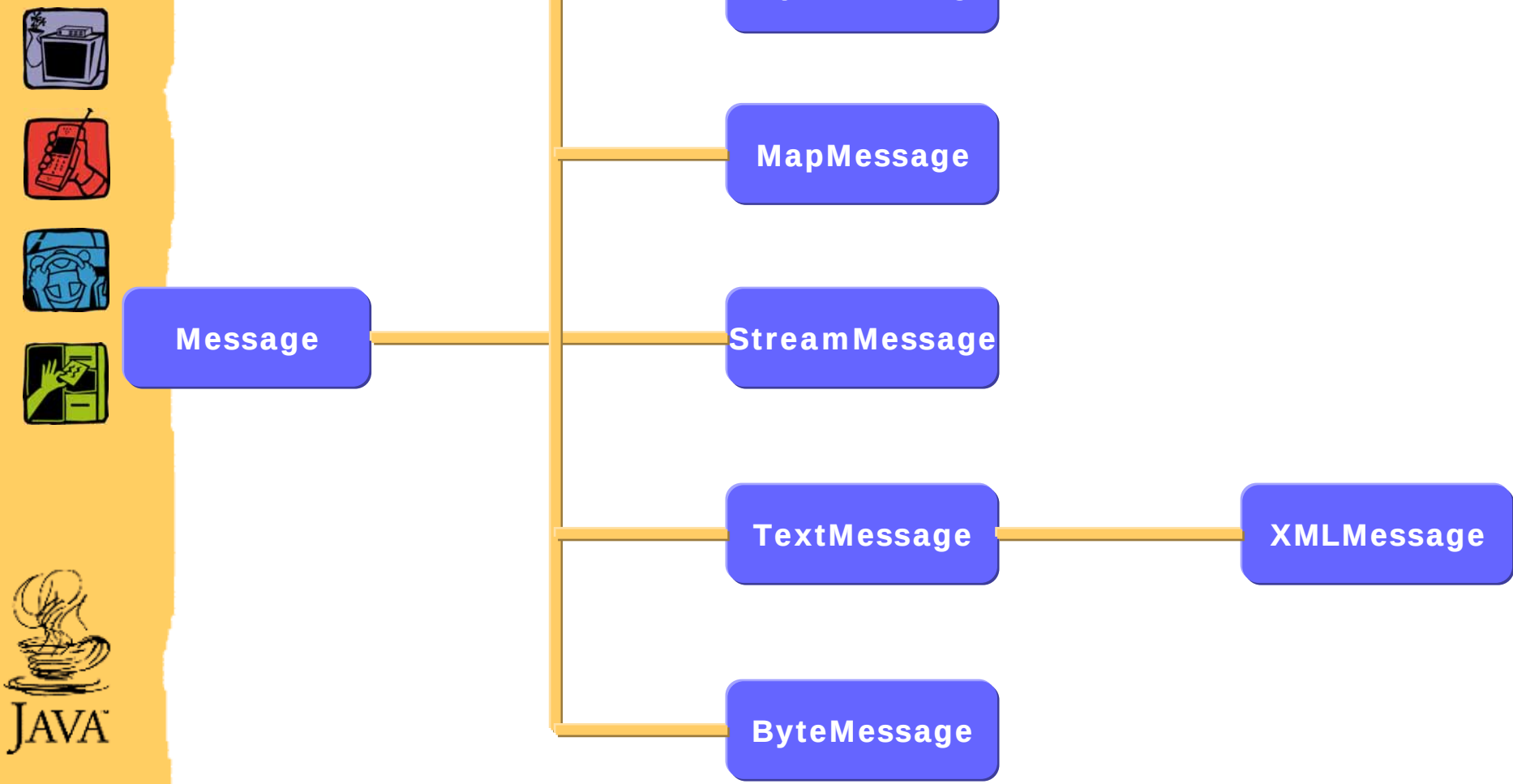
- A header
- Properties (optional)
- A body (optional)

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

Structure of JMS Message



JMS Message Type



What is JMS API?




- The Java Message Service is a Java API that allows applications to create, send, receive, and read messages.

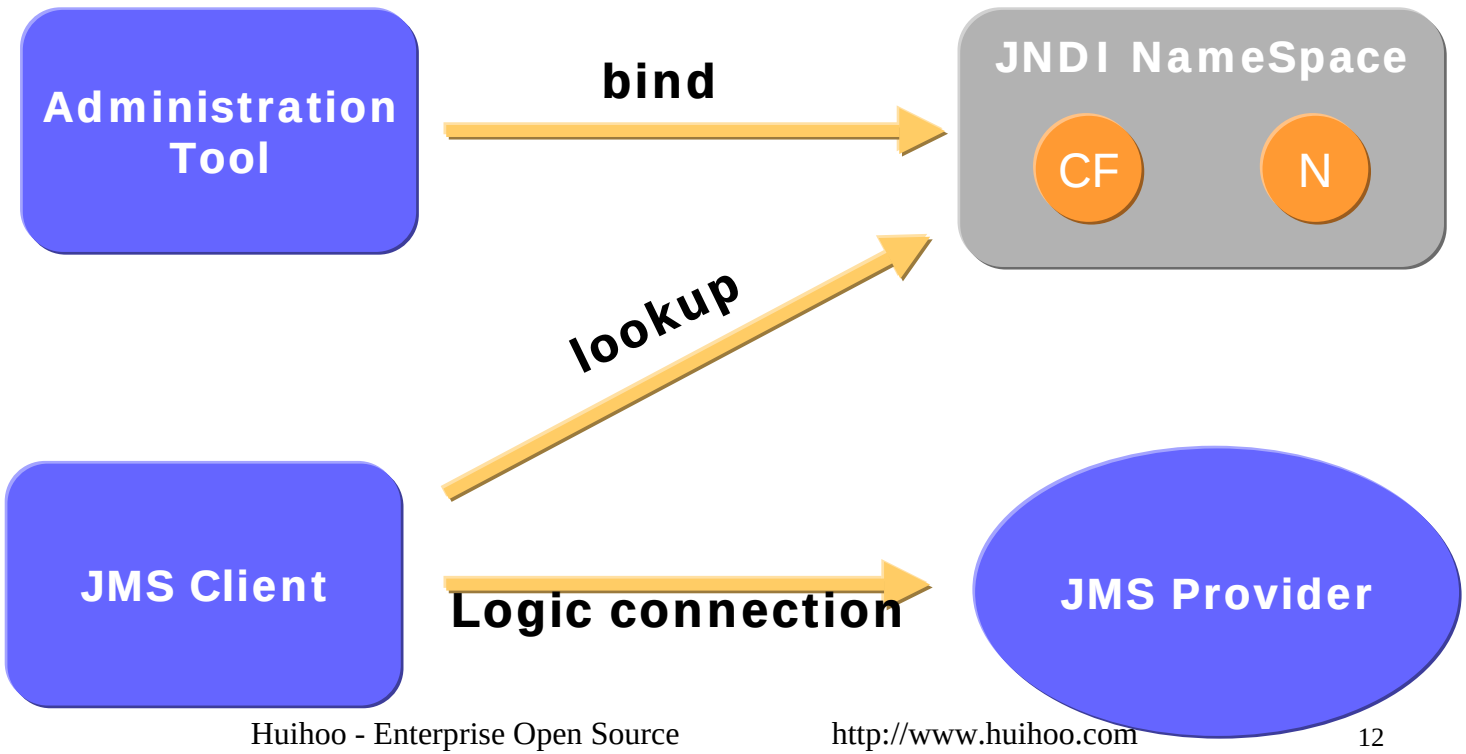
The JMS API enables communication that is not only loosely coupled but also

- **Asynchronous** A JMS provider can deliver messages to a client as they arrive; a client does not have to request messages in order to receive them.
- **Reliable** The JMS API can ensure that a message is delivered once and only once. Lower levels of reliability are available for applications that can afford to miss messages or to receive duplicate messages.

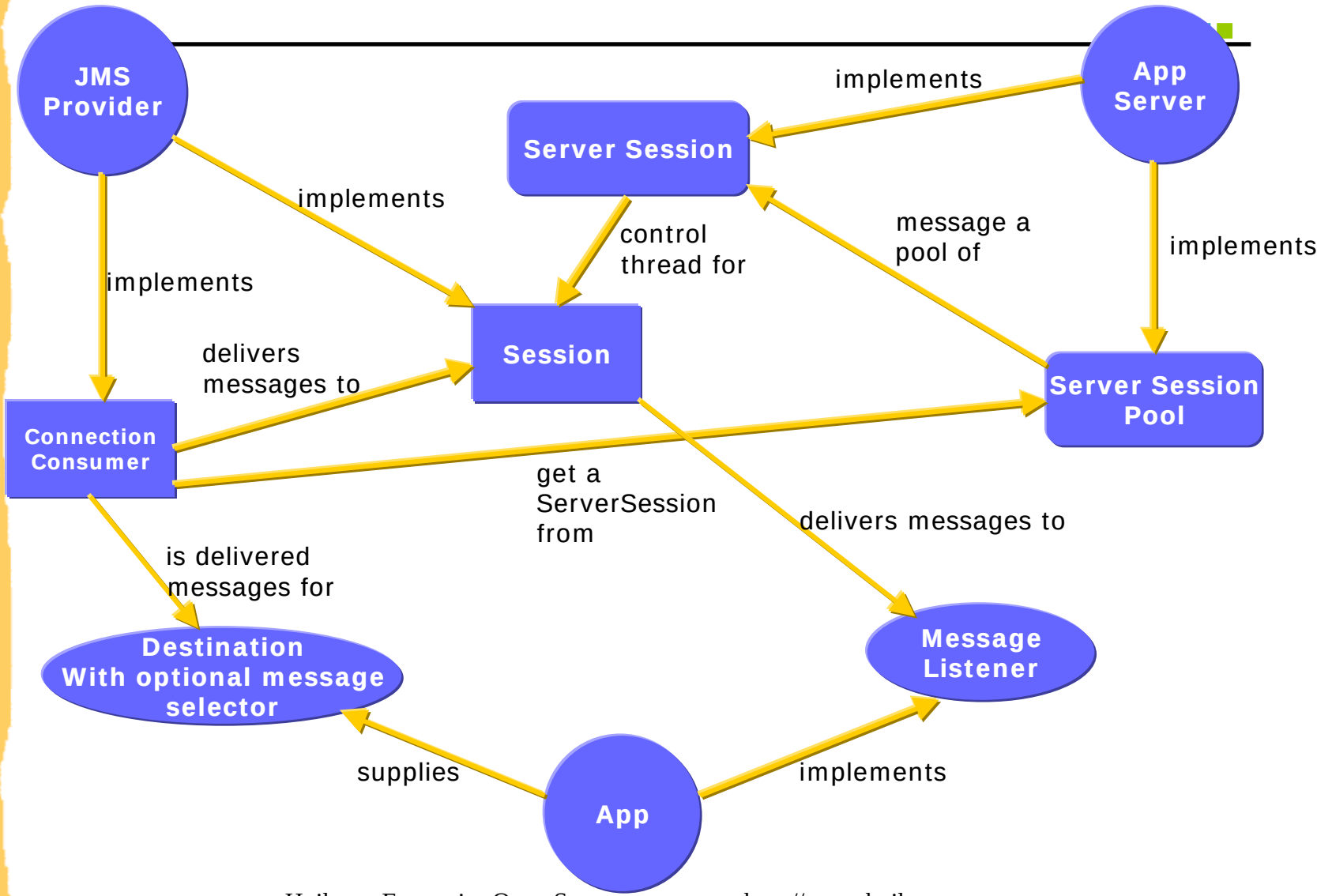
JMS API Architecture

Administrative tools allow you to bind destinations and connection factories into a Java Naming and Directory Interface TM (JNDI) API namespace.

- A JMS client can then look up the administered objects in the namespace and then establish a logical connection to the same objects through the JMS provider.



Relationship --- JMS Provider, App Server, App

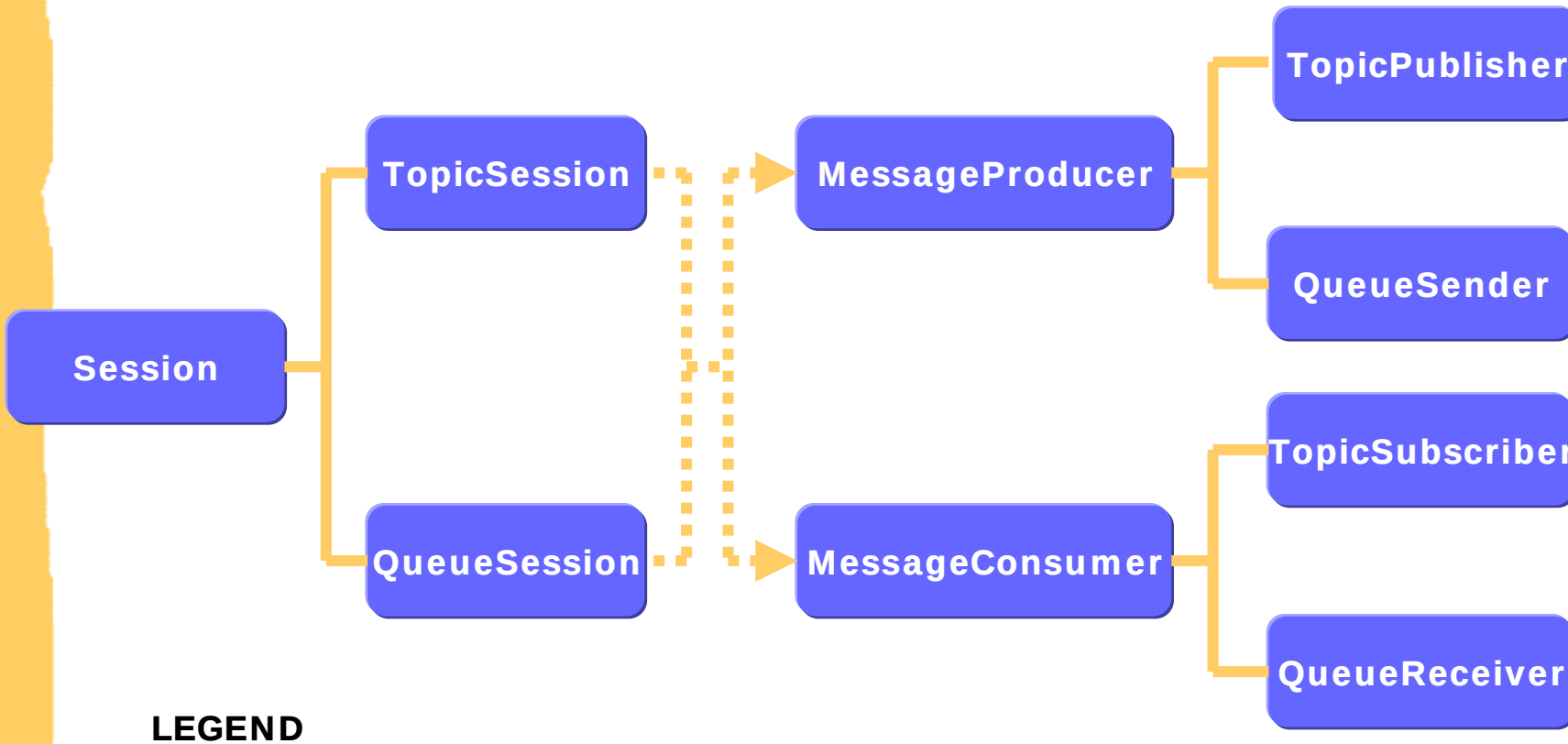


Objects

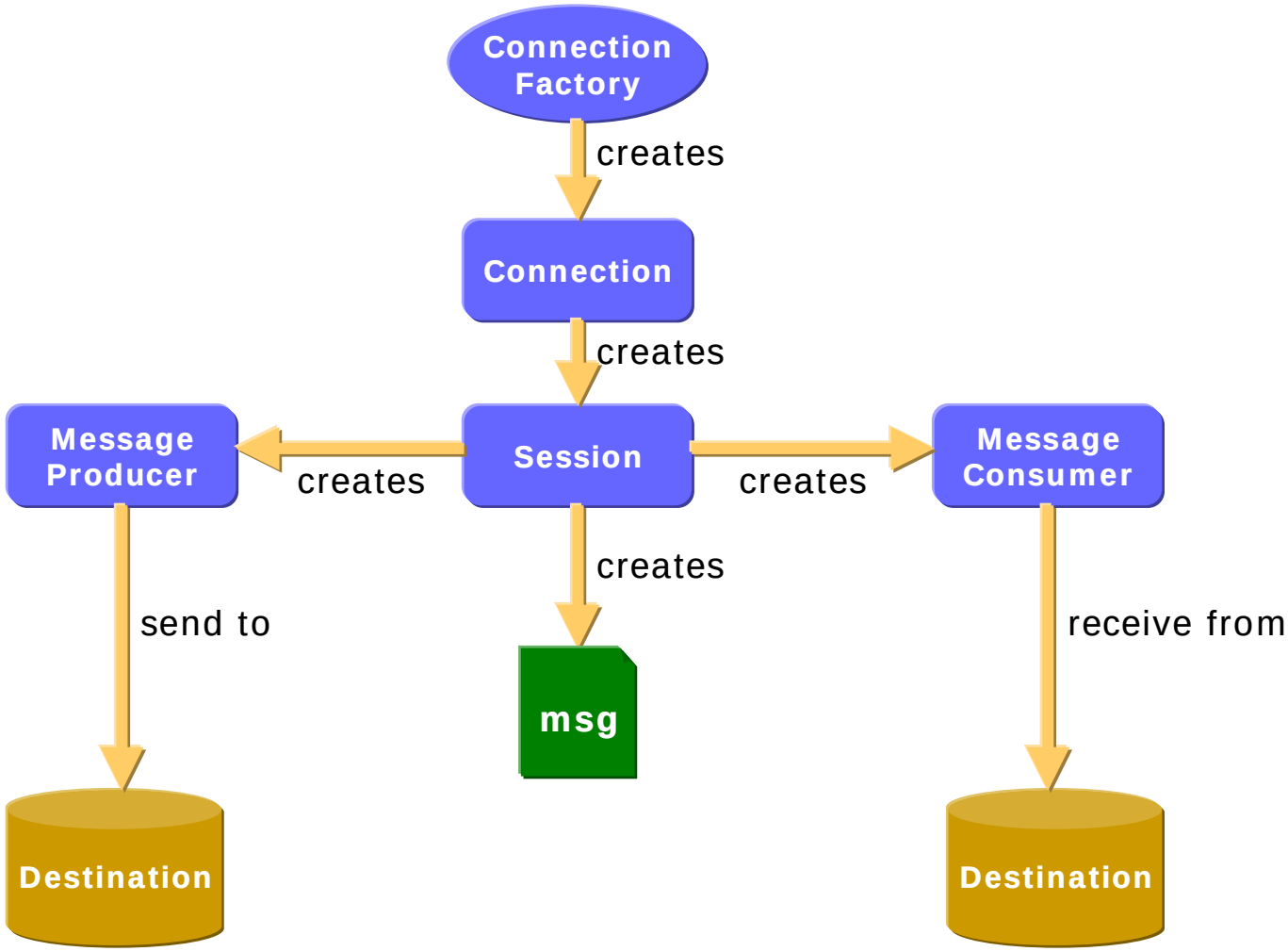
- **ConnectionFactory** - an administered object used by a client to create a Connection
- **Connection** - an active connection to a JMS provider
- **Destination** - an administered object that encapsulates the identity of a message destination
- **Session** - a single-threaded context for sending and receiving messages
- **MessageProducer** - an object created by a Session that is used for sending messages to a destination
- **MessageConsumer** - an object created by a Session that is used for receiving messages sent to a destination

JMS Common Interfaces	PTP-specific Interfaces	Pub/Sub-specific interfaces
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver, QueueBrowser	TopicSubscriber

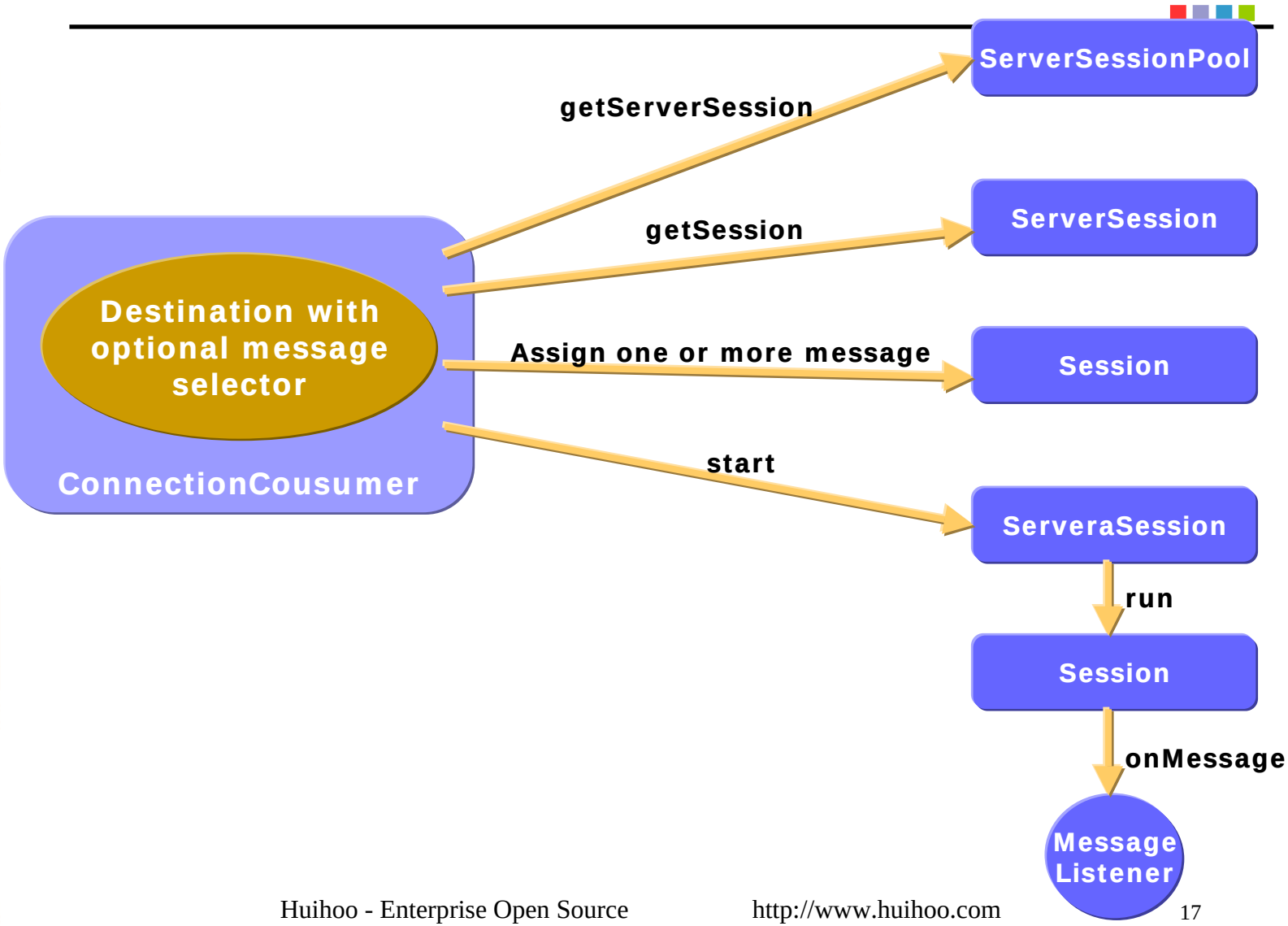
JMS API


Session
TopicSession
QueueSession
MessageProducer
MessageConsumer
TopicPublisher
QueueSender
TopicSubscriber
QueueReceiver
LEGEND
Inheritance (Extend)
Message Flow

JMS API Programming Model



Deliver a message to a MessageListener



Message Consumption

- **Synchronously** A subscriber or a receiver explicitly fetches the message from the destination by calling the receive method. The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit.
- **Asynchronously** A client can register a *message listener* with a consumer. A message listener is similar to an event listener. Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's onMessage method, which acts on the contents of the message.

Reliability Mechanisms



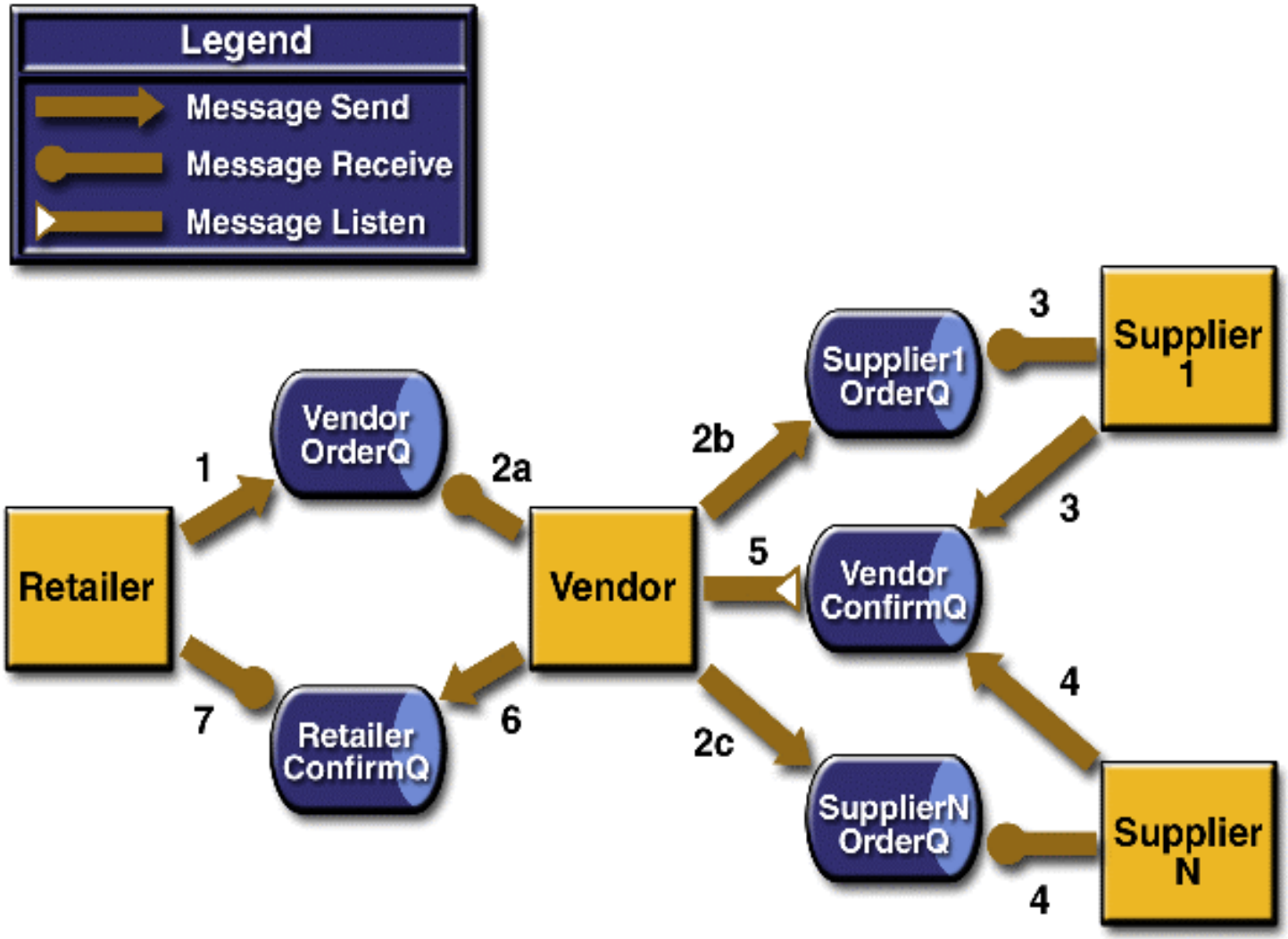

- **Controlling message acknowledgment.** You can specify various levels of control over message acknowledgment.
- **Specifying message persistence**. You can specify that messages are persistent, meaning that they must not be lost in the event of a provider failure.
- **Setting message priority levels.** You can set various priority levels for messages, which can affect the order in which the messages are delivered.
- **Allowing messages to expire**. You can specify an expiration time for messages, so that they will not be delivered if they are obsolete.
- **Creating temporary destinations**. You can create temporary destinations that last only for the duration of the connection in which they are created.
- **Creating durable subscriptions**. You can create durable topic subscriptions, which receive messages published while the subscriber is not active. Durable subscriptions offer the reliability of queues to the publish/subscribe message domain.
- **Using local transactions**. You can use local transactions, which allow you to group a series of sends and receives into an atomic unit of work. Transactions are rolled back if they fail at any time.

Demo

- Point to Point

- Publisher/Subscriber

A PTP Example



Implement

- The program contains five classes: Retailer, Vendor, GenericSupplier, VendorMessageListener, and Order.
- The program also contains a main method and a method that runs the threads of the Retail, Vendor, and two supplier classes.
- All the messages use the MapMessage message type.
- Synchronous receives are used for all message reception except for the case of the vendor processing the replies of the suppliers. These replies are processed asynchronously and demonstrate how to use transactions within a message listener.
- At random intervals, the Vendor class throws an exception to simulate a database problem and cause a rollback.
- All classes except Retailer use transacted sessions.
- The program uses five queues. Before you run the program, create the queues and name them A, B, C, D and E.

Steps

- j2eeadmin -addJmsDestination A queue
- j2eeadmin -addJmsDestination B queue
- j2eeadmin -addJmsDestination C queue
- j2eeadmin -addJmsDestination D queue
- j2eeadmin -addJmsDestination E queue
- java -Djms.properties=$J2EE_HOME/config/jms_client.properties -Dorg.omg.CORBA.ORBInitialHost=192.168.30.10 TransactedExample 2

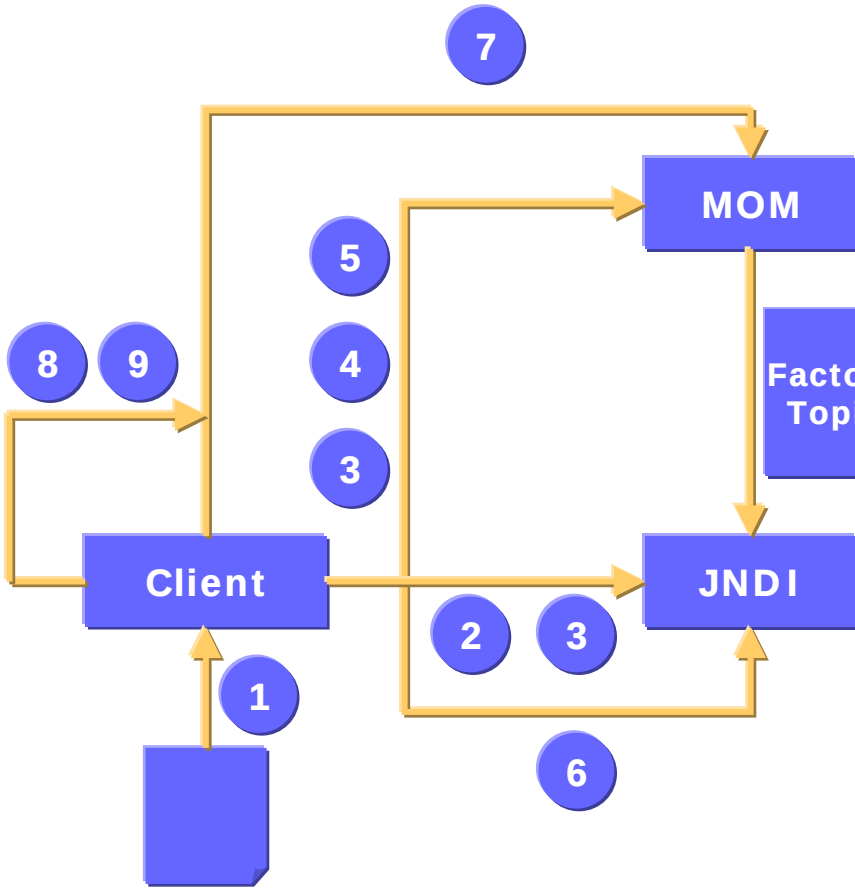
Result

Quantity to be ordered is 2
Java(TM) Message Service 1.0.2 Reference Implementation (build b14)
Java(TM) Message Service 1.0.2 Reference Implementation (build b14)
Java(TM) Message Service 1.0.2 Reference Implementation (build b14)
Java(TM) Message Service 1.0.2 Reference Implementation (build b14)
Retailer: ordered 2 computer(s)
Vendor: Retailer ordered 2 Computer(s)
Vendor: ordered 2 Monitor(s)
Vendor: ordered 2 Hard Drive(s)
  Vendor: committed transaction 1
Monitor Supplier: Vendor ordered 2 Monitor(s)
Monitor Supplier: sent 2 Monitor(s)
Hard Drive Supplier: Vendor ordered 2 Hard Drive(s)
Hard Drive Supplier: sent 2 Hard Drive(s)
  Monitor Supplier: committed transaction
  Hard Drive Supplier: committed transaction
Vendor: Completed processing for order 1
Vendor: sent 2 computer(s)
  Vendor: committed transaction 2
Retailer: Order filled

JMS P/S Message Sequence (asynchronously)

Publisher / Subscriber of a JMS Application

9 Start connection
8 Initialize message listener
7 Create publisher / Subscriber
6 Get topic
5 Create session
4 Create connection
3 Create connection factory
2 Create context
1 Define Properties



Properties, Context (JNDI)

Goal: Set-up JNDI by setting its root context

- Final static String INITCTX = “com.sun.ldap.LdapCtxFactory”
- Hashtable env = new Hashtable()
- env.put(Context.INITIAL_CONTEXT_FACTORY, INITCTX)
- Context context = new Context(env)

ConnectionFactory, Topic

Goal: Retrieve Administered Objects from JNDI

- topicConnectionFactory = (TopicConnectionFactory) context.lookup (“TopicConnectionFactory”)

- topic = (Topic) context.lookup(“topicName”)

ConnectionFactory

- Sole purpose: Create JMS Connection

- The ConnectionFactory can be customized before it is bound with JNDI

  ---Server to connect to (IP Address or DNS name)
  ---Port to connection to
  ---Protocol to use (TCP, UDP, HTTP, SSL)
  ---Reconnection-Strategy to use, etc.

Topic

- Created and administrated by system admin
- Handle for the physical implementation of topic in JMS server
- Encapsulates vendor specific name of physical topic
- Provides a volatile m:n connection of publishers and subscribers

Connection, Session




Goal: Create session with the JMS provider

- TopicConnection topicConnection = TopicConnectionFactory.createTopicConnection()

- TopicSession topicSession = topicConnection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE)

TopicConnection

- Unique, direct connection to the message server
- Factory for TopicSessions
- Expensive!

---Avoid using more than one connection per client

---Except if you want to communicate with more than one JMS provider (Birdge Application)

TopicSession

- Factory for object

---Message (including their type)

---Topic subscriber

---Topic publisher

- Defines transactional behavior
- Defines acknowledgement behavior
- Assigns messages to topics
- Not thread-safe!

Publisher, Session start




Goal: Get ready to Publish Messages

- TopicPuslisher topicPusbisher = topicSession.createPublisher(topic)
- topicConnection.start()

TopicPublisher

- Helper Object to assist in publishing message

Create and Publish a Message

Goal: Publish a Message

- String quoteStr = “I am coming”
- TextMessage quote = topicSession.createTextMessage(quoteStr)
- topicPublisher.publish(quote)

Receive Messages

Goal: Receiver one message synchronous

- TopicSubscriber topicSubscriber = topicSession.createSubscriber(topic)

- Message msg = topicSubscriber.receive()

String message = ((TextMessagg)msg).getText()

System.out.println("Got message : " + message)

TopicSubscriber

- Helper Object to assist in receiver message
- Asynchronous usage needs a MessageListener
- Synchronous usage via receive()
- Also used to register ExceptionListener, to display asynchronous exception

Receive Messages

Goal: Set-up Asynchronous Message Receipt

- TopicSubscriber topicSubscriber = topicSession.createSubscriber(topic)
- topicSubscriber.setMessageListener(new Listener())
- topicConnection.setExceptionListener(new ErrorListener())

MessageListener

Goal: Process Message

```
class listener implements MessageListener {
    public void onMessage(Message m) {
        String message = null;
        try {
            message = ((TextMessage)m).getText():
        } catch (JMSException e) {
            System.out.println("Error")
        }
        System.out.println("Got message : " + message);
    }

}
```

Sample Code

- SimpleTopicPublisher.java
- SimpleTopicSubscriber.java
- TextListener.java

Running the Pub/Sub Clients

- **Starting the JMS Provider**

j2ee -verbose

- **Creating the JMS Administered Objects**

j2eeadmin -addJmsDestination MyTopic topic

- **To verify that the queue has been created, use the following command**

j2eeadmin -listJmsDestination

java -Djms.properties=%J2EE_HOME%\config\jms_client.properties
SimpleTopicSubscriber MyTopic

java -Djms.properties=%J2EE_HOME%\config\jms_client.properties
SimpleTopicPublisher MyTopic 3

- **Deleting the Topic and Stopping the Server**

j2eeadmin -removeJmsDestination MyTopic
j2ee -stop

Output




- The output of the program looks like this:

Topic name is MyTopic

Publishing message: This is message 1

Publishing message: This is message 2

Publishing message: This is message 3

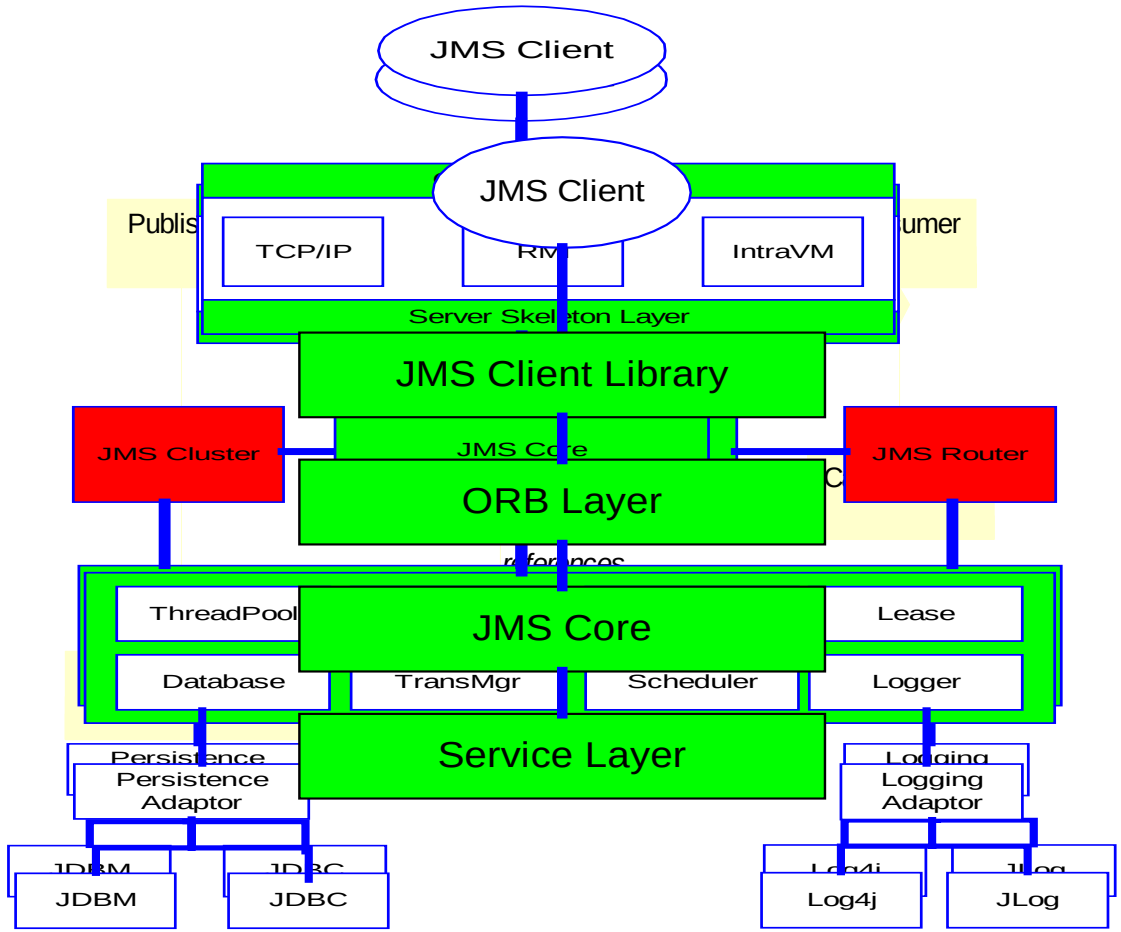
- In the other window, the program displays the following:

Reading message: This is message 1

Reading message: This is message 2

Reading message: This is message 3

Server Internals



结束

谢谢大家！

Allen@huihoo.com
http://www.huihoo.com