Java Message Service (JMS)

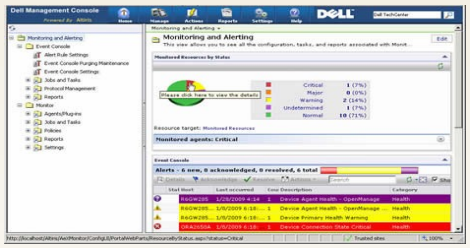
JMS Nedir?

- Java uygulamaları arasında **mesajlaşma ile iletişim kurma**yı sağlar
- Mesajı gönderen ve alan uygulamaların veya sistemlerin **birbirlerini bilmelerine gerek yok**tur
- İki taraf için de sadece **mesaj formatı ve mesajın varış noktası**nın bilinmesi yeterlidir
- Mesaj gönderme ve alma işlemleri **mesaj sunucusu aracı**lığı ile gerçekleştirilir
- Sistemlerin birbirleri ile **asenkron** biçimde **iletişim** kurmalarına imkan sunar

JMS Mimarisi

JNDI Context

J2EE Admin Console

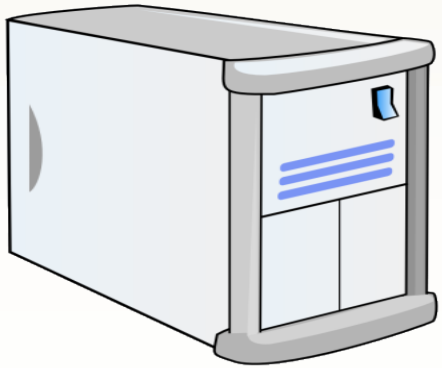


bind

Connection Factory

Destination

lookup

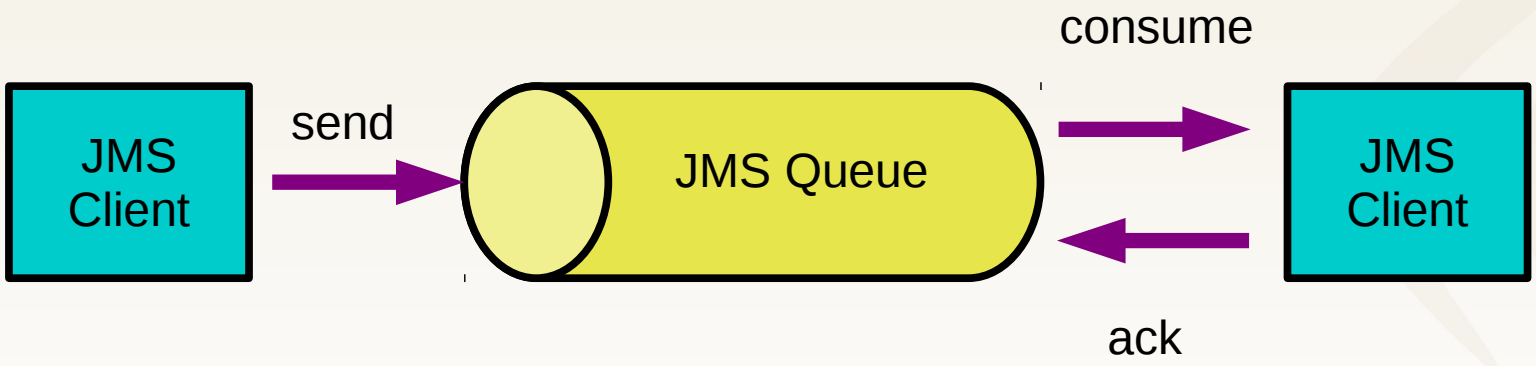


JMS Provider

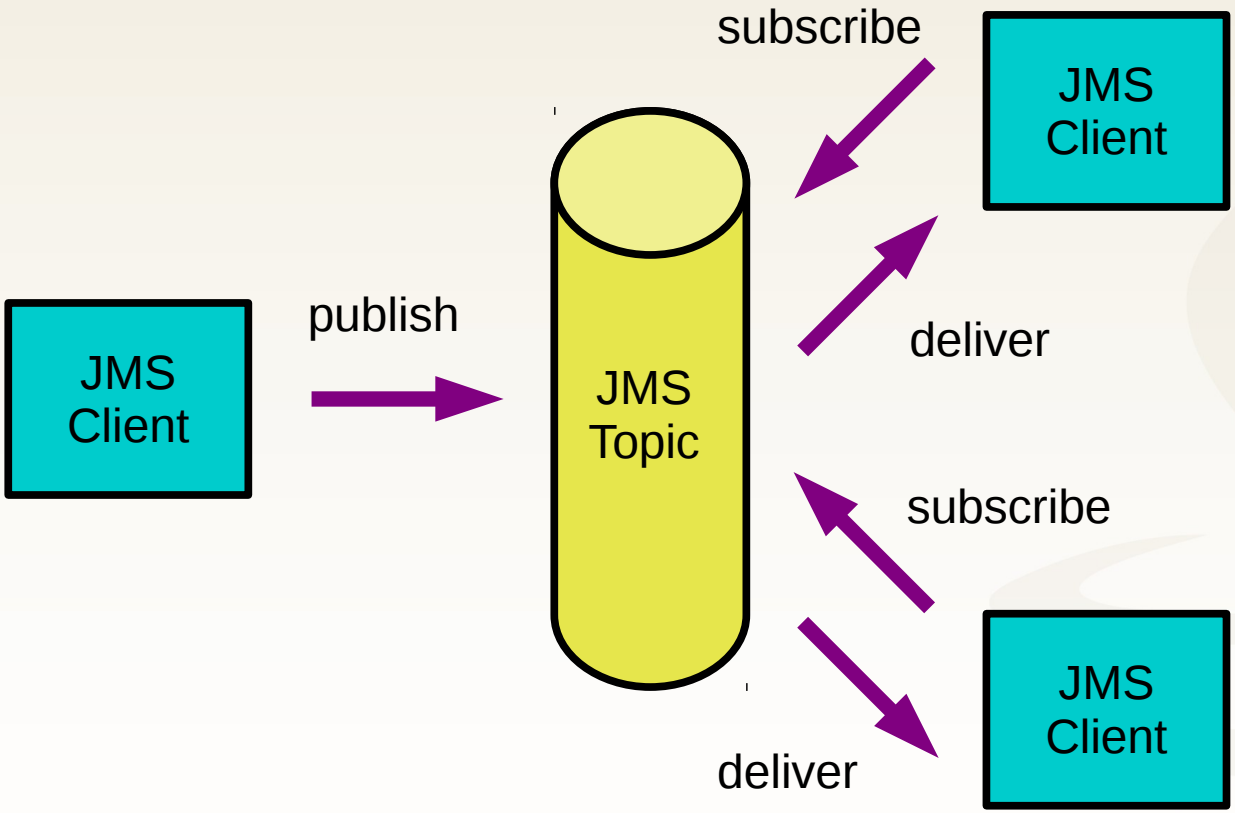
connect

JMS Client

Point-To-Point Mesajlaşma



Publish-Subscribe Tabanlı Mesajlaşma

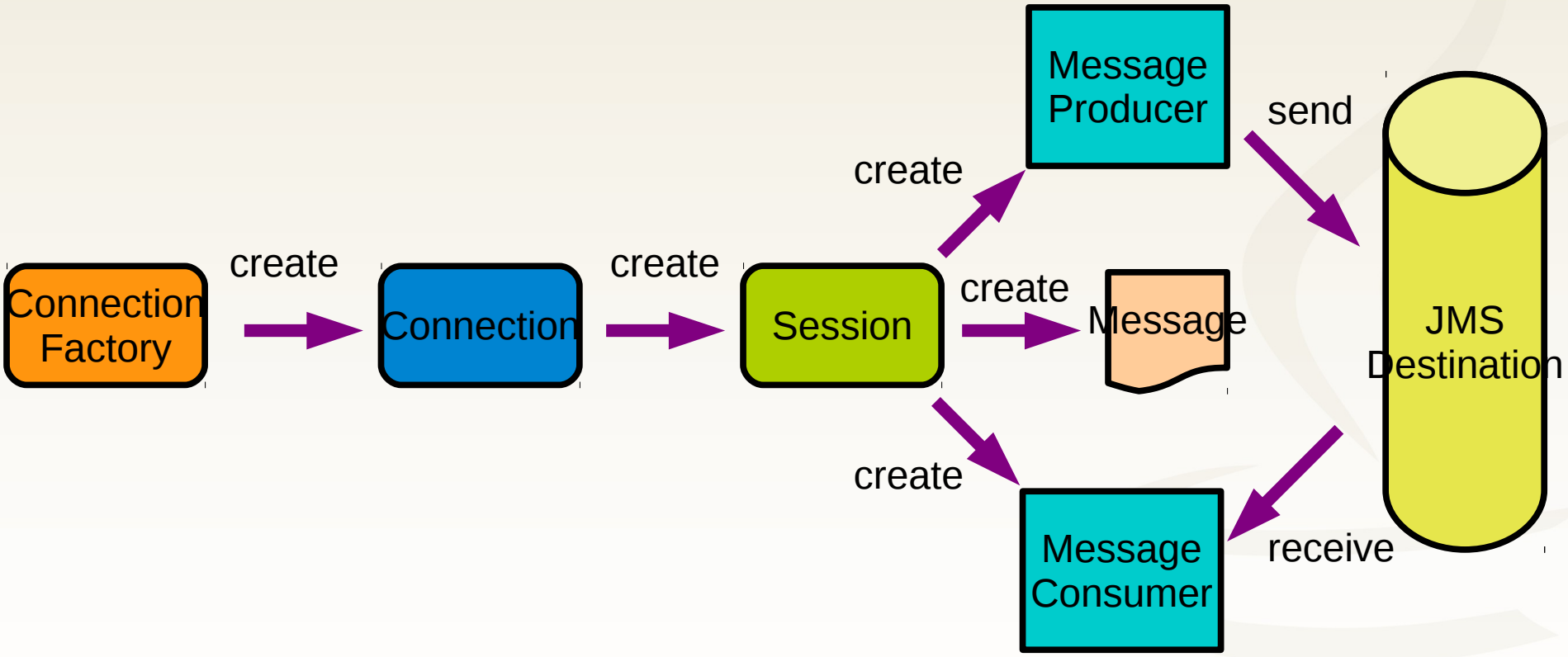


Senkron ve Asenkron Mesaj Tüketimi

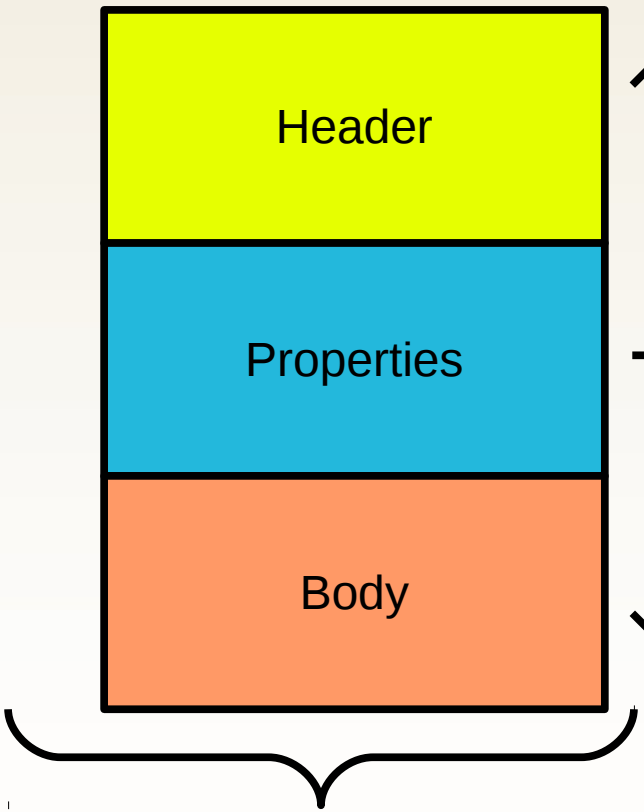
JAVA Eğitimleri

- Senkron mesajlaşmada JMS client mesajı almak için explicit bir **receive()** metot çağrısı yapar
- Bu metot, **mesaj dönen kadar** veya **timeout süresince** süreci **blok**lar
- Asenkron mesajlaşmada ise JMS client **MessageListener** arayüzünü implement eder
- Her ne zaman yeni mesaj gelirse, JMS provider listener'ın **onMessage()** metodunu çağırarak mesajı iletir

JMS Programlama Modeli



JMS Mesaj Yapısı



Mesajı tanımlamak ve yönlendirmek
İçin kullanılırlar, predefined değerlerdir

Çoğunlukla send veya publish metotları
Tarafından otomatik set edilirler, bir kısmı
da client tarafından set edilebilir

Client tarafından oluşturulabilen custom
Değerlerdir, header değerleri yetersiz kalır
Ise kullanılabilir

Predefined değerler de vardır

Kullanımı opsiyoneldir

Değişik mesaj body tipleri vardır, bunlar
Mesaj tipi olarak da bilinir

Session üzerinden elde edilirler

Body kısmı da opsiyoneldir

Message
MapMessage
StreamMessage
TextMessage
BytesMessage
ObjectMessage

JMS Örneği – Topic Server Konfigürasyonu (JBoss)

```
<mbean xmbean-dd="xmdesc/Topic-xmbean.xml"
name="jboss.messaging.destination:service=Topic,name=hellot"
code="org.jboss.jms.server.destination.TopicService">
    <attribute name="JNDIName">topic/hello</attribute>
    <depends optional-attribute-
    name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
    <depends>jboss.messaging:service=PostOffice</depends>
</mbean>
```

Server tarafında ConnectionFactory ve JMS destination (topic veya queue) tanımlamaları yapılması, bunların JNDI'a bind edilmeleri gerekir

Bu işlemler her server'a göre değişir

JBoss hazır bir ConnectionFactory konfigürasyonu ile gelmektedir

Yukarıdaki xml bloğunu JBOSS_HOME/server/default/deploy/messaging/destinations-service.xml dosyası içerisine kopyalayarak topic tanımı yapılabilir

JMS Örneği – Topic Client Initialization

```
InitialContext context = new InitialContext();

TopicConnectionFactory cf = (TopicConnectionFactory)
context.lookup("ConnectionFactory");

Topic topic = (Topic) context.lookup("topic/hello");

TopicConnection c = cf.createTopicConnection();
c.start();

TopicSession session = c.createTopicSession(false,
TopicSession.AUTO_ACKNOWLEDGE);
```

Bu aşamaya kadar ki kısım hem **publisher**, hem de **subscriber** tarafında **aynı**dır

JMS Örneği – Topic Publisher

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


TextMessage message = session.createTextMessage("hello world");
publisher.send(message);


publisher.close();
```

JMS Örneği – Topic Subscriber

```
public class MessageSubscriber implements MessageListener {

    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage)message;
        try {
            System.out.println("Message received :" +
textMessage.getText());
        } catch (JMSException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
TopicSubscriber subscriber = session.createSubscriber(topic);

subscriber.setMessageListener(new MessageSubscriber());

c.start();
```

JMS Örneği – Queue Server Konfigürasyonu (JBoss)

```
<mbean code="org.jboss.jms.server.destination.QueueService"
name="jboss.messaging.destination:service=Queue,name=helloq"
xmbean-dd="xmdesc/Queue-xmbean.xml">
    <attribute name="JNDIName">queue/hello</attribute>
    <depends optional-attribute-
    name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
    <depends>jboss.messaging:service=PostOffice</depends>
</mbean>
```

Server tarafında ConnectionFactory ve JMS destination (topic veya queue) tanımlamaları yapılması, bunların JNDI'a bind edilmeleri gerekir

Bu işlemler her server'a göre değişir

JBoss hazır bir ConnectionFactory konfigürasyonu ile gelmektedir

Yukarıdaki xml bloğunu JBOSS_HOME/server/default/deploy/messaging/destinations-service.xml dosyası içerisne kopyalayarak queue tanımı yapılabilir

JMS Örneği – Queue Client Initialization

```
InitialContext context = new InitialContext();

QueueConnectionFactory cf = (QueueConnectionFactory)
context.lookup("ConnectionFactory");

Queue queue = (Queue) context.lookup("queue/hello");


QueueConnection c = cf.createQueueConnection();
c.start();

QueueSession session = c.createQueueSession(false,
QueueSession.AUTO_ACKNOWLEDGE);
```

Bu aşamaya kadar ki kısım hem **sender**, hem de **receiver** tarafında **aynı**dır

JMS Örneği – Queue Sender

```
QueueSender sender = session.createSender(queue);

TextMessage message = session.createTextMessage("hello world");
sender.send(message);

sender.close();

c.close();
```

JMS Örneği – Queue Receiver

```
QueueReceiver receiver = session.createReceiver(queue);

c.start();

TextMessage message = (TextMessage) receiver.receive();

System.out.println("Message received :" + message.getText());
```

JMS İle Çalışırken Dikkat Edilmesi Gereken Noktalar

- En güvenilir mesaj üretim yöntemi **PERSISTENT** ve **TRANSACTION** içerisinde mesaj üretmektir
- **Persistent** mesaj, **JMS provider fail** etse bile **korunur**, **Non-persistent** mesaj JMS provider fail ettiğinde **kaybolabilir**
- Transaction, send ve receive işlemlerini kapsar
- En güvenilir mesaj tüketimi **non-temporary queue** veya **durable subscription** kullanmaktır
- Message **ACK**, mesajın başarılı biçimde alındığını belirtir

JMS İle Çalışırken Dikkat Edilmesi Gereken Noktalar

- **TX** söz konusu ise **ACK otomatik** olarak gerçekleşir
- **UN-ACK** mesajlar **Point-To-Point** yönteminde **Session kapatıldığında saklanır**, Session açıldığında tekrar iletilmeye çalışılır
- Aynı durum **Pub-Sub** yönteminde **durable subscription** için de geçerlidir
- **Non-durable subscription**'da ise Session kapatıldığında iletilmeyen **mesajlar kaybolur**

JMS İle Çalışırken Dikkat Edilmesi Gereken Noktalar

- Mesajlara **priority**'de verilebilir
- Bu sayede **acil mesajlar** JMS provider tarafından önce teslim edilebilir
- Mesajların belirli bir süre içerisinde tüketilmezlerse **zaman aşımı**na uğramaları sağlanabilir

İletişim

www.harezmi.com.tr

www.java-egitimleri.com

info@harezmi.com.tr

info@java-egitimleri.com

@HarezmiBilisim

@JavaEgitimleri