

JMS (Java Messaging Service)

MOM (message-oriented middleware)

Message-Oriented Middleware (МОМ)

► Аналоги

- E-mail, Viber, Twitter, Facebook
- Почта - провайдеры USPS, FedEx, UPS, and DHL.

Для работы с сообщениями используется вспомогательное программное обеспечение, обычно входящее в поставку сервера приложений

- Приложения могут посыпать асинхронно «сообщения» друг другу используя МОМ (сервис доставки - транспорт сообщений)
- МОМ → реальные сервера, которые являются JMS провайдер

Java Messaging Service (JMS)

- ▶ API с помощью которого Java программа может отправлять сообщения(объекты!!)

МОМ провайдер или поставщик, брокер

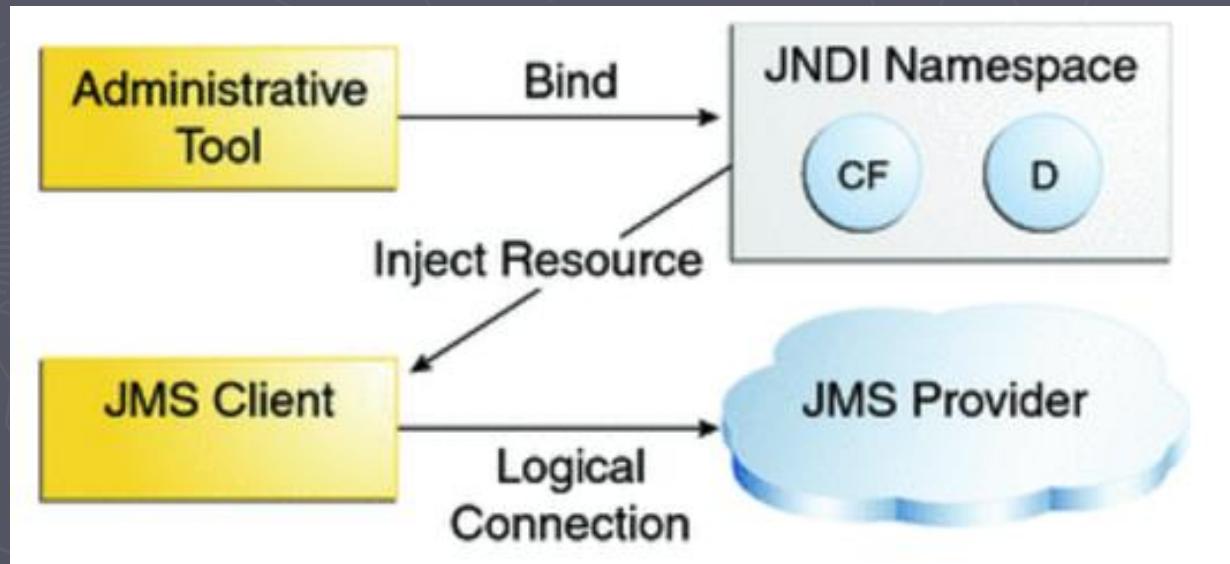


Входит в Java EE спецификацию

Можно использовать в Java SE через jar с JMS классами

Архитектура JMS

- ▶ A **JMS provider** - система которая обеспечивает JMS интерфейс и административный контроль (в Java EE)
- ▶ **JMS clients** - Java программы или компоненты которые производят и потребляют сообщения.
- ▶ **Messages**
- ▶ **Administered objects** - пред сконфигурированные JMS объекты



Администрируемые объекты

- ▶ объекты, которые конфигурируются административно, а не программно
- ▶ Нужно настроить эти объекты и сделать их доступными в пространстве имен JNDI
- ▶ Типы объектов:
- ▶ **фабрики соединений** — используются клиентами для создания подключения к пункту назначения;
- ▶ **места назначения** — точки распространения сообщений, которые получают, хранят и распространяют сообщения; (P2P) (pub-sub).

Способы получения доступа к объектам

- ▶ консоль администрирования
- ▶ командная строка (asadmin)
- ▶ интерфейс REST
- ▶ определить программно (аннотации
@JMSSConnectionFactoryDefinition и
@JMSDestinationDefinition)

Концепция сообщений

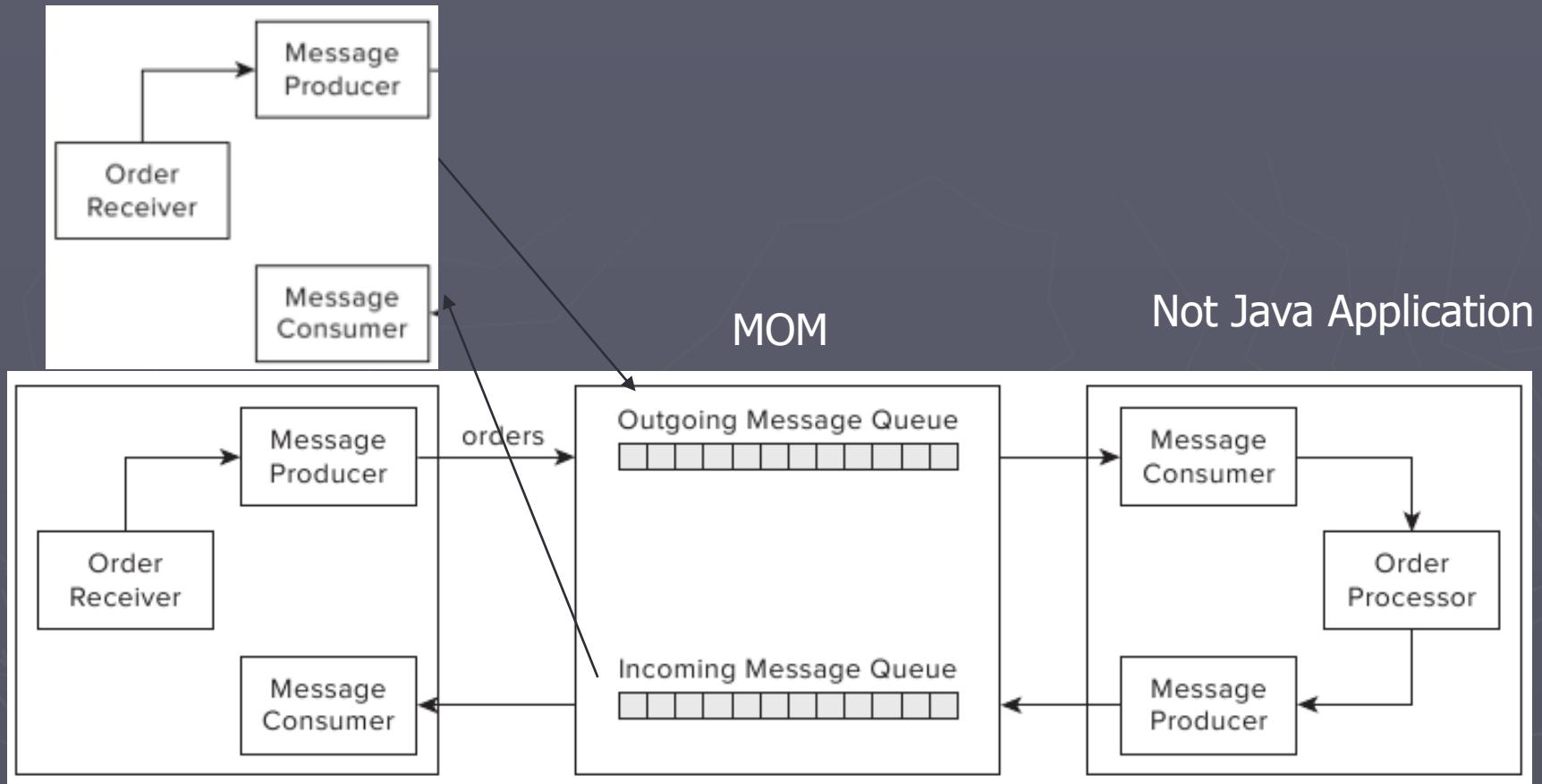
► Методы обмена в распределенных Java приложениях:

- Сокеты
- RMI
- Http-взаимодействие

} Запрос/ответ или
RPC
remote procedure calls

JMS - позволяет выполнять асинхронную связь. При этом МОМ провайдеры могут быть любые:

- WebSphere MQ (IBM)
- EMS (Tibco Software)
- SonicMQ (Progress Software)
- ActiveMQ (open source, Apache)
- Open MQ (open source, Oracle)

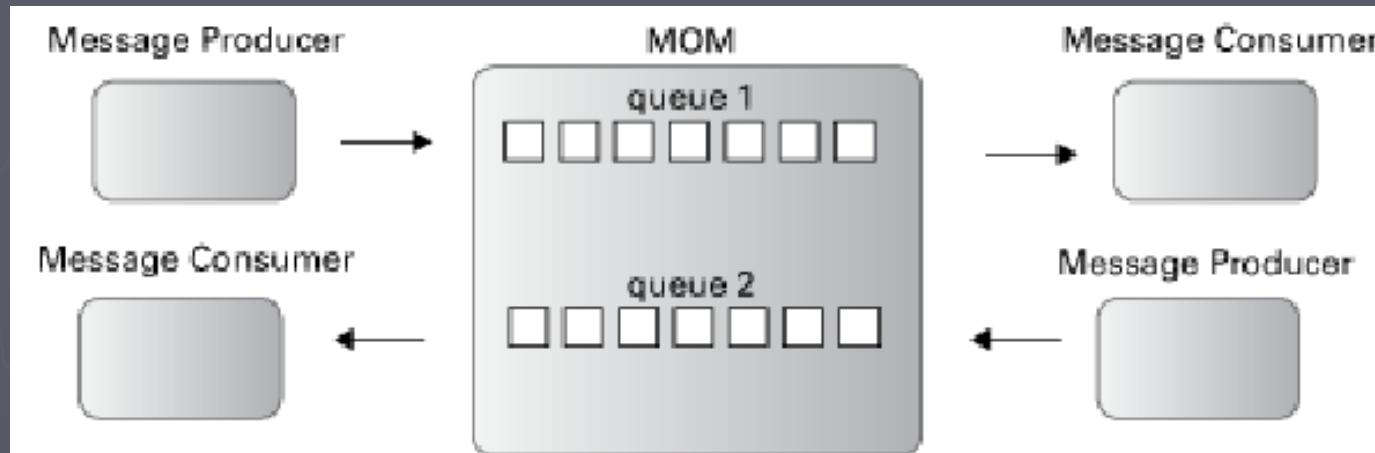


Сообщения могут накапливаться МОМ в случае если принимающая сторона не может обработать
Хранятся в DBMS или файловой системе
Существует не гарантированный режим - принимающее приложение должно быть запущено

Режимы доставки сообщений

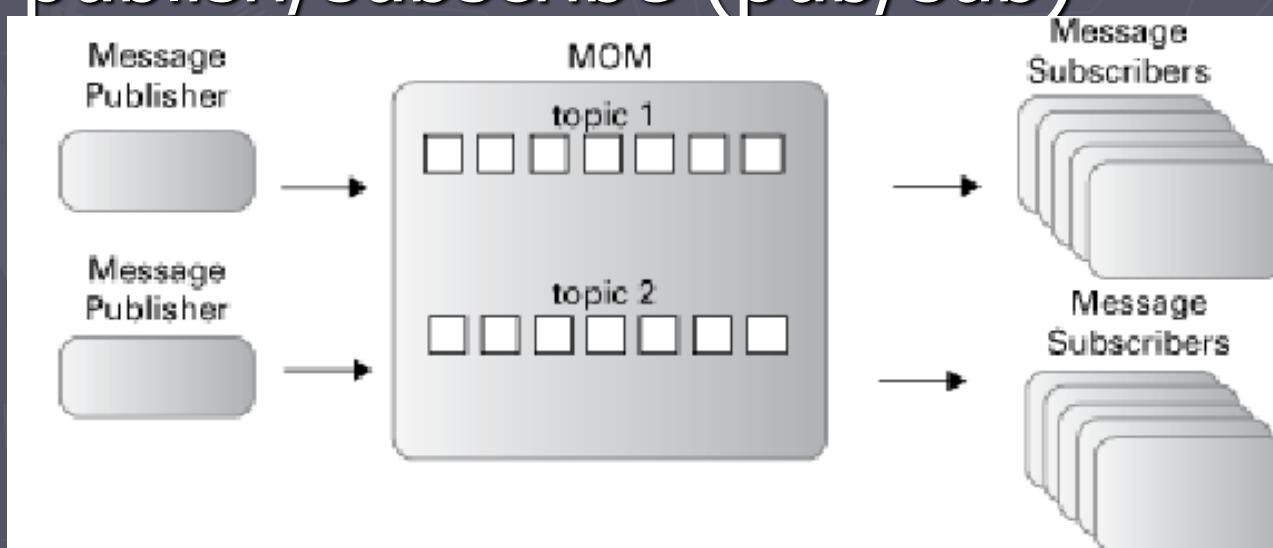
По типу destination могут быть

► point-to-point (P2P)



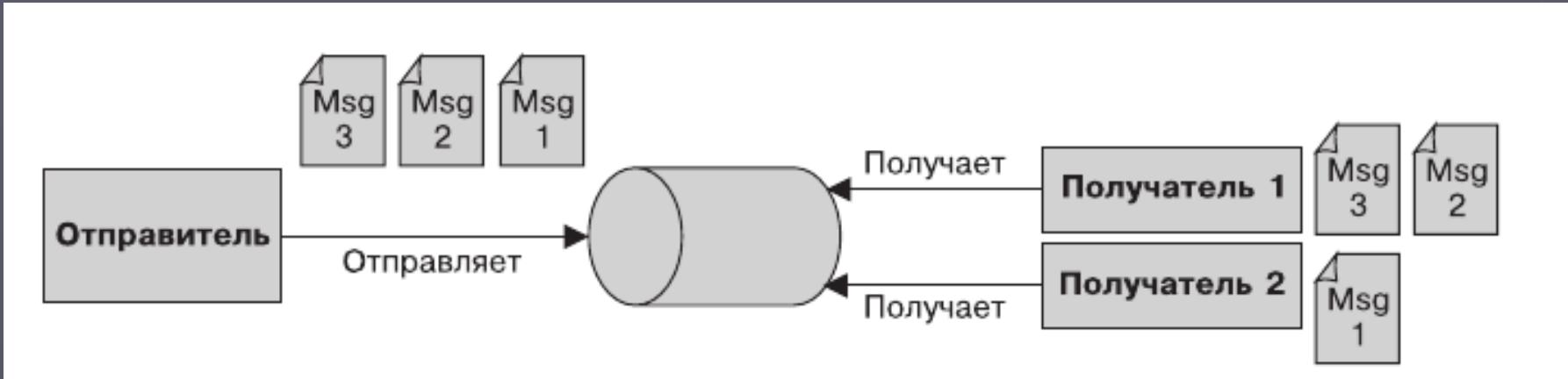
сообщение, которое отправил producer, получает единственный consumer

► publish/subscribe (pub/sub)



одно сообщение может быть прочитано неограниченным количеством consumer, подписанных на этот destination

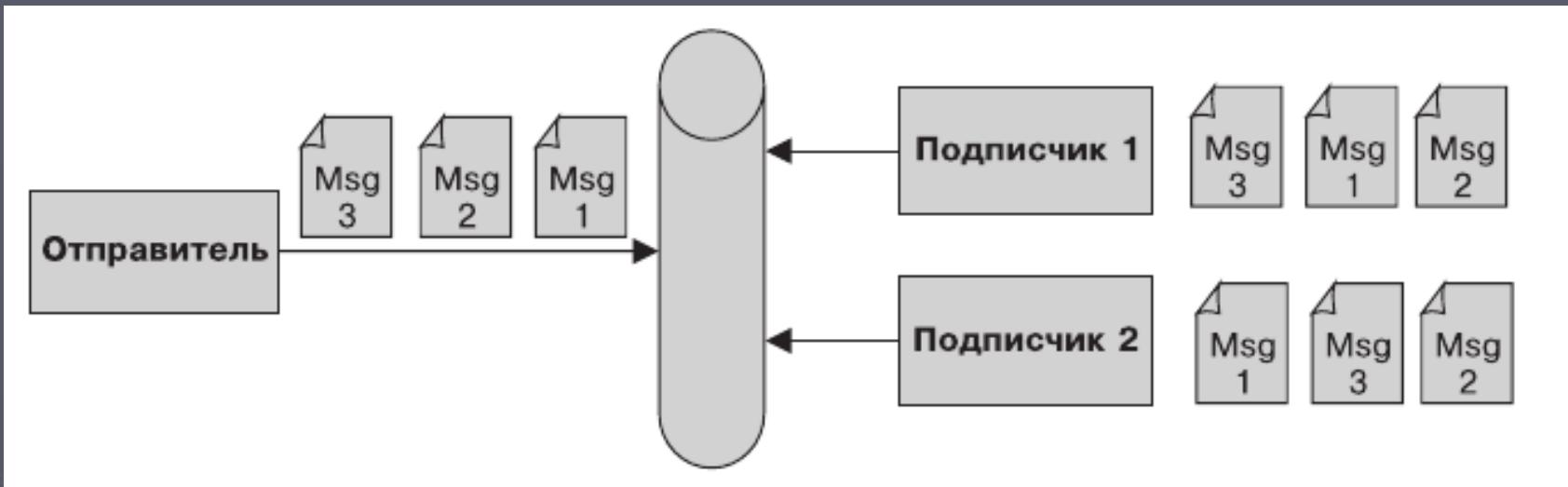
► point-to-point (P2P)



как только получатель потребляет сообщение из очереди, оно оттуда извлекается и никакой другой потребитель не может получить его

не гарантирует доставку сообщений в определенном порядке (нет временной зависимости между отправителем и получателем)

► publish/subscribe (pub/sub)

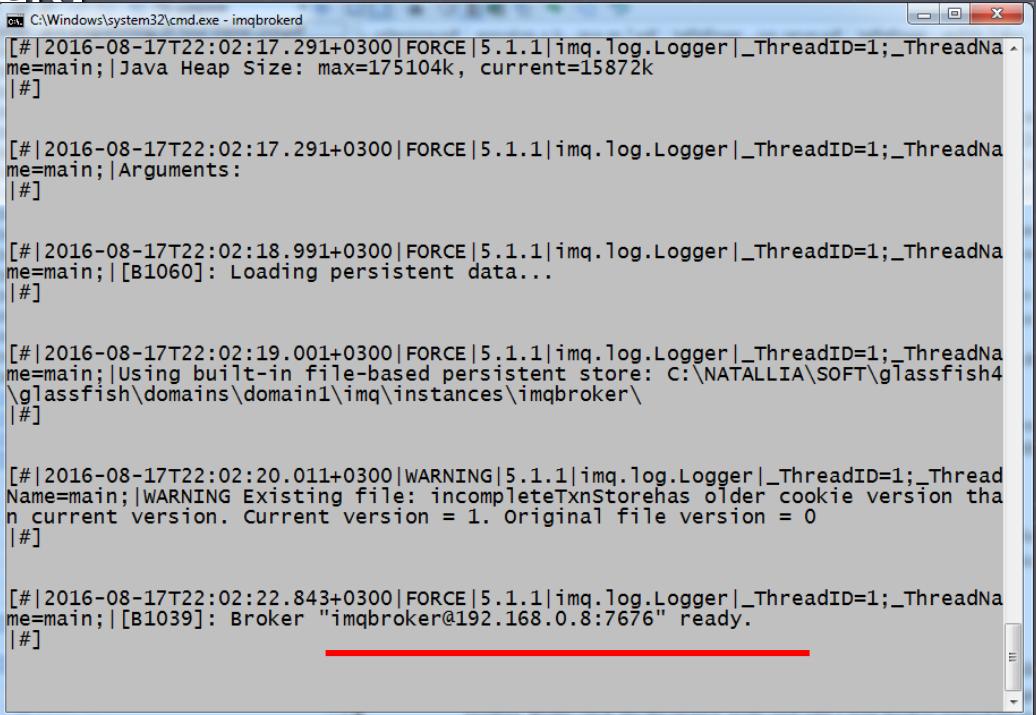


- ПОДПИСЧИКИ не получают сообщения, отправленные до того, как они подписались на тему
- если подписчик неактивен в течение определенного периода времени, он не получит прошлые сообщения

Open MQ

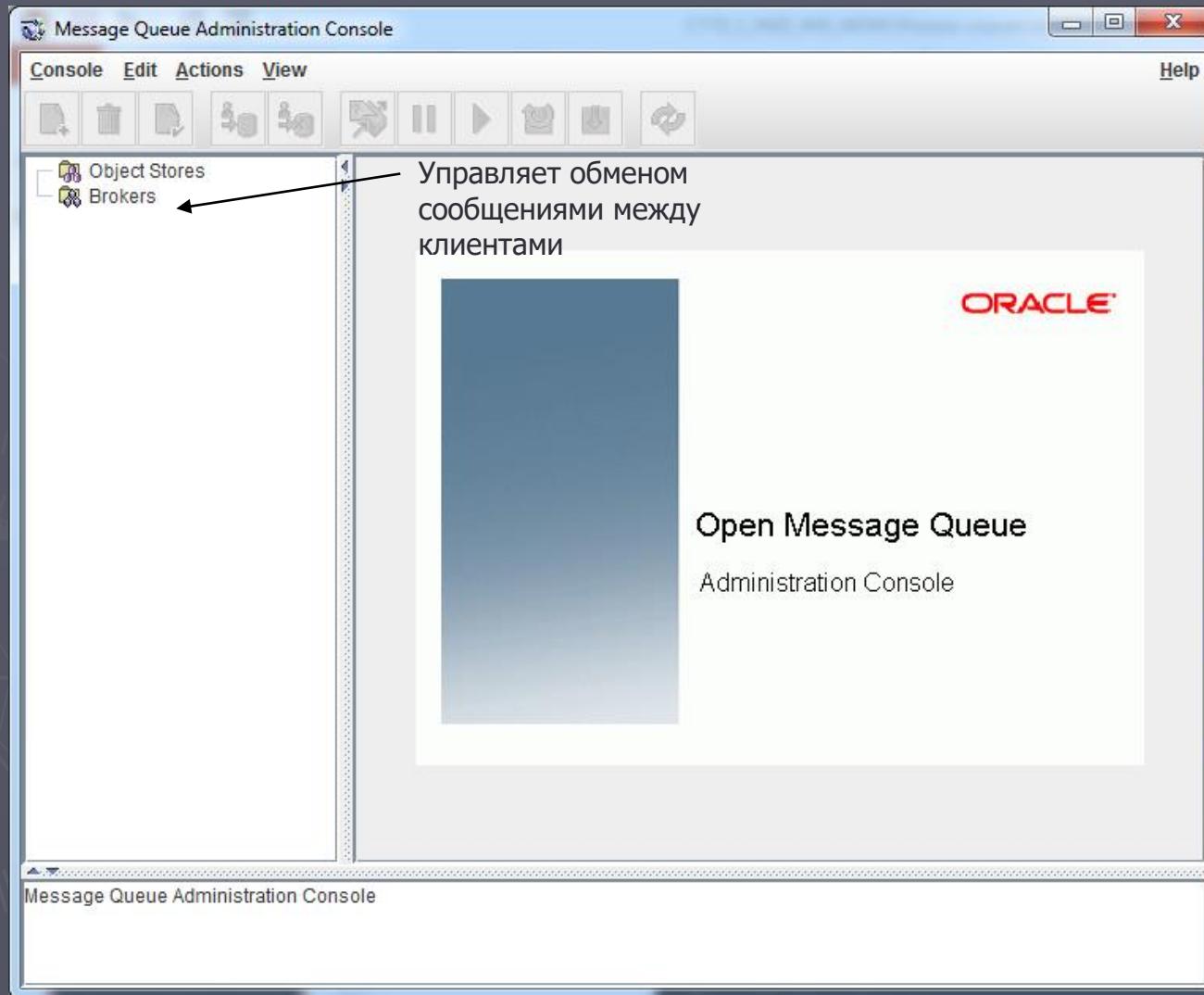
эталонная реализация JMS

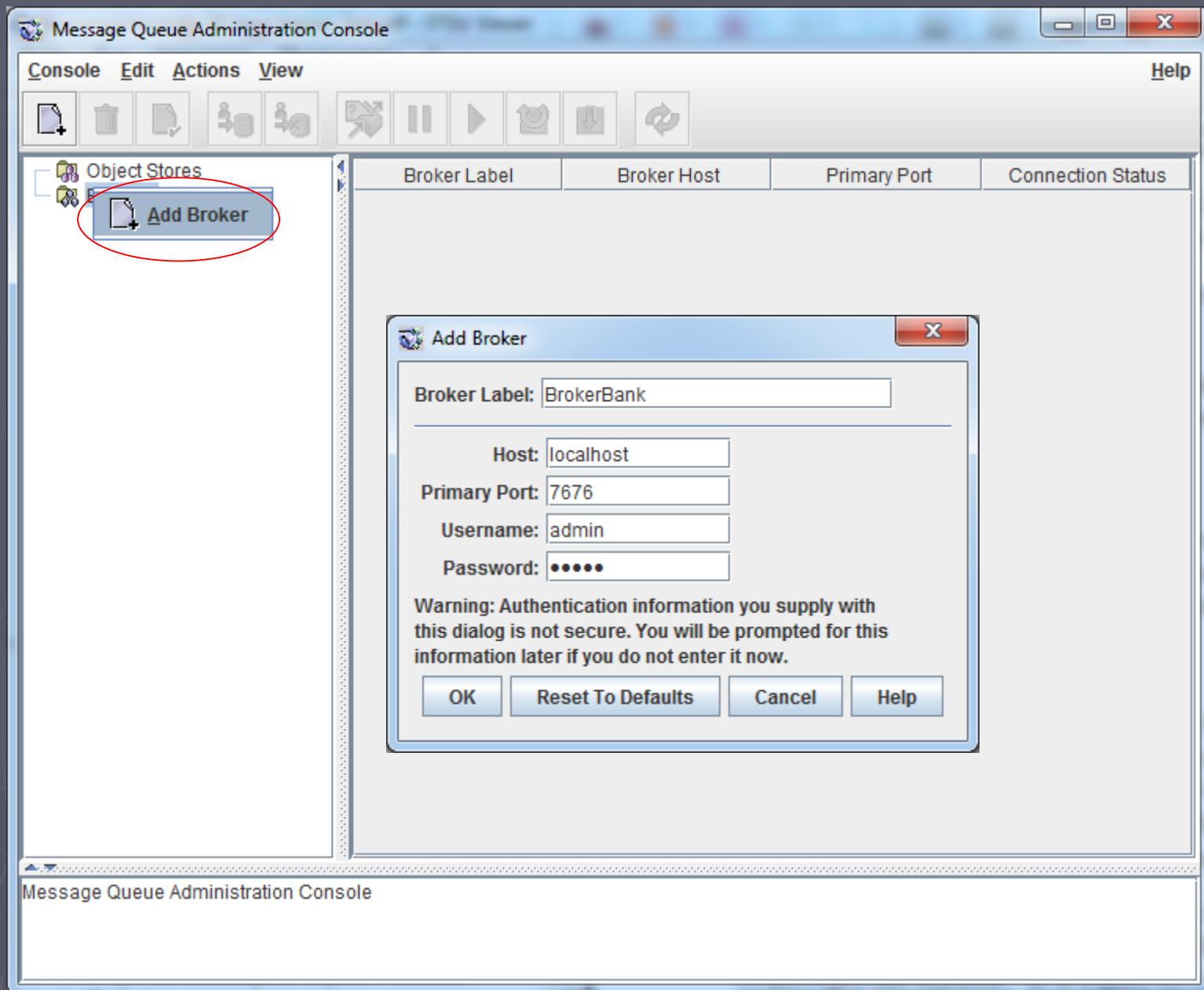
- ▶ Надо запустить МОМ:
- ▶ 1) скачать - вам не надо
<https://mq.java.net/downloads/index.html>
- ▶ 2) найти glassfish4/mq - расположение
- ▶ OpenMQ - поставщик сообщений по умолчанию для GlassFish. Запуск:
- ▶ glassfish4/mq/
/bin/ imqbrokerd.exe

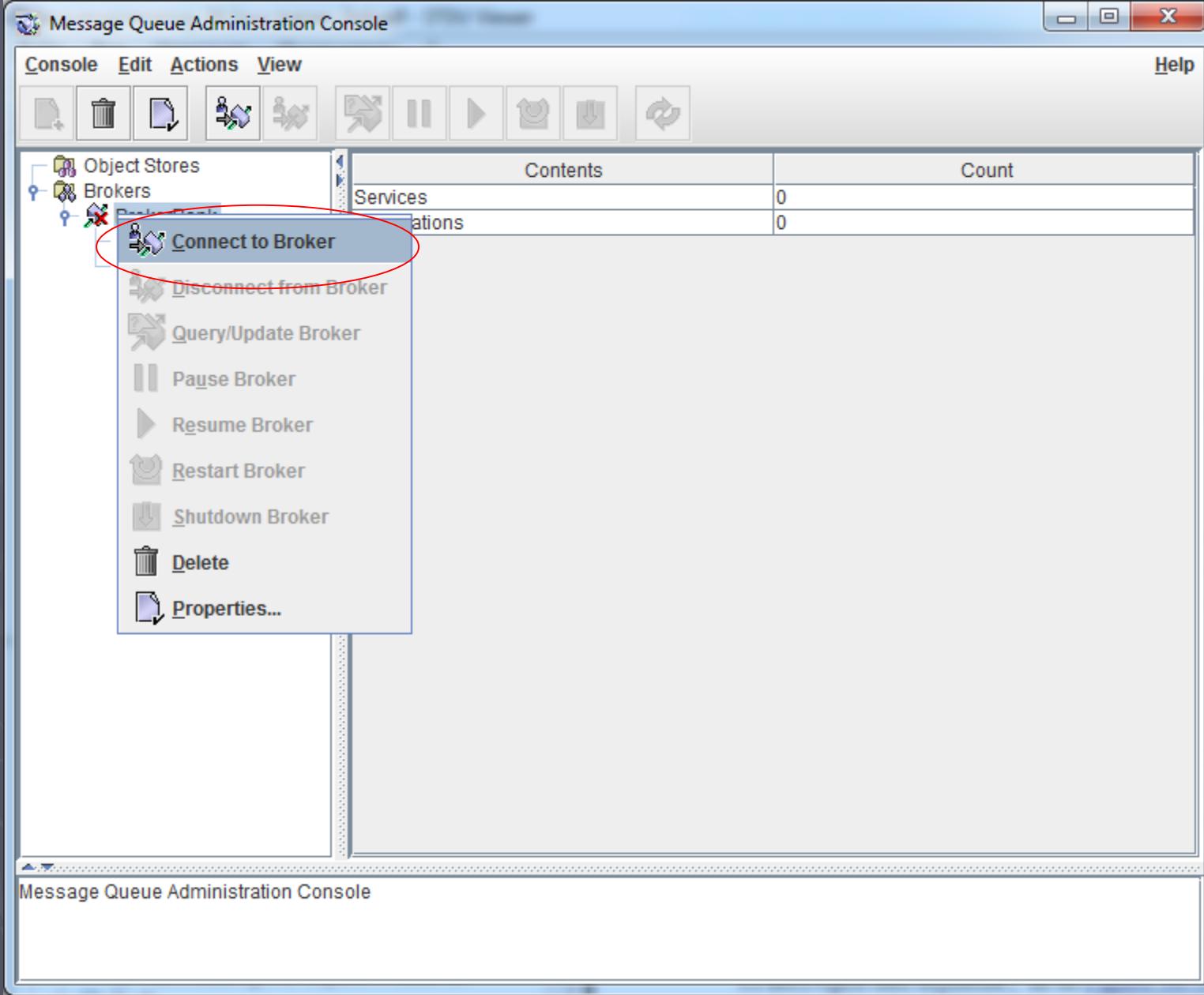


```
C:\Windows\system32\cmd.exe - imqbrokerd
[|#] 2016-08-17T22:02:17.291+0300|FORCE|5.1.1|imq.log.Logger|_ThreadID=1;_ThreadName=main; |Java Heap Size: max=175104k, current=15872k
|#
[|#] 2016-08-17T22:02:17.291+0300|FORCE|5.1.1|imq.log.Logger|_ThreadID=1;_ThreadName=main; |Arguments:
|#
[|#] 2016-08-17T22:02:18.991+0300|FORCE|5.1.1|imq.log.Logger|_ThreadID=1;_ThreadName=main; |[B1060]: Loading persistent data...
|#
[|#] 2016-08-17T22:02:19.001+0300|FORCE|5.1.1|imq.log.Logger|_ThreadID=1;_ThreadName=main; |Using built-in file-based persistent store: C:\NATALIA\SOFT\glassfish4\glassfish\domains\domain1\imq\instances\imqbroker\
|#
[|#] 2016-08-17T22:02:20.011+0300|WARNING|5.1.1|imq.log.Logger|_ThreadID=1;_ThreadName=main; |WARNING Existing file: incompleteTxnStore has older cookie version than current version. Current version = 1. Original file version = 0
|#
[|#] 2016-08-17T22:02:22.843+0300|FORCE|5.1.1|imq.log.Logger|_ThreadID=1;_ThreadName=main; |[B1039]: Broker "imqbroker@192.168.0.8:7676" ready.
```

- ▶ 2) запускаем графическую консоль админа
- ▶ glassfish4/mq/bin/ imqadmin









Добавляем
Получателя

The 'Add Broker Destination' dialog box is open, showing the following configuration:

- Destination Name:** BankOderDestination (highlighted with a red oval)
- Destination Type:** Queue (radio button selected)
- Max Number of Messages:** Unlimited (radio button selected)
- Max Total Message Bytes:** Unlimited (radio button selected)
0 bytes
- Max Bytes per Message:** Unlimited (radio button selected)
0 bytes
- Max Number of Producers:** 100 (radio button selected)
- Max Number of Active Consumers:** 1 (radio button selected)
- Max Number of Backup Consumers:** 0 (radio button selected)

At the bottom are buttons for 'OK', 'Reset To Defaults', 'Cancel', and 'Help'.

Message Queue Administration Console
Successfully connected to the broker 'BrokerBank'.

Message Queue Administration Console

Console Edit Actions View Help

Object Stores Brokers BrokerBank Services Destinations

Destination Name Destination Type Destination State

Destination Name	Destination Type	Destination State
mq.sys.dmq	Queue	RUNNING
BankOderDestination	Queue	RUNNING

Message Queue Administration Console
Successfully connected to the broker 'BrokerBank'.
Successfully added the destination 'BankOderDestination' on broker 'BrokerBank'.

JMS API

- ▶ Классы и интерфейсы

<http://docs.oracle.com/javaee/7/api/javax/jms/package-summary.htm>

- ▶ JMS 1.1

Queue – место для сообщений

QueueConnection – интерфейс, пред. соединение к МОМ

QueueConnectionFactory, - объект, котор . создает Connection

QueueSession – объект сессия между клиентом и МОМ

QueueSender, - объект, посылающий сообщения

QueueReceiver , - получает

Topic, объект в Pub/Sub

TopicPublisher,

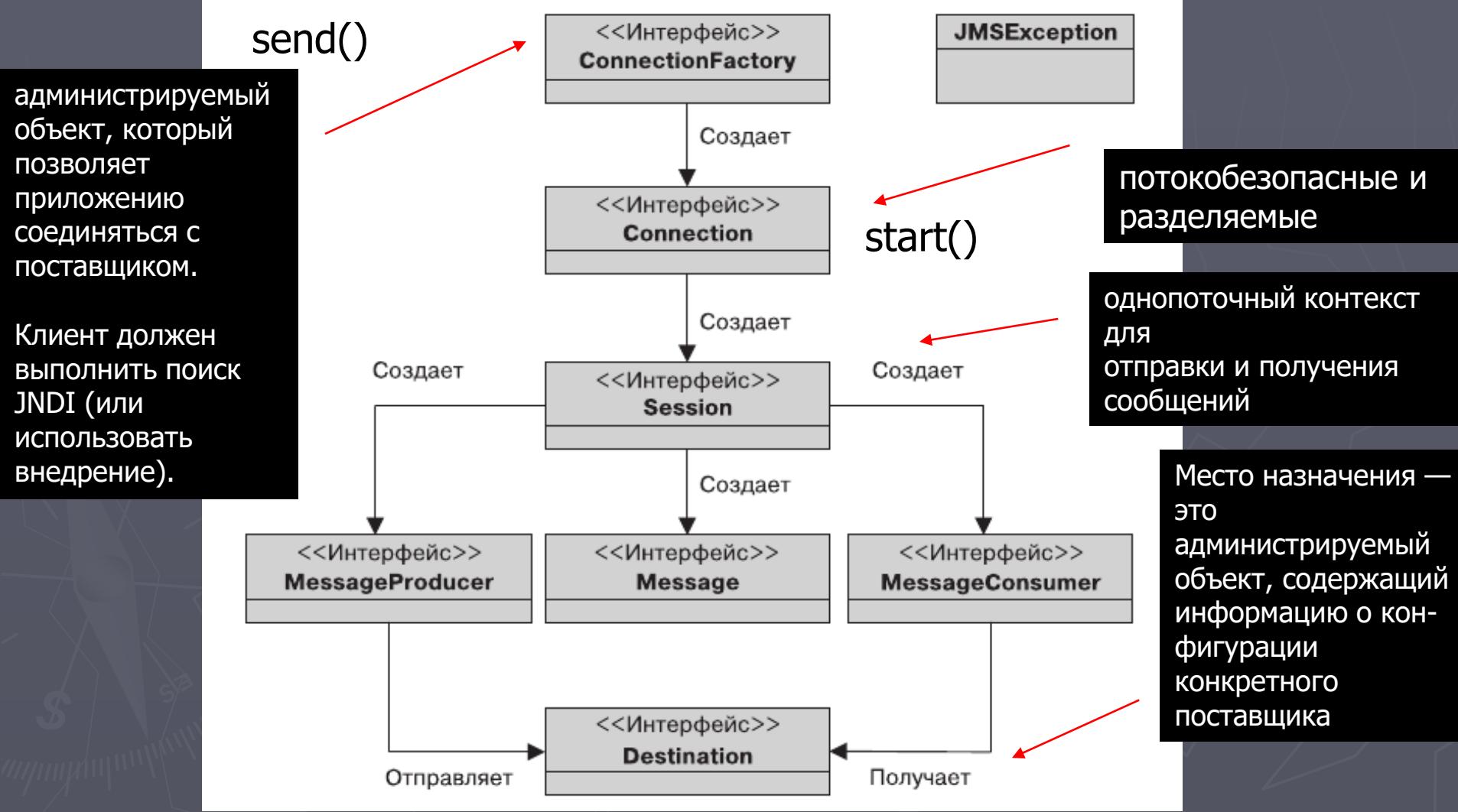
TopicSubscriber, Message – обертка сообщений

стандартный API Java, который позволяет приложениям создавать, отправлять, получать и читать сообщения асинхронно

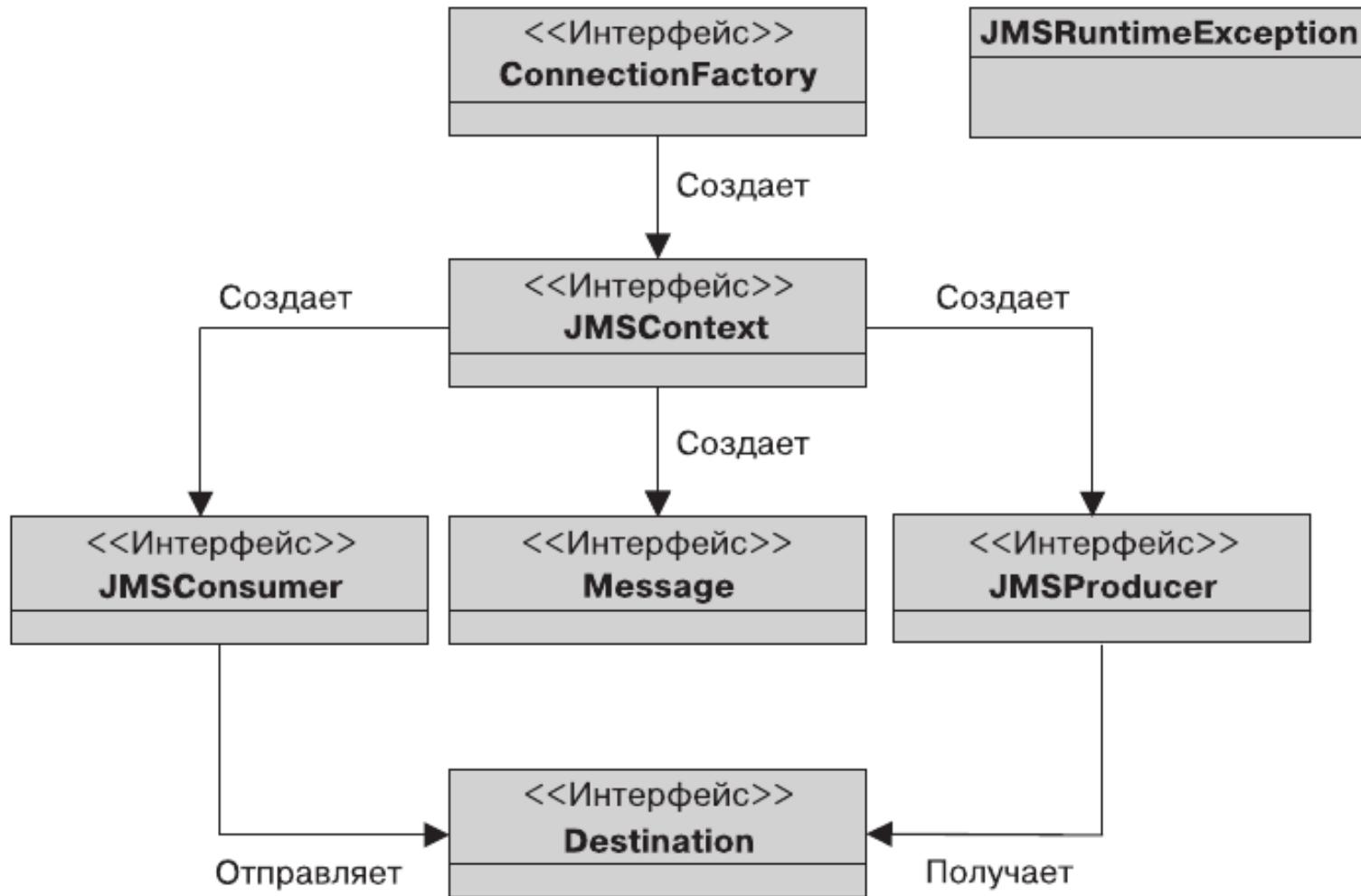
JMS 1.0, 1.1 и 2.0

Классический API	Упрощенный API	Устаревший API (P2P)	Устаревший API (pub-sub)
ConnectionFactory	ConnectionFactory	QueueConnection-Factory	TopicConnection-Factory
Connection	JMSContext	QueueConnection	TopicConnection
Session	JMSContext	QueueSession	TopicSession
Destination	Destination	Queue	Topic
Message	Message	Message	Message
MessageConsumer	JMSConsumer	QueueReceiver	TopicSubscriber
MessageProducer	JMSProducer	QueueSender	TopicPublisher
JMSEException	JMSRuntime-Exception	JMSEException	JMSEException

JMS 1.1 (шаги посыла сообщений)



JMS 2.0



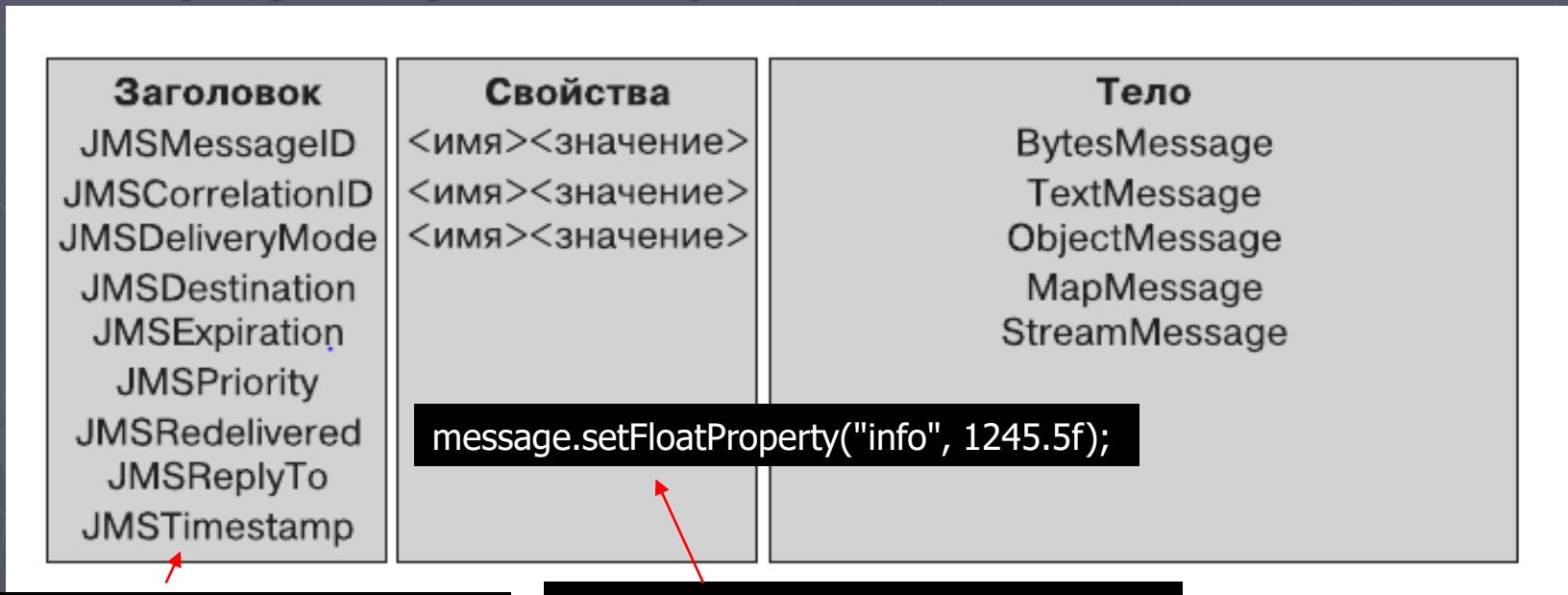
Connection , Session реализуют интерфейс java.lang.AutoCloseable
+JMSRuntimeException + getBody + методы

JMS 2.0

- ▶ *JMSPContext* - объекты JMS соединения и сессии.
Нужно соединение с МОМ, обмен идет в сессии
- ▶ *ConnectionFactory* - объект, который создает
объекты Connection инкапсулированные в
JMSPContext
- ▶ *Queue* и *Topic* - получатели: интерфейсы
наследуются от Destination.
- ▶ *JMSPProducer* интерфейс с методом для посылки
сообщения получателю
- ▶ *JMSPConsumer* интерфейс с методом для получения
сообщения
- ▶ *Message* - интерфейс для всех сообщений.
Содержит заголовок и тело

Структура сообщения JMS

- header - unique message ID, destination, type и т.д. – заполняется автоматически при send() или publish(), клиентом и поставщиком
- доп. свойства (опц.) – запол. программно
- body (опц.) – содержит сообщение для доставки



информация для
идентификации и
маршрутизации сообщений

позволяют фильтровать сообщения на
основе их значений

Типы сообщений

- ▶ **javax.jms.Message**
- ↓
- ▶ TextMessage - объект с Java String
- ▶ ObjectMessage - сериализуемые Java объекты
- ▶ BytesMessage - массив байт.
- ▶ StreamMessage - поток Java примитивов
- ▶ MapMessage - key/value пары
(string/прим. Тип)

► Извлечение сообщения Message.getBody()

```
msg.getBody(String.class); // msg – ссылка на сообщение
```

► Размещение объекта

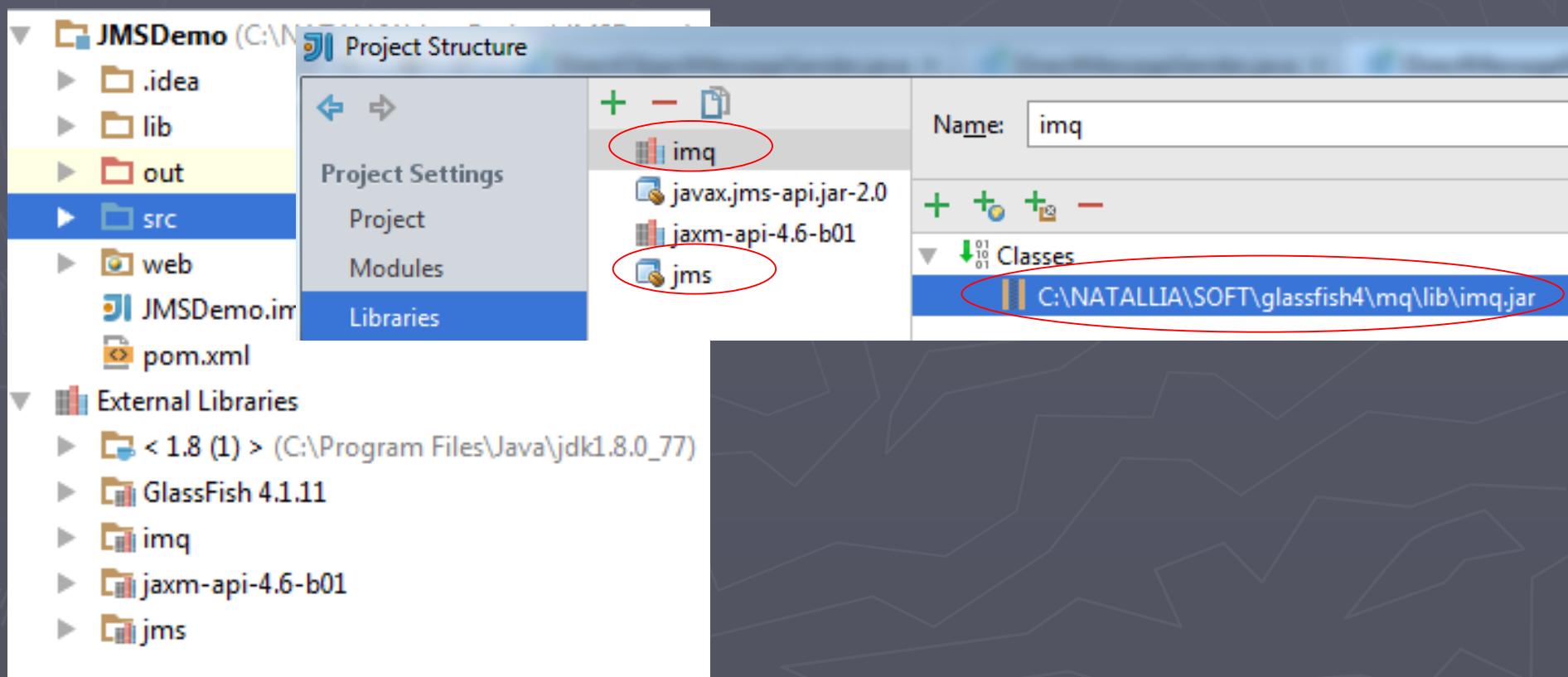
```
Card card = new Card();
ObjectMessage objMsg = context.createObjectMessage(card);
//заворачиваем в ObjectMessage

//получаем
Card receivedOrder = msg.getBody(Card.class);
```

Пример:

Послать сообщение напрямую в МОМ (без запуска Glassfish)

- ▶ queues или topics надо сконфигурировать в МОМ
- ▶ Стартуем с консоли Open MQ



Message Queue Administration Console



Console Edit Actions View

Help



- Object Stores
- Brokers
- BrokerBank
- Services
- Destinations

Destination Name	Destination Type	Destination State
BankCardQueue	Queue	RUNNING
BankCard	Queue	PAUSED
BankOderDestination	Queue	RUNNING
mq.sys.dmq	Queue	RUNNING

```

import javax.jms.*;

import com.sun.messaging.ConnectionFactory;
import com.sun.messaging.ConnectionConfiguration;

public class DirectMessageSender{
    public static void main(String[] args) {
        ConnectionFactory factory;
        factory = new ConnectionFactory();
        try( JMSContext context = factory.createContext("admin", "admin")){
            factory.setProperty(ConnectionConfiguration.imqAddressList,
                "mq://127.0.0.1:7676,mq://127.0.0.1:7676");
            Destination cardsQueue = context.createQueue("BankCardQueue");
            JMSProducer producer = context.createProducer();
            // Send msg about card
            producer.send(cardsQueue, "PNV 100 5634234");
            System.out.println("Placed an information about card transaction to BankCardQueue");
        } catch (JMSEException e){
            System.out.println("Error: " + e.getMessage());
        }
    }
}

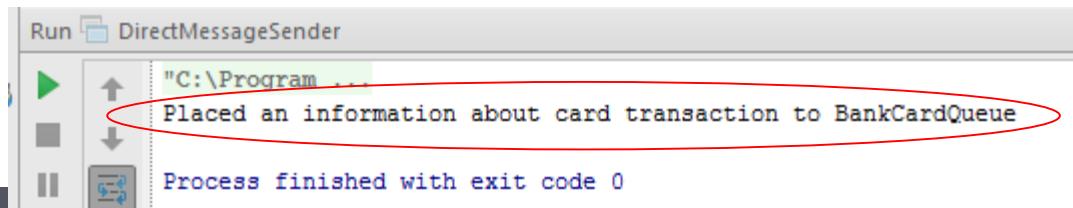
```

1. Объект использует спецификацию реализацию класса предоставленного МОМ

2. Создать JMSContext

3. Создать адресата - Destination

4. Послать сообщение в очередь



Получить сообщение напрямую из МОМ (без запуска Glassfish)

Listener – программа потребитель

Получение сообщений 1) синхронно
receive() - метод

2) асинхронно

Реализация интерфейса MessageListener
и вызов callback метода
onMessage() – получает сообщение как
только оно появилось в очереди

```
public class DirectMessageReceiver implements MessageListener{  
  
    ConnectionFactory factory = new com.sun.messaging.ConnectionFactory();  
    JMSConsumer consumer;  
  
    DirectMessageReceiver() {  
        try( JMSContext context = factory.createContext("admin","admin")) {  
            factory.setProperty(ConnectionConfiguration.imqAddressList,  
                "mq://127.0.0.1:7676,mq://127.0.0.1:7676");  
            Destination cardsQueue = context.createQueue("BankCardQueue");  
            consumer = context.createConsumer(cardsQueue);  
            consumer.setMessageListener(this);  
            System.out.println("Listening to theBankCardQueue...");  
  
            // wait for messages  
            Thread.sleep(100000);  
        } catch (InterruptedException e) {  
            System.out.println("Error: " + e.getMessage());  
        } catch (JMSEException e) {  
            System.out.println("Error: " + e.getMessage());  
        }  
    }  
}
```

Предотвра-
щаем
закрытие
программы

Создать JMS объекты
зарегистрировать listener сообщени

```
public void onMessage(Message msg) {  
  
    try{  
        System.out.println("Got the text message from the BankCardQueue: "  
                           msg.getBody(String.class));  
        System.out.println("\n  
                           = Here's what toString() on the message prints \n" + msg);  
  
    } catch (JMSEException e){  
        System.err.println("JMSEException: " + e.toString());  
    }  
  
}  
  
public static void main(String[] args){  
    new DirectMessageReceiver();  
}
```

код по обработке полученного сообщения

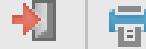
Получаем тело сообщения

Информация о сообщении

Run:

DirectMessageReceiver

DirectMessageSender



```
"C:\Program ...  
Listening to theBankCardQueue...  
Got the text message from the BankCardQueue: PNV 100 5634234  
==== Here's what toString() on the message prints  
  
Text: PNV 100 5634234  
Class: com.sun.messaging.jmq.jmsclient.TextMessageImpl  
getJMSMessageID(): ID:7-192.168.1.10(ae:4b:7:93:47:c)-64172-1471544086636  
getJMSTimestamp(): 1471544086636  
getJMSCorrelationID(): null  
JMSReplyTo: null  
JMSDestination: BankCardQueue  
getJMSDeliveryMode(): PERSISTENT  
getJMSRedelivered(): false  
getJMSType(): null  
getJMSExpiration(): 0  
getJMSDeliveryTime(): 0  
getJMSPriority(): 4  
Properties: {JMSXDeliveryCount=1}
```

Работа с topic

► Создание

```
//публикация топика
Destination priceInfo = context.createTopic("PriceInfo");
// Publish a price
producer.send(priceInfo, "Ерам 100.22");
```

► Подписка

- Долговременная – гарантирует доставку, даже если подписчики не активны
- Не долговременная – получают только активные подписчики (сообщение удаляется после подтверждения)

```
//не долговременная
Destination priceInfo=
    context.createTopic("PriceInfo");
consumer = context.createConsumer(priceInfo);

//долговременная
Destination priceDTopic =
    context.createTopic("PriceDTopic");
context.setClientID("client123");
consumer = context.createDurableConsumer((Topic)priceDTopic,
    "SecurityCenter");

//отписаться от топика
context.unsubscribe("SecurityCenter");
```

Подтверждения и транзакции

- ▶ Как проверить, что сообщение доставлено?
- ▶ JMS API позволяет определить или режим подтверждения получения сообщения, или режим транзакции что позволяет контролировать удаление сообщения из очереди
- ▶

Модели подтверждения

► AUTO_ACKNOWLEDGE

- посылает подтверждение как только успешно выполниться onMessage() (по умолчанию)

► CLIENT_ACKNOWLEDGE

- требует явного подтверждения - acknowledge() из приемника

► DUP_OK_ACKNOWLEDGE

- Нарушена работа сервера. Сообщение может доставляться несколько раз

```
JMSContext contextAsk = factory.createContext("admin", "admin",  
    JMSContext.CLIENT_ACKNOWLEDGE);
```

Транзакции в сессии

```
try (JMSContext context = factory.createContext("admin", "admin",
    JMSContext. SESSION_TRANSACTED)) {

    JMSProducer producer = context.createProducer();
    Destination queue1 = context.createQueue("Queue1");
    Destination queue2 = context.createQueue("Queue2");

    producer.send(queue1, "Msg1");
    producer.send(queue2, "Msg2");

    context.commit(); // commit the JMS transaction
} catch (JMSEException e) {
    context.rollback(); // rollback the JMS transaction
    System.out.println("Error: " + e.getMessage());
}
```

Селекторы сообщений (фильтры)

► Messages consumer

```
//фильтр у receiver  
String selector = "symbol=BSTU";  
context.createConsumer(cardsQueue, selector);
```

Синтаксис выражения основан на подмножестве
условного синтаксиса выражений SQL92

потребитель указывает
критерии отбора сообщений с
использованием выражений-селекторов

Слушатели извлекают
только те сообщения
которые имеют
заданный символ

Устанавливаем
свойство

► Message producers

```
TextMessage outMsg = context.createTextMessage();  
outMsg.setText("PNV 100 2536721");
```

```
outMsg.setStringProperty("symbol", "BSTU");  
Destination ordersQueue=context.createQueue("BankCardQueue");
```

```
JMSProducer producer = context.createProducer();  
producer.send(ordersQueue, outMsg);
```

Производители устанавливают одно
или несколько значений свойств или
полей заголовка

Выражение селектора может использовать

- ▶ логические операторы (NOT , AND , OR),
- ▶ операторы сравнения (= , > , >= , < , <= , <>),
- ▶ арифметические операторы (+ , - , * , /),
- ▶ выражения ([NOT] BETWEEN , [NOT] IN , [NOT] LIKE , IS [NOT] NULL) и т. д.

Настройка параметров времени существования сообщений

- ▶ При большой нагрузке можно указать время существования сообщений, чтобы убедиться, что поставщик удалит их из места назначения по мере устаревания.
- ▶ API JMSProducer

```
context.createProducer().setTimeToLive(1000).send(queue, message);
```

- ▶ Установка поля заголовка JMSExpiration

Задание стойкости сообщения

- ▶ Стойкая доставка гарантирует, что сообщение доставляется потребителю только один раз (по умолчанию)
- ▶ Нестойкое сообщение может быть не доставлено

```
context.createProducer().setDeliveryMode(DeliveryMode.NON_PERSISTENT)  
    .send(queue, message);
```

Определение приоритетов

От 0 до 9

```
context.createProducer().setPriority(2).send(queue, message);
```

```
context.createProducer().setPriority(2)
    .setTimeToLive(1000)
    .setDeliveryMode(DeliveryMode.NON_PERSISTENT)
    .send(queue, message);
```

Механизмы надежности

- ▶ *фильтрация сообщений* — используя селекторы;
- ▶ *настройка времени жизни сообщений* — установите время окончания срока жизни сообщений так, что они не будут доставлены, если устарели;
- ▶ *определение стойкости сообщения* — укажите, что сообщения не пропадают в случае сбоя поставщика;
- ▶ *управление подтверждением* — задайте различные уровни подтверждения сообщений;
- ▶ *создание стойких подписчиков* — убедитесь, что сообщения доставлены недоступному абоненту в модели pub-sub;
- ▶ *определение приоритетов* — установите приоритет для доставки сообщений

Работа с объектами

```
*/  
import java.io.Serializable;  
  
public class Card implements Serializable{  
    public int cardID;  
    public String name;  
    public int code;  
    public float sum;  
  
    public Card(int id, String name, int quantity, float sum){  
        cardID =id;  
        this.name =name;  
        this.code=quantity;  
        this.sum=sum;  
    }  
  
    public String toString(){  
        return "Name: " + name + ", number: "+ cardID +  
            ", sum: " + sum;  
    }  
}
```

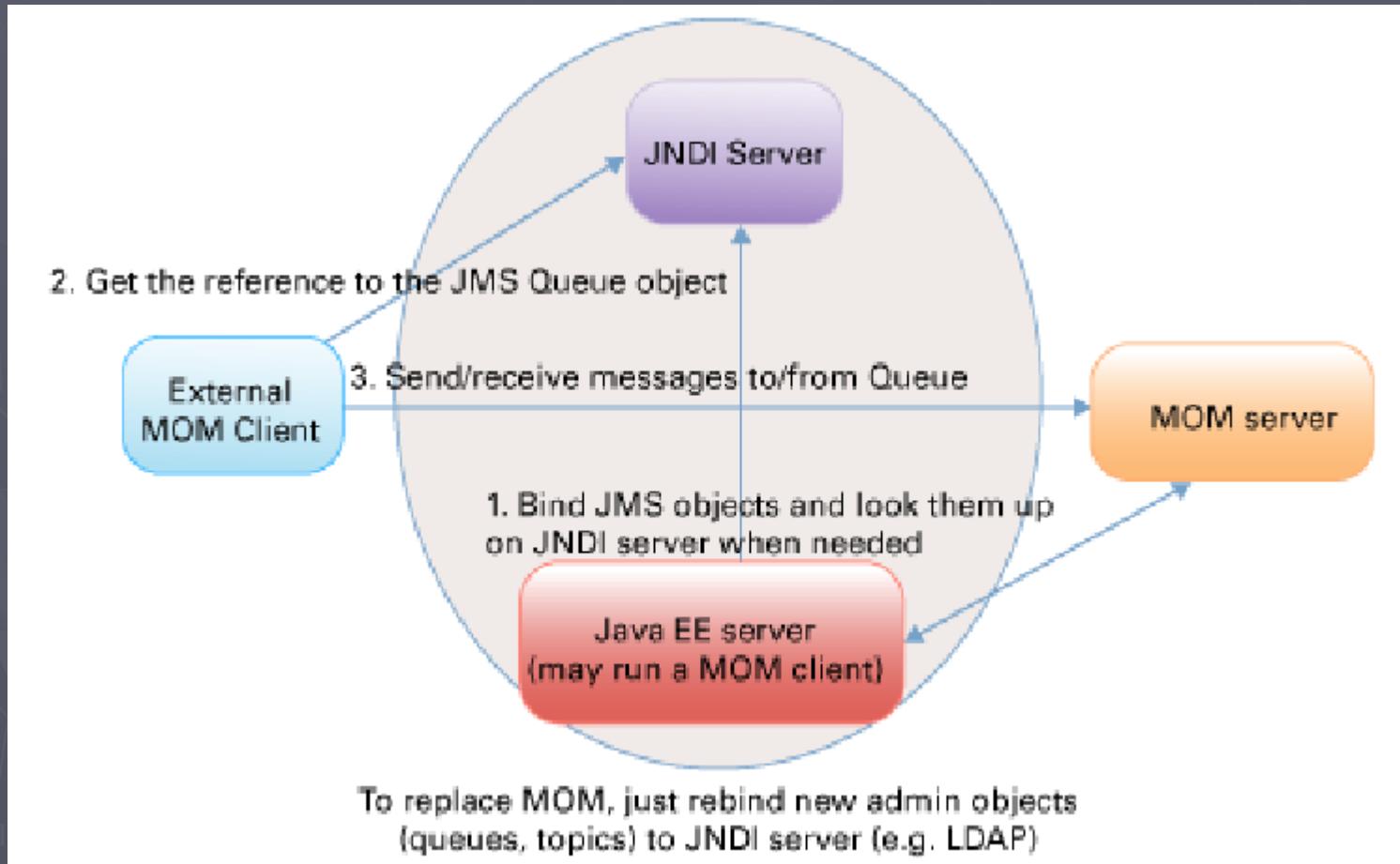
► Message producers

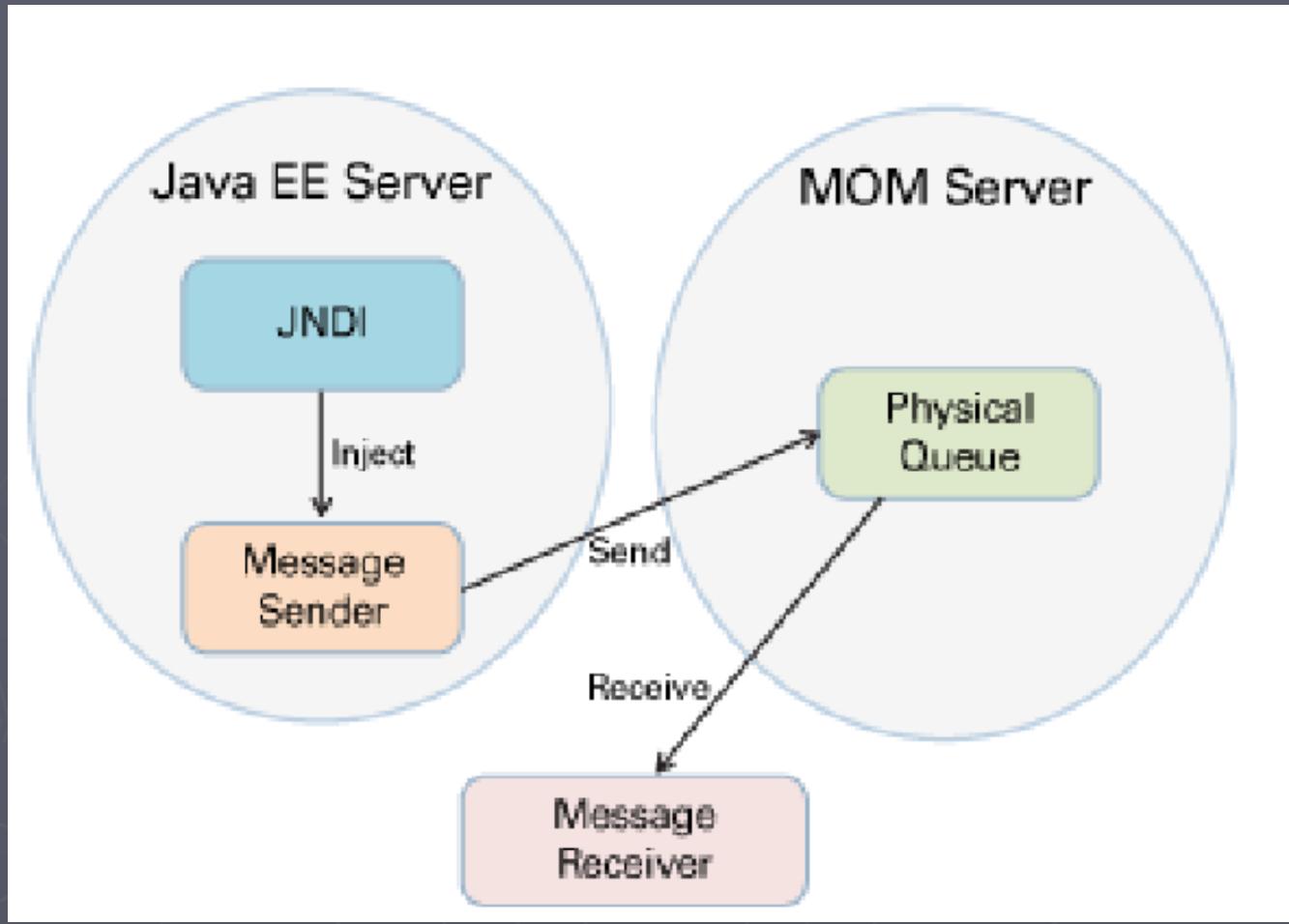
```
Card card = new Card(123456, "PNV", 1456, 100.00f);  
ObjectMessage objMsg = context.createObjectMessage(card);  
  
producer.send(ordersQueue, objMsg);
```

► Messages consumer

```
public void onMessage(Message msg) {  
  
    try {  
        System.out.println("Got the text message  
                           from the TradingCardsQueue: " +  
                           msg.getBody(Card.class));  
  
        System.out.println("\n  
                           Here's what toString() on the message prints \n" + msg);  
  
    } catch (JMSEException e) {  
        System.err.println("JMSEException: " + e.toString());  
    }  
}
```

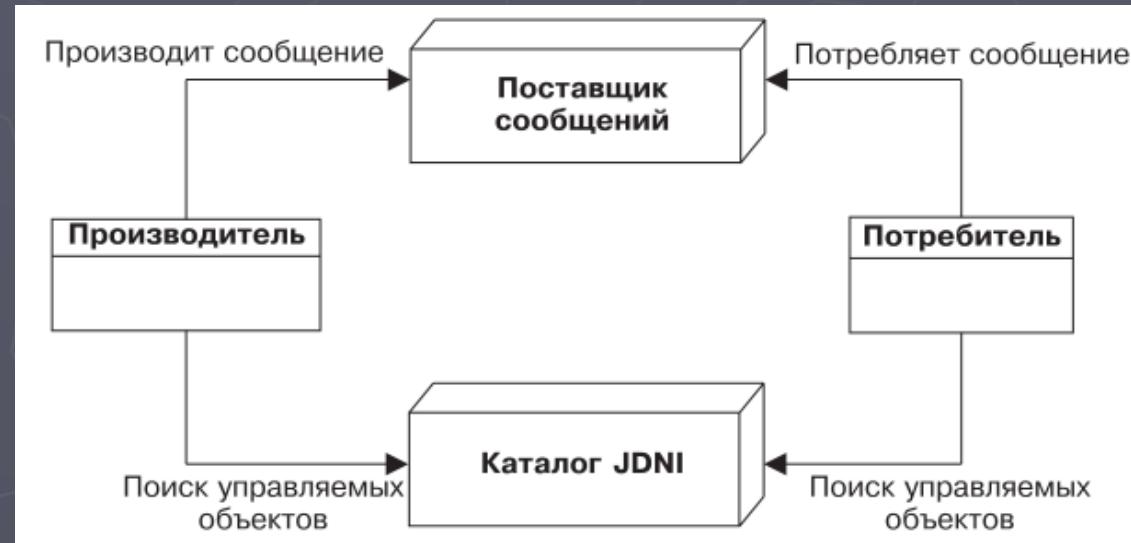
Передача сообщений (объектов) из JAVA EE контейнеров





Java Naming and Directory Interface (JNDI) javax.naming

- ▶ поддержка именования и каталогов
- ▶ API-интерфейс для доступа к службам каталогов, позволяющий клиентам осуществлять привязку и поиск объектов по имени
- ▶ Удобный поиск объектов в распределенных приложениях



- ▶ *A naming service* - позволяет добавлять, изменять или удалять имена объектов, которые существуют в определенной именной иерархии. Java classes могут их искать.
- ▶ Сервис предоставляет уникальные имена каждой записи, которая зарегистрирована (bound to) сервисом.
- ▶ Каждый сервис имеет один и более контекст. Организован в виде дерева — коревой узел - initial context.

- ▶ ***Directory service*** - позволяет искать по именованному дереву на основе атрибутов объектов (например DNS).
- ▶ Java EE сервера (при старте) - связывают объекты EJB, Servlets, JMS, пулы соединений с б.д. с внутренним или внешним ***naming service***

► Способы получения ссылки на зарегистрированный ресурс

- Внутри системы

- @Resource

```
@Resource (name="DefaultJMSConnectionFactory")
private ConnectionFactory factory;
```

- InitialContext → lookup()

- Внешняя система

- InitialContext → lookup()

```
@WebServlet("/MessageSenderServlet")
public class MessageSenderServlet extends HttpServlet {

    @Resource(lookup = "DefaultJMSSConnectionFactory") // JNDI name
    ConnectionFactory factory;

    @Resource(lookup = "BankCard") // JNDI name
    Destination cardsQueue;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{

        try( JMSContext context = factory.createContext("admin", "admin")){
            JMSProducer producer = context.createProducer();
            producer.send(cardsQueue, "PNV 100 12234");
            System.out.println("Placed a card to OutgoingCards");
        }
    }
}
```

Требует конфигурации JMS объектов
через консоль администратора на Java EE server

Производство сообщений внутри контейнера с помощью CDI

- ▶ @Inject
- ▶ @JMSSConnectionFactory

Контейнер выполняет всю работу по внедрению необходимых компонентов и управлению их жизненным циклом.

```
@Inject  
public class Producer {  
  
    @Inject  
    @JMSSConnectionFactory("DefaultJMSSConnectionFactory")  
    private JMSContext context;  
  
    @Resource(lookup = "BankCard")  
    private Queue queue;  
  
  
    public void sendMessage() {  
        context.createProducer().send(queue, "Сообщение отправлено " + new Date());  
    }  
}
```

Администрирование JMS объектами в GlassFish

- ▶ отображение имен JNDI на физические объекты МОМ
- 1) Запустить Glassfish

- Common Tasks
- Domain
 - server (Admin Server)
 - Clusters
 - Standalone Instances
 - Nodes
 - Applications
 - Lifecycle Modules
 - Monitoring Data
 - Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - JNDI
 - JavaMail Sessions
 - Resource Adapter Configs
 - Configurations
 - default-config
 - server-config
 - Update Tool

General Resources Properties Monitor Batch JMS Physical Destinations

JMS Physical Destinations

Java Message Service (JMS) physical destination objects are maintained by Message Queue brokers. The queue named mq.sys.dmq is the system undeliverable messages are redirected. Click New to create a new physical destination.

Instance Name: server

Destinations (4)

Select	Name	Type	Stat
<input type="checkbox"/>	BankCardQueue	queue	Vie
<input type="checkbox"/>	BankCard	queue	Vie
<input type="checkbox"/>	BankOderDestination	queue	Vie
<input type="checkbox"/>	mq.sys.dmq	queue	Vie

Поставщик сообщений настраивает эти объекты и делает их доступными в пространстве имен JNDI

МОМ интегрирован с сервером

JMS Resource -> Connection Factories -> New

Common Tasks

- Domain
 - server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
 - Applications
 - Lifecycle Modules
 - Monitoring Data
- Resources
 - Concurrent Resources
 - Connectors
 - JDBC
- JMS Resources
 - Connection Factories
 - Destination Resources

- JNDI
- JavaMail Sessions
- Resource Adapter Configs
- Configurations
 - default-config
 - server-config
- Update Tool

Edit JMS Connection Factory

Editing a Java Message Service (JMS) connection factory also modifies the associated connector connection pool

[Load Defaults](#)

General Settings

JNDI Name: jms/__defaultConnectionFactory

Logical JNDI Name: java:comp/DefaultJMSConnectionFactory

Resource Type: javax.jms.ConnectionFactory

Description:

Status: Enabled

Pool Settings

Initial and Minimum Pool Size: Connections

Minimum and initial number of connections maintained in the pool

Maximum Pool Size: Connections

Maximum number of connections that can be created to satisfy client requests

Pool Resize Quantity: Connections

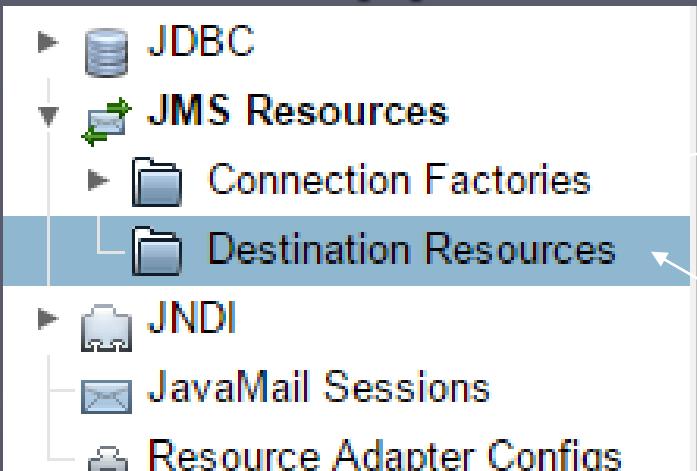
Number of connections to be removed when pool idle timeout expires

Idle Timeout: Seconds

Maximum time that connection can remain idle in the pool

Max Wait Time: Milliseconds

Создание нового JMS ресурса



JMS Resource ->
Destination
Resources ->
New

User: admin | Role: domain1 | Server: localhost
GlassFish™ Server Open Source Edition
Total # of available updates : 1

Common Tasks

- Domain
- server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
- Applications
- Lifecycle Modules
- Monitoring Data
- Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - Connection Factories
 - Destination Resources
 - JNDI
 - JavaMail Sessions
 - Resource Adapter Configs

New JMS Destination Resource

The creation of a new Java Message Service (JMS) destination resource also creates an admin object res...

JNDI Name: *

Physical Destination Name *

Destination name in the Message Queue broker. If the destination does...

Resource Type: *

javax.jms.Queue

Description:

Status: Enabled

Additional Properties (0)

Add Property Delete Properties

Select	Name	Value
No items found.		

P2P + устаревший API

```
public class MySender {  
    public static void main(String[] args) {  
        try {  
            //создать соединение  
            InitialContext ctx=new InitialContext();  
            QueueConnectionFactory f=  
(QueueConnectionFactory)ctx.lookup("java:comp/DefaultJMSSelector");  
            QueueConnection con=f.createQueueConnection();  
            con.start();  
            //2) создать queue session  
            QueueSession session=  
con.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);  
            //3) получить Queue object  
            Queue queue=(Queue)ctx.lookup("BankCard");  
            //4) создать QueueSender object  
            QueueSender sender=session.createSender(queue);  
            //5) создать TextMessage object  
            ObjectMessage msg=session.createObjectMessage();  
        }  
    }  
}
```

Посыпаем

Не будут поддерживаться транзакции

```
//6) записать сообщение
BufferedReader b=
new BufferedReader(new InputStreamReader(System.in));
while(true)
{
    System.out.println("Enter Msg, end to terminate:");
    String s=b.readLine();
    if (s.equals("end"))
        break;
    MyClass myClass = new MyClass();
    myClass.setId(1);
    myClass.setText(s);
    msg.setObject(myClass);
    //7) послать
    sender.send(msg);
    System.out.println("Message successfully sent.");
}
//8) закрыть
con.close();
}catch(Exception e){System.out.println(e);}
}
```

```
public class MyClass
implements Serializable

static final long
serialVersionUID = 42L

private int id;
private String tex
...
```



Принимаем синхронно

```
public class MyReceiver {  
    public static void main(String[] args) {  
        try {  
            //1) создать и стартовать connection  
            InitialContext ctx=new InitialContext();  
            QueueConnectionFactory f=  
                (QueueConnectionFactory)ctx.lookup("java:comp/DefaultJMSServerFactory");  
            QueueConnection con=f.createQueueConnection();  
            con.start();  
            //2) создать Queue session  
            QueueSession session=con.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);  
            //3) полуслить Queue object  
            Queue queue=(Queue)ctx.lookup("BankCard");  
            //4) создать QueueReceiver  
            QueueReceiver receiver=session.createReceiver(queue);  
        }  
    }  
}
```

//синхронно

```
MessageConsumer consumer = session.createConsumer(queue);  
Message message = consumer.receive();  
if (message instanceof ObjectMessage) {  
    ObjectMessage objectMessage = (ObjectMessage) message;  
    MyClass msg = null;  
    try {  
        msg = (MyClass) objectMessage.getObject();  
    } catch (JMSException e) {  
        e.printStackTrace();  
    }  
    System.out.println(msg.toString());  
}
```

```
//асинхронно
```

```
//5) создать listener object
```

```
MyListener listener=new MyListener();
```

```
//6) регистрируем listener object приемником
```

```
receiver.setMessageListener(listener);
```

```
System.out.println("Waiting for messages...");
```

```
System.out.println("press Ctrl+c to shutdown...");
```

```
while(true) {
```

```
    Thread.sleep(1000);
```

```
}
```

```
} catch (Exception e) {System.out.println(e);}
```

```
}
```

```
public class MyListener implements MessageListener {
```

```
    public void onMessage(Message m) {
```

```
        if (m instanceof ObjectMessage) {
```

```
            ObjectMessage objectMessage = (ObjectMessage) m;
```

```
            MyClass msg = null;
```

```
            try {
```

```
                msg = (MyClass) objectMessage.getObject();
```

```
            } catch (JMSEException e) {
```

```
                e.printStackTrace();
```

```
}
```

```
            System.out.println("following message is received: "+msg.toString());
```

```
}
```

Принимаем асинхронно

Topic B GlssFish

```
public class MySender {
    public static void main(String[] args) {
        try
        {   //Create and start connection
            InitialContext ctx=new InitialContext();
            TopicConnectionFactory f=(TopicConnectionFactory)ctx.lookup("myTopicConnectionFactory");
            TopicConnection con=f.createTopicConnection();
            con.start();
            //2) create queue session
            TopicSession ses=con.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
            //3) get the Topic object
            Topic t=(Topic)ctx.lookup("myTopic");
            //4)create TopicPublisher object
            TopicPublisher publisher=ses.createPublisher(t);
            //5) create TextMessage object
            TextMessage msg=ses.createTextMessage();

            //6) write message
            BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
            while(true)
            {
                System.out.println("Enter Msg, end to terminate:");
                String s=b.readLine();
                if (s.equals("end"))
                    break;
                msg.setText(s);
                //7) send message
                publisher.publish(msg);
                System.out.println("Message successfully sent.");
            }
            //8) connection close
            con.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

```
public class MyReceiver {
    public static void main(String[] args) {
        try {
            //1) Create and start connection
            InitialContext ctx=new InitialContext();
            TopicConnectionFactory f=(TopicConnectionFactory)ctx.lookup("myTopicConnectionFactory");
            TopicConnection con=f.createTopicConnection();
            con.start();
            //2) create topic session
            TopicSession ses=con.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
            //3) get the Topic object
            Topic t=(Topic)ctx.lookup("myTopic");
            //4)create TopicSubscriber
            TopicSubscriber receiver=ses.createSubscriber(t);

            //5) create listener object
            MyListener listener=new MyListener();

            //6) register the listener object with subscriber
            receiver.setMessageListener(listener);

            System.out.println("Subscriber1 is ready, waiting for messages...");
            System.out.println("press Ctrl+c to shutdown...");
            while(true){
                Thread.sleep(1000);
            }
        }catch(Exception e){System.out.println(e);}
    }
}
```

Написание компонентов, управляемых сообщениями

- ▶ MDB — это асинхронный потребитель, который вызывается контейнером в результате прихода сообщений (часть спецификации EJB, нуждаются в полном стеке технологий Java EE).

```
@MessageDriven(mappedName = "jms/Topic")  
  
public class BillMDB implements MessageListener {  
    public void onMessage(Message message) {  
        System.out.println("Сообщение получено: " +  
            message.getBody(String.class));  
    }  
}
```

контейнер управляет многопоточностью,
безопасностью и операциями