**Лекция 15**

**Введение**

*JMS (Java Messaging System)* представляет собой интерфейс к внешним системам, ориентированный на работу через сообщения. JMS является "старой" технологией - первая спецификация была опубликована в 1998 г. В настоящее время пакет **javax.jms** входит в комплект jdk, а *Sun Application Server* реализует поддержку JMS в качестве одного из сервисов.

При разработке JMS в качестве основной задачи рассматривалось создание обобщенного Java API для приложений, ориентированных на работу с сообщениями ( message-oriented application programming ), и обеспечение независимости от конкретных реализаций соответствующих служб обработки сообщений.

Таким образом, программа, написанная с использованием JMS, будет корректно работать с любой системой сообщений, поддерживающей эту спецификацию (или имеющую соответствующие интерфейсы).

Поскольку JMS является лишь оболочкой, или интерфейсом, описывающей доступные для приложения методы, для работы приложения понадобится определенная реализация этих интерфейсов JMS, называемая провайдером JMS. Они создаются независимыми производителями, и в настоящее время таких реализаций существует достаточно много (в том числе, например, реализация, включенная в Sun Application Server и распространяемая вместе с J2EE, а также MQSeries от IBM, служба JMS WebLogic от BEA, SonicMQ от Progress и другие).

Модель обмена сообщениями (и JMS ) удобно использовать в том случае, если распределенное приложение обладает следующими характеристиками:

* взаимодействие между компонентами является асинхронным;
* информация (сообщение) должна передаваться нескольким или даже всем компонентам системы (семантика передачи от одного ко многим);
* передаваемая информация используется многими внешними системами, часть из которых неизвестна на момент проектирования системы или интерфейсы которых подвержены частым изменениям (концепция ESB - Enterprise Service Bus );
* обменивающиеся информацией (сообщениями) компоненты выполняются в разное время, что требует наличия посредника для промежуточного хранения переданной информации.

**Архитектура JMS**

Архитектура JMS выглядит следующим образом ([рис. 13.1](https://www.intuit.ru/studies/courses/633/489/lecture/11092?page=1#image.13.1)):

* прикладные программы Java, использующие JMS, называются *клиентами JMS* ( JMS client );
* система обработки сообщений, управляющая маршрутизацией и доставкой сообщений, называется *JMS-провайдером* ( JMS provider );
* приложение *JMS* ( JMS application ) - это прикладная система, состоящая из нескольких JMS клиентов, и, как правило, одного JMS -провайдера. JMS -клиент, посылающий сообщение, называется *поставщиком* ( producer ). JMS -клиент, принимающий сообщение, называется потребителем ( consumer ). Один и тот же JMS -клиент может быть одновременно и поставщиком, и потребителем в разных актах взаимодействия;
* *сообщения* ( Messages ) - это объекты, передающиеся и принимающиеся компонентами (клиентами JMS );
* *средства администрирования* ( Administrative tools ) - средства управления ресурсами, использующимися клиентами.



**Рис. 13.1.** Архитектура JMS

JMS предоставляет два подхода к передаче сообщений. Первый называется *"издание-подписка"* ( publish an subscribe ) ([рис. 13.3](https://www.intuit.ru/studies/courses/633/489/lecture/11092?page=1#image.13.3)) и используется в том случае, если сообщение, отправленное одним клиентом, должно быть получено несколькими.

Второй подход называется *"точка-точка"* ( point to point ) ([рис. 13.2](https://www.intuit.ru/studies/courses/633/489/lecture/11092?page=1#image.13.2)) и служит для реализации обмена сообщениями между двумя компонентами.

Спецификация JMS называет эти два подхода зонами сообщений ( messaging domains ).



**Рис. 13.2.** Модель взаимодействия "точка-точка"



**Рис. 13.3.** Модель взаимодействия "издание-подписка"

Модель передачи сообщений "точка-точка" предоставляет возможность клиентам JMS посылать и принимать сообщения (как синхронно, так и асинхронно) через виртуальные каналы, называемые *очередями* ( queues ). Модель основывается на методе опроса, при котором сообщения явно запрашиваются (считываются) клиентом из очереди. Несмотря на то, что чтение из очереди могут осуществлять несколько клиентов, каждое сообщение будет прочитано только единожды - провайдер JMS это гарантирует.

**Модель взаимодействия "издание-подписка"**

При использовании модели взаимодействия "издание-подписка" один клиент (поставщик) может посылать сообщения многим клиентам (потребителям) через виртуальный канал, называемый *темой* ( topic ). Потребители могут выбрать подписку ( subscribe ) на любую тему. Все сообщения, направляемые в тему, передаются всем потребителям данной темы. Каждый потребитель принимает копию каждого сообщения. Модель передачи сообщений "издание-подписка", по существу, представляет собой модель сервера, инициирующего соединение и "проталкивающего" информацию на клиента. В JMS эта концепция реализуется с помощью специальных *"слушателей"* (листенеров), регистрируемых в системе. При возникновении нового события листенер, закрепленный за данной темой, возбуждается.

Следует отметить, что при использовании модели "издание-подписка" клиенты JMS могут устанавливать долговременные подписки, позволяющие потребителям отсоединиться и позже снова подключиться и получать сообщения, поступившие во время отключения связи.

#### Первое знакомство

Использование JMS предполагает выполнение разработчиком последовательности шагов. Ниже приведена общая схема применения JMS API.

Любой компонент, использующий JMS, прежде всего должен создать соединение с JMS -провайдером - собственно системой, обеспечивающей всю функциональности управления сообщениями.



**Рис. 13.4.** Общая схема использования JMS API

Для этого используется специальная фабрика объектов - **ConnectionFactory. ConnectionFactory** на самом деле является интерфейсом, от которого наследуют **QueueConnectionFactory, TopicConnectionFactory, XAQueueConnectionFactory** и **XATopicConnectionFactory**.Таким образом, мы будем иметь дело с реализацией одного из этих интерфейсов. Для того чтобы получить доступ к **ConnectionFactory**,программа должна извлечь соответствующую ссылку из JNDI. С использованием механизма аннотаций соответствующий код будет иметь следующий вид:

@>Resource(mappedName="jms/ConnectionFactory")

private static ConnectionFactory connectionFactory;

Указанный фрагмент кода извлекает ресурс с JNDI -именем *jms/ConnectionFactory* и связывает его с переменной connectionFactory.

Естественно, перед первым применением **ConnectionFactory** должна быть создана командой

asadmin.bat create-jms-resource --user admin --passwordfile

admin-password.txt --host localhost --port 4848 --restype

javax.jms.ConnectionFactory --enabled=true jms/ConnectionFactory

Данная команда создает ресурс типа **javax.jms.ConnectionFactory** с именем *jms/ConnectionFactory* .

Следующим шагом является создание соединения с JMS провайдером. Соответствующий код будет выглядеть следующим образом:

Connection connection = connectionFactory.createConnection();

Следует отметить, что соединение должно быть закрыто после того, как оно перестает использоваться **(connection.close())**.

После того как соединение создано, могут быть созданы виртуальные каналы, в рамках которых будет осуществляться передача сообщений. Существуют два типа таких каналов - *очередь* ( Queue ) и *тема* ( Topic ). Следует отметить, что создание виртуального канала требует наличия соответствующего *"пункта назначения"* ( destination ), созданного в JNDI. Таким образом, прежде чем программа сможет задействовать очередь (или тему), соответствующий объект должен быть создан в JMS -провайдере. Для *Sun Application Server* соответствующая команда будет иметь вид:

asadmin.bat create-jms-resource --user admin --passwordfile

admin-password.txt --host localhost --port 4848 --restype

javax.jms.Queue --enabled=true --property Name=PhysicalQueue jms/Queue

где **restype** указывает тип объекта - в данном случае это **javax.jms.Queue**,а параметр **property** задает JNDI имя объекта - **jms/Queue**.

Для того, чтобы в программе получить ссылку на соответствующий объект, можно использовать код вида

@>Resource(mappedName="jms/Queue") private static Queue queue;

для очереди, или, если нужно получить ссылку на объект типа тема:

@>Resource(mappedName="jms/Topic") private static Topic topic;

Еще раз обращаем внимание на то, что объекты с именами *jms/Queue* или *jms/Topic* должны быть предварительно созданы.

После того как соединение с JMS провайдером установлено и получены ссылки на соответствующие пункты назначения (очередь или тема), может начаться собственно процесс обмена сообщениями.

Весь процесс обмена происходит в рамках *сессии* ( Session ), которая представляет собой однопотоковый контекст для обмена сообщениями. Сессия также предоставляет транзакционный контекст для обеспечения атомарности выполнения групп передач/приемов сообщений1. Сессия может быть создана следующим образом:

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

где первый параметр указывает на то, что секция не транзакционна, а второй параметр указывает на то, что необходимо автоматически подтверждать успешную доставку сообщений.

В рамках сессии приложение может создать ряд объектов, обеспечивающих отправку и приемку сообщений. Это **Message Producers** и **Message Consumers**.

Создаются эти объекты следующим образом:

MessageProducer producer = session.createProducer(queue);

MessageConsumer consumer = session.createConsumer(queue);

В качестве аргумента методы создания требуют указания пункта назначения. После создания посредством **MessageProducer** приложение может отправлять сообщения, а посредством **MessageConsumer** - принимать. Для отправки сообщения достаточно вызвать метод send, передав ему в качестве аргумента объект типа **Message**:

producer.send(message)

Для синхронного приема сообщения необходимо сначала запустить диспетчеризацию сообщения, затем вызвать метод receive соответствующего **MessageConsumer**.Метод **receive** является блокирующим, однако он может быть вызван с указанием количества миллисекунд, в течение которых он должен ожидать чтения сообщения.

connection.start();

Message m = consumer.receive();

при этом управление вернется в момент прихода сообщения, или

connection.start();

Message m = consumer.receive(2000);

При этом управление вернется в момент прихода сообщения или по истечении 2 секунд.

Для асинхронного приема сообщения может быть создан специальный класс - листенер, который должен быть зарегистрирован в рамках данного **MessageConsumer** и метод которого **onMessage** будет вызван в момент прихода сообщения.

Listener myListener = new Listener();

consumer.setMessageListener(myListener);

**Типы сообщений**

Сообщение в JMS - это объект Java, состоящий из двух частей: заголовка ( header ) и тела ( body ) сообщения. В заголовке находится служебная информация, тело сообщения содержит в себе пользовательские данные, которые могут быть разной формы: текстовой, сериализуемых объектов, байтовых потоков и т. д.

JMS API определяет несколько типов сообщений:

* **BytesMessage** предназначен для передачи потока байт, который система никак не интерпретирует;
* **MapMessage** предназначен для передачи множества элементов типа "имя-значение", где имена являются объектами строкового типа, а значения - объектами примитивных типов данных Java ;
* **ObjectMessage** предназначен для передачи сериализуемых объектов;
* **StreamMessage** предназначен для передачи множества элементов примитивных типов данных Java (они могут быть последовательно записаны, а затем прочитаны из тела сообщения этого типа);
* **TextMessage** предназначен для передачи текстовой информации.

**Первая программа**

Приведем практический пример применения рассмотренных технологий и пройдем по шагам все этапы создания приложения с использованием JMS.

Во-первых, нам потребуется JMS -провайдер. В качестве такового будет использован встроенный сервис Sun Java System Application Server, поставляемый в составе пакета J2EE. Инсталляция этого сервера приложений была рассмотрена ранее.



**Рис. 13.5.** Административная консоль. Создание ConnectionFactory

Как уже говорилось ранее, первым шагом при разработке JMS приложения является создание необходимых ресурсов. Первый ресурс, который будет нами создан, - **ConnectionFactory**.В нашем случае это делается командой

asadmin.bat create-jms-resource -user admin --passwordfile

admin-password.txt --host localhost --port 4848 --restype

javax.jms.ConnectionFactory --enabled=true

jms/Exаmple1ConnectionFactory

Убедиться, что указанный ресурс создан, можно с помощью административной консоли2.

Следующим ресурсом, который необходимо создать, является "пункт назначения". Для нашего примера будет создан пункт назначения типа "очередь" с именем **Example1Queue**:



**Рис. 13.6.** Административная консоль. Создание пункта назначения

asadmin.bat create-jms-resource --user admin --passwordfile

admin-password.txt --host localhost --port 4848 --restype

javax.jms.Queue --enabled=true --property Name=PhysicalQueue

jms/Exаmple1Queue

Как и в случае с **ConnectionFactory**,созданный объект доступен в административной консоли.

Теперь, после того как все необходимые ресурсы созданы, можно приступить к созданию приложения. В качестве первого примера мы разработаем простое приложение, которое состоит из двух компонентов, обменивающихся текстовыми сообщениями в синхронном режиме.