GLOBAL 2007

Обучение Java

Тема **EJB 3.0.**

Часть **Передача сообщений и разработка MDB**

Автор **ASKIL (omendba@gmail.com)**

14.07.2007

Основные концепции передачи сообщений

Когда мы говорим о передаче сообщений в контексте J2EE, то мы реально подразумеваем процесс свободной двухсторонней, асинхронной коммуникации между компонентами системы. Большинство коммуникаций между компонентами синхронно, типа простого вызова метода или Java RMI. Синхронная коммуникация также означает, что клиент должен ждать ответ от сервиса перед продолжением.

Можно провести аналогию синхронного общения – когда человек звонит и говорит с кем-то по телефону. Но что, если собеседник недоступен. Тогда человек оставляет голосовое сообщение. Голосовое сообщение делает коммуникацию асинхронной, для этого сообщение сохраняется, чтобы приемник мог прослушать его позже и ответить. Ориентированное на сообщение программное обеспечение (MOM) реализует передачу сообщений почти таким же способом, действует как посредник между отправителем сообщения и приемником.

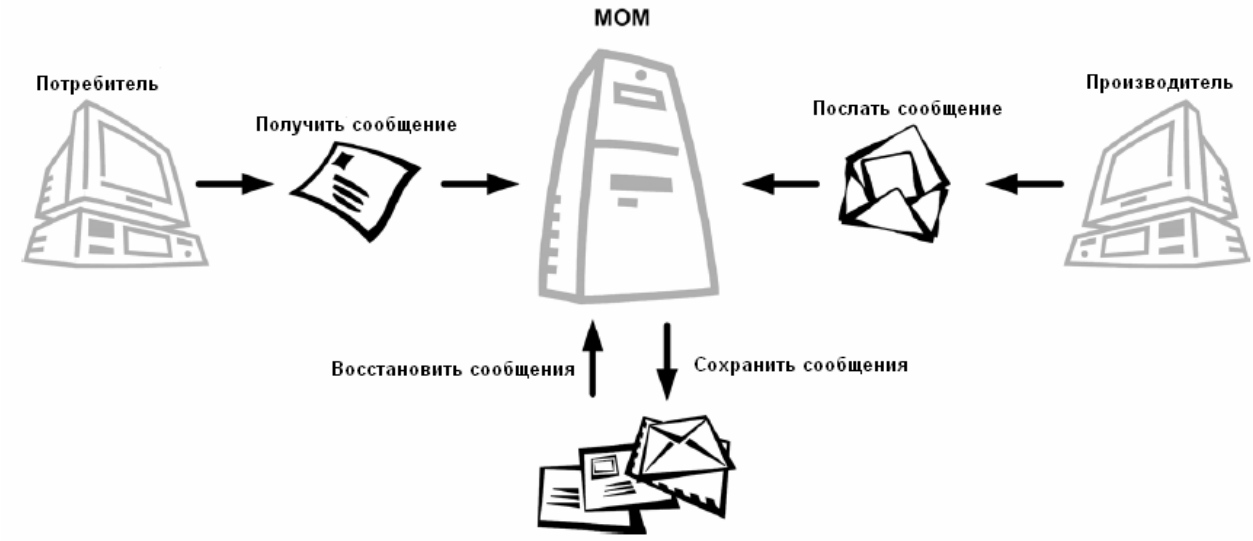


Иллюстрация №1. Когда производитель посылает сообщение программному обеспечению, оно сохраняется немедленно и позже забирается потребителем.

MOM

Когда посылают сообщение, MOM хранит сообщение в месте, под названием *destination* (предназначение) и подтверждает получение немедленно. Отправителя сообщения называют *producer* (производитель). Компоненты программного обеспечения, получающие сообщения называют *consumers* (потребитель).

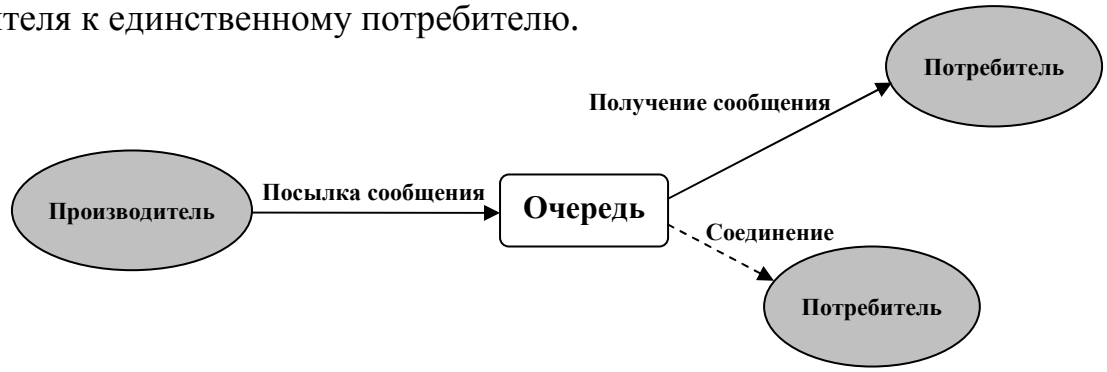
Продукты MOM включают IBM WebSphere MQ, TIBCO Rendezvous, SonicMQ, ActiveMQ, и Oracle Advanced Queuing.

Модели передачи сообщений

Передающая модель – просто путь передачи сообщений, когда множество отправителей и потребителей работают с системой одновременно. Две популярных передающих модели стандартизированы в Java EE: *point to-point* (PTP) и *publish-subscribe* (PUB/SUB).

Point to-point (PTP)

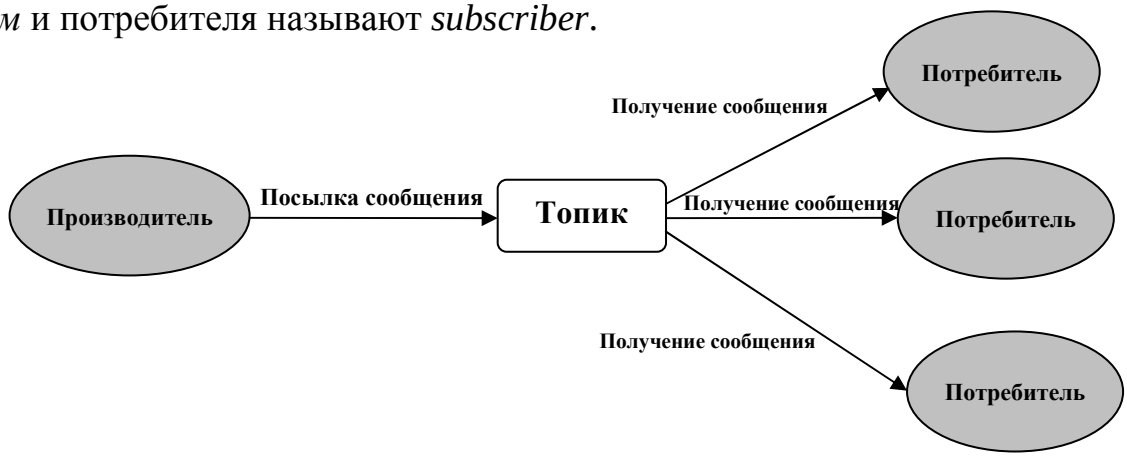
В схеме PTP, единственное сообщение идет от единственного производителя к единственному потребителю.



Предназначенное место для сообщений PTP называют *очередью*. Если больше чем один потенциальный приемник существует для сообщения, будет выбран случайный приемник. Это похоже на классическую историю “сообщение в бутылке” – является хорошей аналогией передачи сообщений PTP. Сообщение в бутылке оставлено потерпевшим кораблекрушение (производитель). Океан (очередь) несет сообщение анонимному обитателю берега (потребитель), и сообщение может быть "найдено" только однажды.

Publish-subscribe (PUB/SUB)

Единственный производитель производит сообщение, которое может быть получено любым числом потребителей, которые в данный момент связаны с очередью. Предназначение сообщения в этой модели называют *топиком* и потребителя называют *subscriber*.



Введение в Java Messaging Service

Java Message Service (JMS) – это API, разработанный Sun Microsystems и являющийся частью Java 2 Platform, Enterprise Edition (J2EE). Эта система предоставляет Java-программистам возможности создания, отправки, получения и обработки сообщений электронной почты.

Так как системы передачи сообщений являются асинхронными, JMS-клиент может посылать сообщения, не дожидаясь ответа на них. Эти системы – противоположность системам, основанным на RPC, которые применяются в технологиях EJB, CORBA и Java RMI. Клиент, использующий RPC, должен ждать, пока вызванный метод ответит. После этого клиент может выполнять следующую команду. При использовании JMS клиент может отправлять сообщение разделу, к которому подключено множество клиентов JMS. После отправки сообщения клиенту JMS не требуется ждать ответа на это сообщение.

Чтобы обеспечить доставку и очередность сообщений, JMS-приложения используют базы данных. Для поиска и получения объектов, а также для присвоения им имен JMS-приложения могут воспользоваться JNDI. С появлением спецификации EJB 2.0 интеграция EJB в JMS-приложения упростилась. Новый управляемый сообщениями bean-компонент EJB может асинхронно получать и обрабатывать асинхронные сообщения в контейнере. Создав несколько экземпляров таких управляемых сообщениями bean-компонентов, можно обеспечить параллельную обработку очереди сообщений.

JMS является интерфейсом, связывающим Java-приложения и MOM-программы. JMS позволяет клиентам (или узлам) обмениваться данными в виде сообщений. Основными преимуществами использования сообщений являются:

- Простота интеграции несовместимых систем
- Асинхронность связи
- Возможность передачи информации из одного источника нескольким адресатам
- Гарантия прохождения сообщений
- Передача сообщений посредством транзакций

Составные части JMS-приложения

- *JMS-провайдер*

Хост-приложение, под управлением которого запускается JMS-приложение. JMS-провайдер общается с JMS-приложениями и предоставляет им механизмы, необходимые для работы с сообщениями.

- *Администрируемые объекты*

JMS-объекты, созданные и поддерживаемые администратором и предназначенные для использования JMS-клиентами.

- *Клиенты*

Приложения, которые могут отправлять и/или получать сообщения.

• *Сообщения*

Пакеты информации, которыми обмениваются приложения. В каждом приложении определены типы данных, которые могут содержаться в сообщении.

Режимы связи

В большинстве приложений используется синхронная связь. При этом запрашивающий процесс не может обрабатывать никакие команды до тех пор, пока не получит ответ на запрос (или пока не истечет время ожидания).

Синхронная связь

Сеансы синхронной связи проводятся между двумя сторонами. Обе стороны при этом должны быть активны и обе должны подтвердить получение сообщения (блокирующий вызов) перед началом сеанса. По мере роста объемов передаваемых данных растет необходимая пропускная способность каналов связи и появляется необходимость в установке дополнительной аппаратуры. Подобные системы легко выводятся из равновесия отказами аппаратуры, ПО и сети, что приводит к задержке передачи сообщений и повторам. При перегрузке каналов связи информация обычно теряется безвозвратно.

Асинхронная связь

При асинхронной связи обе стороны являются равноправными и могут отправлять получать сообщения по своему усмотрению. При асинхронной связи не требуется подтверждения получения сообщения в реальном времени; запрашивающий процесс может обрабатывать новые команды сразу после отправки сообщений (неблокирующий вызов).

Примером асинхронной связи может служить электронная почта.

Разработка JMS производителя сообщений

Фабрика соединения и предназначение

В JMS административные объекты подобны JDBC javax.sql.DataSource объектам. Эти ресурсы создаются и конфигурируются без кода и хранятся в JNDI. JMS имеет два административных объекта: javax.jms.ConnectionFactory и javax.jms.Destination. С EJB 3 использование ресурсов происходит намного легче: нет необходимости иметь дело со сложностью JNDI или формировать ссылки ресурса в дескрипторах поставки.

Открытие соединения и сессия

Javax.jms.Connection объект представляет соединение с MOM, которое мы создаем, используя *createConnection* метод фабрики соединения. Соединения потоко-безопасны и разработаны с учетом обеспечения совместного доступа, потому что открытие нового соединения – емкий ресурс. Сессия JMS (javax.jms.Session) обеспечивает единственный способ, ориентируемый на задачу посылки и получения сообщения. Мы создаем сессию от соединения, используя *createSession* метод. Первый параметр метода определяет, является ли сессия транзакционной. Если сессия транзакционная, то должны установить этот параметр в true. Если сессия будет не транзакционной, то сообщения будут посылаться, как только будет вызван метод посылки. Второй

параметр createSession метода определяет способ и имеет эффект для нетранзакционных сессий, получающих сообщения.

Подготовка и посылка сообщения

Сессия непосредственно не используется для того, чтобы посылать или получать сообщения. Вместо этого javax.jms.MessageProducer должен посылать сообщения, создается, используя *createProducer* метод сессии. Затем мы создаем и заполняем javax.jms.Message, которое будет послано.

Удаление ресурсов

Большое количество ресурсов занимается и для сессии и для объектов соединения, поэтому важно явно закрыть оба, как только мы закончили с ними, вызвав методы *session.close()* и *connection.close()*.

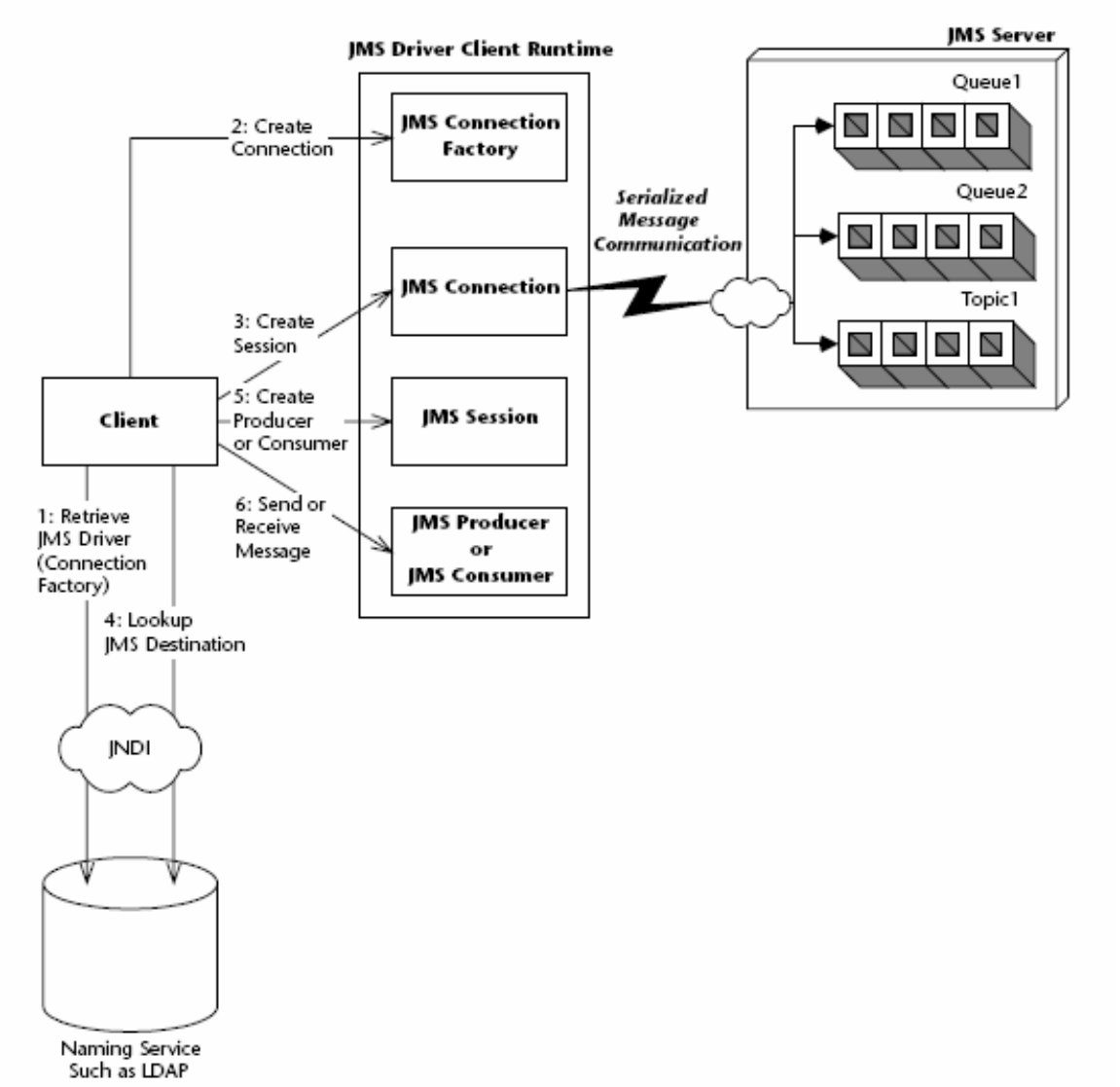


Иллюстрация №2. Представление клиента о системе JMS.

Интерфейс сообщения JMS

Сообщение представляет собой блок данных, пересылаемый процессом, запущенным на одном компьютере, другим процессам, запущенным на том же или другом компьютере. Сообщение может содержать либо только текст, либо более сложную структуру данных, например такую, как объект Java Hashtable. Однако объект должен относиться к типу Serializable. Объект тако-

го типа может быть превращен в последовательность байтов, передан по сети и затем воссоздан в первоначальном виде. Обычно для передачи объектов по сети с одного компьютера на другой используется именно этот способ.

Сообщения JMS представляются в простом, но гибком формате, который позволяет отправителю создавать сообщения, совпадающие по структуре с сообщениями, созданными не в JMS-программах на различных платформах. JMS-сообщения представляют собой данные в определенном формате (такие, как запрос, событие или сообщение о состоянии), передаваемые одними приложениями другим. Сообщение может содержать данные как простых, так и сложных типов. JMS-приложения могут также передавать сериализуемые объекты Java другим Java-приложениям.

JMS-сообщение состоит из одной необходимой части и двух других, без которых можно обойтись:

- Заголовок (необходимо)
- Свойства (необязательно)
- Тело сообщения (необязательно)

Поля заголовка

Заголовок JMS-сообщения – несколько полей, в которых хранится информация, применяющаяся для идентификации сообщений и определения маршрута их следования. У каждого из этих полей заголовка есть соответствующие методы get/set. Большая часть значений автоматически устанавливается методами send() и publish(), но некоторые из них может установить и клиент.

Поле заголовка	Описание
JMSMessagelD	Уникальный идентификатор сообщения.
JMSDeliveryMode	*PERSISTENT* означает, что доставка сообщения гарантирована. Сообщение будет существовать до тех пор, пока его не получат все запрашивавшие адресаты. Сообщение доставляется лишь единожды. *NON-PERSISTENT* означает, что система выполнит все необходимые для доставки сообщения действия, но в случае каких-либо неполадок сообщение может быть потеряно. Такие сообщения могут быть доставлены максимум один раз.
JMSExpiration	Длительность существования сообщения (в миллисекундах) до его удаления. Если в поле JMSExpiration установить значение 0, сообщение не будет удалено вообще.
JMSPriority	Хотя никаких гарантий этого и нет, сообщения с более высоким приоритетом обычно доставляются раньше сообщений с низким приоритетом. Приоритет 0 – самый низкий, а 9 – самый высокий. По умолчанию приоритет принимает значение 4. Значения от 0 до 4 являются степенями нормального приоритета, а с 5 до 9 - высокого.
JMSRedelivered	Уведомляет клиента о том, что он, вероятно, уже по меньшей мере однажды получал это сообщение раньше, но по какой-то причине не подтвердил его получения. Обычно JMS-провайдер устанавливает этот флаг во время восстановления после сбоя.
JMSReplyTo	Содержит адрес, по которому должны отправляться ответы (этот адрес предоставляется клиентом). Заполнение данного поля обычно означает, что на сообщения ожидается ответ (хотя он и не обязателен), принятие решения об отправке ответа полностью возлагается на клиента.

Свойства

Свойства – это значения, дополняющие содержащуюся в полях заголовка информацию. API JMS определяет несколько стандартных имен свойств, которые могут поддерживаться провайдером (их имена начинаются с префикса JMS_).

Пользоваться свойствами необязательно. Если вы все же решите их использовать, в них можно хранить данные типов boolean, byte, short, int, long, float, double или string. Устанавливать их значения можно при отправке сообщения – но это может происходить и при получении. Эти свойства и поля заголовка могут применяться в сочетании с MessageSelector для фильтрации и маршрутизации сообщений по установленным критериям.

Тело сообщения

Существует пять различных форматов или типов тел сообщений, которые позволяют JMS-клиентам отправлять и получать данные самых различных типов, а также обеспечивают совместимость с существующими форматами сообщений.

Формат	Содержимое
ByteMessage	Поток неинтерпретированных байтов. Этот тип тела сообщения обеспечивает наибольшую совместимость с ранее существовавшими типами сообщений.
MapMessage	Набор пар имя/значение, подобный типу HashMap. Имя – это объект типа String, а значение – примитив Java.
ObjectMessage	Один сериализуемый объект Java или несколько таких объектов.
StreamMessage	Поток значений примитивов Java, которые вводятся и считываются последовательно.
TextMessage	Текст, отформатированный в виде java.lang.String. Этот формат отлично подходит для обмена XML-данными.

Управление сеансами

Далее приведена информация об отдельных аспектах сеансов JMS. Если не утверждается обратное, эта информация относится как к модели PUB/SUB, так и к PTP.

Транзакционный сеанс

Транзакционный сеанс – это взаимосвязанная группа потребленных и произведенных сообщений, обрабатывающихся как одно целое. Транзакция может быть либо подтверждена, либо отменена.

При вызове метода commit транзакции потребленные сообщения подтверждаются, а связанные с ними произведенные сообщения отправляются. При вызове метода rollback транзакции произведенные сообщения уничтожаются, а потребленные – восстанавливаются.

Дублирующиеся сообщения

Клиенты должны иметь возможность посылать сообщения, зная, что JMS доставит их лишь единожды. Поэтому JMS-провайдер не должен посылать сообщения дважды или посылать копии сообщений, получение которых уже было подтверждено.

В заголовке сообщения есть поле флага повторной доставки. Установка данного флага “сигнализирует” клиенту о том, что сообщение, возможно, уже было доставлено ранее, но по какой-то причине JMS-сервер не принял от клиента подтверждения его получения. Флаг повторной доставки устанавли-

вается приложением JMS-провайдера, и происходит это обычно при восстановлении после сбоя.

Подтверждение получения

При выполнении транзакций сеансов связи JMS получение сообщений автоматически выполняется механизмом подтверждения транзакции, а восстановление – отменой транзакции.

Если транзакция не выполняется, восстановление необходимо проводить вручную, и подтверждение получения сообщений может производиться одним из следующих способов:

Способ подтверждения	Описание	Поддерживается MDB
AUTO_ACKNOWLEDGE	Сеанс автоматически подтверждает получение клиентом каждого сообщения при успешном завершении вызова клиента или MessageListener, вызванного сеансом для обработки сообщения	ДА
CLIENT_ACKNOWLEDGE	Клиент подтверждает получение сообщения вызовом его метода acknowledge. Таким образом, подтверждается получение и всех обработанных за время сеанса сообщений	НЕТ
DUPS_OK_ACKNOWLEDGE	При сбое JMS может произойти дублирование сообщений, но зато этот вариант позволяет сэкономить системные ресурсы при работе сеанса связи	ДА

Производство и потребление сообщений

Сообщения в JMS-приложениях создаются и принимаются при помощи четырех объектов: MessageProducer, MessageConsumer, MessageListener и MessageSelector.

MessageProducer

Объект MessageProducer создается сеансом и используется для передачи сообщений к месту назначения. В модели PTP место назначения – это очередь, в PUB/SUB – топик. При создании объекта MessageProducer можно указать:

- Режим доставки по умолчанию (setDeliveryMode) – либо *NON-PERSISTENT*, требующий меньшего количества ресурсов, так как сообщение при использовании данного режима не вносится в журнал, либо *PERSISTENT*, применение которого требует ведения журнала отправки сообщений, обычно – в базе данных.
- Приоритет сообщения (setPriority) – 0 является низшим приоритетом, 9 – высшим. Приоритет по умолчанию – 4. Значения от 0 до 4 являются степенями нормального приоритета, а с 5 до 9 – высокого.
- Продолжительность существования (setTimeToLive) – длительность существования сообщения (в миллисекундах) до его удаления. Если значение этого свойства установить в 0, сообщение не будет удалено вообще.

MessageConsumer

Объект MessageConsumer создается сеансом и используется для получения сообщений, отправленных по адресу Destination. Сообщения могут приниматься одним из двух способов:

• Синхронно – клиент вызывает один из методов получения сообщений (receive или receiveNoWait) после запуска consumer.

• Асинхронно – клиент регистрирует MessageListener и запускает объект consumer.

MessageListener

MessageListener представляет собой интерфейс, использование которого обязательно при асинхронном получении сообщений. Асинхронно получить сообщение можно, выполнив следующие действия:

• Создать объект, в котором реализован интерфейс MessageListener. В реализацию должен быть включен код метода onMessage().

• Уведомить QueueReceiver о том, сообщения каких объектов должны передаваться через метод setMessageListener().

• Вызвать метод Start() объекта Connection для запуска процесса получения сообщений.

MessageSelector

MessageSelector – это объект, используемый клиентом для фильтрации сообщений по указанным критериям. MessageSelector просматривает заголовки и свойства сообщений, сравнивая их с выражением, содержащимся в строке параметра. Синтаксис этой строки является подмножеством синтаксиса записи условных выражений SQL92.

JMS-компоненты

Компоненты, использующиеся приложениями, которые работают с сообщениями при помощи JMS:

Компонент	Назначение
Destination	Топик для модели PUB/SUB и очередь для модели PTP
ConnectionFactory	Получает указатель (содержащий IP-адрес) на JMS-сервер
Connection	Физическое соединение с JMS-сервером
Session	Отвечает за отправку/получение сообщений, обработку транзакций и подтверждений

Администрируемые объекты

Администрируемый объект – объект, созданный администратором приложения JMS-провайдера. Администрируемые объекты размещаются в пространстве имен Java Naming and Directory Interface (JNDI), ими можно управлять с обычной консоли.

В JMS есть два типа администрируемых объектов: Destination и ConnectionFactory.

• *Destination*

Объект Destination содержит информацию, предоставленную JMS-провайдером. Клиент использует этот объект для указания места назначения отправляемых сообщений и/или адреса сервера, с которого он получает сообщения. Существует два типа интерфейсов Destination – Queue для модели Point-to-Point и Topic для модели Publish/Subscribe.

• *ConnectionFactory*

ConnectionFactory можно получить через JNDI-поиск. Этот объект содержит информацию, позволяющую JMS-клиенту установить соединение с JMS-сервером. Несмотря на то, что JMS-клиенты могут работать с этими объектами через JNDI, реализация системы управления администрируемыми объектами у каждого провайдера своя. JMS-провайдер предоставляет инструментарий, позволяющий администратору создавать и удалять такие объекты.

Интерфейсные классы

В JMS определены интерфейсные классы, создающие соединения с MOM-программами и управляющие этими соединениями. Набор необходимых интерфейсов зависит от модели сообщений, которой пользуется разработчик.

Основные интерфейсы	**Интерфейсы Point-to-Point**	**Интерфейсы Publish/Subscribe**
Destination	Queue	Topic
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver и QueueBrowse	TopicSubscriber

Работа с MDB

Управляемый сообщениями bean-компонент (MDB-компонент) – это компонент, не имеющий состояния, который вызывается контейнером после получения соответствующего сообщения JMS. MDB-компоненты были созданы для того, чтобы иметь EJB-компоненты, которые могли бы асинхронно вызываться для обработки поступающих JMS-сообщений подобно тому, как работает прослушиватель событий MessageListener, и при этом иметь доступ к тем же сервисным средствам EJB-контейнера, что и bean-компоненты сеанса и объекта. MDB-компонент ведет себя как потребитель сообщения. Он, подобно стандартным потребителям JMS-сообщений, получает сообщения из раздела JMS и выполняет бизнес-логику в зависимости от содержания сообщения. MDB-компоненты получают JMS-сообщения от клиентов таким же способом, как и любой другой обслуживающий объект JMS.

Сходство MDB-компонентов и не имеющих состояния EJB-компонентов сеанса заключается в том, что они создаются динамически, и их экземпляры существуют только на протяжении конкретного вызова метода. Отличие же MDB-компонентов от bean-компонентов сеанса, не имеющих состояния и от bean-компонентов объектов состоит в том, что MDB-компоненты не имеют ни домашнего, ни удаленного интерфейсов. Следовательно, к MDB-компоненту нельзя получить доступ непосредственно из внутреннего или внешнего клиента. Клиенты могут взаимодействовать с bean-компонентами, управляемыми сообщениями, только косвенно, посылая сообщение в JMS-раздел.

MDB и параллельная обработка

В модели разработки EJB развернутый bean-компонент сообщения представляет единственного получателя сообщения, но сам bean-компонент обслуживается одним или несколькими экземплярами bean-компонента. Каждый экземпляр bean-компонента может получать сообщения, полученные bean-компонентом сообщения. Это означает, что для получения сообщений bean-компонентом должен применяться не последовательный, а параллельный способ передачи сообщений, за счет чего обеспечивается возможность создания системы уровня предприятия.

Сервисные средства контейнера для бина MDB

EJB-контейнер обеспечивает следующие сервисные средства:

- *Управление жизненным циклом MDB-компонента* – жизненный цикл MDB-компонента соответствует продолжительности промежутка жизни EJB-контейнера, в котором он развернут. Поскольку MDB-компоненты не имеют состояния, экземпляры bean-компонентов обычно помещаются EJB-контейнером в пул и извлекаются, когда сообщение становится доступным разделу, для которого он является получателем сообщения.
- *Обработка исключений* – при обработке сообщений MDB-компоненты не могут генерировать исключения уровня приложения. Другими словами,

единственные исключения, которые могут быть сгенерированы MDB-компонентом – это исключения времени выполнения, указывающие на ошибку системного уровня. Контейнер должен обработать это исключение, удалив экземпляр bean-компонента и совершив откат любой транзакции, запущенной экземпляром bean-компонента или контейнером.

- *Параллелизм и потоки* – предполагается, что экземпляр MDB-компонента может выполняться в единственном потоке, что упрощает задачу разработчика. EJB-контейнер обеспечивает такую возможность. Кроме того, EJB-контейнер может обеспечивать режим работы, при котором несколько сообщений могут обрабатываться одновременно отдельными экземплярами bean-компонента. В этой опции доставки используются классы экспертного уровня, которые определены в JMS-спецификации. Производителю JMS не требуется обеспечивать реализацию этих классов, так что EJB-контейнер не получает преимуществ от отдельной реализации JMS. Использование этих классов обеспечивает компромисс между производительностью и преобразованием в последовательную форму посылаемых серверу сообщений.
- *Подтверждение сообщений* – контейнер всегда обрабатывает подтверждение сообщения для MDB-компонента; для bean-компонента запрещено использование клиентских методов подтверждения получения сообщения, определенных в JMS-спецификации. Подтверждение сообщения может принимать значения DUPS_OK_ACKNOWLEDGE или AUTO_ACKNOWLEDGE. Первое позволяет передавать повторные сообщения после ошибки, в то время как последнее дает гарантию, что сообщение будет передано только один раз.
- *Безопасность* – поскольку MDB-компонент не имеет клиента, то на контейнер при получении сообщения не распространяются никакие правила защиты. Модель разработки EJB обеспечивает для метода bean-компонента средства выполнения в декларативно определенной роли. В результате MDB-компонент может быть конфигурирован для выполнения в контексте защиты, который может быть распространен на другие EJB-компоненты, использующиеся во время обработки сообщения. Это обеспечивает поддержку безопасности MDB-компонентов на уровне метода.

Служба сообщений Java и стандартные протоколы

MDB-часть модели разработки EJB обеспечивает простую, но реализующую качество на уровне предприятия компонентную модель для обработки сообщений, получаемых через JMS. MDB-компоненты вообще настолько практичны, что этой технологии “грозит” возможное развитие в стандартную компонентную модель обработки программных сообщений любого вида, а не только сообщений, поставляемых сервером JMS. Если сервер приложений имеет средство для преобразования протоколов, таких как HTTP, FTP, и электронной почты в JMS-сообщение, MDB-компонент может обрабатывать сообщения, посланные в соответствии с любым из этих протоколов, так, как будто они были посланы как JMS-сообщение. Это обеспечи-

вает стандартную и переносимую компонентную модель, которая может обрабатывать любое сообщение, посланное любым протоколом.

MDB-компоненты в сравнении с bean-компонентами сеансов и объектов

Между MDB-компонентами и EJB-компонентами сеансов и объектов имеется несколько различий. Bean-компонент, управляемый сообщением, поскольку он не является RPC-компонентом, не имеет ни удаленного, ни домашнего интерфейсов. MDB-компоненты не имеют методов бизнес-логики, которые синхронно вызываются клиентами EJB-компонентов. MDB-компонент прослушивает виртуальный канал раздела сообщений и получает сообщения, поставляемые другими JMS-клиентами на этот канал.

MDB-компонент похож на bean-компонент сеанса, не имеющий состояния, тем, что оба эти bean-компоненты не поддерживают никакого состояния между запросами. Кроме того, они оба содержат переменные экземпляров, которые поддерживаются на протяжении жизни экземпляров bean-компонента.

MDB-компоненты могут получать сообщения от любого раздела, обеспеченного поставщиком JMS. Сообщения, получаемые MDB-компонентами, могут исходить из других bean-компонентов, таких как bean-компоненты сеансов, объектов или сообщений, Java-приложений, не использующих JMS, или приложений, не являющихся Java-приложениями, которые использует поставщик JMS.

Почему используются MDB?

Мультипоточность

Ваше бизнес приложение может требовать мультипоточное потребление сообщений. Бины управляют входящими сообщениями множеством экземпляров (пул бинов), которые не имеют никакого специального кода мультипоточности. Как только новое сообщение достигает предназначения, экземпляр MDB восстанавливается из пула, чтобы работать с сообщением.

Упрощенное кодирование

Кроме того, MDB освобождают от кодирования механических аспектов обработки сообщений, типа поиска фабрик соединения или предназначения, создание соединений, открытия сессий, создание потребителей и слушателей.

Правила программирования

Как все EJB, MDB – простые java объекты, которые следуют за простым набором правил и иногда имеют аннотации.

• Класс MDB должен непосредственно (при использовании ключевого слова implements в декларации класса) или косвенно (через аннотации, или описатели) осуществить интерфейс слушателя сообщения.

• Класс MDB должен быть конкретным. Он не может быть или final или abstract.

• MDB должен быть классом POJO и не подклассом другого MDB.

• Класс MDB должен быть объявлен как public.

• Класс бина должен иметь конструктор без аргументов. Если не будет никаких конструкторов в классе, то компилятор создаст конструктор по умолчанию. Контейнер использует этот конструктор, чтобы создать экземпляр бина.

• Вы не можете определить finalize метод в классе бина. Если какой-нибудь код уборки необходим, это должно быть определено в методе, определяемом, как PreDestroy.

• Вы должны осуществить методы, определенные в интерфейсе слушателя сообщения. Эти методы должны быть public и не могут быть static или final.

• Вы не должны обрабатывать javax.rmi.RemoteException или любые другие исключения во время выполнения.

Использование @MessageDriven аннотации

MDB – одни из самых простых видов EJB, и они поддерживают наименьшее число аннотаций. Фактически, @MessageDriven аннотация и @ActivationConfigProperty аннотация – единственные MDB-определенные аннотации. @MessageDriven аннотация определена следующим образом:

```
@Target(TYPE)
@Retention(RUNTIME)
public @interface MessageDriven {
    String name() default "";
    Class messageListenerInterface default Object.class;
    ActivationConfigProperty[] activationConfig() default {};
    String mappedName();
    String description();
}
```

Заметьте, что все три аргумента аннотации являются дополнительными. Первый элемент, *name*, определяет название MDB. Если элемент name опущен, код использует название класса. Второй параметр, *messageListenerInterface*, определяет, какого слушателя сообщения MDB осуществляет. Последний параметр, *activationConfig*, используется, чтобы определить определенные для слушателя свойства конфигурации.

Реализация MessageListener

MDB реализует интерфейс javax.jms.MessageListener слушателя сообщения. Контейнер использует интерфейс слушателя, чтобы связать MDB с по-

ставщиком сообщения и передавать поступающие сообщения, вызывая реализованные методы слушателя сообщения.

Использование ActivationConfigProperty

ActivationConfig свойство @MessageDriven аннотации позволяет обеспечить передачу сообщений через определенную для системы информацию конфигурации через массив из экземпляров ActivationConfigProperty. ActivationConfigProperty определен следующим образом:

```
public @interface ActivationConfigProperty {
    String propertyName();
    String propertyValue();
}
```

Каждое свойство активации – по существу пара значений, которую поставщик понимает и использует, чтобы настроить MDB.

acknowledgeMode

Сообщения фактически не удаляются из очереди, пока потребитель не подтвердит их получение. Есть много “способов”, с помощью которых это можно сделать. По умолчанию, для основной сессии JMS этот параметр равен AUTO_ACKNOWLEDGE, что означает, что сессия подтвердила сообщение от нашего имени автоматически. Можно изменить данный способ на DUPS_OK_ACKNOWLEDGE следующим образом:

```
@ActivationConfigProperty(propertyName="acknowledgeMode",
                          propertyValue="DUPS_OK_ACKNOWLEDGE")
```

subscriptionDurability

Если MDB слушает топик, мы можем определить, будет ли подписка топика *durable* (длительной) или *nondurable* (недолговременной). Так как в PUB/SUB модели сообщение отсылается всем потребителям в текущий момент времени, которые подписались. Это очень походит на широковещательное сообщение, и подписчик, который не связан с топиком в текущий момент, когда идет рассылка, не получит копию сообщения. Исключение из этого правила – длительная подписка (durable subscription).

Как только потребитель подписывается на длительную подписку, все сообщения, посланные в топик, гарантированно будут доставлены этому потребителю. Если длительный подписчик не связан с топиком, когда сообщение прислано в топик, MOM сохраняет копию сообщения, когда подписчик вновь соединится с MOM. Следующие код показывает как создать длительного подписчика:

```
MessageConsumer ClientSubscriber = session.createDurableSubscriber(DoomTopic,"Jey");
```

Здесь мы создаем потребителя с идентификационным номером Jey, подписанного на длительный срок, на сообщения из топика DoomTopic. С этого момента времени, все сообщения в топике будут доставлены потребителю с ID Jey. Вы можете удалить эту подписку следующим кодом:

```
session.unsubscribe("Jey");
```

Если нужно, чтобы MDB был длительным подписчиком, то пример ActivationConfigProperty:

```
@ActivationConfigProperty(propertyName="destinationType",
                          propertyValue="javax.jms.Topic"),
@ActivationConfigProperty(propertyName="subscriptionDurability",
                          propertyValue="Durable")
```

Для недолговременных подписок, явно установите значение свойства subscriptionDurability в NonDurable, который является также значением по умолчанию.

messageSelector

Если необходимо, то можно фильтровать сообщения, используя message selector – создает критерий отбора по заголовкам сообщения и свойствам сообщений. Например, если мы хотим получать все запросы, чей параметр PAR1 установлен в TRUE, мы должны использовать следующий код:

```
MessageConsumer consumer = session.createConsumer(destination,"PAR1 IS TRUE");
```

Используя наш пример селектора сообщения JMS, мы могли определить так:

```
@ActivationConfigProperty(propertyName="messageSelector",
                          propertyValue="PAR1 IS TRUE")
```

Тип	Описание	Пример
Литералы	Может иметь строковый тип, точные и приближенные числовые значения, или булевы значения	BidManagerMDB 100 TRUE
Идентификаторы	Может быть свойством сообщения или названием заголовка; чувствителен к регистру.	RECIPIENT NumOfBids Fragile JMSTimestamp
Пробелы	Так же как определено в Java-спецификации	
Операторы сравнения	Операторы сравнения, типа =,>,>=,<=,<>	RECIPIENT='BidManagerMDB' NumOfBids>=100
Логические операторы	Все три типа логических операторов поддерживаются – NOT,AND,OR.	RECIPIENT='BidManagerMDB' AND NumOfBids>=100
NULL сравнение	IS NULL и IS NOT NULL сравнения	FirstName IS NOT NULL
TRUE/FALSE сравнение	IS [NOT] TRUE и IS [NOT] FALSE сравнения	Fragile IS TRUE Fragile IS FALSE

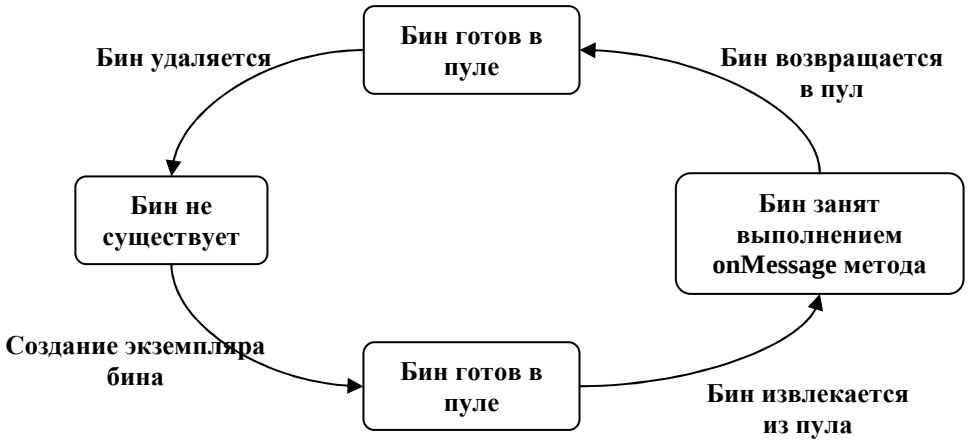
Жизненный цикл MDB

Контейнер выполняет следующее:

- Создает MDB экземпляр и настраивает его
- Инъецирует ресурсы, включая контекст MD
- Размещает экземпляр в пуле
- Выбирает необходимый бин из пула, когда сообщение приходит
- Выполняет метод слушателя сообщения, например onMessage метод

• Когда метод onMessage заканчивает свое выполнение, бин забирается обратно в пул

• Когда это необходимо, бины удаляются из пула



В ЖЦ MDB вызываются два метода – *PostConstruct*, который вызывается немедленно после того, как MDB создан и настроен, и все ресурсы инъецированы, и *PreDestroy*, который вызывается прежде, чем экземпляр бина удаляется.

Управление транзакциями MDB

Если о параметрах транзакции ничего не известно MDB, то он управляет транзакцией по умолчанию. По умолчанию, контейнер начинает транзакцию прежде, чем onMessage метод вызывается и фиксирует транзакцию, когда метод возвращается, если транзакция была отмечена как *rollback* в контексте MDB.

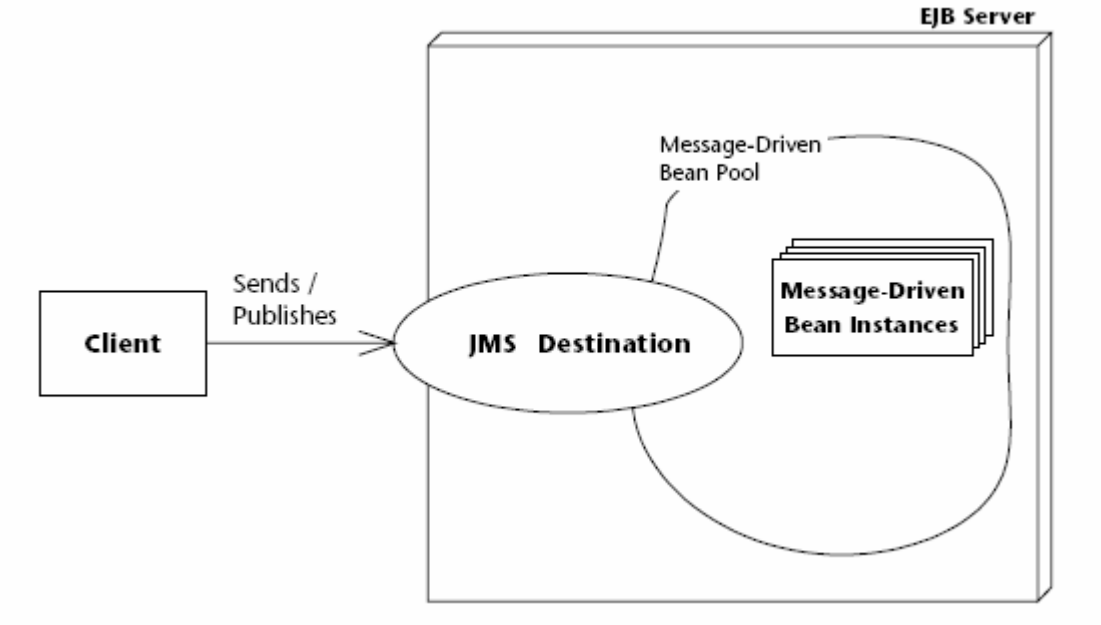


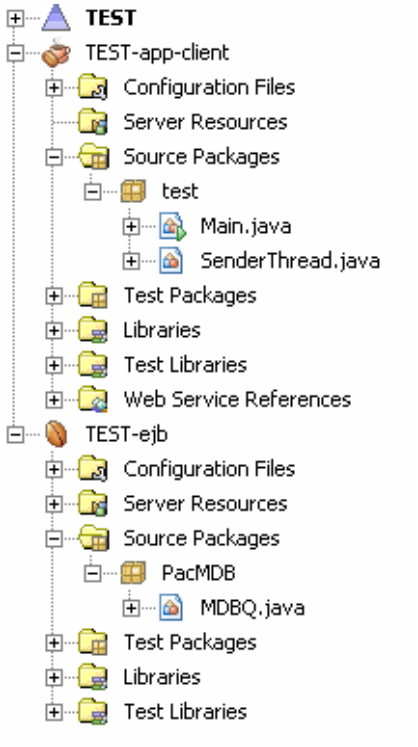
Иллюстрация №3. Клиент, посылающий сообщения MDB бинам посредством JMS.

Лучшие практические рекомендации разработки MDB

Как все технологии, технология MDB имеет некоторые ловушки.

- Выбирайте модели сообщений тщательно.
- Помните модуляризацию. Поскольку MDB подобны бинам сессии, лучше помещать бизнес логику прямо в методы слушателя сообщений. Отличное решение – разместить бизнес логику в сессионных бинах и вызывать их из onMessage метода.
- Используйте фильтры сообщения.
- Выберите типы сообщения тщательно. Выбор типа сообщения не всегда очевиден, как это кажется. Например, хорошая идея использовать XML-формат сообщений.
- Опасайтесь побочных сообщений. Сообщения могут прийти в формате, который ваш бин не в состоянии отработать. Необходимо тщательно обрабатывать входящие сообщения.

Пример MDB №1




```
package test;
import javax.jms.*;
import javax.naming.*;
import javax.annotation.*;
public class Main
{
    public static void main(String[] args) throws Exception
    {
      (new SenderThread()).start();
      Thread.sleep(10000);
      (new SenderThread()).start();
    }
}
```

```
package test;
import javax.jms.*;
import javax.naming.*;
import javax.annotation.*;
public class SenderThread extends Thread
{
    public void run()
    {
      InitialContext ic;
      Queue q;
      QueueConnectionFactory qcf;
      QueueConnection qc;
      QueueSession qs;
      QueueSender qsend;
      try
      {
        ic = new InitialContext();
        q = (Queue) ic.lookup("jms/MDBQ");
        qcf = (QueueConnectionFactory) ic.lookup("jms/MDBQFactory");
        qc = qcf.createQueueConnection();
        qs = qc.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
        qsend = qs.createSender(q);
        qsend.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
        TextMessage tmes = qs.createTextMessage();
        for(int i=1;i<20;i++)
        {
            tmes.clearBody();
            tmes.setText("отослано сообщение № "+i+" от "+((new java.util.Date()).toString())+
                         " потоком - "+this.getName());
            qsend.send(tmes);
            Thread.sleep(1000);
        }
        qsend.close();
        qs.close();
        qc.close();
        System.out.println("Поток "+this.getName()+" закончил отсылать сообщения в "+
                    ((new java.util.Date()).toString()));
```

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

```
package PacMDB;
import javax.ejb.*;
import javax.jms.*;
import javax.annotation.*;
@MessageDriven(mappedName = "jms/MDBQ", activationConfig =  {
        @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "clientId", propertyValue = "MDBQ")
    })
public class MDBQ implements MessageListener
{
    private int count;
    @PostConstruct
    public void init()
    {
       System.out.println("Начало обработки очереди!");
       count=1;
    }
    public void onMessage(Message message)
    {
        if (message instanceof TextMessage)
        {
          TextMessage tm = (TextMessage) message;
          try
          {
              String text = tm.getText();
              System.out.println("Получено сообщение: "+"\""+text+"\"");
              System.out.println("Обработано сообщение № "+String.valueOf(count)+"!");
              System.out.println("    JMSMessageID: "+tm.getJMSMessageID());
              count++;
         }
         catch (JMSException e){e.printStackTrace();}
        }
    }
}
```

Серверная консоль:

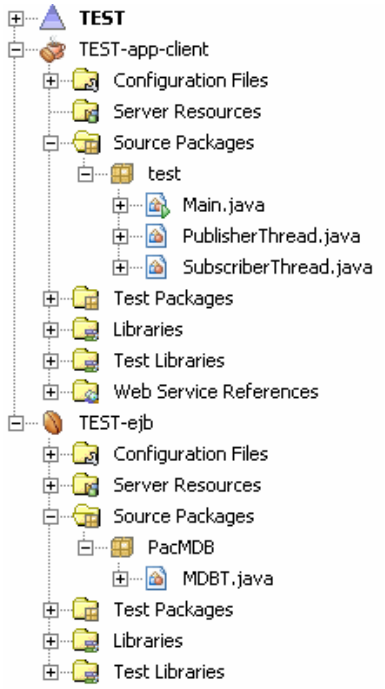
```
Начало обработки очереди!
Получено сообщение: "отослано сообщение № 1 от Sat Jul 14 11:51:11 MSD 2007 потоком - Thread-0"
Обработано сообщение № 1!
    JMSMessageID: ID:40-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399471250
Получено сообщение: "отослано сообщение № 2 от Sat Jul 14 11:51:12 MSD 2007 потоком - Thread-0"
Обработано сообщение № 2!
    JMSMessageID: ID:41-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399472250
Получено сообщение: "отослано сообщение № 3 от Sat Jul 14 11:51:13 MSD 2007 потоком - Thread-0"
Обработано сообщение № 3!
    JMSMessageID: ID:42-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399473265
Получено сообщение: "отослано сообщение № 4 от Sat Jul 14 11:51:14 MSD 2007 потоком - Thread-0"
Обработано сообщение № 4!
    JMSMessageID: ID:43-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399474265
Получено сообщение: "отослано сообщение № 5 от Sat Jul 14 11:51:15 MSD 2007 потоком - Thread-0"
Обработано сообщение № 5!
    JMSMessageID: ID:44-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399475281
Получено сообщение: "отослано сообщение № 6 от Sat Jul 14 11:51:16 MSD 2007 потоком - Thread-0"
Обработано сообщение № 6!
    JMSMessageID: ID:45-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399476328
Получено сообщение: "отослано сообщение № 1 от Sat Jul 14 11:51:16 MSD 2007 потоком - Thread-24"
Обработано сообщение № 7!
    JMSMessageID: ID:50-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399476750
Получено сообщение: "отослано сообщение № 7 от Sat Jul 14 11:51:17 MSD 2007 потоком - Thread-0"
Обработано сообщение № 8!
    JMSMessageID: ID:51-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399477343
Получено сообщение: "отослано сообщение № 2 от Sat Jul 14 11:51:17 MSD 2007 потоком - Thread-24"
Обработано сообщение № 9!
    JMSMessageID: ID:52-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399477750
Получено сообщение: "отослано сообщение № 8 от Sat Jul 14 11:51:18 MSD 2007 потоком - Thread-0"
Обработано сообщение № 10!
    JMSMessageID: ID:53-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399478343
Получено сообщение: "отослано сообщение № 3 от Sat Jul 14 11:51:18 MSD 2007 потоком - Thread-24"
Обработано сообщение № 11!
    JMSMessageID: ID:54-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399478750
Получено сообщение: "отослано сообщение № 9 от Sat Jul 14 11:51:19 MSD 2007 потоком - Thread-0"
Обработано сообщение № 12!
    JMSMessageID: ID:55-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399479359
Получено сообщение: "отослано сообщение № 4 от Sat Jul 14 11:51:19 MSD 2007 потоком - Thread-24"
Обработано сообщение № 13!
    JMSMessageID: ID:56-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399479765
Получено сообщение: "отослано сообщение № 10 от Sat Jul 14 11:51:20 MSD 2007 потоком - Thread-0"
Обработано сообщение № 14!
    JMSMessageID: ID:57-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399480359
Получено сообщение: "отослано сообщение № 5 от Sat Jul 14 11:51:20 MSD 2007 потоком - Thread-24"
Обработано сообщение № 15!
    JMSMessageID: ID:58-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399480765
Получено сообщение: "отослано сообщение № 11 от Sat Jul 14 11:51:21 MSD 2007 потоком - Thread-0"
```

```
Обработано сообщение № 16!
    JMSMessageID: ID:59-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399481359
Получено сообщение: "отослано сообщение № 6 от Sat Jul 14 11:51:21 MSD 2007 потоком - Thread-24"
Обработано сообщение № 17!
    JMSMessageID: ID:60-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399481765
Получено сообщение: "отослано сообщение № 12 от Sat Jul 14 11:51:22 MSD 2007 потоком - Thread-0"
Обработано сообщение № 18!
    JMSMessageID: ID:61-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399482375
Получено сообщение: "отослано сообщение № 7 от Sat Jul 14 11:51:22 MSD 2007 потоком - Thread-24"
Обработано сообщение № 19!
    JMSMessageID: ID:62-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399482781
Получено сообщение: "отослано сообщение № 13 от Sat Jul 14 11:51:23 MSD 2007 потоком - Thread-0"
Обработано сообщение № 20!
    JMSMessageID: ID:63-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399483375
Получено сообщение: "отослано сообщение № 8 от Sat Jul 14 11:51:23 MSD 2007 потоком - Thread-24"
Обработано сообщение № 21!
    JMSMessageID: ID:64-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399483781
Получено сообщение: "отослано сообщение № 14 от Sat Jul 14 11:51:24 MSD 2007 потоком - Thread-0"
Обработано сообщение № 22!
    JMSMessageID: ID:65-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399484390
Получено сообщение: "отослано сообщение № 9 от Sat Jul 14 11:51:24 MSD 2007 потоком - Thread-24"
Обработано сообщение № 23!
    JMSMessageID: ID:66-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399484781
Получено сообщение: "отослано сообщение № 15 от Sat Jul 14 11:51:25 MSD 2007 потоком - Thread-0"
Обработано сообщение № 24!
    JMSMessageID: ID:67-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399485390
Получено сообщение: "отослано сообщение № 10 от Sat Jul 14 11:51:25 MSD 2007 потоком - Thread-24"
Обработано сообщение № 25!
    JMSMessageID: ID:68-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399485781
Получено сообщение: "отослано сообщение № 16 от Sat Jul 14 11:51:26 MSD 2007 потоком - Thread-0"
Обработано сообщение № 26!
    JMSMessageID: ID:69-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399486406
Получено сообщение: "отослано сообщение № 11 от Sat Jul 14 11:51:26 MSD 2007 потоком - Thread-24"
Обработано сообщение № 27!
    JMSMessageID: ID:70-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399486796
Получено сообщение: "отослано сообщение № 17 от Sat Jul 14 11:51:27 MSD 2007 потоком - Thread-0"
Обработано сообщение № 28!
    JMSMessageID: ID:71-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399487406
Получено сообщение: "отослано сообщение № 12 от Sat Jul 14 11:51:27 MSD 2007 потоком - Thread-24"
Обработано сообщение № 29!
    JMSMessageID: ID:72-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399487796
Получено сообщение: "отослано сообщение № 18 от Sat Jul 14 11:51:28 MSD 2007 потоком - Thread-0"
Обработано сообщение № 30!
    JMSMessageID: ID:73-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399488406
Получено сообщение: "отослано сообщение № 13 от Sat Jul 14 11:51:28 MSD 2007 потоком - Thread-24"
Обработано сообщение № 31!
    JMSMessageID: ID:74-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399488812
Получено сообщение: "отослано сообщение № 19 от Sat Jul 14 11:51:29 MSD 2007 потоком - Thread-0"
Обработано сообщение № 32!
    JMSMessageID: ID:75-127.0.0.1(fc:b4:e3:e0:ad:e5)-3642-1184399489421
Получено сообщение: "отослано сообщение № 14 от Sat Jul 14 11:51:29 MSD 2007 потоком - Thread-24"
Обработано сообщение № 33!
    JMSMessageID: ID:76-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399489812
Получено сообщение: "отослано сообщение № 15 от Sat Jul 14 11:51:30 MSD 2007 потоком - Thread-24"
Обработано сообщение № 34!
    JMSMessageID: ID:79-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399490812
Получено сообщение: "отослано сообщение № 16 от Sat Jul 14 11:51:31 MSD 2007 потоком - Thread-24"
Обработано сообщение № 35!
    JMSMessageID: ID:80-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399491812
Получено сообщение: "отослано сообщение № 17 от Sat Jul 14 11:51:32 MSD 2007 потоком - Thread-24"
Обработано сообщение № 36!
    JMSMessageID: ID:81-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399492828
Получено сообщение: "отослано сообщение № 18 от Sat Jul 14 11:51:33 MSD 2007 потоком - Thread-24"
Обработано сообщение № 37!
    JMSMessageID: ID:82-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399493828
Получено сообщение: "отослано сообщение № 19 от Sat Jul 14 11:51:34 MSD 2007 потоком - Thread-24"
Обработано сообщение № 38!
    JMSMessageID: ID:83-127.0.0.1(8c:e9:37:a4:9:e0)-3652-1184399494843
```

Клиентская консоль:

```
Поток Thread-0 закончил отсылать сообщения в Sat Jul 14 11:51:30 MSD 2007
Поток Thread-24 закончил отсылать сообщения в Sat Jul 14 11:51:35 MSD 2007
```

Пример MDB №2




```
package test;
import java.util.*;
public class Main
{
    public static void main(String[] args) throws Exception
    {
      (new SubscriberThread()).start();
      Thread.sleep(1000);
      (new SubscriberThread()).start();
      Thread.sleep(1000);
      (new SubscriberThread()).start();
      Thread.sleep(3000);
      (new PublisherThread()).start();
    }
}
```

```
package test;
import javax.jms.*;
import javax.naming.*;
public class PublisherThread extends Thread
{
    public void run()
    {
      InitialContext ic;
      Topic t;
      TopicConnectionFactory tcf;
      TopicConnection tc;
      TopicSession ts;
      TopicPublisher tpub;
      try
      {
        ic = new InitialContext();
        t = (Topic) ic.lookup("jms/MDBT");
        tcf = (TopicConnectionFactory) ic.lookup("jms/MDBTFactory");
        tc = tcf.createTopicConnection();
        ts = tc.createTopicSession(false,Session.AUTO_ACKNOWLEDGE);
        tpub = ts.createPublisher(t);
        tpub.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
        TextMessage m = ts.createTextMessage();
        for(int i=1;i<51;i++)
        {
            m.clearBody();
            m.setText("\"Опубликовано сообщение № "+i+" от "+((new java.util.Date()).toString())+"\"");
            tpub.publish(m);
            Thread.sleep(2000);
        }
        tpub.close();
        ts.close();
        tc.close();
      }
      catch(Exception e){System.out.println(e.getMessage());}
    }
}
```

```
package test;
import com.sun.tools.ws.processor.model.java.JavaArrayType;
import javax.jms.*;
import javax.naming.*;
import java.util.*;
public class SubscriberThread extends Thread
{
    InitialContext ic;
    Topic t;
    TopicConnectionFactory tcf;
    TopicConnection tc;
    TopicSession ts;
    TopicSubscriber tsub;
    public void run()
    {
       Random gen = new Random();
       int rand = (gen.nextInt(3)+1)*1000;
       try
       {
        ic = new InitialContext();
        t = (Topic) ic.lookup("jms/MDBT");
        tcf = (TopicConnectionFactory) ic.lookup("jms/MDBTFactory");
        tc = tcf.createTopicConnection();
        ts = tc.createTopicSession(false,Session.AUTO_ACKNOWLEDGE);
        tsub = ts.createSubscriber(t);
        tc.start();
        System.out.println("Поток "+this.getName()+" начал подписку с периодом "+
                                                          String.valueOf(rand)+"!");
        for(int i=0;i<20;i++)
        {
          TextMessage m = (TextMessage)tsub.receive();
          System.out.println("Получено сообщение потоком "+this.getName()+" "+m.getText());
          Thread.sleep(rand);
        }
        System.out.println("Поток "+this.getName()+" закончил подписку!");
        tsub.close();
        ts.close();
        tc.close();
       }
       catch(Exception e){System.out.println(e.getMessage());}
    }
}
```

```
package PacMDB;
import javax.ejb.*;
import javax.jms.*;
@MessageDriven(mappedName = "jms/MDBT", activationConfig =  {
        @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
        @ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "Durable"),
        @ActivationConfigProperty(propertyName = "clientId", propertyValue = "MDBT"),
        @ActivationConfigProperty(propertyName = "subscriptionName", propertyValue = "MDBT")
    })
public class MDBT implements MessageListener
{
    private int count=0;
    public void onMessage(Message message)
    {
        System.out.println("onMessage MDBT! "+String.valueOf(count));
        count++;
    }
}
```

Серверная консоль:

```
onMessage MDBT! 0
onMessage MDBT! 1
onMessage MDBT! 2
onMessage MDBT! 3
onMessage MDBT! 4
onMessage MDBT! 5
onMessage MDBT! 6
onMessage MDBT! 7
onMessage MDBT! 8
onMessage MDBT! 9
onMessage MDBT! 10
onMessage MDBT! 11
onMessage MDBT! 12
onMessage MDBT! 13
onMessage MDBT! 14
onMessage MDBT! 15
onMessage MDBT! 16
onMessage MDBT! 17
onMessage MDBT! 18
onMessage MDBT! 19
onMessage MDBT! 20
onMessage MDBT! 21
onMessage MDBT! 22
```

```
onMessage MDBT! 23
onMessage MDBT! 24
onMessage MDBT! 25
onMessage MDBT! 26
onMessage MDBT! 27
onMessage MDBT! 28
onMessage MDBT! 29
onMessage MDBT! 30
onMessage MDBT! 31
onMessage MDBT! 32
onMessage MDBT! 33
onMessage MDBT! 34
onMessage MDBT! 35
onMessage MDBT! 36
onMessage MDBT! 37
onMessage MDBT! 38
onMessage MDBT! 39
onMessage MDBT! 40
onMessage MDBT! 41
onMessage MDBT! 42
onMessage MDBT! 43
onMessage MDBT! 44
onMessage MDBT! 45
onMessage MDBT! 46
onMessage MDBT! 47
onMessage MDBT! 48
onMessage MDBT! 49
```

Клиентская консоль:

```
Поток Thread-0 начал подписку с периодом 2000!
Поток Thread-5 начал подписку с периодом 1000!
Поток Thread-3 начал подписку с периодом 3000!
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 1 от Fri Jul 13 21:41:00 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 1 от Fri Jul 13 21:41:00 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 1 от Fri Jul 13 21:41:00 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 2 от Fri Jul 13 21:41:02 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 2 от Fri Jul 13 21:41:02 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 2 от Fri Jul 13 21:41:02 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 3 от Fri Jul 13 21:41:04 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 3 от Fri Jul 13 21:41:04 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 4 от Fri Jul 13 21:41:06 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 3 от Fri Jul 13 21:41:04 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 4 от Fri Jul 13 21:41:06 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 5 от Fri Jul 13 21:41:08 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 5 от Fri Jul 13 21:41:08 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 4 от Fri Jul 13 21:41:06 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 6 от Fri Jul 13 21:41:10 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 6 от Fri Jul 13 21:41:10 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 7 от Fri Jul 13 21:41:12 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 5 от Fri Jul 13 21:41:08 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 7 от Fri Jul 13 21:41:12 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 8 от Fri Jul 13 21:41:14 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 8 от Fri Jul 13 21:41:14 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 6 от Fri Jul 13 21:41:10 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 9 от Fri Jul 13 21:41:16 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 9 от Fri Jul 13 21:41:16 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 10 от Fri Jul 13 21:41:18 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 7 от Fri Jul 13 21:41:12 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 10 от Fri Jul 13 21:41:18 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 11 от Fri Jul 13 21:41:20 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 11 от Fri Jul 13 21:41:20 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 8 от Fri Jul 13 21:41:14 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 12 от Fri Jul 13 21:41:22 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 12 от Fri Jul 13 21:41:22 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 13 от Fri Jul 13 21:41:24 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 9 от Fri Jul 13 21:41:16 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 13 от Fri Jul 13 21:41:24 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 14 от Fri Jul 13 21:41:26 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 14 от Fri Jul 13 21:41:26 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 10 от Fri Jul 13 21:41:18 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 15 от Fri Jul 13 21:41:28 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 15 от Fri Jul 13 21:41:28 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 16 от Fri Jul 13 21:41:30 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 11 от Fri Jul 13 21:41:20 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 16 от Fri Jul 13 21:41:30 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 17 от Fri Jul 13 21:41:32 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 17 от Fri Jul 13 21:41:32 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 12 от Fri Jul 13 21:41:22 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 18 от Fri Jul 13 21:41:34 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 18 от Fri Jul 13 21:41:34 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 19 от Fri Jul 13 21:41:36 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 13 от Fri Jul 13 21:41:24 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 19 от Fri Jul 13 21:41:36 MSD 2007"
Получено сообщение потоком Thread-5 "Опубликовано сообщение № 20 от Fri Jul 13 21:41:38 MSD 2007"
Получено сообщение потоком Thread-0 "Опубликовано сообщение № 20 от Fri Jul 13 21:41:38 MSD 2007"
Поток Thread-5 закончил подписку!
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 14 от Fri Jul 13 21:41:26 MSD 2007"
Поток Thread-0 закончил подписку!
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 15 от Fri Jul 13 21:41:28 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 16 от Fri Jul 13 21:41:30 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 17 от Fri Jul 13 21:41:32 MSD 2007"
```

```
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 18 от Fri Jul 13 21:41:34 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 19 от Fri Jul 13 21:41:36 MSD 2007"
Получено сообщение потоком Thread-3 "Опубликовано сообщение № 20 от Fri Jul 13 21:41:38 MSD 2007"
Поток Thread-3 закончил подписку!
```