Методы и средства построения распределенных программных систем с использованием технологии Java

JMS

- Другой подход к построению распределенных приложений: системы построенные на обмене сообщениями
- Что такое JMS?
  - История
  - Область применимости
  - Элементы архитектуры
- Пример использования

Модель, основанная на событиях

- Объекты реагируют на события (например, вызовом методов)
- События могут изменять состояние объектов
- Множество объектов, расположенных на различных узлах могут извещаться о наступлении событий определенного типа
- Используется модель «издатель-подписчик»
  - Объект, генерирующий события, называется издателем - он публикует список типов событий, на которые остальные объекты могут подписаться
  - Объекты, подписывающиеся на события определенного типа называются подписчиками. Они получают уведомление о наступлении события, на которое они подписались

Модель, основанная на событиях

- Связь между компонентами – слабая
  - Компонент может получить событие и преобразовать его к нужному виду
- Не требуется одновременного наличия «на связи» обменивающихся сообщениями компонентов
  - Синхронные и асинхронные механизмы приема и обработки сообщений
  - Сохранение сообщений в буфере, в случае отсутствия получателя
- Возможность как широковещательной рассылки сообщений, так и избирательной

JMS

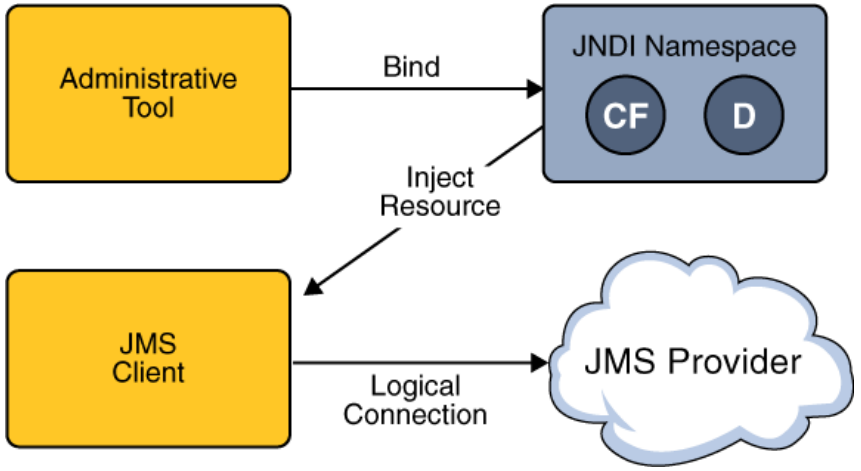
- Реализация событийной модели от Sun
- Входит в состав J2EE
- Обеспечивает механизм обмена сообщениями:
  - Асинхронный
  - Надежный
- Проверенная временем технология
  - Первая спецификация опубликована в 1998 г.
- Реализуется во многих продуктах сторонних разработчиков

Рекомендуется использовать, если…

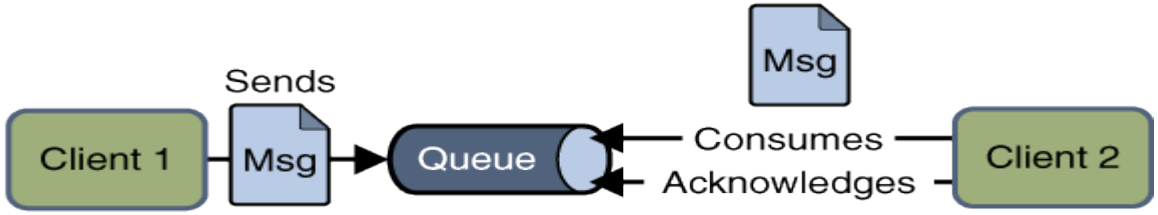
- необходимо обеспечить независимость модуля от интерфейса взаимодействия с другим модулем (rpc и rmi требуют знания интерфейса удаленного модуля)
- компоненты должны работать независимо и могут выполняться в разное (!) время
- структура приложения такова, что информацию можно передать в другой модуль, не дожидаясь ответа и каких-либо результатов обработки

Архитектура

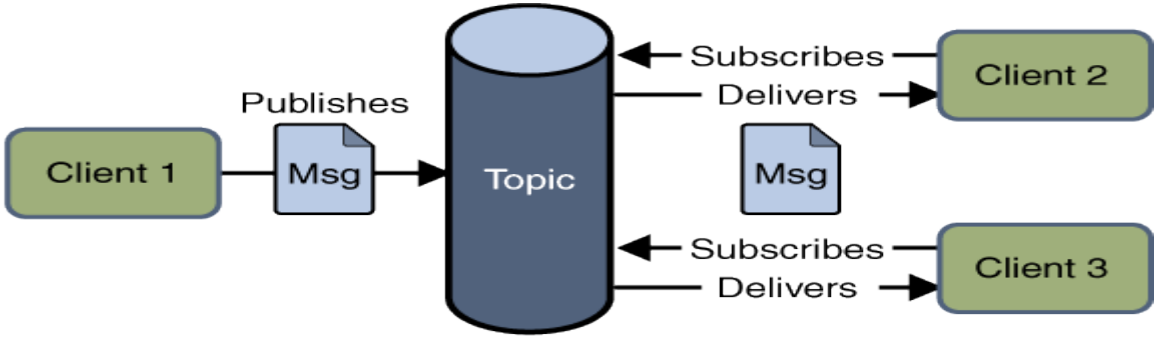
- JMS Provider – система, реализующая интерфейс JMS и предоставляющая средства администрирования
- JMS Clients – компоненты, посылающие и принимающие сообщения
- Messages – сообщения (объекты, передающиеся и принимающиеся компонентами)
- Administrative tools – средства администрирования



JMS – два подхода



- Модель точка-точка для обмена сообщениями между двумя компонентами



- Модель «издатель-подписчик» для обмена сообщениями между многими компонентами

Модель точка-точка

- Область применимости:
  - Взаимодействие двух компонентов
- Элементы модели:
  - Очередь
  - Отправитель
  - Получатель
- Каждое сообщение обрабатывается только одним клиентом
- Зависимости по времени между отправителем и получателем нет
  - Получатель может запуститься и обработать сообщения, когда отправитель работу уже закончил (или нет)

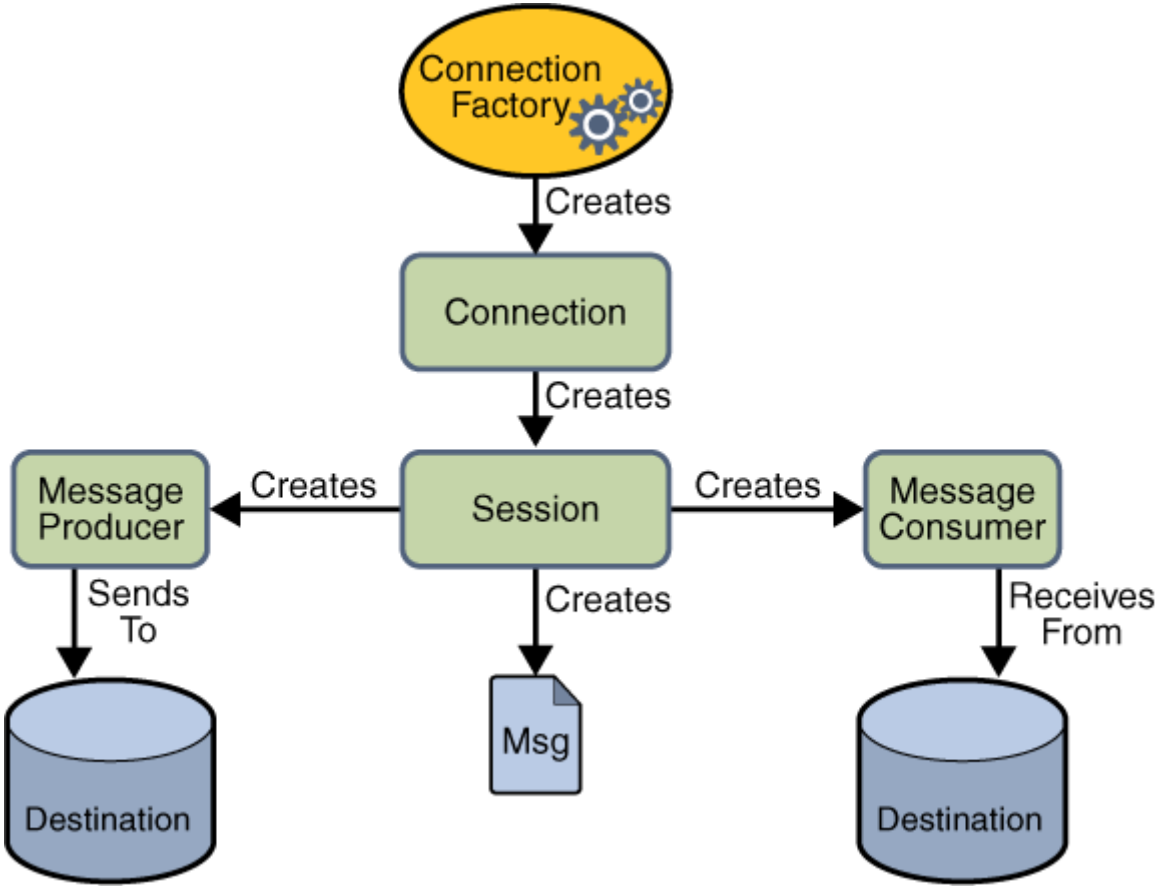
Модель «издатель - подписчики»

- Область применимости:
  - Взаимодействие нескольких компонентов
- Элементы модели:
  - «тема» (topic)
  - Издатель
  - Подписчик
- Сообщение может быть получено несколькими подписчиками
- Подписчик получает только те сообщения, что были отправлены после его подписки на «тему»

Обработка сообщений

- Синхронная
  - Клиент вызывает блокирующий метод receive
- Асинхронная
  - Клиент регистрирует «слушателя» (listener), метод которого вызывается при поступлении сообщения

Схема работы с API JMS



Connection Factory

- Используется для создания соединения с провайдером
- Как правило связывается с определенным ресурсом посредством механизма аннотаций

```
@Resource(mappedName="jms/ConnectionFactory")
private static ConnectionFactory connectionFactory;
```

Destination

- Используется для указания объекта назначения для сообщения
- Поддерживается два типа:
  - Очередь (Queue)
  - Тема (Topic)
- Как правило связывается с определенным ресурсом посредством механизма аннотаций

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

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

Connection

- Инкапсулирует соединение с провайдером
- Используется для создания сессий

```
Connection connection = connectionFactory.createConnection();
```

- После использования соединение должно быть закрыто

```
connection.close();
```

Session

- Сессия представляет собой однопоточный контекст для создания следующих объектов:
  - Отправителей сообщений (Message Producers)
  - Приемщиков сообщений (Message Consumers)
  - Сообщений (Messages)
  - Просмотрщиков очередей (Queue Browsers)
  - Вспомогательных очередей и тем (Temporary queues and topics)
- Создание сессии:

```
Session session = connection.createSession(false,
  Session.AUTO_ACKNOWLEDGE);
```

  - Первый аргумент – поддерживаются транзакции или нет
  - Оповещение об успешной обработке сообщения

Типы сообщений

- TextMessage – текстовое сообщение
- MapMessage – пары «ключ-значение»
- BytesMessage – поток байт
- StreamMessage – поток примитивных типов Java
- ObjectMessage – сериализуемый объект
- Message – пустое сообщение

Схема программы – отправителя сообщений

```
@Resource(mappedName="jms/ConnectionFactory")
private static ConnectionFactory connectionFactory;

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

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

Destination dest = queue;
Connection connection =
  connectionFactory.createConnection();
Session session = connection.createSession(false,
  Session.AUTO_ACKNOWLEDGE);
MessageProducer producer =
        session.createProducer(dest);
TextMessage message = session.createTextMessage();

message.setText("This is message " + (i + 1));
producer.send(message);

connection.close();
```

- Создается объект Connection, с которым связывается ресурс (механизм аннотаций)
- Создается очередь (тема)

- Определяется Destination
- Создается соединение и сессия

- Создается MessageProducer и сообщение (в данном случае - текстовое)
- В тело сообщения записываются текстовые данные и сообщение посылается
- Соединение закрывается

Схема программы – читателя сообщений (синхронный прием)

```
@Resource(mappedName="jms/ConnectionFactory")
private static ConnectionFactory connectionFactory;

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

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

Destination dest = queue;
Connection connection =
  connectionFactory.createConnection();
Session session = connection.createSession(false,
  Session.AUTO_ACKNOWLEDGE);
consumer = session.createConsumer(dest);
connection.start();

 Message m = consumer.receive();
message = (TextMessage) m;
     System.out.println(message.getText());

connection.close();
```

- Создается объект Connection, с которым связывается ресурс (механизм аннотаций)
- Создается очередь (тема)

- Определяется Destination
- Создается соединение и сессия

- Создается MessageConsumer и запускается обработка сообщений
- Читается и печатается сообщение (receive - блокирующий)
- Соединение закрывается

Схема программы – читателя сообщений (асинхронный прием)

```
@Resource(mappedName="jms/ConnectionFactory")
private static ConnectionFactory connectionFactory;

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

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

Destination dest = queue;
Connection connection =
  connectionFactory.createConnection();
Session session = connection.createSession(false,
  Session.AUTO_ACKNOWLEDGE);
consumer = session.createConsumer(dest);

listener = new XXXListener();
consumer.setMessageListener(listener);
connection.start();
```

- Создается объект Connection, с которым связывается ресурс (механизм аннотаций)
- Создается очередь (тема)

- Определяется Destination
- Создается соединение и сессия

- Создается MessageConsumer
- Создается листенер
- Листенер связывается с MessageConsumer и запускается обработка сообщений
- При возникновении сообщения будет вызван метод листенера public void onMessage(Message message)

Схема компиляции и запуска

- Создание ресурсов (очередей или тем)
- Компиляция
  - Программы отправляющей сообщений
  - Программы принимающей сообщения
- Запуск
  - Программы отправляющей сообщений
  - Программы принимающей сообщения

Создание ресурсов и компиляция

- Создание ConnectionFactory

asadmin.bat create-jms-resource --user admin --passwordfile
admin-password.txt --host localhost --port 4848 --restype
javax.jms.ConnectionFactory --enabled=true jms/ConnectionFactory

- Создание очереди

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

- Компиляция

javac -classpath .; activation.jar; appserv-ws.jar; javaee.jar; jsf-impl.jar;
appserv-jstl.jar; -d build *.java

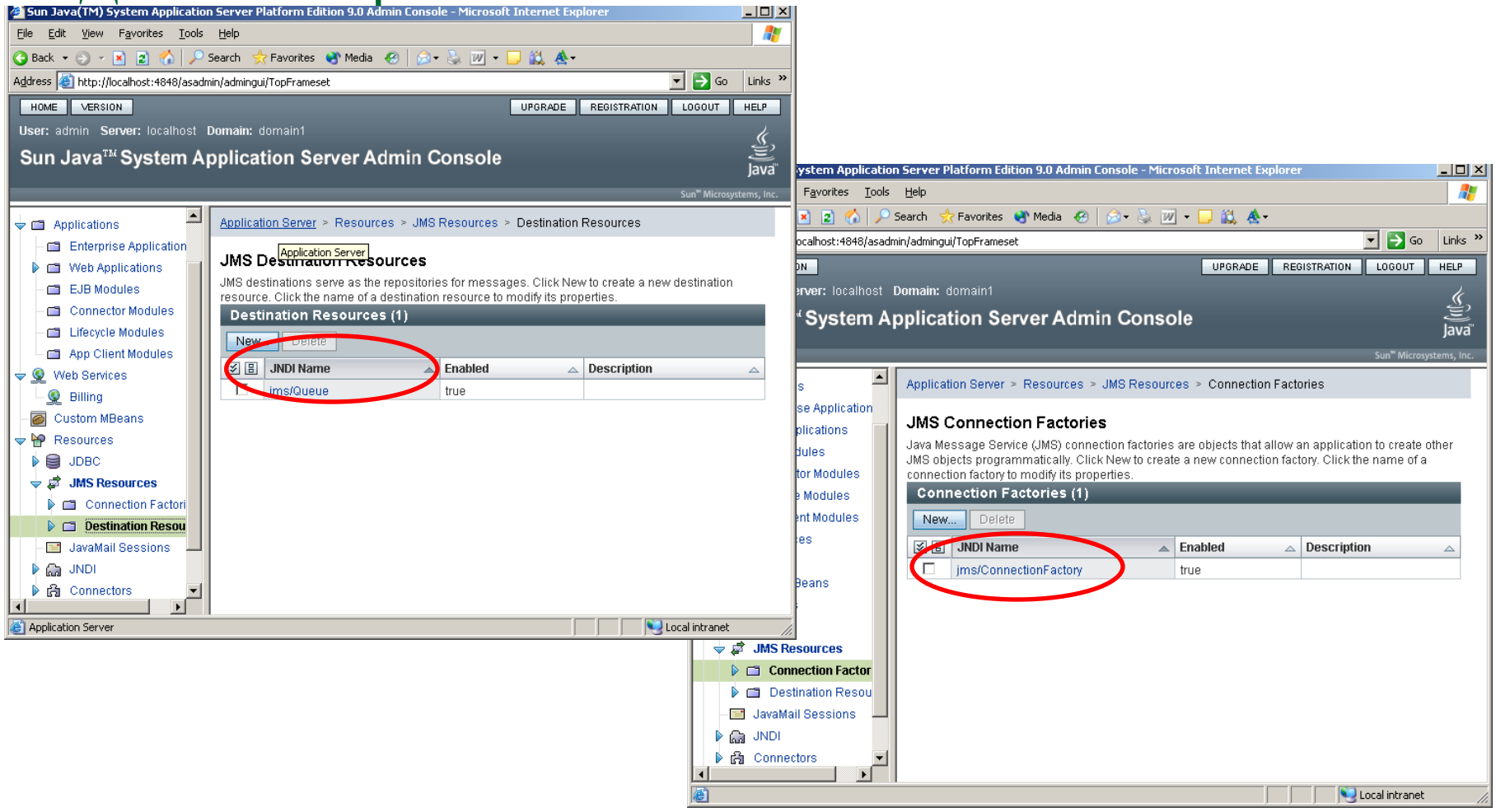
- Создание архива

jar -cvfm cl.jar manifest.mf -C ./build

- Выполнение

appclient -client cl.jar

Просмотр ресурсов в административной консоли



Итоги

- JMS – реализация концепции построения приложений, основанных на обмене событиями
- Существуют различные реализации JMS
- Поддерживает две концепции обмена:
  - Точка-точка
  - Издатель–подписчик
- Поддерживает как синхронный, так и асинхронный обмен сообщениями
- Является удобным механизмом для создания класса систем, нуждающихся в обмене сообщениями