Apache ActiveMQ and Apache ServiceMix

Bruce Snyder
Senior Software Engineer
SpringSource
Agenda

- Message-based integration
  - Apache ActiveMQ
    - Installing ActiveMQ
    - Configuring ActiveMQ
    - Using ActiveMQ with Spring JMS
    - Some ActiveMQ Features
    - Message routing with Apache Camel
  - Apache ServiceMix
    - Enterprise Service Bus
    - Java Business Integration
    - Apache ServiceMix ESB
    - More message routing with Apache Camel
What is Message-Oriented Middleware?
"Software that connects separate systems in a network by carrying and distributing messages between them. MOM infrastructure is typically built around a queuing system that stores messages pending delivery, and keeps track of whether and when each message has been delivered."

(LooselyCoupled.com)
"A client/server infrastructure that increases the interoperability, portability, and flexibility of an application by allowing it to be distributed over multiple heterogeneous platforms. It reduces the complexity of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces.”

(Wikipedia.com)
Message-Oriented Middleware
Enterprise Carrier Pigeon ;-)
What is the Java Message Service?
What is JMS?

"JMS provides a common way for Java programs to create, send, receive and read an enterprise messaging system’s messages."

(JMS spec)
What is ActiveMQ?

- Open source
- Message-oriented middleware
- Apache project
- Apache licensed
- JMS 1.1 compliant
- Goal:
  - To achieve standards-based, message-oriented application integration across many languages and platforms
Apache Software Foundation
Installing ActiveMQ

- Download it
- Unzip it
- Run it

It’s really that simple!
Configuring ActiveMQ

(Spring)

<xml />

(conf/activemq.xml)
Configuring ActiveMQ

- Not only XML configuration
  - Embedding via Java is also very common
    - Lightweight
    - Damn easy to use

```java
BrokerService broker = new BrokerService();
broker.setPersistence(false);
TransportConnector connector = broker.addConnector("tcp://localhost:61616");
broker.start();

... 

connector.stop();
broker.stop();
```
ActiveMQ Uses URIs

<protocol>://<host>:<port>?<transport-options>

vm://embedded?broker.persistent=false

tcp://localhost:61616?jms.useAsyncSend=true

stomp://localhost:61613

failover:(tcp://host1:61616,tcp://host2:61616)?initialReconnectDelay=100
Wire Formats

- **OpenWire**
  - The default in ActiveMQ; a binary protocol
  - Clients for C++, Java and .NET

- **STOMP**
  - Simple Text Oriented Messaging Protocol; a text based protocol
  - Clients for C, Javascript, Perl, PHP, Python, Ruby and more

- **XMPP**
  - The Jabber XML protocol

- **REST**
  - HTTP POST and GET

- **AMQP**
  - Not yet fully supported
Two Types of Transports

- Client-to-broker communications

- Broker-to-broker communications
Transport Connectors

- Client-to-broker connections
  - Similar to JDBC connections to a database

- Protocols are supported:
  - TCP
  - UDP
  - NIO
  - SSL
  - HTTP/S
  - VM
  - XMPP
Network Connectors

- Broker-to-broker connections
  - A cluster of ActiveMQ instances
    - Known as a network of brokers

Protocols supported:
- Static
- Failover
- Multicast
- Zeroconf
- Peer
- Fanout
- Discovery
Message Persistence

- AMQ Store
- JDBC
- Journaled JDBC
AMQ Message Store

- Transactional message storage solution
- Fast and reliable
- Composed of two parts:
  - Data Store - holds messages in a transactional journal
  - Reference store - stores message locations for fast retrieval
- The default message store in ActiveMQ 5
Non-Journaled JDBC

- Transactional message storage solution
- Reliable but not fast
  - JDBC connection overhead is prohibitively slow
Journaled JDBC

- Transactional message storage solution
- Reliable and faster than non-journaled
- Two-piece store
  - Journal - A high-performance, transactional journal
  - Database - A relational database of your choice
- Default persistence in ActiveMQ 4.x
  - Default database in ActiveMQ 4.x was Apache Derby
Somewhat Related -

- Messages are no longer stored in memory
  - Previous to 5.1, message references were stored in memory
- Messages are paged in from storage when space is available in memory
- Lots of configurability
Master/Slave Configurations

- Pure master/slave
- Shared filesystem master/slave
- JDBC master/slave
Pure Master/Slave

- Shared nothing, fully replicated topology
  - Does not depend on shared filesystem or database

- A Slave broker consumes all commands from the master broker (messages, acks, tx states)

- Slave does not start any networking or transport connectors

- Master broker will only respond to client when a message exchange has been successfully passed to the slave broker
Pure Master/Slave

- If the master fails, the slave optionally has two modes of operation:
  - Start up all its network and transport connectors
    - All clients connected to failed Master resume on Slave
  - Close down completely
    - Slave is simply used to duplicate state from Master
Shared Filesystem Master/

- Utilizes a directory on a shared filesystem
- No restriction on number of brokers
- Simple configuration (point to the data dir)
- One master selected at random
JDBC Master/Slave

- Recommended when using a shared database
- No restriction on the number of brokers
- Simple configuration
- Clustered database negates single point of failure
- One master selected at random
Clients should use failover transport for auto-reconnect:

```java
failover:(tcp://broker1:61616, \n tcp://broker2:61616, \n tcp://broker3:61616)? \n initialReconnectDelay=100
```
Tips for HA and Fault

- RAIDed disks
- A Storage Area Network
- Clustered relational databases
- Clustered JDBC via C-JDBC
  - [http://sequoia.continuent.org/](http://sequoia.continuent.org/)
Broker Security

- **Authentication**
  - I.e., are you allowed to connect to ActiveMQ?
  - File based implementation
  - JAAS based implementation

- **Authorization**
  - I.e., do you have permission to use that ActiveMQ resource?
  - Destination level
  - Message level via custom plugin

- **Additional Security Plugins**
  - [http://ttmsolutions.com/](http://ttmsolutions.com/)
Many distributed ActiveMQ brokers working together
Provides large scalability by clustering brokers
ActiveMQ store-and-forward allows messages to traverse brokers in the network
   Demand-based forwarding
   Some people call this distributed queues
Many possible configurations or topologies are supported
Topology Example

Broker ——— Broker ——— Broker ——— Broker
Topology Example
Topology Example
Topology Example
Using ActiveMQ

Three options:

1. DIY
2. EJB Message-Driven Beans
3. JMS Message-Driven POJOs
Do It Yourself (DIY)

Advantages
- Do whatever you want - it’s a green field!

Disadvantages
- Manual creation of MessageProducers and MessageConsumers
- Manual concurrency
- Manual thread management
- Manual transaction management
- Manual resource management
  - ConnectionFactory, Connections, Destinations
EJB: Message-Driven Beans

- Advantages
  - Automatic Transaction management
  - Automatic Concurrency
  - Automatic resource management
    - ConnectionFactory, Connections, Destinations

- Disadvantages
  - Requires EJB container and therefore a JEE server
  - Exception: Apache OpenEJB (http://openejb.apache.org/)
  - Increased overhead
Spring JMS

Advantages

- No EJB container required (no JEE container)
- Simplified resource management
  - ConnectionFactory, Connections, Destinations
- Simplified concurrency management
- Simplified transaction management

Disadvantages

- Are there any? ;-)
The Spring JmsTemplate

- **browse()**
  - Browse messages in a queue
- **convertAndSend()**
  - Send messages synchronously
  - Convert a Java object to a JMS message
- **execute()**
  - Provides access to callbacks for more complex scenarios
- **receive() and receiveAndConvert()**
  - Receive messages synchronously
- **receiveSelected() and receiveSelectedAndConvert()**
  - Receive filtered messages synchronously
- **send()**
  - Send a message synchronously using a MessageCreator
Sending Messages

- Using convertAndSend()

```java
JmsTemplate jmsTemplate = (JmsTemplate) context.getBean("jmsTemplate");

jmsTemplate.convertAndSend("Hello World!");
```
Sending Messages

- Using send() with a MessageCreator
  - Provides access to Session for more complex message creation

```java
Destination destination = (Destination) context.getBean("queue");
JmsTemplate jmsTemplate = (JmsTemplate) context.getBean("jmsTemplate");

jmsTemplate.send(destination, new MessageCreator() {
    public Message createMessage(Session session) throws JMException {
        return session.createTextMessage("Hello World!");
    }
});
```
Using `execute()` the `SessionCallback` provides access to the Session for flexibility.
Using `execute()` with the `ProducerCallback`

Provides access to the `Session` and the `MessageProducer` for more complex scenarios

```java
JmsTemplate jmsTemplate = (JmsTemplate) context.getBean("jmsTemplate");

jmsTemplate.execute(new ProducerCallback() {
    public Object doInJms(Session session, MessageProducer producer) throws JMSException {
        Message message = session.createTextMessage("Hello World!");
        producer.send(destination, message);
        return null;
    }
});
```
Receiving Messages

Using receiveAndConvert()

```java
Destination destination = (Destination) context.getBean("queue");
JmsTemplate jmsTemplate = (JmsTemplate) context.getBean("jmsTemplate");

jmsTemplate.receiveAndConvert(destination, new MessageCreator() {
    public Message createMessage(Session session) throws JMSException {
        return session.createTextMessage("Hello World!");
    }
});
```
Spring Message-Driven POJOs

- **SimpleMessageListenerContainer**
  - Very basic
  - No dynamic resizing offered

- **DefaultMessageListenerContainer**
  - Most common
  - Allows for dynamic resizing
  - Can participate in transactions

- **ServerSessionMessageListenerContainer**
  - Most powerful
  - Less commonly used
  - Relies upon the ServerSessionPool support from the JMS provider
Spring Message-Driven POJOs

Three types

- javax.jms.MessageListener interface
  - Standard JEE interface
  - Threading is up to you
- SessionAwareMessageListener interface
  - Spring-specific interface
  - Provides access to the Session object
    - Useful for request-response messaging
  - Client must handle exceptions
- MessageListenerAdapter interface
  - Spring-specific interface
  - Allows for type-specific message handling
  - No JMS dependencies whatsoever

Asynchronous
Receiving Messages

(Spring 2.5+)

```xml
<bean id="amqConnectionFactory"
     class="org.apache.activemq.spring.ActiveMQConnectionFactory">
    <property name="brokerURL" value="tcp://localhost:61616" />
</bean>

<bean id="messageReceiver" class="com.mycompany.MessageReceiver" />

<jms:listener-container connection-factory="connectionFactory"
                          concurrency="3">
    <jms:listener id="myMessageListener"
                  ref="messageReceiver"
                  destination="MY.TEST.QUEUE"
                  method="processMessage" />
</jms:listener-container>

Asynchronous
public class MessageReceiver {

    public void processMessage(String message) {
        System.out.println("Received a String message: " + message);
    }

    public void processMessage(Map<String, Object> message) {
        System.out.println("Received a Map message: " + message);
    }
}

Asynchronous
Consumer Options

- Message prefetch
- Exclusive consumer
- Consumer priority
- Message groups
- Redelivery policies
- Retroactive consumer
- Selectors
- Some slow consumer strategies
Message Prefetch

- Used for slow consumer situations
- Prevents flooding the consumer
- FIFO buffer on the consumer side
<bean id="amqConnectionFactory"
    class="org.apache.activemq.spring.ActiveMQConnectionFactory">
  <property name="brokerURL"
    value="tcp://localhost:61616?jms.prefetchPolicy.queuePrefetch=1" />
</bean>

<bean id="messageReceiver" class="com.mycompany.jms.MessageReceiver" />

<jms:listener-container connection-factory="connectionFactory"
    concurrency="3">
  <jms:listener id="myMessageListener"
    ref="messageReceiver"
    destination="MY.TEST.QUEUE"
    method="processMessage" />
</jms:listener-container>
Exclusive Consumer

- Anytime more than one consumer is consuming from a queue, message order is lost
- Allows a single consumer to consume all messages on a queue to maintain message ordering
Exclusive Consumer

```xml
<bean id="amqConnectionFactory"
     class="org.apache.activemq.spring.ActiveMQConnectionFactory">
  <property name="brokerURL"
            value="tcp://localhost:61616?jms.prefetchPolicy.queuePrefetch=1"/>
</bean>

<bean id="messageReceiver" class="com.mycompany.jms.MessageReceiver"/>

<bean id="queue" class="org.apache.activemq.command.ActiveMQQueue">
  <constructor-arg value="TEST.FOO?consumer.prefetchSize=10"/>
</bean>

<jms:listener-container connection-factory="ConnectionFactory"
                         concurrency="3">
  <jms:listener id="myMessageListener"
                ref="messageReceiver"
                destination="MY.TEST.QUEUE"
                method="processMessage"/>
</j/ms:listener-container>
```
Consumer Priority

- Gives a consumer preference for message delivery
- Allows for the weighting of consumers to optimize network traversal for message delivery
Consumer Priority

```xml
<bean id="amqConnectionFactory"
    class="org.apache.activemq.spring.ActiveMQConnectionFactory">
    <property name="brokerURL"
        value="tcp://localhost:61616?jms.prefetchPolicy.queuePrefetch=1"/>
</bean>

<bean id="messageReceiver" class="com.mycompany.jms.MessageReceiver"/>

<bean id="queue" class="org.apache.activemq.command.ActiveMQQueue">
    <constructor-arg value="TEST.FOO?consumer.priority=10"/>
</bean>

<jms:listener-container connection-factory="ConnectionFactory"
    concurrency="3">
    <jms:listener id="myMessageListener"
        ref="messageReceiver"
        destination="MY.TEST.QUEUE"
        method="processMessage"/>
</jms:listener-container>
```
Message Groups

- Uses the JMSXGroupID property to define a group
- One consumer receives all messages in the group
- Allows one consumer to handle related messages
Message Groups

Set programmatically:

```java
JmsTemplate jmsTemplate = (JmsTemplate) context.getBean("jmsTemplate");

jmsTemplate.execute(new SessionCallback() {
    public Object doInJms(Session session) throws JMSException {
        Queue queue = session.createQueue("MY.TEST.QUEUE");
        MessageProducer producer = session.createProducer(queue);
        Message message = session.createBytesMessage();
        message.writeObject(someObject);
        message.setStringProperty("JMSXGroupID", "COLLATERAL_TRADE_X");
        producer.send(message);
    }
});
```
Redelivery Policy

Messages are redelivered to a client when:
- A transacted session is rolled back
- A transacted session is closed before commit
- A session is using CLIENT_ACKNOWLEDGE and Session.recover() is explicitly called

Clients can override the redelivery policy
- Dead Letter Strategy can also be configured in activemq.xml
Clients can override the redelivery policy programatically:

```java
RedeliveryPolicy policy = connectionFactory.getRedeliveryPolicy();
policy.setInitialRedeliveryDelay(500);
policy.setBackOffMultiplier((short) 2);
policy.setUseExponentialBackOff(true);
policy.setMaximumRedeliveries(4);
...```
Redelivery Policy

Also configurable via activemq.xml:

```
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic=">">
        <!-- 1 minutes worth -->
        <subscriptionRecoveryPolicy>
          <timedSubscriptionRecoveryPolicy recoverDuration="60000"/>
        </subscriptionRecoveryPolicy>
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>
```
Retroactive Consumer

- Message replay at start of a subscription
  - At the start of every subscription, send any old messages that the consumer may have missed
- Configurable via recovery policies
Retroactive Consumer

- **FixedSizedSubscriptionRecoveryPolicy**
  - Keep a fixed amount of memory in RAM for message history which is evicted in time order

- **FixedCountSubscriptionRecoveryPolicy**
  - Keep a fixed count of last messages

- **LastImageSubscriptionRecoveryPolicy**
  - Keep only the last message

- **NoSubscriptionRecoveryPolicy**
  - Disables message recovery

- **QueryBasedSubscriptionRecoveryPolicy**
  - Perform a user specific query to load any message they may have missed (uses JMS selectors)

- **TimedSubscriptionRecoveryPolicy**
  - Keep a timed buffer (in ms) of messages around in memory and use that to recover new subscriptions
Redelivery Policy

Give each queue it’s own DLQ:

```xml
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic="">
        <deadLetterStrategy>
          <individualDeadLetterStrategy queuePrefix="DLQ."
          useQueueForQueueMessages="true" />
        </deadLetterStrategy>
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>
```

* Configured in conf/activemq.xml
Message Selectors

- Used to attach a filter to a subscription

- **JMS selectors**
  - Defined using a subset SQL 92 syntax
  - Filters only message properties
    - JMSType = ‘stock’ and trader = ‘bob’ and price < ‘105’

- **XPath selectors**
  - Defined using XPath queries
  - Filters message bodies that contain XML
    - ‘/message/cheese/text() = 'swiss'’
Other Handy Features

- Destination Policies
- Virtual Destinations
- Total Ordering of Messages
- Mirrored Queues

* Configured in conf/activemq.xml
Wildcards on Destination Policies

```
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic="Price.Stock.">
        memoryLimit="128mb">
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>
```

- Price.>
- Price.Stock.>
- Price.Stock.NASDAQ.*
- Price.Stock.*.IBM

> - Everything recursively
* - Everything at that level
Virtual Destinations
Total Ordering

- A guaranteed order of messages for each consumer
Mirrored Queues
What is Apache Camel?

Java API for message routing

```java
package com.mycompany.routes;

public class MyRoute extends RouteBuilder {
    public void configure() {
        from("activemq:TEST.QUEUE").
            to("file:///opt/inbox/text.txt").
            to("log:MyLog?showProperties=true");
    }
}
```
XML flavor as well

```xml
<camelContext id="camel"
    xmlns="http://activemq.apache.org/camel/schema/spring">
    <package>com.mycompany</package>
    <route>
        <from uri="activemq:example.A" />
        <to uri="file:///opt/inbox/text.txt" />
        <to uri="log:MyLog?showProperties=true" />
    </route>
</camelContext>
```
RouteBuilder simpleChoiceRoute = new RouteBuilder() {
    public void configure() {
        from("activemq:QUEUE.A").choice().
            when(header("foo").isEqualTo("bar")).
                to("file:/opt/inbox").
            when(header("foo").isEqualTo("cheese")).
                to("jbi:service:http://com/mycompany/MyService").
            otherwise().
                to("file:/opt/outbox-foo");
    }
};
<camelContext id="simpleChoiceRoute">
  <route>
    <from uri="activemq:QUEUE.A" />
    <choice>
      <when>
        <predicate>
          <header name="foo" />
          <isEqualTo value="bar" />
        </predicate>
        <to uri="file:/opt/inbox" />
      </when>
      <when>
        <predicate>
          <header name="foo" />
          <isEqualTo value="cheese" />
        </predicate>
        <to uri="jbi:service:http://com/mycompany/MyService" />
      </when>
      <otherwise>
        <to uri="file:/opt/outbox-foo" />
      </otherwise>
    </choice>
  </route>
</camelContext>
Coming Soon: ActiveMQ in Action
Let’s Take a Break

Up Next ...

Apache ServiceMix
Integration is Messy!
INTEGRATION

Just because you can, doesn’t mean you should.
What is an ESB?
"An Enterprise Service Bus (ESB) is a new architecture that exploits Web services, messaging middleware, intelligent routing, and transformation. ESBs act as a lightweight, ubiquitous integration backbone through which software services and application components flow."

(Gartner)
What is an ESB?

An ESB acts as a shared messaging layer for connecting applications and other services throughout an enterprise computing infrastructure. It supplements its core asynchronous messaging backbone with intelligent transformation and routing to ensure messages are passed reliably. Services participate in the ESB using either web services messaging standards or JMS (LooselyCoupled.com)
What is an ESB?

An ESB is an open standards, message-based, distributed, integration solution that provides routing, invocation, and mediation services to facilitate the interactions of disparate distributed information technology resources (applications, services, information, platforms) in a reliable manner.

(Brenda Michelson, Elemental Links)
Do I need an ESB?

- Article at IBM developerWorks
  - Written by Bobby Woolf
  - General idea
    - Don’t adopt an ESB without planning out the services first

**ESB-oriented architecture:**

_The wrong approach to adopting SOA_

Use a Planning Process

ESB Planning Process

1. Your business purpose
2. Your functional requirements
3. Your architectural decisions
4. Your criteria for evaluating ESBs
What is JBI?
What is JBI?

*JBI defines an architecture that allows the construction of integration systems from plug-in components, that interoperate through the method of mediated message exchange.*

*(JBI 1.0 Spec)*
Java Business Integration

Normalized Message Router

JBI Environment
Java Business Integration

JBI Environment

- Service Engine
- Service Engine
- Service Engine
- Service Engine

Standardized interfaces for Service Engines

Normalized Message Router

Standardized interfaces for Binding Components

- Binding Component
- Binding Component
- Binding Component
- Binding Component

External Service Provider
External Service Consumer

Local providers/consumers
Protocol handlers
Remote providers/consumers

JMX-based mgmt app

- Installation
- Deployment
- Control
- Monitoring
JBI Normalized Message
JBI Packaging

Service Assembly (.jar/.zip)

Service Assembly:
1) META-INF/jbi.xml

Service Unit (.jar/.zip)

Service Unit:
1) META-INF/jbi.xml
2) dependencies
Apache ServiceMix

http://servicemix.apache.org/
Apache ServiceMix Architecture
ServiceMix Features

- Supports many protocols
  - File, FTP, HTTP/S, JMS, SMTP, SOAP, TCP, XMPP

- Supports many engines
  - Apache Camel, Apache CXF, Apache ODE, Drools, OS Workflow, POJOs, Quartz, Scripting, Saxon XQuery and XSLT, WS-Notification

- Supports Security
  - JAAS, WS-Security

- Web Container/App Server Integration
  - Geronimo, JBoss, Jetty, Tomcat, Weblogic, Websphere
Apache Software Foundation
Message Routing
Hello World with ServiceMix
Involves configuring existing components using XML
<beans xmlns:file='http://servicemix.apache.org/file/1.0'
       xmlns:test='urn:test'>

<file:poller service="test:file"
             endpoint="endpoint"
             targetService="test:wiretapIn"
             file="/Users/bsnyder/smxdropbox"/>

</beans>
The Wiretap

```xml
<beans xmlns:eip="http://servicemix.apache.org/eip/1.0"
       xmlns:test="urn:test">

  <eip:wire-tap service="test:wiretapIn" endpoint="endpoint">
    <eip:target>
      <eip:exchange-target service="test:cbr"/>
    </eip:target>
    <eip:inListener>
      <eip:exchange-target service="test:logger"/>
    </eip:inListener>
  </eip:wire-tap>

</beans>
```
public class MyLogger extends RouteBuilder {
    public void configure() {
        from("jbi:service:urn:test:logger").
            process(new Processor() {
                public void process(Exchange exchange) {
                    Message in = exchange.getIn();
                    in.setBody(in.getBody(String.class) +
                            "<foo>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</foo>");
                }
            }).
            to("log:Devoxx-DEMO");
    }
}
The Logger: Camel Config

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                          http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
                          http://activemq.apache.org/camel/schema/spring
                          http://activemq.apache.org/camel/schema/spring/camel-spring.xsd">

  <camelContext id="camel"
                xmlns="http://activemq.apache.org/camel/schema/spring">
    <package>com.mycompany</package>
  </camelContext>

</beans>
```
<beans xmlns:eip="http://servicemix.apache.org/eip/1.0"
      xmlns:test="urn:test">
  <eip:content-based-router service="test:cbr"
      endpoint="endpoint">
    <eip:rules>
      <eip:routing-rule>
        <eip:predicate>
          <eip:xpath-predicate
              xpath="/message/cheese/text() = 'gouda'" />
        </eip:predicate>
        <eip:target>
          <eip:exchange-target service="test:queue1" />
        </eip:target>
      </eip:routing-rule>
    </eip:rules>
  </eip:content-based-router>
...
The Content-Based Router

...<eip:routing-rule>
  <eip:predicate>
    <eip:xpath-predicate>
      xpath="/message/cheese/text() = 'swiss'" />
    </eip:xpath-predicate>
  </eip:predicate>
  <eip:target>
    <eip:exchange-target service="test:queue2" />
  </eip:target>
</eip:routing-rule>
</eip:routing-rule>
</eip:rules>
</eip:content-based-router>
The JMS Sender

```xml
<beans xmlns:jms="http://servicemix.apache.org/jms/1.0"
    xmlns:test="urn:test"
    xmlns:amq="http://activemq.org/config/1.0">

    <jms:endpoint service="test:queue1"
        endpoint="myProvider"
        role="provider"
        destinationStyle="queue"
        jmsProviderDestinationName="queue1"
        connectionFactory="#connectionFactory" />

    <jms:endpoint service="test:queue2"
        endpoint="myProvider"
        role="provider"
        destinationStyle="queue"
        jmsProviderDestinationName="queue2"
        connectionFactory="#connectionFactory" />

</beans>
```
The JMS Sender

...<jms:endpoint service="test:queue3"
    endpoint="myProvider"
    role="provider"
    destinationStyle="queue"
    jmsProviderDestinationName="queue3"
    connectionFactory="#connectionFactory"/>

<amq:connectionFactory id="connectionFactory"
    brokerURL="tcp://localhost:61616" />

</beans>
Hello World with ServiceMix
JBI Packaging

Service Assembly (.jar/.zip)

Service Assembly:
1) Maven project (pom.xml)
2) Included Maven projects (pom.xml)

Service Unit (.jar/.zip)

Service Unit:
1) Maven project (pom.xml)
2) Spring XML (xbean.xml)
Distributed ServiceMix Containers
Eclipse IDE Tooling For ServiceMix

- Eclipse SOA Tooling Platform (STP) Project
  - [http://eclipse.org/stp](http://eclipse.org/stp)

- Spagic
  - [http://spagic.com/](http://spagic.com/)

- Sopera
Eclipse Tooling

Eclipse SOA Tools Platform (STP) Project
Apache ServiceMix 4.0
Lessons Learned From JBI 1.0

✓ Normalized Exchange and Messages
✓ Normalized Message Router
  ✓ The only way to plug in third party components

- XML normalization can be problematic
- Packaging/classloaders not very flexible
- Creating components is not easy
- Not always the best fit for a given API
ServiceMix 4.0 Building Blocks

- Runtime: OSGi (Apache Felix)
  - JBI support still intact
  - NMR is an OSGi service
- Message Broker: Apache ActiveMQ
- Web Services: Apache CXF
- Routing Engine: Apache Camel
ServiceMix 4 Architecture

NMR
- JBI 1.0
- JAX-WS
- Camel

NMR
- ActiveMQ
- Transaction Manager

Kernel
- Console
- Logging
- Deployer
- Provisionning
- Admin
- Spring DM

OSGI
ServiceMix 4 Architecture

- Spring
- Web
- REST / JSR 311
- EJB
- OSGi
- JBI 1.0
- JBI 2.0
- Camel
- JAX-WS
Do You Have Information Overload Yet?
Thank You!

Questions?