Tutorial JMS com ActiveMQ

Nível: Introdutório
Autor: José Damico (jdamico@br.ibm.com) IBM
Data: 26 de novembro de 2006

O objetivo deste tutorial é oferecer uma documentação básica passo-a-passo e um exemplo prático de aplicação para que o programador possa instalar um *broker JMS* e aprenda a usá-lo corretamente. Assume-se que o usuário deste tutorial tenha o conhecimento conceitual sobre a tecnologia JMS. Para aqueles que não possuem tal conhecimento, sugiro que leiam o artigo **Introducing the Java Message Service** (https://www6.software.ibm.com/developerworks/education/j-jms/index.html) e depois voltem a este.

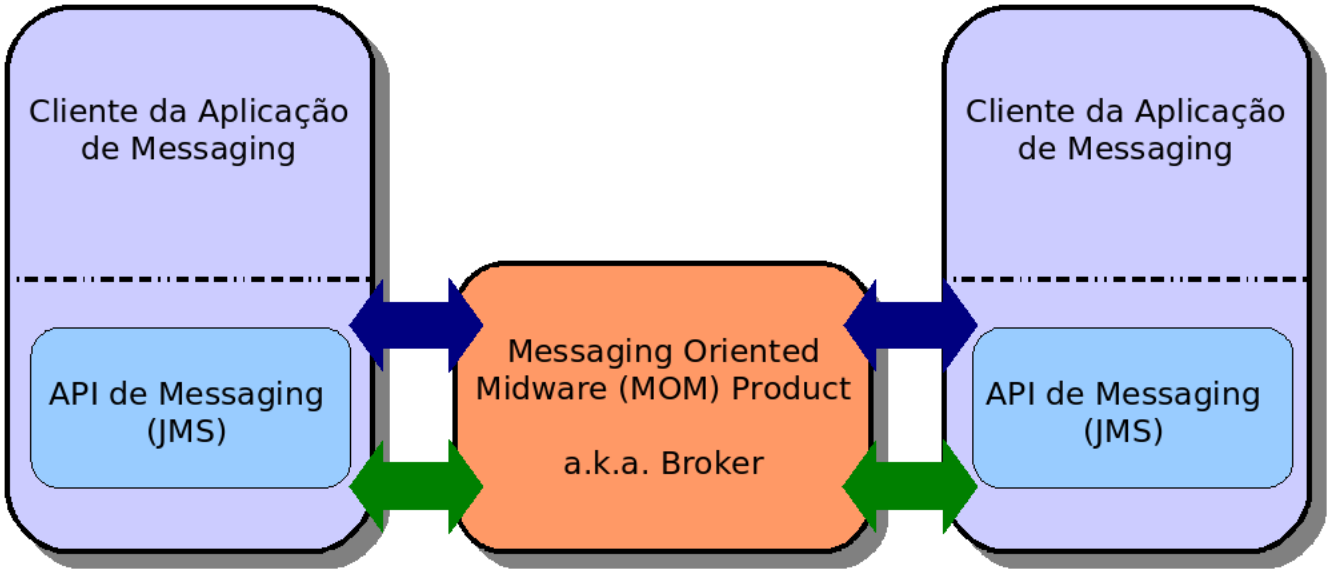
Revisão:

Antes de tudo é importante reforçar que os conceitos tratados aqui fazem parte de uma área maior, a computação distribuída e de que a metodologia de programação com suporte a mensagens entre sistemas é uma das formas de se garantir maior segurança, estabilidade e confiabilidade em ambientes com múltiplas aplicações que rodam em diferentes máquinas* interligadas via rede sob diferentes situações.

Façamos então uma revisão dos dois principais conceitos usados neste tutorial. São eles JMS e Brokers:

JMS: É uma API criada pela Sun que permite que programas escritos em java possam fazer uso dos recursos de um Broker em conformidade com a especificação JMS.

Broker: Chamamos de Broker um produto baseado no conceito MOM (Message Oriented Middleware) que precede a criação do JMS e consiste basicamente em um sistema intermediário a outras aplicações o qual recebe, envia e redireciona mensagens destas e para estas aplicações de forma a suportar processos assíncronos e distribuídos com maior confiabilidade. Um Broker pode possuir suas próprias interfaces de comunicação com seus clientes e pode também oferecer suporte a API JMS. Brokers possuem comumente em seus nomes a sigla MQ que nada mais é do que Message Queue. Outro detalhe comum é o de que as mensagens que trafegam entre as aplicações também podem ser chamadas de eventos.



> * *Uma abordagem comum e eficaz do uso de JMS e brokers é a de executar processos em background em um mesmo servidor no qual rodam aplicações Web sobre um Application Server. Com o crescimento acelerado dos serviços disponíveis na Internet, é cada vez mais importante desvincular processos longos e com grande movimentação de dados, da execução das páginas dinâmicas que rodam no servidor. São necessários recursos que permitam que o código relacionado com as páginas web apenas iniciem processos e consultem sobre o status dos mesmos, com isso o carregamento de tais páginas poderá ser independente do término do processo iniciado.*

Instalação do ActiveMQ:

> ***Importante:***
> *Todos os passos utilizados neste tutorial foram executados e testados em 3 computadores com sistema operacional Linux (kernel 2.6) conectados em uma mesma rede local sem restrições de comunição sob o protocolo TCP/IP entre elas.*

1. Defina corretamente sua variável JAVA_HOME

2. Faça o download do empacotamento binário para Unix.
   O download pode ser feito do seguinte endereço:
   http://incubator.apache.org/activemq/download.html
   (Durante a escrita deste tutorial a versão disponível para download do ActiveMQ era a 4.0.2)

3. Depois de feito o download, descompacte o arquivo.
   tar zxvf nome-do-arquivo.tar.gz

4. Com o arquivo descompactado entre no seguinte diretório nome-do-arquivo/bin e dê permissão de execução para o arquivo activemq (chmod 755 activemq) e saia do diretório bin

5. Finalmente inicie o ActiveMQ: bin/activemq

6. Para verificar o funcionamento do serviço iniciado use o seguinte comando: ps -ef | grep activemq o resultado deverá ser algo semelhante a isso:
   root 4120 2396 33 19:27 pts/0 00:00:19 /usr/bin/java -Xmx512M -Dorg.apache.activemq.UseDedicatedTaskRunner=true -Dderby.system.home=../data -Dderby.sto rage.fileSyncTransactionLog=true -Dcom .sun.management.jmxremote -classpath (...)

> ***Importante:***
> *Não há necessidade de mudar nenhuma configuração padrão do ActiveMQ para a execução dos testes e códigos deste tutorial*

Aplicativos de Envio e Recebimento de mensagens:

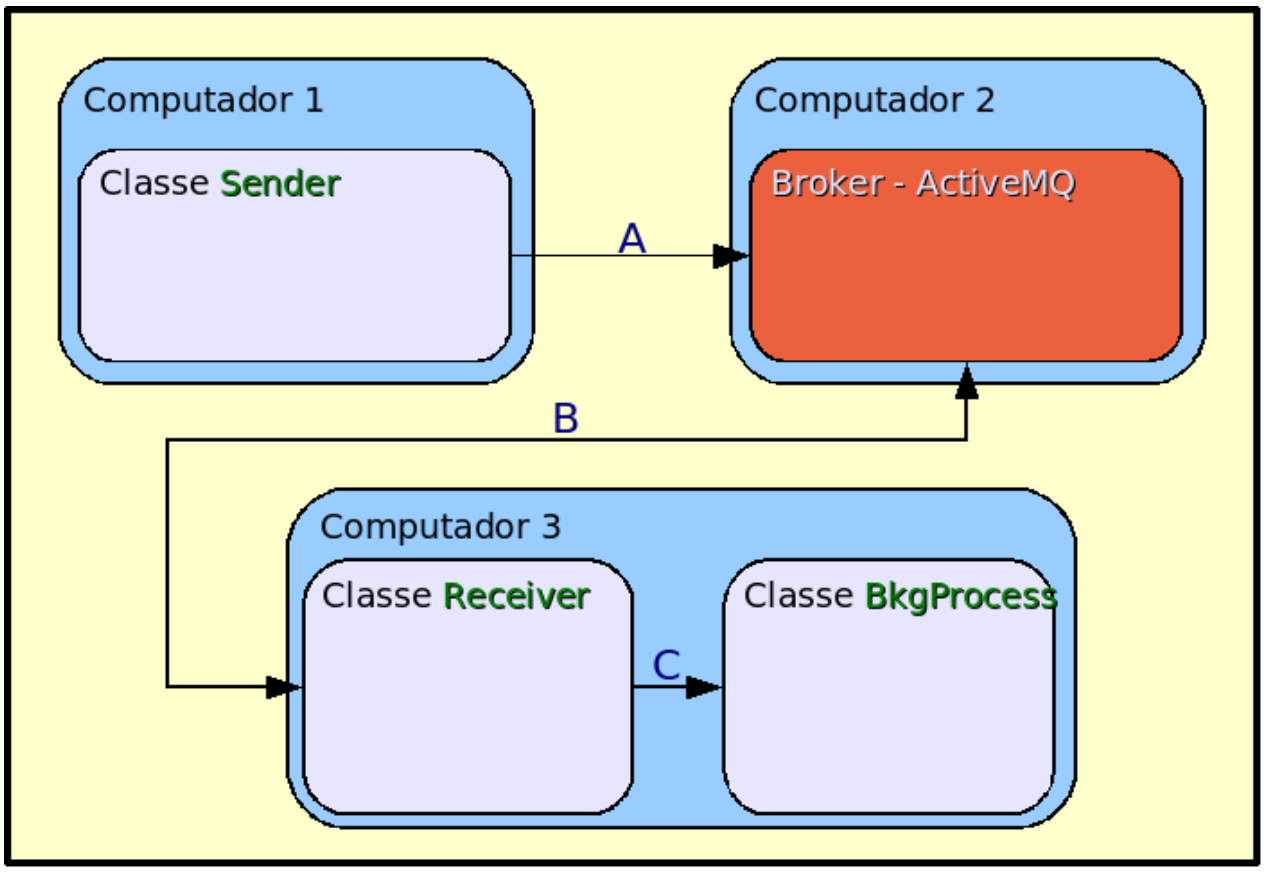
A melhor forma de testar o broker que acabamos de instalar é através de uso de aplicativos funcionais que implementam as características básicas de um sistema JMS.

O que proponho aqui é desenvolver 3 classes distintas que simularão um ambiente distribuído de computação no qual um sistema de mensagens controlará a ativação de processos remotos. Para entender essa proposta precisamos imaginar um cenário de rede com 3 computadores que rodarão estas 3 classes e 1 aplicação distinta:

Computadores	Classes	Descrição
1	Sender	Aplicação cliente com envio de mensagem para requisição de execução de processo em background
2	---	Broker
3	Receiver, BkgProcess	Aplicação de recebimento de mensagem e execução de processo em background

Com este cenário em mente, podemos em 3 (A,B e C) momentos explicar o ciclo de vida deste sistema: No **momento A** o programa cliente, especialmente a classe *Sender* que roda no computador 1 envia uma mensagem com conteúdo texto para uma fila específica de mensagens no computador 2 que possui o Broker instalado. O **momento B** acontece quando a classe *Receiver* é executada no computador 3, esta inicia um loop infinito que busca por novas mensagens em uma fila específica no

Broker. Quando uma nova mensagem é identificada, a mesma é recebida e seu conteúdo verificado. De acordo com o padrão encontrado dentro da mensagem a classe *BkgProcess* é executada iniciando por sua vez um processo em background, este é o **momento C**, que ocorre também no computador 3. O diagrama abaixo exemplifica estas 3 situações.



Uma vez que o fluxo de operações foi entendido podemos prosseguir com a análise do código-fonte do qual os itens mais importantes estão nas classes Sender e Receiver, que na verdade são derivações do exemplo Hello-World contido no código App.java encontrado no site do ActiveMQ (http://incubator.apache.org/activemq/hello-world.html)

Análise das classes *Sender* e *Receiver*

Vamos começar com uma parte comum aos arquivos Sender.java e Receiver.java. O primeiro item que deve ser notado são os imports contidos no código, eles nos remetem a necessidade de termos em nosso classpath os jars da API JMS 1.1 da Sun bem como os jars do ActiveMQ. Ao todo os jars necesários serão:

- javax.jms.jar
- jms.jar

- activeio-core
- activemq-core-4.0.jar
- commons-logging-1.0.4.jar
- backport-util-concurrent-2.1.jar
- geronimo-j2ee-management_1.0_spec-1.0.jar
- geronimo-jms_1.1_spec-1.0.jar
- log4j-1.2.12.jar

Com os imports resolvidos podemos analisar outras partes em comum destas 2 classes:

- Iniciam em seu método main as threads principais referentes a cada classe no caso da Sender.java a thread StartProducer e no caso da Receiver.java a thread StartConsumer além de iniciarem o processo no Broker.

- Possuem a classe estática StartBroker reponsável pela inicialização e teste de conectividade com o Broker.

Depois dos itens em comum, vamos as diferenças:

- Em Sender.java temos a classe estática StartProducer que implementa uma thread do tipo Runnable. Nessa classe temos o método run() que se encarrega de todos os passos para que uma mensagem seja enviada ao Broker, isto é:

    1. Cria um connection Factory
    2. Cria uma conexão
    3. Cria uma sessão
    4. Estabelece um destination do tipo queue
    5. Constrói um MessageProducer a partir da sessão criada para a destination estabelecida
    6. Cria uma mensagem de texto
    7. Envia a mensagem através do MessageProducer construído
    8. Fecha a sessão
    9. Fecha a conexão

- Em Receiver.java a diferença está na classe estática StartConsumer a qual também implementa uma thread Runnable, mas neste caso tem a função, através do método run(), de “consumir” mensagens de um Broker. Este método possui os passos de 1 a 4 iguais ao da classe Sender.java, portanto o que a diferencia está abaixo:

    5. Constrói um MessageConsumer a partir da sessão criada para a destination estabelecida
    6. Espera por mensagens
    7. Executa o método onMessage() assim que uma mensagem é recebida

8. Fecha a sessão
9. Fecha a conexão

- O método onMessage() da classe estática StartConsumer é responsável por filtrar o conteúdo da mensagem recebida (consumida) e de acordo com o que foi filtrado, disparar métodos da classe BkgProcess.

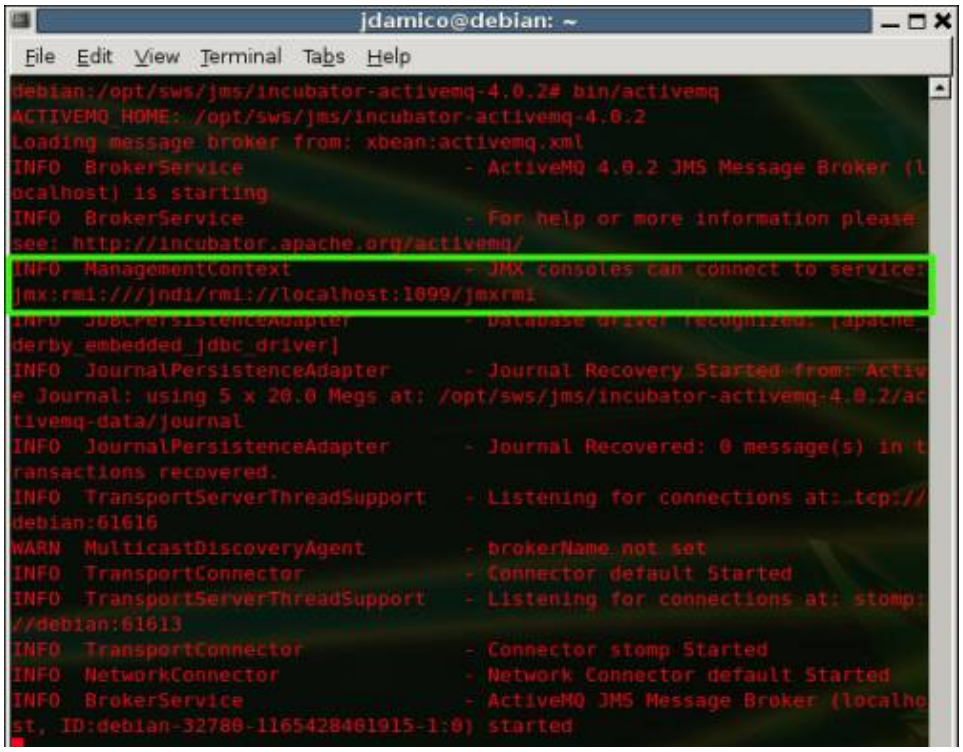
- Ainda na classe estática StartConsumer, podemos notar um último método, onException(), responsável pelo controle de exceptions desta classe.

Análise da classe *BkgProcess*

Bem mais simples que as classes já estudadas, a classe BkgProcess além de seu construtor, possuí 1 método que é responsável pela a execução do processo em background. Neste tutorial, usaremos como exemplo um processo que testa durante 1 minuto a conectividade de um computador em rede.

Monitoramento

Além do desenvolvimento das classes e implementação da API JMS, é de extrema importância saber como monitorar o funcionamento do broker. Para fazer isso no ActiveMQ uma abordagem eficaz é o uso do JMX com o Jconsole. A utilização é simples, basta você saber o endereço JNDI de acesso ao JMX. Essa informação pode ser obtida no momento da inicialização do ActiveMQ, veja o exemplo abaixo:

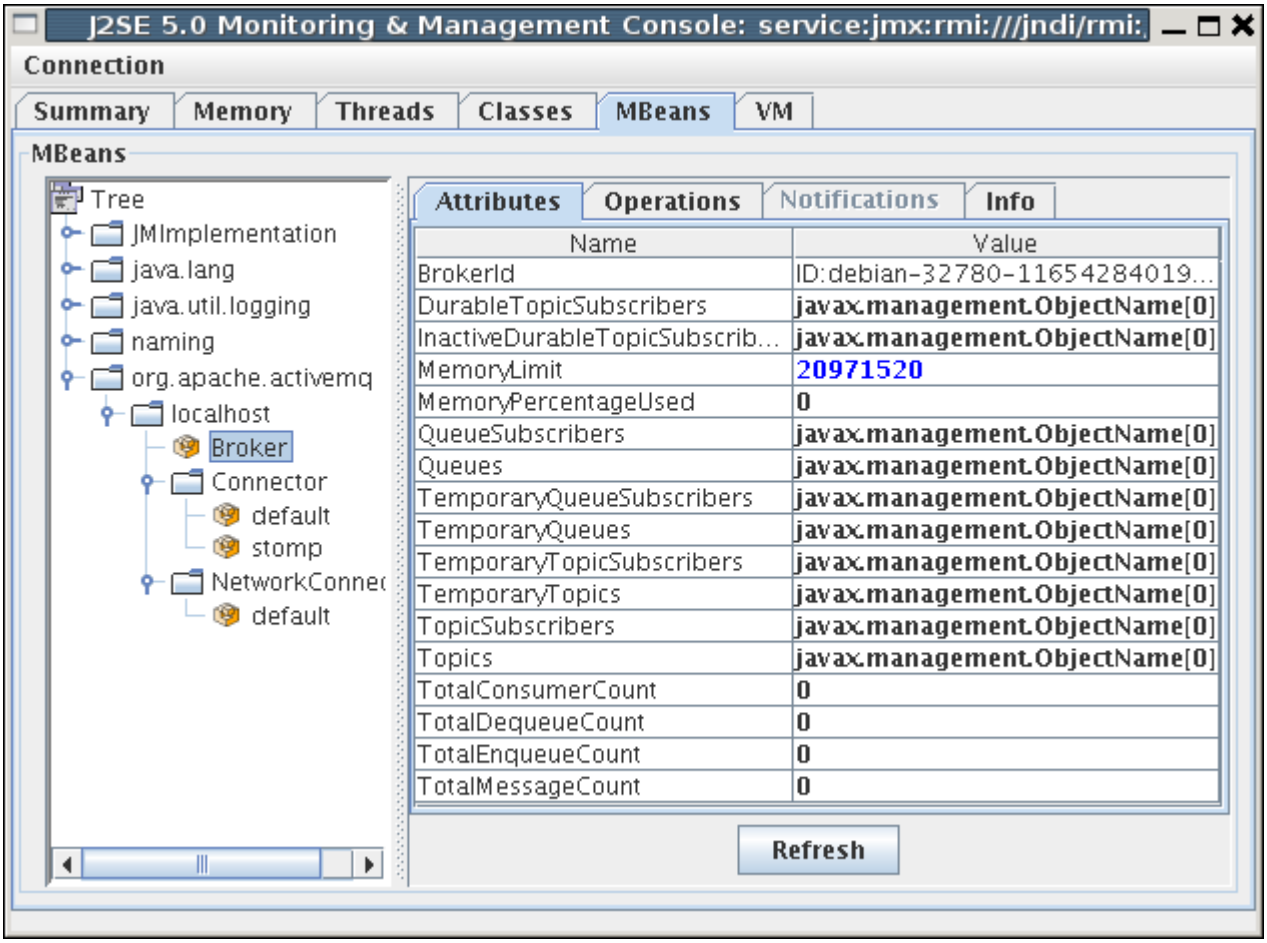


Através dessa imagem podemos observar que o endereço do serviço JMX é *jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi*, dessa forma basta acessar o jconsole com a seguinte sintaxe: *jconsole service:<endereço jmx>*, ou seja:

jconsole service:*jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi*

Porém, se o ActiveMQ estiver instalado em outro servidor, substitua *localhost* pelo endereço do servidor.

O resultado será uma janela com uma série de informações em tempo real sobre a JVM que está rodando no servidor indicado. Para acessar as informações específicas do ActiveMQ, selecione a aba *Mbeans* e depois a opção *org.apache.activemq*, como mostra a imagem abaixo.



Importante:
O jconsole é um aplicativo que está disponível na pasta $JAVA_HOME/bin de sua JDK 1.5

Conclusão

Com este tutorial percorremos de forma prática os passos básicos para a instalação de um broker e a implementação de clientes de envio e recebimento de mensagens em uma fila com o uso da API JMS. Para quem deseja continuar o estudo nessa área o caminho natural a seguir é substituir a fila por tópicos e implementar assinaturas e publicações de mensagens em determinados tópicos e para isso sugiro a leitura do seguinte artigo **JMS Applications with WebSphere Studio V5 -- Part 2: Developing a JMS Publish-Subscribe Application**
(http://www-128.ibm.com/developerworks/websphere/library/techarticles/0307_wilkinson/wilkinson2.html)

Anexos - Códigos-fonte

Sender.java:

```
import java.text.Format;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.activemq.*;
import org.apache.activemq.broker.BrokerService;
import javax.jms.Connection;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;

/**
 * Classe Sender
 */
public class Sender {

      private static String host = "100.100.100.100";
      private static String port = "61616";
      private static String queue_name = "SERVER.TEST";
      private static Format formatter = new
SimpleDateFormat("HH:mm:ss");
      private static String stime = null;

    public static void main(String[] args) throws Exception {

       thread(new StartBroker(), true);
       thread(new StartProducer(), false);
    }

    public static void thread(Runnable runnable, boolean daemon) {
        Thread brokerThread = new Thread(runnable);
        brokerThread.setDaemon(daemon);
        brokerThread.start();
    }

    public static class StartBroker implements Runnable {

        public StartBroker() {
            //Just constructor
        }

        public void run() {
            try{
                    BrokerService broker = new BrokerService();
                    // configure the broker
                    broker.addConnector("tcp://"+host+":"+port+"");
                    broker.start();
            }catch(Exception e){
```

```
                    System.out.println("Caught: " + e);
                    e.printStackTrace();
                }
            }
        }

        public static class StartProducer implements Runnable {
            public void run() {
                try {
                    // Create a ConnectionFactory
                    ActiveMQConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("tcp://"+host+":"+port+"");

                    // Create a Connection
                    Connection connection =
connectionFactory.createConnection();
                    connection.start();

                    // Create a Session
                    Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

                    // Create the destination (Topic or Queue)
                    Destination destination =
session.createQueue(queue_name);

                    // Create a MessageProducer from the Session to the
Topic or Queue
                    MessageProducer producer =
session.createProducer(destination);
                    producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

                    // Create a messages
                    String text = "address:100.100.100.2";
                    TextMessage message = session.createTextMessage(text);


                    Date date = new Date();
                      stime = formatter.format(date);

                    // Tell the producer to send the message
                    System.out.println("Mensagem enviada: "+ text+"
["+stime+"]");
                    producer.send(message);

                    // Clean up
                    session.close();
                    connection.close();
                }
                catch (Exception e) {
                    System.out.println("Caught: " + e);
                    e.printStackTrace();
                }
            }
```

```
    }


}
```

Receiver.java:

```
import org.apache.activemq.*;
import org.apache.activemq.broker.BrokerService;
import javax.jms.Connection;
import javax.jms.Destination;
import javax.jms.ExceptionListener;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageListener;
import javax.jms.Session;
import javax.jms.TextMessage;

/**
 * Hello world!
 */
public class Receiver {

      private static String host = "100.100.100.9";
      private static String port = "61616";
      private static String queue_name = "SERVER.TEST";

    public static void main(String[] args) throws Exception {
        thread(new StartBroker(), true);


       while(true){

         thread(new StartConsumer(), false);
         Thread.sleep(5000);

       }







    }

    public static void thread(Runnable runnable, boolean daemon) {
        Thread brokerThread = new Thread(runnable);
        brokerThread.setDaemon(daemon);
        brokerThread.start();
    }
```

```
    public static class StartBroker implements Runnable {


        public StartBroker() {

        }

        public void run() {
            try{
                    BrokerService broker = new BrokerService();
                    // configure the broker
                    broker.addConnector("tcp://"+host+":"+port+"");
                    broker.start();
            }catch(Exception e){
                    System.out.println("Caught: " + e);
                    e.printStackTrace();
            }
        }
    }



    public static class StartConsumer implements Runnable,
ExceptionListener {
        public void run() {
            try {

                // Create a ConnectionFactory
                ActiveMQConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("tcp://"+host+":"+port+"");

                // Create a Connection
                Connection connection =
connectionFactory.createConnection();
                connection.start();

                connection.setExceptionListener(this);

                // Create a Session
                Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

                // Create the destination (Topic or Queue)
                Destination destination =
session.createQueue(queue_name);


                // Create a MessageConsumer from the Session to the
Topic or Queue
                MessageConsumer consumer =
session.createConsumer(destination);

                // Wait for a message
                Message message = consumer.receive(1000);
```

```
                onMessage(message);

                consumer.close();
                session.close();
                connection.close();
            } catch (Exception e) {
                System.out.println("Caught: " + e);
                e.printStackTrace();
            }
        }


        public void onMessage(javax.jms.Message jmsMessage) {
            try {


             if (jmsMessage instanceof TextMessage) {
                    TextMessage textMessage = (TextMessage)
jmsMessage;
                    String text = textMessage.getText();
                    System.out.println("Received: -" + text);

                    if(text.contains("address")){
                       System.out.println(">>> Sending...");
                       BkgProcess bp = new
BkgProcess(text.substring(8));
                       bp.StartBkgProcess();

                    }

             }


            } catch(javax.jms.JMSException ex) {
                ex.printStackTrace();




            } catch (Exception e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
        }

        public synchronized void onException(JMSException ex) {
            System.out.println("JMS Exception occured.  Shutting down
client.");
        }
    }

}
```

BkgProcess.java:

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.text.Format;
import java.text.SimpleDateFormat;
import java.util.Date;


public class BkgProcess{

	private static String host = null;
	private static Format formatter = new
SimpleDateFormat("HH:mm:ss");
	private static String stime = null;

	public BkgProcess(String fhost){
		host = fhost;
	}

	public void StartBkgProcess(){
		Runnable tr = new BkgProcessThread();
		int i = 0;
		while(i < 6){
			Thread thread = new Thread(tr);
			thread.start();
			try {
				thread.sleep(3000);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
			i++;
		}
	}

	public static class BkgProcessThread implements Runnable{

		public BkgProcessThread(){

			Date date = new Date();
			stime = formatter.format(date);


			System.out.println("Objeto instanciado ["+stime+"]");
		}

		public void run() {

			Date date = new Date();
			stime = formatter.format(date);
			System.out.println("Host testado: "+host+"
["+stime+"]");
```

```
                  try{
                InetAddress address = InetAddress.getByName(host);
                address.isReachable(2000);
            }catch (UnknownHostException uhe) {
                uhe.printStackTrace();
            }catch (IOException ioe) {
                ioe.printStackTrace();
            }

            }

      }
}
```