EJB 3.0 – Message-Driven Beans

Este exercício trata de um componente **EJB MDB (*Message-Driven Beans*)**.

Message-Driven Bean é um componente que permite que aplicações Java EE processem mensagens assincronamente. Este tipo de bean normalmente age como um listener de mensagem JMS (Java Message Service), que é similar a um listener de eventos, como eventos em uma interface gráfica, mas recebe mensagens JMS ao invés de eventos propriamente ditos.

As mensagens podem ser enviadas por qualquer componente Java EE, como um servlet, outro componente EJB, aplicações JSF, ou até mesmo por aplicações não Java EE, como um cliente Java executando o método main().

Diferenças entre Message-Driven Beans e Session Beans

A diferença de maior destaque entre **MDB** e **SB** é que clientes não acessam MDB através de interfaces. Diferente de SB, um MDB possui somente uma classe bean.

Em muitos aspectos, um MDB se assemelha a um Stateless Session Bean.

As variáveis de instância de um MDB podem conter dados através da manipulação de mensagens de clientes, como por exemplo, uma conexão JMS, conexão com um banco de dados, ou uma referência para um objeto EJB.

Componentes cliente não localizam MDB's ou invocam seus métodos diretamente. Ao invés disso, um cliente acessa um MDB através de JMS enviando mensagens para o destino onde o MDB é um *MessageListener*. A configuração de um MDB ocorre durante o deploy no servidor.

Um MDB possui as seguintes características:

- Executam sobre uma simples mensagem do cliente.
- São invocados assincronamente.
- Possuem relativamente vida curta.
- Não representam dados em uma base de dados, mas acessam esta base.
- São *stateless*.

Quando usar MDB's

Session beans permitem o envio e recebimento de mensagens sincronamente. A fim de evitar bloqueio de recursos no servidor, use MDB's.

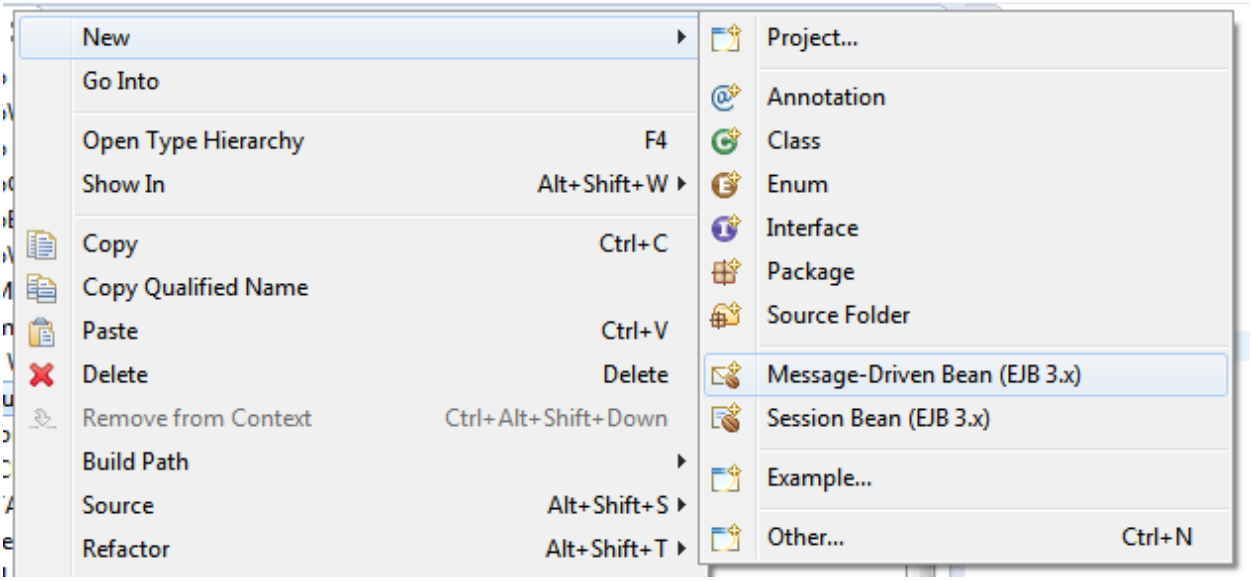
Nas etapas seguintes desenvolveremos um MDB.

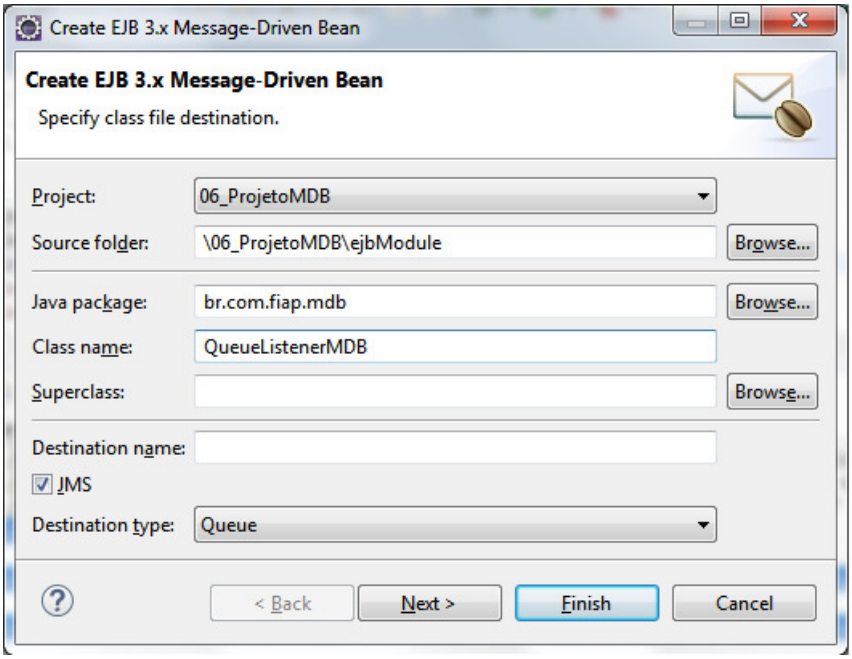
1. Criar um projeto EJB Project chamado **06_ProjetoMDB**.

2. Neste projeto, incluir a classe **Cliente**, conforme modelo abaixo:

```
package br.com.fiap.mdb.classes;

import java.io.Serializable;

public class Cliente implements Serializable {

   private static final long serialVersionUID = 1L;

   private int id;
   private String nome, telefone, email;
   public int getId() {
      return id;
   }
   public void setId(int id) {
      this.id = id;
   }
   public String getNome() {
      return nome;
   }
   public void setNome(String nome) {
      this.nome = nome;
   }
   public String getTelefone() {
      return telefone;
   }
   public void setTelefone(String telefone) {
      this.telefone = telefone;
   }
   public String getEmail() {
      return email;
   }
   public void setEmail(String email) {
      this.email = email;
   }

   @Override
   public String toString(){
      return "Cliente [id=" + this.getId() + ", nome=" +
            this.getNome() + ", telefone=" + this.getTelefone() +
            ", email=" + this.getEmail() + "]";
   }

}
```

3. Criar o consumidor Message-Driven Bean. Crie uma classe com as configurações apresentadas na interface abaixo:





4. Acrescentar as alterações na classe resultante, conforme modelo:

```
package br.com.fiap.mdb;

import java.util.Date;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;
import javax.jms.TextMessage;
```

```
import br.com.fiap.mdb.classes.Cliente;

@MessageDriven(
     activationConfig = { @ActivationConfigProperty(
          propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
                  @ActivationConfigProperty(
          propertyName= "destination", propertyValue = "queue/ExemploQueue")
     })

public class QueueListenerMDB implements MessageListener {

    public QueueListenerMDB() {
        // TODO Auto-generated constructor stub
    }

    public void onMessage(Message message) {
     try {
            if (message instanceof TextMessage) {
                System.out.println("Queue: TextMessage recebida em " + new Date());
                TextMessage msg = (TextMessage) message;
                System.out.println("Message is : " + msg.getText());
            } else if (message instanceof ObjectMessage) {
                System.out.println("Queue: ObjectMessage recebida em " + new Date());
                ObjectMessage msg = (ObjectMessage) message;
                Cliente cliente = (Cliente) msg.getObject();
                System.out.println("Detalhes do cliente: ");
                System.out.println(cliente.getId());
                System.out.println(cliente.getNome());
                System.out.println(cliente.getTelefone());
                System.out.println(cliente.getEmail());
            } else {
                System.out.println("Nenhuma mensagem válida!");
            }

        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
   }
```

5. Localizar o arquivo **standalone.xml** na pasta **[JBoss]\standalone\configuration**. Dentro da tag **<extensions>** incluir o elemento:

```
<extension module="org.jboss.as.messaging"/>
```

6. Dentro do elemento **<profile>**, adicionar o seguinte elemento :

```
<subsystem xmlns="urn:jboss:domain:messaging:2.0">
     <hornetq-server>
         <persistence-enabled>true</persistence-enabled>
         <journal-file-size>102400</journal-file-size>
         <journal-min-files>2</journal-min-files>

         <connectors>
             <http-connector name="http-connector" socket-binding="http">
                 <param key="http-upgrade-endpoint" value="http-acceptor"/>
```

```
    </http-connector>
    <http-connector name="http-connector-throughput" socket-binding="http">
        <param key="http-upgrade-endpoint" value="http-acceptor-throughput"/>
        <param key="batch-delay" value="50"/>
    </http-connector>
    <in-vm-connector name="in-vm" server-id="0"/>
</connectors>

<acceptors>
    <http-acceptor http-listener="default" name="http-acceptor"/>
    <http-acceptor http-listener="default" name="http-acceptor-throughput">
        <param key="batch-delay" value="50"/>
        <param key="direct-deliver" value="false"/>
    </http-acceptor>
    <in-vm-acceptor name="in-vm" server-id="0"/>
</acceptors>

<security-settings>
    <security-setting match="#">
        <permission type="send" roles="guest"/>
        <permission type="consume" roles="guest"/>
        <permission type="createNonDurableQueue" roles="guest"/>
        <permission type="deleteNonDurableQueue" roles="guest"/>
    </security-setting>
</security-settings>

<address-settings>
    <address-setting match="#">
        <dead-letter-address>jms.queue.DLQ</dead-letter-address>
        <expiry-address>jms.queue.ExpiryQueue</expiry-address>
        <redelivery-delay>0</redelivery-delay>
        <max-size-bytes>10485760</max-size-bytes>
        <page-size-bytes>2097152</page-size-bytes>
        <address-full-policy>PAGE</address-full-policy>
        <message-counter-history-day-limit>
           10
        </message-counter-history-day-limit>
    </address-setting>
</address-settings>

<jms-connection-factories>
    <connection-factory name="InVmConnectionFactory">
        <connectors>
            <connector-ref connector-name="in-vm"/>
        </connectors>
        <entries>
            <entry name="java:/ConnectionFactory"/>
        </entries>
    </connection-factory>
    <connection-factory name="RemoteConnectionFactory">
        <connectors>
            <connector-ref connector-name="http-connector"/>
        </connectors>
        <entries>
            <entry name="java:jboss/exported/jms/RemoteConnectionFactory"/>
        </entries>
    </connection-factory>
    <pooled-connection-factory name="hornetq-ra">
        <transaction mode="xa"/>
        <connectors>
            <connector-ref connector-name="in-vm"/>
        </connectors>
```

```
                    <entries>
                        <entry name="java:/JmsXA"/>
                        <entry name="java:jboss/DefaultJMSConnectionFactory"/>
                    </entries>
                </pooled-connection-factory>
            </jms-connection-factories>

            <jms-destinations>
                <jms-queue name="ExemploQueue">
                    <entry name="queue/ExemploQueue"/>
                    <entry name="java:jboss/exported/jms/queue/ExemploQueue"/>
                </jms-queue>
                <jms-topic name="professorTopic">
                    <entry name="topic/professorTopic"/>
                    <entry name="java:jboss/exported/jms/topic/professorTopic"/>
                </jms-topic>
            </jms-destinations>
        </hornetq-server>
    </subsystem>
```

Obs.:

- Hornetq ie um projeto opensource que permite a construção de sistemas assíncronos multiplataforma. A partir da versão 7.0 do JBoss, um grupo de configurações no arquivo standalone.xml é suficiente. Em versões anteriores era necessário usar a API Hornetq, disponível em **http://hornetq.jboss.org/docs.html**.

- Projetado pela Sun MicroSystems com o apoio de empresas associadas, e lançado para o mercado em agosto de 1998, o JMS surgiu para permitir que os aplicativos escritos na linguagem Java pudessem criar, receber e enviar mensagens destinadas ou oriundas de outros aplicativos. A principal característica deste tipo de processamento, classificado como fracamente acoplado, é que todas as operações que envolvem a troca de mensagens são feitas de forma assíncrona, fazendo com que as aplicações participantes não precisem ficar bloqueadas esperando o término de alguma computação remotamente solicitada, como ocorre naquelas aplicações que utilizam o Remote Procedure Call(RPC) ou mesmo o Remote Method Invocation(RMI)

7. Dentro do elemento **<socket-binding-group>** adicionar os elementos:

```
<socket-binding name="messaging" port="5445"/>
<socket-binding name="messaging-throughput" port="5455"/>
```

8. Localize o elemento **<subsystem xmlns="urn:jboss:domain:ejb3:2.0">**. Em seguida, adicione o elemento:

```
<mdb>
   <resource-adapter-ref
      resource-adapter-name="${ejb.resource-adapter-name:hornetq-ra.rar}"/>
   <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>
</mdb>
```

9. Inicie o servidor. Após sua inicialização, verifique as mensagens:

```
Bound messaging object to jndi name java:/queue/ExemploQueue
Started message driven bean 'QueueListenerMDB' with 'hornetq-ra.rar' resource adapter
```

10. Para criar o cliente JMS, usaremos uma aplicação Web com um servlet. Definir um novo Dynamic Web Project com o nome **06_MDBCliente**. Usar o mesmo servidor de aplicações usado no MDB.

11. Definir um arquivo .jar contendo a classe Cliente que você criou no primeiro projeto deste exercício.

12. Adicionar o arquivo **.jar** criado na pasta **WEB-INF/lib** da aplicação Web.

13. Definir, no projeto Web, um servlet cujas informações são dadas no código:

```
package br.com.fiap.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.annotation.Resource;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.MessageProducer;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import br.com.fiap.mdb.classes.Cliente;

@WebServlet("/mdb")
public class ServletMdb extends HttpServlet {
   private static final long serialVersionUID = 1L;

   @Resource(mappedName = "java:/queue/ExemploQueue")
   private Queue fila;

   @Resource(mappedName = "java:/ConnectionFactory")
   private ConnectionFactory connectionFactory;

   public ServletMdb() {
      super();
      // TODO Auto-generated constructor stub
   }

   protected void doGet(HttpServletRequest request,
         HttpServletResponse response) throws ServletException, IOException {


      PrintWriter out = response.getWriter();
      response.setContentType("text/html");
      try {
         System.out.println(getClass() + "Inicio........");
```

```
        Connection connection = connectionFactory.createConnection();
        try {
           Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
           MessageProducer messageProducer = session.createProducer(fila);

               //1. Enviando objeto TextMessage
               TextMessage message = session.createTextMessage();
               message.setText("Exemplo EJB3 MDB Queue!!!");
               messageProducer.send(message);
               out.println("1. Enviando mensagem tipo TextMessage");

               //2. Enviando objeto ObjectMessage
           ObjectMessage objMessage = session.createObjectMessage();
           Cliente cliente = new Cliente();
           cliente.setId(1500);
           cliente.setNome("Fiap Pós");
           cliente.setTelefone("3385-8010");
           cliente.setEmail("emilio.celso@fiap.com.br");

              objMessage.setObject(cliente);
           messageProducer.send(objMessage);

           messageProducer.close();
        } finally {
           connection.close();
        }
        System.out.println(getClass() + "Fim........");
        out.print("<H1>Objeto enviado com sucesso! JMS 1.0</H1>");

     } catch (Exception ex) {
        ex.printStackTrace();
     }
  }

  protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
  }
}
```

14. Incluir o projeto Web ao servidor.

15. Executar o servlet. O resultado no browser deverá ser semelhante a:

e no log do servidor, a informação:

```
18:20:48,421 INFO  [stdout] (default task-2) class
br.com.fiap.servlet.ServletMdbInicio........

18:20:48,496 INFO  [stdout] (Thread-0 (HornetQ-client-global-threads-1516768974))
Queue: TextMessage recebida em Fri Jun 26 18:20:48 BRT 2015

18:20:48,497 INFO  [stdout] (Thread-0 (HornetQ-client-global-threads-1516768974))
Message is : Exemplo EJB3 MDB Queue!!!

18:20:48,501 INFO  [stdout] (Thread-1 (HornetQ-client-global-threads-1516768974))
Queue: ObjectMessage recebida em Fri Jun 26 18:20:48 BRT 2015

18:20:48,509 INFO  [stdout] (Thread-1 (HornetQ-client-global-threads-1516768974))
Detalhes do cliente:

18:20:48,509 INFO  [stdout] (Thread-1 (HornetQ-client-global-threads-1516768974))
1500

18:20:48,509 INFO  [stdout] (Thread-1 (HornetQ-client-global-threads-1516768974))
Fiap Pós

18:20:48,509 INFO  [stdout] (Thread-1 (HornetQ-client-global-threads-1516768974))
3385-8010

18:20:48,510 INFO  [stdout] (Thread-1 (HornetQ-client-global-threads-1516768974))
emilio.celso@fiap.com.br

18:20:48,523 INFO  [stdout] (default task-2) class
br.com.fiap.servlet.ServletMdbFim........
```

Bom trabalho!

Prof. Emilio