Module 11

Developing Message-Driven Beans

Objectives

Upon completion of this module, you should be able to:

- Describe the properties and life cycle of message-driven beans
- Create a JMS message-driven bean
- Create lifecycle event handlers for a JMS message-driven bean

Introducing Message-Driven Beans

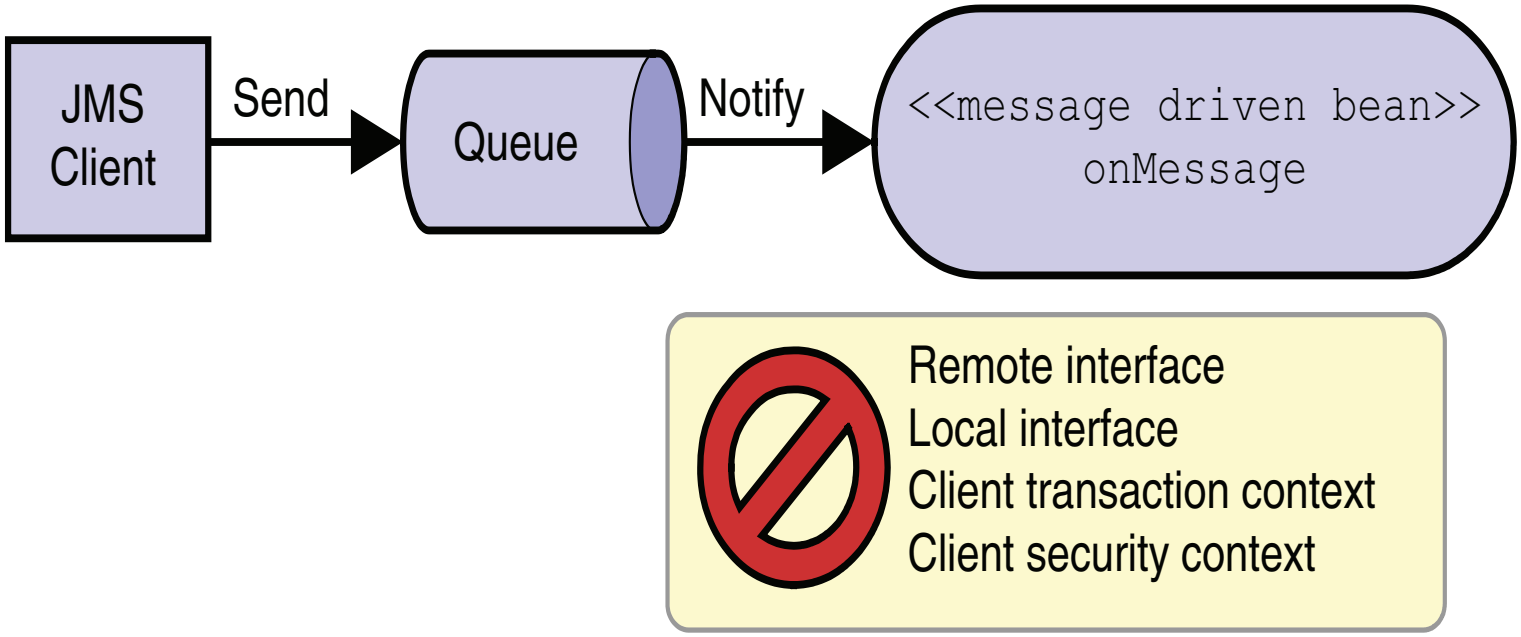
Message-driven beans are intended as an asynchronous message consumer.

Message-driven beans implement a message listener interface:

- For example, JMS message-driven beans implement the JMS API `MessageListener` interface.
- The bean designer is responsible for the message listener method code.

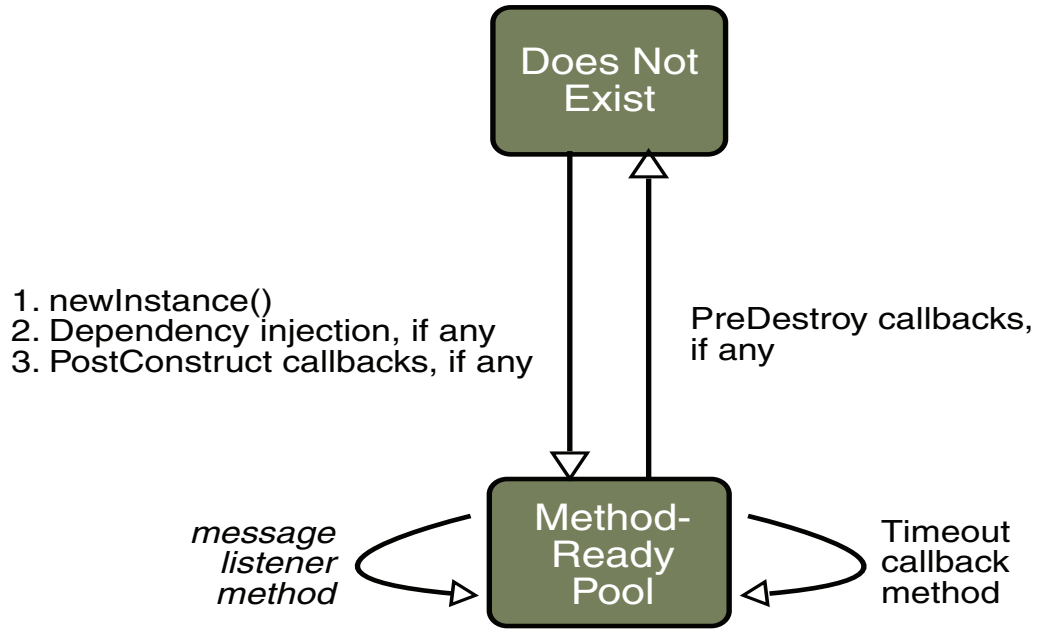
The deployer supplies information for the container to register the message-driven bean as a message listener with the destination.

Java EE Technology Client View of Message-Driven Beans



Containers can create and pool multiple instances of message-driven beans to service the same message destination.

Life Cycle of a Message-Driven Bean



Types of Message-Driven Beans

JMS message-driven beans:

- Are annotated with `@javax.ejb.MessageDriven` and implement the `javax.jms.MessageListener` interface.
- Application server can integrate support or use connector.

Non JMS message-driven beans:

- Are annotated with `@javax.ejb.MessageDriven` and implement a message listener interface specific to the messaging service.
- Application server requires a connector.

Creating a JMS Message-Driven Bean

1. Declare the message-driven bean class:
   - Class must be public but not final or abstract.
2. Annotate the message-driven bean class using the `MessageDriven` metadata annotation
   - Specify values with the following attributes:

```
mappedName = nameOfMessageDestination

activationConfig = {
        @ActivationConfigProperty(propertyName = "acknowledgeMode”,
propertyValue = "Auto-acknowledge”),
        @ActivationConfigProperty(propertyName = "destinationType”,
propertyValue = "javax.jms.Queue")
}
```

Creating a JMS Message-Driven Bean

3. Optionally, use resource injection to get a `MessageDrivenContext` instance.
4. Include a public no-argument empty constructor.
5. Write the `onMessage` method of the `MessageListener` interface.
6. Do not define a `finalize` method.

Message-Driven Bean Class Simple Example

```
1   import javax.ejb.*;
2   import javax.jms.*;
3
4   @MessageDriven(mappedName = "jms/MyQueue", activationConfig =  {
5           @ActivationConfigProperty(propertyName = "acknowledgeMode",
6             propertyValue = "Auto-acknowledge"),
7           @ActivationConfigProperty(propertyName = "destinationType",
8             propertyValue = "javax.jms.Queue")
9       })
10  public class SampleMDB implements MessageListener {
11
12      public void onMessage(Message message) {
13      }
14
15  }
```

Coding the Message-Driven Bean Class

```
1   import javax.ejb.*;
2   import javax.annotation.*;
3   import javax.naming.*;
4   import javax.jms.*;
5
6   /**
7    * The MessageBean class is a JMS message-driven bean. It
8    * indirectly implements javax.jms.MessageListener interface.
9    * It is defined as public (but not final or abstract).
10   * It defines a constructor and the onMessage method.
11   */
12  @MessageDriven {
13    mappedName="destinationType",
14    messageListenerInterface = javax.jms.MessageListener,
15    activationConfig = {
16      @ActivationConfigProperty(propertyName="messageSelector",
17        propertyValue="RECIPIENT = 'MDB'")
18    }
19  }
```

Coding the Message-Driven Bean Class

```
20  public class MessageBean {
21    @Resource private MessageDrivenContext mdc = null;
22
23    // Constructor, which is public and takes no arguments.
24    public MessageBean() {
25      System.out.println("In MessageBean.MessageBean()");
26    }
27
28    // onMessage method, public not final or static
29    // Casts the incoming Message to a TextMessage and displays
30    // the text.
31
32    public void onMessage(Message inMessage) {
33      TextMessage msg = null;
34      try {
35        if (inMessage instanceof TextMessage) {
36          msg = (TextMessage) inMessage;
```

Coding the Message-Driven Bean Class

```
37          System.out.println("MESSAGE BEAN: Message " +
38            "received: " + msg.getText());
39        } else {
40          System.out.println("Message of wrong type: " +
41            inMessage.getClass().getName());
42        }
43      } catch (JMSException e) {
44        System.err.println("MessageBean.onMessage: " +
45          "JMSException: " + e);
46           mdc.setRollbackOnly();
47      } catch (Throwable te) {
48        System.err.println("MessageBean.onMessage: " +
49          "Exception: " + te);
50      }
51    }
52  }
```

Complex Message-Driven Bean Classes

- Using message header message selectors

```
@MessageDriven(
  activationConfig={
    @ActivationConfigProperty(
      propertyName="messageSelector",
      propertyValue="JMSType = 'car' AND color = 'blue' and weight >2500"
    )
  }
)
```

- Specifying message acknowledgement

```
@MessageDriven(
  activationConfig={
    @ActivationConfigProperty(
      propertyName="acknowledgeMode",
      propertyValue="Dups-ok-acknowledge"
    )
  }
)
```

Callback Methods in a Bean Class

```
1   import javax.ejb.*;
2   import javax.jms.*;
3   import javax.annotation.*;
4
5   @MessageDriven {
6   public class MessageBean {
7     @Resource private MessageDrivenContext mdc;
8
9     public MessageBean() { } // constructor
10
11    @PostConstruct void obtainResources() { }
12
13    @PreDestroy void releaseResources() { }
14
15    public void onMessage(Message inMessage) { }
16  }
```