



## Messaging (2)



Corso di Laurea in Informatica, Programmazione Distribuita  
Delfina Malandrino, [dmandrino@unisa.it](mailto:dmandrino@unisa.it)  
<http://www.unisa.it/docenti/delfinamalandrino>

1

## Organizzazione della lezione

2

- Meccanismi di affidabilità
- Message-Driven Beans
- Un esempio conclusivo
  - Il codice
  - Configurazione
  - I progetti
- Conclusioni

2

## Organizzazione della lezione

3

- **Meccanismi di affidabilità**
- Message-Driven Beans
- Un esempio conclusivo
  - Il codice
  - Configurazione
  - I progetti
- Esercizi
- Conclusioni

3

## Come assicurare un recapito affidabile

4

- JMS definisce diversi livelli di affidabilità per assicurare che i messaggi siano instradati correttamente
  - anche se il provider va in crash o è sotto carico elevato

4

## Come assicurare un recapito affidabile

5

- I meccanismi sono i seguenti
  - ▣ Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - ▣ Setting message time-to-live: Scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - ▣ Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - ▣ Controlling acknowledgment: controllo degli ack a vari livelli
  - ▣ Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - ▣ Setting priorities: priorità di messaggi

5

## Come assicurare un recapito affidabile

6

- I meccanismi sono i seguenti
  - ▣ Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - ▣ Setting message time-to-live: Scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - ▣ Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - ▣ Controlling acknowledgment: controllo degli ack a vari livelli
  - ▣ Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - ▣ Setting priorities: Priorità di messaggi

6

## Filtering dei messaggi

7

- Si fa in modo che arrivino **solo** i messaggi a cui si è interessati
  - ▣ Messaggio mandato in broadcast a diversi client, si definisce un selector in modo che venga consumato solo da consumer interessati
- Nessuno spreco di tempo e banda per ricevere cose non di interesse
- Si può fare selezione su headers o metadati (JMSPriority < 6) o su proprietà custom (orderAmount < 200)
- Il message selector è una stringa che contiene una espressione:

```
context.createConsumer(queue, "JMSPriority < 6").receive();  
context.createConsumer(queue, "JMSPriority < 6 AND orderAmount < 200").receive();  
context.createConsumer(queue, "orderAmount BETWEEN 1000 AND 2000").receive();
```

7

## Filtering dei messaggi

8

- Il messaggio viene creato dal Producer usando metodi per settare proprietà e priorità (nell'header)

```
context.createTextMessage().setIntProperty("orderAmount", 1530);  
context.createTextMessage().setJMSPriority(5);
```

8

## Filtering dei messaggi

9

### Producer

```
context.createTextMessage().setIntProperty("orderAmount", 1530);  
context.createTextMessage().setJMSPriority(5);
```

### Consumer

```
context.createConsumer(queue, "JMSPriority < 6").receive();  
context.createConsumer(queue, "JMSPriority < 6 AND orderAmount < 200").receive();  
context.createConsumer(queue, "orderAmount BETWEEN 1000 AND 2000").receive();
```

9

## Filtering dei messaggi

10

- Selector expression possono usare:
  - ▣ logical operators (NOT, AND, OR)
  - ▣ comparison operators (=, >, >=, <, <=, <>)
  - ▣ Arithmetic operators (+, -, \*, /)
  - ▣ expressions ([NOT] BETWEEN, [NOT] IN, [NOT] LIKE, IS [NOT] NULL)
  - ▣ and so on.

10

## Come assicurare un recapito affidabile

11

- I meccanismi sono i seguenti
  - Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - Setting message time-to-live: Scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - Controlling acknowledgment: controllo degli ack a vari livelli
  - Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - Setting priorities: Priorità di messaggi

11

## Evitare messaggi obsoleti

12

- Un setting del time-to-live può essere di aiuto per evitare che messaggi obsoleti vengano recapitati ai destinatari
- Si setta il tempo in millisecondi, passato il quale il provider (il broker) rimuove il messaggio
- Si utilizza il metodo del producer:

```
context.createProducer().setTimeToLive(1000).send(queue, message);
```

12

## Come assicurare un recapito affidabile

13

- I meccanismi sono i seguenti
  - ▣ Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - ▣ Setting message time-to-live: Scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - ▣ Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - ▣ Controlling acknowledgment: controllo degli ack a vari livelli
  - ▣ Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - ▣ Setting priorities: Priorità di messaggi

13

## Gestire la persistenza

14

- JMS supporta 2 modalità di message delivery: persistent e nonpersistent
  - ▣ **Persistent delivery**: messaggio salvato sul provider (disk/database)
    - Messaggi non persi in caso di restart del broker
  - ▣ **Non-persistent delivery**: messaggio non salvato
  - ▣ Persistent delivery è il valore di default... che può essere "degradato" per migliorare le prestazioni

```
context.createProducer().setDeliveryMode(DeliveryMode.NON_PERSISTENT).send(queue, message);
```

14

## Come assicurare un recapito affidabile

15

- I meccanismi sono i seguenti
  - ▣ Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - ▣ Setting message time-to-live: Scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - ▣ Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - ▣ **Controlling acknowledgment**: controllo degli ack a vari livelli
  - ▣ Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - ▣ Setting priorities: Priorità di messaggi

15

## Controllo degli Acknowledgment

16

- Si vuole ricevere una verifica del recapito del messaggio al destinatario
- Diverse modalità di acknowledgment:
  - ▣ `AUTO_ACKNOWLEDGE`: la sessione automaticamente fa ack di un messaggio
  - ▣ `CLIENT_ACKNOWLEDGE`: ack esplicito del client
    - chiamando il metodo `Message.acknowledge()`

16



## Controllo degli Acknowledgment

17

- Esempio di Producer che usa l'annotazione `@JMSSessionMode` per settare l'acknowledgment mode

```
// Producer
@Inject
@JMSConnectionFactory("jms/connectionFactory")
@JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
private JMSContext context;
...
context.createProducer().send(queue, message);
```

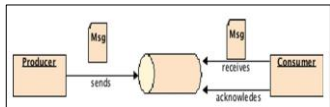
17

## Controllo degli Acknowledgment

18

- Esempio di consumer che fa esplicito acknowledges del messaggio chiamando il metodo `acknowledge()`

```
// Consumer
message.acknowledge();
```



18

## Come assicurare un recapito affidabile

19

- I meccanismi sono i seguenti
  - ▣ Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - ▣ Setting message time-to-live: Scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - ▣ Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - ▣ Controlling acknowledgment: controllo degli ack a vari livelli
  - ▣ **Creating durable subscribers**: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - ▣ Setting priorities: Priorità di messaggi

19

## Durable Consumer

20

- Nel modello publish-subscribe un consumer che non è in esecuzione perde i messaggi che vengono postati sul topic
- Con i durable consumer si può controllare che i messaggi vengano mantenuti dal provider fino a quando tutti i consumer li hanno ricevuti
- Con i durable subscribers, un consumer che si riconnette riceve i messaggi che sono arrivati durante la disconnessione
- Creazione attraverso JMSContext con una id specifica "unica"

```
context.createDurableConsumer(topic, "uniqueID").receive();
```

```
context.createDurableConsumer(topic, "javaee7DurableSubscription").receive();
```

20

## Durable Consumer

21

```
context.createDurableConsumer(topic,"uniqueID").receive();
```

- A questo punto il client inizia la connessione e riceve messaggi
- Il nome **"uniqueID"** (nel nostro esempio `javaee7DurableSubscription`) è usato come identificatore della durable subscription (identificativo del subscriber)
- Ogni durable consumer deve avere un unique ID
  - ▣ che corrisponde alla dichiarazione di un'unica connection factory
  - ▣ per ogni potenziale durable consumer

21

## Durable Consumer

21

```
context.createDurableConsumer(topic,"uniqueID").receive();
```

- L'identificativo unico associato al durable subscriber serve al JMS server per memorizzare messaggi arrivati mentre il subscriber non è attivo
- Quando il subscriber si riconnette il JMS server provvede a inviare tutti i messaggi ancora validi accumulati fino a quel momento

22

## Come assicurare un recapito affidabile

22

- I meccanismi sono i seguenti
  - ▣ Filtering messages: usando i selector è possibile ricevere solo i messaggi che si desiderano
  - ▣ Setting message time-to-live: Scegliere il time-to-live (expiration time) in modo da non instradare messaggi se obsoleti
  - ▣ Specifying message persistence: specificare la persistenza di messaggi (nonostante possibili malfunzionamento del provider)
  - ▣ Controlling acknowledgment: controllo degli ack a vari livelli
  - ▣ Creating durable subscribers: assicurare instradamento di messaggi verso un "unavailable" subscriber in un pub-sub model
  - ▣ Setting priorities: Priorità di messaggi

23

## Scegliere priorità del messaggio

23

- Si istruisce il JMS provider a fare il delivery di messaggi urgenti per primi
- Contenute nell'header del messaggio
- Valori da 0 (bassa priorità) a 9 (alta priorità)
- Esempio:

```
context.createProducer().setPriority(2).send(queue, message);
```

- Concatenazione di diversi meccanismi:

```
context.createProducer().setPriority(2)
                        .setTimeToLive(1000)
                        .setDeliveryMode(DeliveryMode.NON_PERSISTENT)
                        .send(queue, message);
```

24

## Organizzazione della lezione

24

- Meccanismi di affidabilità
- **Message-Driven Beans**
- Un esempio conclusivo
  - Il codice
  - Configurazione
  - I progetti
- Esercizi
- Conclusioni

25

## EJB che scambiano messaggi

25

- Un Message Driven Bean (MDB) è un consumatore di messaggi, asincrono, invocato dal container quando arriva un messaggio
- Parte delle specifiche di Enterprise JavaBeans: simili a stateless
- Tramite CDI può accedere a altri EJB, JDBC, risorse JMS, entity manager, etc.
- **Perché usare un MDB anziché un JMS Client?**
  - ▣ Il vantaggio di usare un MDB (rispetto ad un JMS Client) è che transazione, multithread, sicurezza, etc., sono gestiti dal container

26

## Un esempio di un semplice MDB

```
@MessageDriven(mappedName =
    "jms/javaee7/Topic")
public class BillingMDB
    implements MessageListener

    public void onMessage(Message message)
    {   System.out.println("Received:"
        + message.getBody(String.class));
    }
```

Definisce un MDB

27

## Un esempio di un semplice MDB

```
@MessageDriven(mappedName =
    "jms/javaee7/Topic")
public class BillingMDB
    implements MessageListener

    public void onMessage(Message message)
    {   System.out.println("Received:"
        + message.getBody(String.class));
    }
```

Definisce un MDB

Implementa un  
**MessageListener** e quindi il  
metodo **onMessage()**

28

## Un esempio di un semplice MDB

```
@MessageDriven(mappedName =
    "jms/javaee7/Topic")
public class BillingMDB
    implements MessageListener

    public void onMessage(Message message)
    {
        System.out.println("Received:"
            + message.getBody(String.class));
    }
```

- > Definisce un MDB
- > Implementa un MessageListener e quindi il metodo onMessage()

Cosa fare quando si riceve un messaggio

29

## Come è fatto un MDB

29

- MDB non è parte del modello EJB Lite: serve una implementazione full EE
- Annotazione con @javax.ejb.MessageDriven (o XML equivalente)
- Implementare l'interfaccia del listener
- Definita come public, non final o abstract
- Deve esserci un costruttore senza argomenti, per permetterne l'istanziamento automatico da parte del container
- La classe non deve avere il metodo finalize()

30

## Come configurare un MDB

30

- Possibile usare tutti i setting realizzabili con JMS

Property	Description
acknowledgeMode	The acknowledgment mode (default is <code>AUTO_ACKNOWLEDGE</code> )
messageSelector	The message selector string used by the MDB
destinationType	The destination type, which can be <code>TOPIC</code> or <code>QUEUE</code>
destinationLookup	The lookup name of an administratively-defined Queue or Topic
connectionFactoryLookup	The lookup name of an administratively defined ConnectionFactory
destination	The name of the destination.
subscriptionDurability	The subscription durability (default is <code>NON_DURABLE</code> )
subscriptionName	The subscription name of the consumer
shareSubscriptions	Used if the message-driven bean is deployed into a clustered
clientId	Client identifier that will be used when connecting to the JMS provider

31

## Come usare le proprietà di un MDB

```

@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount < 3000")
})
public class BillingMDB implements MessageListener {
    public void onMessage(Message message) {
        System.out.println("Message received: "+
            message.getBody(String.class));
    }
}

```

Nella definizione del MDB

32



## Come usare le proprietà di un MDB

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount < 3000")
})
public class BillingMDB implements MessageListener {
    public void onMessage(Message message) {
        System.out.println("Message received: "+
            message.getBody(String.class));
    }
}
```

Nella definizione del MDB

Viene definita la  
configurazione ...

33

## Come usare le proprietà di un MDB

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount < 3000")
})
public class BillingMDB implements MessageListener {
    public void onMessage(Message message) {
        System.out.println("Message received: "+
            message.getBody(String.class));
    }
}
```

Nella definizione del MDB

Viene definita la  
configurazione ...

... con le sue proprietà

34

## Come usare le proprietà di un MDB

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount < 3000")
})
public class BillingMDB implements MessageListener {
    public void onMessage(Message message) {
        System.out.println("Message received: "+
            message.getBody(String.class));
    }
}
```

> Nella definizione del MDB

Viene definita la configurazione ...

> ... con le sue proprietà

> ... filtro di messaggi

35

## Come usare le proprietà di un MDB

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig={
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount < 3000")
})
public class BillingMDB implements MessageListener {
    public void onMessage(Message message) {
        System.out.println("Message received: "+
            message.getBody(String.class));
    }
}
```

> Nella definizione del MDB

> Viene definita la configurazione ...

> ... con le sue proprietà

> ... filtro di messaggi

> Metodo per gestire i messaggi

36

## Dependency Injection

37

- Come per tutti gli EJB, gli MDBs possono usare Dependency Injection per ottenere riferimenti a risorse come JDBC datasources, EJBs, o altri oggetti
- Injection è il meccanismo attraverso il quale il container inserisce le dipendenze automaticamente dopo aver creato l'oggetto
- Queste risorse devono essere disponibili nel container oppure nell'environment context

```
@PersistenceContext
private EntityManager em;
@Inject
private InvoiceBean invoice;
@Resource(lookup = "jms/javaee7/ConnectionFactory")
private ConnectionFactory connectionFactory;
```

The MDB context can also be injected using the @Resource annotation:

```
@Resource private MessageDrivenContext context;
```

37

## MDB context: MessageDrivenContext

37

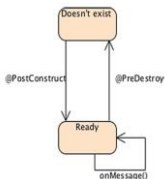
- Questa interfaccia fornisce accesso al runtime context, che il container fornisce per una istanza di un MDB
- Il container passa l'interfaccia MessageDrivenContext all'istanza, che rimane associata per il lifetime dell'MDB
- Permette all'MDB di fare roll back di una transazione, ottenere il caller (user principal), ecc.

Method	Description
getCallerPrincipal	Returns the java.security.Principal associated with the invocation
getRollbackOnly	Tests whether the current transaction has been marked for rollback
getTimerService	Returns the javax.ejb.TimerService interface
getUserTransaction	Returns the javax.transaction.UserTransaction interface to use to demarcate transactions. Only MDBs with bean-managed transaction (BMT) can use this method
isCallerInRole	Tests whether the caller has a given security role
lookup	Enables the MDB to look up its environment entries in the JNDI naming context
setRollbackOnly	Allows the instance to mark the current transaction as rollback. Only MDBs with BMT can use this method

38

## Il ciclo di vita di un MDB

38



- › Simile a quello degli stateless beans
- › Possibile inserire interceptors dei metodi

39

## MDB as a consumer

40

- Per natura, gli MDBs sono progettati per funzionare come asynchronous message consumers
- Gli MDBs implementano una message listener interface, che viene "risvegliata" (triggered) dal container quando un messaggio arriva

40

## MDB as a consumer

41

- Può un MDB essere un *synchronous consumer*?
- Sì, ma non è raccomandato
  - ▣ Synchronous message consumers bloccano le risorse del server (gli EJBs si bloccheranno in un loop senza eseguire nessun lavoro ed il container non sarà in grado di liberarli)
  - ▣ Gli MDBs, come gli stateless session beans, vivono in un pool di una certa taglia
    - Quando il container ha bisogno di una istanza la prende dal pool e la usa
    - Se l'istanza va in un loop infinito, il pool si svuoterà e tutte le risorse saranno bloccate in un busy looping
  - ▣ L' EJB container può generare nuove istanze di MDB incrementando il pool ma aumentando così il consumo di memoria
  - ▣ Per questa ragione, session beans e MDBs non dovrebbero essere usati come synchronous message consumers

41

## MDB as a consumer

42

- Può un MDB essere un *synchronous consumer*?
- Sì, ma non è raccomandato

*Table 13-10. MDB Compared with Session Beans*

Enterprise Beans	Producer	Synchronous Consumer	Asynchronous Consumer
Session beans	Yes	Not recommended	Not possible
MDB	Yes	Not recommended	Yes

42

## MDB as a producer

43

- Gli MDBs possono ANCHE diventare message producers
  - ▣ Workflow che prevede che essi ricevano messaggi da una destinazione, li processino, e li rinviino ad un'altra destinazione
- Per aggiungere questa capacità bisogna usare le API JMS
- Vediamo un esempio....

43

## Esempio di MDB che riceve ed invia messaggi

```

@MessageDriven(mappedName = "jms/javaee7/Topic", ← MDB con il topic
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    { context.createProducer().send(printingQueue,
        "Message has been received and resent");
    }
}

```

44

## Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    {
        context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
}
```

MDB con il topic

Configurazione

45

## Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    {
        context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
}
```

MDB con il topic

Configurazione

Auto-ack

46

## Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    { context.createProducer().send(printingQueue,
        "Message has been received and resent");
    }
}
```

- › MDB con il topic
- › Configurazione
- › Auto-ack
- › Filtro messaggi

47

## Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    { context.createProducer().send(printingQueue,
        "Message has been received and resent");
    }
}
```

- › MDB con il topic
- › Configurazione
- › Auto-ack
- › Filtro messaggi
- › Implementa listener

48



## Esempio di MDB che riceve ed invia messaggi

```

@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    { context.createProducer().send(printingQueue,
        "Message has been received and resent");
    }
}

```

- › MDB con il topic
- › Configurazione
- › Auto-ack
- › Filtro messaggi
- › Implementa listener
- › Context iniettato dalla CF

49

## Esempio di MDB che riceve ed invia messaggi

```

@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    { context.createProducer().send(printingQueue,
        "Message has been received and resent");
    }
}

```

- › MDB con il topic
- › Configurazione
- › Auto-ack
- › Filtro messaggi
- › Implementa listener
- › Context iniettato dalla CF
- › ...nella variabile

50

## Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    { context.createProducer().send(printingQueue,
        "Message has been received and resent");
    }
}
```

- › MDB con il topic
- › Configurazione
- › Auto-ack
- › Filtro messaggi
- › Implementa listener
- › Context iniettato dalla CF
- › ...nella variabile
- › Destinazione iniettata dal container

51

## Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    { context.createProducer().send(printingQueue,
        "Message has been received and resent");
    }
}
```

- › MDB con il topic
- › Configurazione
- › Auto-ack
- › Filtro messaggi
- › Implementa listener
- › Context iniettato dalla CF
- › ...nella variabile
- › Destinazione iniettata dal container
- › Metodo quando riceve messaggi

52

## Esempio di MDB che riceve ed invia messaggi

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount BETWEEN 3 AND 7")
})
public class BillingMDB implements MessageListener {
    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    @JMSSessionMode(JMSContext.AUTO_ACKNOWLEDGE)
    private JMSContext context;

    @Resource(lookup = "jms/javaee7/Queue")
    private Destination printingQueue;

    public void onMessage(Message message) {
        System.out.println("Message received:"
            + message.getBody(String.class));
        sendPrintingMessage();
    }

    private void sendPrintingMessage() throws JMSException
    {
        context.createProducer().send(printingQueue,
            "Message has been received and resent");
    }
}
```

- › MDB con il topic
- › Configurazione
- › Auto-ack
- › Filtro messaggi
- › Implementa listener
- › Context iniettato dalla CF
- › ...nella variabile
- › Destinazione iniettata dal container
- › Metodo quando riceve messaggi
- › Metodo per inviare

53

## Transazioni... anche per i messaggi

54

- Transazioni per scambio di messaggi: un certo numero di messaggi vanno recapitati tutti insieme o nessuno
- In quanto EJB, le transazioni possono essere Bean-managed oppure Container-managed

54

## Organizzazione della lezione

54

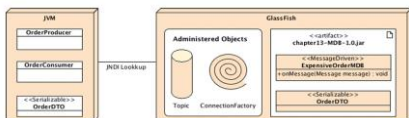
- Meccanismi di affidabilità
- Message-Driven Beans
- Un esempio conclusivo
  - ▣ Il codice
    - Configurazione
    - I progetti
- Esercizi
- Conclusioni

55

## Lo schema

56

- Producer stand-alone, `OrderProducer`, che invia messaggi a un topic, con un valore di amount (settato su linea di comando del producer)
- Il consumer `OrderConsumer` riceve tutti i messaggi
- ... mentre l'MDB `ExpensiveOrderMDB` riceve solamente quelli sopra i 1000



56

## L'ordine

```
public class OrderDTO implements Serializable {  
    private Long orderId;  
    private Date creationDate;  
    private String customerName;  
    private Float totalAmount;  
  
    //Constructors, getters, setters
```

Deve essere trasmesso in  
un messaggio

57

## L'ordine

```
public class OrderDTO implements Serializable {  
    private Long orderId;  
    private Date creationDate;  
    private String customerName;  
    private Float totalAmount;  
  
    //Constructors, getters, setters
```

Deve essere trasmesso in  
un messaggio

Campi dell'ordine

58

## L'ordine

```
public class OrderDTO implements Serializable {
    private Long orderId;
    private Date creationDate;
    private String customerName;
    private Float totalAmount;
    //Constructors, getters, setters
}
```

- > Deve essere trasmesso in un messaggio
- > Campi dell'ordine
- > Tra cui l'ammontare totale

59

## Il Produttore

```
public class OrderProducer {
    public static void main(String[] args)
        throws NamingException {

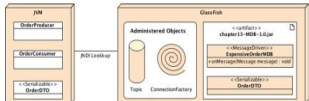
        Float totalAmount = Float.valueOf(args[0]);
        OrderDTO order = new OrderDTO(1234L, new Date(),
            "Betty Moreau", totalAmount);

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try {
            JMSContext jmsContext =
                connectionFactory.createContext() {
            jmsContext.createProducer().setProperty("orderAm",
                totalAmount).send(topic, order);
            }
        }
    }
}
```

Parametro passato su linea di comando



60

## Il Produttore

```
public class OrderProducer {
    public static void main(String[] args)
        throws NamingException {

        Float totalAmount = Float.valueOf(args[0]);

        OrderDTO order = new OrderDTO(12341, new Date(),
            "Betty Moreau", totalAmount);

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext =
            connectionFactory.createContext()) {
            jmsContext.createProducer().setProperty("orderAmount",
                totalAmount).send(topic, order);
        }
    }
}
```

Parametro passato su linea di comando

Si crea un nuovo ordine

61

## Il Produttore

```
public class OrderProducer {
    public static void main(String[] args)
        throws NamingException {

        Float totalAmount = Float.valueOf(args[0]);

        OrderDTO order = new OrderDTO(12341, new Date(),
            "Betty Moreau", totalAmount);

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext =
            connectionFactory.createContext()) {
            jmsContext.createProducer().setProperty("orderAmount",
                totalAmount).send(topic, order);
        }
    }
}
```

Parametro passato su linea di comando

Si crea un nuovo ordine

Si ottiene il contesto JNDI

62

## Il Produttore

```
public class OrderProducer {
    public static void main(String[] args)
        throws NamingException {

        Float totalAmount = Float.valueOf(args[0]);

        OrderDTO order = new OrderDTO(12341, new Date(),
            "Betty Moreau", totalAmount);

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext =
            connectionFactory.createContext()) {
            jmsContext.createProducer().setProperty("orderAmount",
                totalAmount).send(topic, order);
        }
    }
}
```

Parametro passato su linea di comando

- › Si crea un nuovo ordine
- › Si ottiene il contesto JNDI

Lookup degli oggetti administered: CF

63

## Il Produttore

```
public class OrderProducer {
    public static void main(String[] args)
        throws NamingException {

        Float totalAmount = Float.valueOf(args[0]);

        OrderDTO order = new OrderDTO(12341, new Date(),
            "Betty Moreau", totalAmount);

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext =
            connectionFactory.createContext()) {
            jmsContext.createProducer().setProperty("orderAmount",
                totalAmount).send(topic, order);
        }
    }
}
```

Parametro passato su linea di comando

- › Si crea un nuovo ordine
- › Si ottiene il contesto JNDI

Lookup degli oggetti administered: CF

... e topic

64



## Il Produttore

```
public class OrderProducer {
    public static void main(String[] args)
        throws NamingException {

        Float totalAmount = Float.valueOf(args[0]);

        OrderDTO order = new OrderDTO(12341, new Date(),
            "Betty Moreau", totalAmount);

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try(JMSContext jmsContext =
            connectionFactory.createContext()) {
            jmsContext.createProducer().setProperty("orderAmount",
                totalAmount).send(topic, order);
        }
    }
}
```

- › Parametro passato su linea di comando
- › Si crea un nuovo ordine
- › Si ottiene il contesto JNDI
- › Lookup degli oggetti administered: CF
- › ... e topic
- › Con il contesto creato...

65

## Il Produttore

```
public class OrderProducer {
    public static void main(String[] args)
        throws NamingException {

        Float totalAmount = Float.valueOf(args[0]);

        OrderDTO order = new OrderDTO(12341, new Date(),
            "Betty Moreau", totalAmount);

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try(JMSContext jmsContext =
            connectionFactory.createContext()) {
            jmsContext.createProducer().setProperty("orderAmount",
                totalAmount).send(topic, order);
        }
    }
}
```

- › Parametro passato su linea di comando
- › Si crea un nuovo ordine
- › Si ottiene il contesto JNDI
- › Lookup degli oggetti administered: CF
- › ... e topic
- › Con il contesto creato...
- › Si crea un produttore con una proprietà

66

## Il Produttore

```
public class OrderProducer {
    public static void main(String[] args)
        throws NamingException {

        Float totalAmount = Float.valueOf(args[0]);

        OrderDTO order = new OrderDTO(12341, new Date(),
            "Betty Moreau", totalAmount);

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext =
            connectionFactory.createContext()) {
            jmsContext.createProducer().setProperty("orderAmount",
                totalAmount).send(topic, order);
        }
    }
}
```

Parametro passato su linea di comando

- > Si crea un nuovo ordine
- > Si ottiene il contesto JNDI
- > Lookup degli oggetti administered: CF
- > ... e topic

Con il contesto creato...

Si crea un produttore con una proprietà

e si invia il messaggio

67

## Il consumatore standard JSE

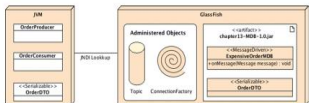
```
public class OrderConsumer {
    public static void main(String[] args)
        throws NamingException {

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext =
            connectionFactory.createContext()) {
            while(true) {
                OrderDTO order = jmsContext.createConsumer(topic)
                    .receiveBody(OrderDTO.class);
                System.out.println("Order received:" + order);
            }
        }
    }
}
```

Il contesto dall'ambiente di esecuzione



68

## Il consumatore standard JSE

```
public class OrderConsumer {
    public static void main(String[] args)
        throws NamingException {

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try(JMSContext jmsContext=
            connectionFactory.createContext()) {
            while(true) {
                OrderDTO order = jmsContext.createConsumer(topic)
                    .receiveBody(OrderDTO.class);
                System.out.println("Order received:" + order);
            }
        }
    }
}
```

Il contesto dall'ambiente di esecuzione

Lookup oggetti administered: CF e destinazione

69

## Il consumatore standard JSE

```
public class OrderConsumer {
    public static void main(String[] args)
        throws NamingException {

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try(JMSContext jmsContext=
            connectionFactory.createContext()) {
            while(true) {
                OrderDTO order = jmsContext.createConsumer(topic)
                    .receiveBody(OrderDTO.class);
                System.out.println("Order received:" + order);
            }
        }
    }
}
```

Il contesto dall'ambiente di esecuzione

Lookup oggetti administered: CF e destinazione

Con il contesto ottenuto

70

## Il consumatore standard JSE

```
public class OrderConsumer {
    public static void main(String[] args)
        throws NamingException {

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext=
            connectionFactory.createContext()) {
            while(true) {
                OrderDTO order = jmsContext.createConsumer(topic)
                    .receiveBody(OrderDTO.class);
                System.out.println("Order received:"+ order);
            }
        }
    }
}
```

Il contesto dall'ambiente di esecuzione

- › Lookup oggetti administered: CF e destinazione
- › Con il contesto ottenuto

Va in loop infinito...

71

## Il consumatore standard JSE

```
public class OrderConsumer {
    public static void main(String[] args)
        throws NamingException {

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext=
            connectionFactory.createContext()) {
            while(true) {
                OrderDTO order = jmsContext.createConsumer(topic)
                    .receiveBody(OrderDTO.class);
                System.out.println("Order received:"+ order);
            }
        }
    }
}
```

Il contesto dall'ambiente di esecuzione

- › Lookup oggetti administered: CF e destinazione
- › Con il contesto ottenuto

Va in loop infinito...

Crea un consumer

72

## Il consumatore standard JSE

```
public class OrderConsumer {
    public static void main(String[] args)
        throws NamingException {

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext=
            connectionFactory.createContext()) {
            while(true) {
                OrderDTO order = jmsContext.createConsumer(topic)
                    .receiveBody(OrderDTO.class);
                System.out.println("Order received:" + order);
            }
        }
    }
}
```

Il contesto dall'ambiente di esecuzione

- > Lookup oggetti administered: CF e destinazione
- > Con il contesto ottenuto
- > Va in loop infinito...

Crea un consumer

> Dal quale riceve messaggi

73

## Il consumatore standard JSE

```
public class OrderConsumer {
    public static void main(String[] args)
        throws NamingException {

        Context jndiContext = new InitialContext();

        ConnectionFactory connectionFactory = (ConnectionFactory)
            jndiContext.lookup("jms/javaee7/ConnectionFactory");
        Destination topic = (Destination)
            jndiContext.lookup("jms/javaee7/Topic");

        try (JMSContext jmsContext=
            connectionFactory.createContext()) {
            while(true) {
                OrderDTO order = jmsContext.createConsumer(topic)
                    .receiveBody(OrderDTO.class);
                System.out.println("Order received:" + order);
            }
        }
    }
}
```

Il contesto dall'ambiente di esecuzione

- > Lookup oggetti administered: CF e destinazione
- > Con il contesto ottenuto
- > Va in loop infinito...

Crea un consumer

> Dal quale riceve messaggi

> Che stampa a video

74

## Message-driven Bean

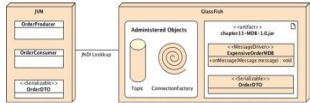
74

```

@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {
    public void onMessage(Message message)
    {
        try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                                + order);
        }catch(JMSException e) {
            e.printStackTrace();
        }
    }
}

```

› MDB con il nome del topic da usare



75

## Message-driven Bean

75

```

@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {
    public void onMessage(Message message)
    {
        try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                                + order.toString());
        }catch(JMSException e) {
            e.printStackTrace();
        }
    }
}

```

› MDB con il nome del topic da usare

› Configurazione: auto-ack

76

## Message-driven Bean

76

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {
    public void onMessage(Message message)
    {
        try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                               + order.toString());
        }catch(JMSException e) {
            e.printStackTrace();
        }
    }
}
```

- › MDB con il nome del topic da usare
- › Configurazione: auto-ack
- › ... e selettore messaggi con ammontare alto

77

## Message-driven Bean

77

```
@MessageDriven(mappedName = "jms/javaee7/Topic",
activationConfig = {
    @ActivationConfigProperty(propertyName="acknowledgeMode",
        propertyValue = "Auto-acknowledge"),
    @ActivationConfigProperty(propertyName="messageSelector",
        propertyValue = "orderAmount > 1000")
})
public class ExpensiveOrderMDB implements MessageListener {
    public void onMessage(Message message)
    {
        try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                               + order.toString());
        }catch(JMSException e) {
            e.printStackTrace();
        }
    }
}
```

- › MDB con il nome del topic da usare
- › Configurazione: auto-ack
- › ... e selettore messaggi con ammontare alto
- › Definizione dell'MDB

78

## Message-driven Bean

78

```

@MessageDriven(mappedName = "jms/javaee7/Topic",
    activationConfig = {
        @ActivationConfigProperty(propertyName="acknowledgeMode",
            propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName="messageSelector",
            propertyValue = "orderAmount > 1000")
    })
public class ExpensiveOrderMDB implements MessageListener {

    public void onMessage(Message message)
    {
        try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                               + order.toString());

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

```

- › MDB con il nome del topic da usare
- › Configurazione: auto-ack
- › ... e selettore messaggi con ammontare alto
- › Definizione dell'MDB
- › Riceve il messaggio

79

## Message-driven Bean

79

```

@MessageDriven(mappedName = "jms/javaee7/Topic",
    activationConfig = {
        @ActivationConfigProperty(propertyName="acknowledgeMode",
            propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName="messageSelector",
            propertyValue = "orderAmount > 1000")
    })
public class ExpensiveOrderMDB implements MessageListener {

    public void onMessage(Message message)
    {
        try{
            OrderDTO order = message.getBody(OrderDTO.class);
            System.out.println("Expensive order received:"
                               + order.toString());

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

```

- › MDB con il nome del topic da usare
- › Configurazione: auto-ack
- › ... e selettore messaggi con ammontare alto
- › Definizione dell'MDB
- › Riceve il messaggio
- › Lo stampa a video

80



## Organizzazione della lezione

80

- Meccanismi di affidabilità
- Message-Driven Beans
- Un esempio conclusivo
  - Il codice
  - Configurazione
  - I progetti
- Esercizi
- Conclusioni

81

## La configurazione degli oggetti administered

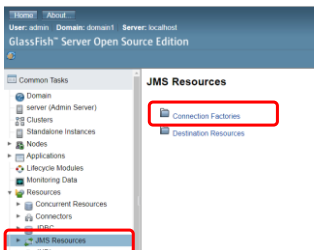
81

- Necessario che il server abbia:
  - Il Connection Factory
  - Il Topic
- Configurazione possibile da Console Web e da linea di comando
- Nella creazione del Topic necessario anche creare la destinazione fisica (creata di default con linea di comando)

82

## Configurazione via Web Console

82



83

## Configurazione via Web Console

83

### JMS Connection Factories

Java Message Service (JMS) connection factories are objects that allow an application to create other JMS objects programmatically. Click New to create a new connection factory. Click the name of a connection factory to modify its properties.

Connection Factories (2)					
Select	JNDI Name	Logical JNDI Name	Enabled	Resource Type	Description
<input type="checkbox"/>	jms/_defaultConnectionFactory	java.comp.DefaultJMSConnectionFactory	<input checked="" type="checkbox"/>	javax.jms.ConnectionFactory	

84

## Configurazione via Web Console

84

### New JMS Connection Factory

The creation of a new Java Message Service (JMS) connection factory also creates a connector connection pool for

#### General Settings

JNDI Name:

Resource Type:

Description:

Status: ☒ Enabled

85

## Configurazione via Web Console

85








86

## Configurazione via Web Console

87

### JMS Destination Resources

JMS destinations serve as the repositories for messages. Click **New** to create a new destination resource. Click the name of a destination resource to modify its properties.

Destination Resources				
				
<b>New</b>	<b>Edit</b>	<b>Delete</b>	<b>Refresh</b>	<b>Help</b>
<b>Select</b>	<b>JNDI Name</b>	<b>Enabled</b>	<b>Resource Type</b>	<b>Description</b>

87

## Configurazione via Web Console

87

Home / Administration / JMS / JMS Destinations

User: admin, Domain: domain-1, Server: localhost

GlassFish® Server Open Source Edition

Common Tools

- Domain
  - server (Admin Server)
    - Clusters
    - Stand-alone Instances
    - Nodes
      - Applications
      - Lifecycle Modules
      - Monitoring Data
      - Resources
        - Connection Resources
        - Connectors
        - JMS
          - JMS Resources
            - Connection Factories
              - InitialContextFactory
              - InitialContextFactory
              - Destination Resources
            - JNDI
              - Annotation Resources
              - Resource Adapter Configs

### New JMS Destination Resource

The creation of a new Java Message Service (JMS) destination resource also creates an control object resource.

**JNDI Name:**

**Physical Destination Name:**

**Resource Type:**

**Description:**

**State:** ☒ Enabled

**Advanced Properties (0)**

**Add Property** **Delete Property**

Property Name	Value	Description
No items found		

**OK** **Cancel**

88

## Configurazione via Web Console

88

### New JMS Destination Resource

The creation of a new Java Message Service (JMS) destination resource also creates an admin object resource.

JNDI Name: \*

Physical Destination Name: \*

Resource Type: \*

Description:

Status: ☒ Enabled

Additional Properties (0)

[Add Property](#) [Delete Properties](#)

Select	Name	Value	Description
No items found.			

89

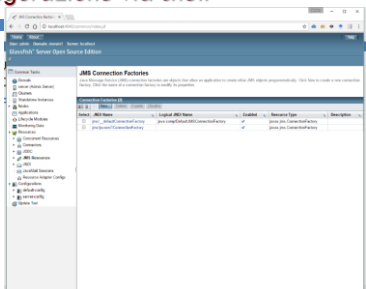
## Configurazione via shell

89

- Uso della shell asadmin

- Comandi utili:

```
asadmin create-
jms-resource --
restype
javax.jms.ConnectionFactory
jms/javaee7/ConnectionFactory
```

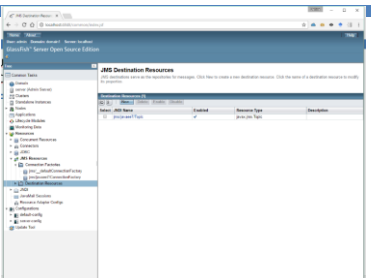


90

## Configurazione via shell

90

- Uso della shell asadmin
- Comandi utili:
  - `asadmin create-jms-resource --restype javax.jms.Topic jms/javaee7/Topic`

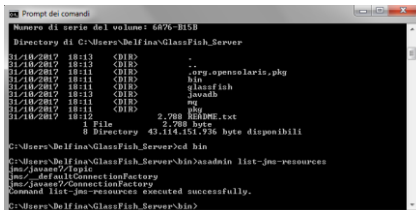


91

## Configurazione via shell

91

- Uso della shell asadmin
- Comandi utili:
  - `asadmin list-jms-resources`



92

## Organizzazione della lezione

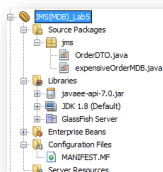
92

- Meccanismi di affidabilità
- Message-Driven Beans
- **Un esempio conclusivo**
  - Il codice
  - Configurazione
  - **I progetti**
- Esercizi
- Conclusioni

93

## Il Message-Driven Bean

93

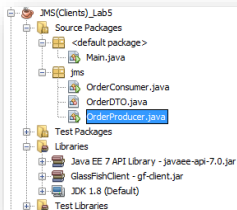


- › Progetto per EJB
- › Libreria di Java EE 7 aggiunta
- › Build & deploy

94

## Il Produttore

95

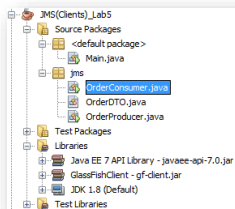


- › Progetto standard JSE
- › Libreria di Java EE 7 aggiunta
- › Libreria di Glassfish client aggiunta
- › Build e run
- › Parametro su linea di comando da configurazione di lancio (right-click, Properties, Run)

95

## Il Consumatore

95



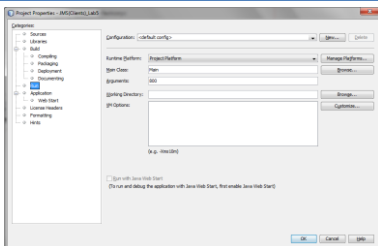
- › Progetto standard JSE (lo stesso di prima)
- › Run

96



## Esecuzione: set dei parametri

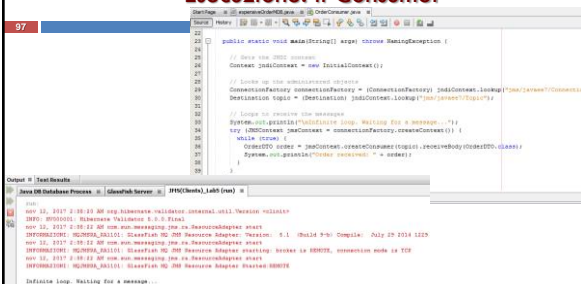
96



97

## Esecuzione: Il Consumer

97



98

## Esecuzione: Il Producer (800)

98

```

Output | Test Results
Java DB Database Process | GlassFish Server | JMS(Clients)_Lab5 (run) | JMS(Clients)_Lab5 (run) #2 |
RUN:
Sending message with amount = 800
nov 12, 2017 2:55:19 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
nov 12, 2017 2:55:20 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMAZIONI: MQMDSRA_RAI101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 5-b) Compile: July 29 2014 1229
nov 12, 2017 2:55:20 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMAZIONI: MQMDSRA_RAI101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
nov 12, 2017 2:55:20 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMAZIONI: MQMDSRA_RAI101: GlassFish MQ JMS Resource Adapter Started:REMOTE

Order sent : OrderDTO{orderId=1234, creationDate=Sun Nov 12 02:55:00 CET 2017, customerName='Serge Gainsbourg', totalAmount=800.0}
BUILD SUCCESSFUL (total time: 28 seconds)

```

99

## Esecuzione: MDB sul Container non riceve nulla

99

```

Output | Test Results
Java DB Database Process | GlassFish Server | JMS(Clients)_Lab5 (run) | JMS(Clients)_Lab5 (run) #2 |
Informazioni: Grizzly Framework 2.3.15 started in: 6ms - bound to [/0.0.0.0:8181]
Informazioni: Created HTTP listener http-listener-1 on host/port 0.0.0.0:8080
Informazioni: Grizzly Framework 2.3.15 started in: 98ms - bound to [/0.0.0.0:8080]
Informazioni: Initiating Jersey application, version Jersey: 2.10.4 2014-08-08 15:09:00...
Informazioni: Listening to REST requests at context: /management/domain.
Informazioni: Registered com.sun.enterprise.glassfish.bootstrap.osgi.EmbeddedOSGiGlassFishImpl@9967
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: Initializing Mojarra 2.2.7 ( 20140610-1547 https://svn.java.net/svn/mojarra-svn/tags/
Informazioni: Loading application [_adminui] at [/]
Informazioni: Loading application _adminui done in 12.477 ms
Informazioni: visiting unvisited references
Informazioni: visiting unvisited references
Informazioni: endpoint.determine.destinationType
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] private org.glassfish.jersey.gf.cdi.inte
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] public org.glassfish.jms.injection.JMSCC
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] org.glassfish.sse.impl.ServerSentEventCd
Informazioni: _JMS_MDB_Lab5 was successfully deployed in 2.484 milliseconds.

```

100

## Esecuzione: Il Consumer riceve il messaggio (800)

100

```

Output | Test Results
Java DB Database Process | GlassFish Server | JMS(Clients)_Lab5 (run) | JMS(Clients)_Lab5 (run) #2
run:
nov 12, 2017 2:55:00 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
nov 12, 2017 2:55:00 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMATION: MQJMSRA_RAI101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 9-b) Compile: July 29 2014 1229
nov 12, 2017 2:55:00 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMATION: MQJMSRA_RAI101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
nov 12, 2017 2:55:00 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMATION: MQJMSRA_RAI101: GlassFish MQ JMS Resource Adapter Started: REMOTE

Infinite loop. Waiting for a message...
Order received: OrderDTO{orderId=1234, creationDate=Sun Nov 12 02:55:00 CET 2017, customerName='Serge Gainsbourg', totalAmount=800.0}
  
```

101

## Esecuzione: Il Producer (2000)

101

```

Output | Test Results
Java DB Database Process | GlassFish Server | JMS(Clients)_Lab5 (run) | JMS(Clients)_Lab5 (run) #2
run:
Sending message with amount = 2000
nov 12, 2017 2:57:34 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
nov 12, 2017 2:57:35 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMATION: MQJMSRA_RAI101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 9-b) Compile: July 29 2014 1229
nov 12, 2017 2:57:35 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMATION: MQJMSRA_RAI101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
nov 12, 2017 2:57:35 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMATION: MQJMSRA_RAI101: GlassFish MQ JMS Resource Adapter Started: REMOTE

Order sent : OrderDTO{orderId=1234, creationDate=Sun Nov 12 02:57:34 CET 2017, customerName='Serge Gainsbourg', totalAmount=2000.0}
BUILD SUCCESSFUL (total time: 26 seconds)
  
```

102

## Esecuzione: Il Consumer riceve il messaggio (2000)

102

```

Output | Test Results
Java DB Database Process | GlassFish Server | JHS(Clients)_Lab5 (run) | JHS(Clients)_Lab5 (run) #2
run:
nov 12, 2017 2:53:09 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.0.0.Final
nov 12, 2017 2:53:09 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMATION: MQJMSRA_RAI101: GlassFish MQ JMS Resource Adapter: Version: 5.1 (Build 9-b) Compile: July 29 2014 1229
nov 12, 2017 2:53:09 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMATION: MQJMSRA_RAI101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCF
nov 12, 2017 2:53:09 AM com.sun.messaging.jmx.ra.ResourceAdapter start
INFORMATION: MQJMSRA_RAI101: GlassFish MQ JMS Resource Adapter Started-REMOTE

Infinite loop. Waiting for a message...
Order received: OrderTCO/orderId=1234, creationDate=Sun Nov 12 02:53:00 CET 2017, customerName='Serge Gainsbourg', totalAmount=800.0)
Order received: OrderTCO/orderId=1234, creationDate=Sun Nov 12 02:57:34 CET 2017, customerName='Serge Gainsbourg', totalAmount=2000.0)

```

103

## Esecuzione: l'MDB? ANCHE

103

```

Output | Test Results
Java DB Database Process | GlassFish Server | JHS(Clients)_Lab5 (run) | JHS(Clients)_Lab5 (run) #2
Information: Created HTTP listener http-listener-1 on host/port 0.0.0.0:8080
Information: Grizzly Framework 2.3.18 started on: N/A - bound to [/0.0.0.0:8080]
Information: Initializing Jersey application, Version Jersey: 2.12-4 2014-08-08 15:09:00...
Information: Listening to REST requests at context: /management/domain
Information: Registered com.sun.enterprise.glassfish.bootstrap.osgi.EmbeddedOSGiGlassFishDpi@1947022 as OSGi service registration: org.apache.felix.framework.Bea
Information: Visiting unvisited references
Information: Visiting unvisited references
Information: Initialising Resource 5.0.7 + 20140610-1547 http://www.java.net/em/mojaxra-em/tags/2.2.7813242 for context ""
Information: Loading application _wsmngui at 1/1
Information: Loading application _wsmngui done in 12.477 ms
Information: Visiting unvisited references
Information: Visiting unvisited references
Information: endpoint determine derivation type
WARN: WELD-00141: Observer method [BackedAnnotatedMethod] private org.glassfish.jersey.gf.cdi.internal.CdiComponentProvider.processAnnotatedType(@Observer Process
WARN: WELD-00141: Observer method [BackedAnnotatedMethod] public org.glassfish.jmx.ra.Extension.MQJMSRA_RAI101Extension.processAnnotatedType(@Observer ProcessAnnotatedType@
WARN: WELD-00141: Observer method [BackedAnnotatedMethod] org.glassfish.jmx.ra.Extension.MQJMSRA_RAI101Extension.processAnnotatedType(@Observer ProcessAnnotatedType@
Information: _JMS_WSP_Lab5 was successfully deployed in 2.816 milliseconds
Information: Registering order received: OrderTCO/orderId=1234, creationDate=Sun Nov 12 02:57:34 CET 2017, customerName='Serge Gainsbourg', totalAmount=2000.0)

```

104

## Organizzazione della lezione

104

- Meccanismi di affidabilità
- Message-Driven Beans
- Un esempio conclusivo
  - ▣ Il codice
  - ▣ Configurazione
  - ▣ I progetti
- Conclusioni



### Nelle prossime lezioni:

Service Oriented Architecture  
Web Services