

Messaging (1)

Corso di Laurea in Informatica, Programmazione Distribuita
Delfina Malandrino, dmalandrino@unisa.it
http://www.unisa.it/docenti/delfinamalandrino

1

Organizzazione della lezione

2

- Introduzione
- Messaging
- Java Messaging Service API
- Message Producers
- Message Consumers
- Conclusioni

2

Organizzazione della lezione

2

- Introduzione
- Messaging
- Java Messaging Service API
- Message Producers
- Message Consumers
- Conclusioni

3

Understanding messaging

4

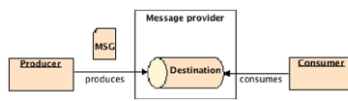
- MOM (Message-oriented middleware) è un software (provider) che permette lo scambio di messaggi asincroni fra sistemi eterogenei
- Può essere visto come un buffer che produce e consuma messaggi
- È intrinsecamente loosely coupled dal momento che i produttori non sanno chi è all'altra estremità del canale di comunicazione a consumare il messaggio
- Il produttore e il consumatore non devono essere disponibili contemporaneamente per comunicare

4

Understanding messaging

5

- Quando un messaggio viene inviato, il software che memorizza il messaggio e lo invia è detto **Provider (broker)**
- Il sender del messaggio è chiamato **Producer** e la locazione in cui il messaggio è memorizzato è detta **destinazione**
- La componente che riceve il messaggio è detta Consumer
- Ogni componente interessata ad un messaggio in una particolare destinazione può consumarlo



5

Java Message Service (JMS)

6

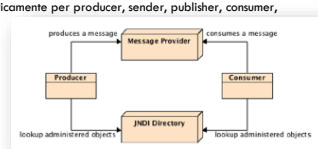
- In Java EE, l'API ch gestisce questi concetti è Java Message Service (JMS)
  - Set di interfacce e classi per
    - Connettersi ad un provider
    - Creare un messaggio
    - Inviare un messaggio
    - Ricevere un messaggio
- In un EJB container, Message-Driven Beans (MDBs) possono essere usati per ricevere messaggi in container-managed way

6

Messaging Architecture

7

- Componenti di un'architettura di messaging:
  - ***Un Provider***: componente necessaria per instradare messaggi
    - Gestisce buffering e delivery dei messaggi
  - ***Clients***: una qualunque applicazione Java o una componente che produce o consuma messaggi per/da un provider
    - Il termine "Client" si usa genericamente per producer, sender, publisher, consumer, receiver, subscriber

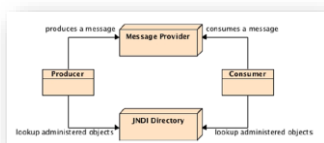


7

Messaging Architecture

8

- Componenti di un'architettura di messaging:
  - ***Messages***: oggetti che i client inviano/ricevono dal provider
  - ***Administered objects***: oggetti (connection factories e destinazioni) fornite attraverso JNDI lookups o injection



8

Messaging Architecture

9

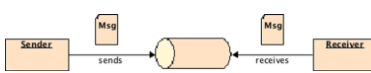
- Il Provider permette comunicazione asincrona fornendo una destinazione dove i messaggi possono essere mantenuti finché non vengono instradati verso un client
- Esistono due differenti tipi di destination:
  - *Point-to-point (P2P) model: la destination è chiamata coda*
    - Il client inserisce un messaggio in coda, mentre un altro client riceve il messaggio
    - Una volta fatto acknowledge, il message provider rimuove il messaggio dalla coda
  - *Publish-subscribe (pub-sub) model:* la destination è chiamata *topic*
    - Il client pubblica un messaggio con un topic, e tutti i sottoscrittori al topic riceveranno il messaggio

9

Point-to-Point model

10

- Il messaggio viaggia da un singolo producer verso un singolo consumer



- Ogni messaggio viene inviato ad una specifica coda, ed il receiver riceve il messaggio dalla coda
- La coda mantiene i messaggi finché non vengono consumati o scadono

10

Point-to-Point model

11

- Nel modello P2P, esiste un solo receiver per ogni messaggio
- Una coda può avere consumers multipli…
  - …ma quando un receiver consuma il messaggio, questo viene tolto dalla coda, e nessun altro receiver potrà consumarlo

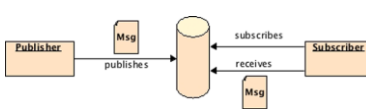
- Il modello P2P non garantisce che i messaggi siano instradati in un particolare ordine
  - Un provider può riceverli in un particolare ordine, o random, o in qualunque altro ordine

11

Publish-Subscribe Model

12

- Mel modello pub-sub model, un singolo messaggio è inviato da un singolo producer per potenzialmente diversi consumers
- Il modello è costruito intorno ai concetti di topics, publishers, e subscribers
  - I Consumers sono chiamati *subscribers*
    - Hanno necessità di sottoscriversi ad un topic
    - Forniscono il meccanismo di subscribing/unsubscribing, che occorre dinamicamente

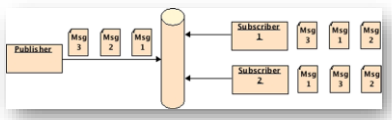


12

Publish-Subscribe Model

13

- Il topic conserva i messaggi fino a quando non vengono distribuiti a tutti i subscribers
- Dipendenza temporale fra publisher e subscriber
  - I subscribers NON ricevono i messaggi inviati PRIMA della loro sottoscrizione e, se il subscriber è inattivo per un periodo di tempo determinato, esso non riceve messaggi vecchi quando diventa nuovamente attivo
- Multipli subscribers possono consumare lo stesso messaggio
- Utile per broadcast-type applications: singolo messaggio recapitato a diversi consumatori



13

Administered Objects

14

- Oggetti che si configurano amministrativamente, e non programmatically
- Il provider permette di configurare questi oggetti e li rende disponibili nello spazio dei nomi JNDI
- Come JDBC datasources questi oggetti vengono creati solo una volta. I due tipi di oggetti amministrati sono:
  - Connection factory: usato dai clienti per creare una connessione a una destinazione
  - Destinazioni: punti di distribuzione del messaggio che ricevono, mantengono, e distribuiscono messaggi
    - Le destinazioni possono essere code (P2P) o topic (pub-sub)

14

Message-Driven Beans

15

- Message-Driven Beans (MDBs) sono message consumer asincroni eseguiti in un EJB container
- L'EJB container si occupa dei servizi (transactions, security, concurrency, message acknowledgment, etc.), mentre l'MDB si occupa di consumare messaggi
- MDBs sono stateless
  - L'EJB container può avere numerose istanze, eseguite in concorrenza per processare messaggi provenienti da diversi producers
- In generale gli MDBs sono in ascolto su una destination (queue o topic) e, quando il messaggio arriva, lo consuma e lo processa
- Poiché sono stateless, gli MDBs non mantengono stato attraverso invocazioni separate

15

Message-Driven Beans

16

- Gli MDBs rispondono a messaggi ricevuti dal container….
- … laddove gli stateless session beans rispondono a richieste client attraverso una interfaccia appropriata
  - local, remote, o no-interface

16

Java Messaging Service API

17

- JMS è un insieme di standard Java API che permette alle applicazioni di creare, inviare, ricevere e leggere messaggi in maniera asincrona
- Definisce un insieme di interfacce e classi per la comunicazione con altri message providers
- JMS è analogo a JDBC:
  - JDBC permette la connessione a differenti databases (Derby, MySQL, Oracle, DB2, etc.)
  - JMS permette la connessione a diversi providers (OpenMQ, MQSeries, SonicMQ, etc.)

17

Connection Factory

21

- Connection factories sono administered objects
- L'interfaccia `javax.jms.ConnectionFactory` incapsula i parametri definiti da un amministratore
- Per usare un administered object come una ConnectionFactory, il client deve eseguire una JNDI lookup (o usare injection)
- Per esempio, nel seguente code fragment si ottiene un JNDI InitialContext object e lo si usa per fare lookup di una connectionFactory attraverso il suo JNDI name:

```
Context ctx = new InitialContext();
ConnectionFactory ConnectionFactory = ➥
                (ConnectionFactory) ctx.lookup("jms/javaee7/ConnectionFactory");
```

21

Destination

22

- Una destination è un administered object che contiene provider-specific configuration information come ad esempio un destination address
- Questo meccanismo è nascosto al JMS client attraverso l'uso dell'interfaccia `javax.jms.Destination`
- Come per le connection factory, una JNDI lookup è necessaria per restituire tali oggetti:

```
Context ctx = new InitialContext();
Destination queue = (Destination) ctx.lookup("jms/javaee7/Queue");
```

22

Simplified API

23

- Le altre interfacce:
  - JMSContext: active connection ad un MS provider e single-threaded context per inviare e ricevere messaggi
  - JMSProducer: oggetto creato da un JMSContext per inviare messaggi ad una coda o ad un topic
  - JMSConsumer: oggetto creato da un JMSContext per ricevere messaggi inviati ad una coda o ad un topic

23

JMS Context API

24

Property	Description
`void start()`	Starts (or restarts) delivery of incoming messages
`void stop()`	Temporarily stops the delivery of incoming messages
`void close()`	Closes the `JMSContext`
`void commit()`	Commits all messages done in this transaction and releases any locks currently held
`void rollback()`	Rolls back any messages done in this transaction and releases any locks currently held
`BytesMessage createBytesMessage()`	Creates a `BytesMessage` object
`MapMessage createMapMessage()`	Creates a `MapMessage` object
`Message createMessage()`	Creates a `Message` object
`ObjectMessage createObjectMessage()`	Creates an `ObjectMessage` object
`StreamMessage createStreamMessage()`	Creates a `StreamMessage` object
`TextMessage createTextMessage()`	Creates a `TextMessage` object
`Topic createTopic(String topicName)`	Creates a `Topic` object
`Queue createQueue(String queueName)`	Creates a Queue object
`JMSConsumer createConsumer(Destination destination)`	Creates a `JMSConsumer` for the specified destination
`JMSConsumer createConsumer(Destination destination, String messageSelector)`	Creates a `JMSConsumer` for the specified destination, using a message selector
`JMSProducer createProducer()`	Creates a new `JMSProducer` object which can be used to configure and send messages
`JMSContext createContext(int sessionMode)`	Creates a new `JMSContext` with the specified session mode

24

JMS Producer API

25

Property	Description
`get/set[Type]Property`	Sets and returns a message property where `[Type]` is the type of the property and can be `Boolean`, `Byte`, `Double`, `Float`, `Int`, `Long`, `Object`, `Short`, `String`
`JMSProducer clearProperties()`	Clears any message properties set
`Set<String> getPropertyNames()`	Returns an unmodifiable `Set` view of the names of all the message properties that have been set
`boolean propertyExists(String name)`	Indicates whether a message property with the specified name has been set
`get/set[Message Header]`	Sets and returns a message header where `[Message Header]` can be `DeliveryDelay`, `DeliveryMode`, `JMSCorrelationID`, `JMSReplyTo`, `JMSType`, `Priority`, `TimeToLive`
`JMSProducer send(Destination destination, Message message)`	Sends a message to the specified destination, using any send options, message properties and message headers that have been defined
`JMSProducer send(Destination destination, String body)`	Sends a `TextMessage` with the specified body to the specified destination

25

JMS Consumer API

26

Property	Description
`void close()`	Closes the `JMSConsumer`
`Message receive()`	Receives the next message produced
`Message receive(long timeout)`	Receives the next message that arrives within the specified timeout interval
`<T> T receiveBody(Class<T> c)`	Receives the next message produced and returns its body as an object of the specified type
`Message receiveNoWait()`	Receives the next message if one is immediately available
`void setMessageListener(MessageListener listener)`	Sets the `MessageListener`
`MessageListener getMessageListener()`	Gets the `MessageListener`
`String getMessageSelector()`	Gets the message selector expression

26

Organizzazione della lezione

27

- Introduzione
- Messaging
- Java Messaging Service API
- Message Producers
- Message Consumers
- Conclusioni

27

Writing Message Producers

28

- Con le nuove API di JMS 2.0, la scrittura di produttori e consumatori diventa meno verbosa
- Rimangono gli administered objects `ConnectionFactory` e `Destination`
- A seconda se si è fuori o dentro un container, si usa JNDI lookup oppure injection
- Tre esempi:
  - Produttore fuori da un container
  - Produttore in un container
  - Produttore in un container con CDI

28

Writing Message Producers

28

- Con le nuove API di JMS 2.0, la scrittura di produttori e consumatori diventa meno verbosa
- Rimangono gli administered objects `ConnectionFactory` e `Destination`
- A seconda se si è fuori o dentro un container, si usa JNDI lookup oppure injection
- Tre esempi:
  - **Produttore fuori da un container**
  - Produttore in un container
  - Produttore in un container con CDI

29

Produttore fuori da un Container

29

- Un oggetto `JMSProducer` viene creato da un `JMSContext` e usato per inviare messaggi
- I passi da seguire:
  - Ottenere una connection factory ed una coda con JNDI
  - Creare un `JMSContext` usando la factory
  - Creare un `JMSProducer` usando il contesto
  - Inviare un messaggio usando il metodo send() del producer

30

Produttore fuori da un Container

```
public class Producer {
 public static void main(String[] args)
 {  try{
   Context jndiContext = new InitialContext();

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

   Destination queue = (Destination)
                       jndiContext.lookup("jms/javaee7/Queue");

   try(JMSContext context =
         connectionFactory.createContext()) {
       context.createProducer().send(queue,
                          "Text message sent at"+
                                new  Date());
   }
  }catch(NamingException e) {
     e.printStackTrace();
  }
 }
}
```

Prende il contesto da JNDI

Listing 13.4

La classe Producer produce un Message in una Queue

31

Produttore fuori da un Container

```
public class Producer {
 public static void main(String[] args)
 {  try{
   Context jndiContext = new InitialContext();

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

   Destination queue = (Destination)
                    jndiContext.lookup("jms/javaee7/Queue");

   try(JMSContext context =
         connectionFactory.createContext()) {
       context.createProducer().send(queue,
                        "Text message sent at"+
                              new  Date());
   }
  }catch(NamingException e) {
      e.printStackTrace();
  }
 }
}
```

Listing 13.4

- Prende il contesto da JNDI
- Cerca per l'administered object di ConnectionFactory

La classe Producer produce un Message in una Queue

32

Produttore fuori da un Container

```
public class Producer {
 public static void main(String[] args)
 {  try{
   Context jndiContext = new InitialContext();

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

   Destination queue = (Destination)
                    jndiContext.lookup("jms/javaee7/Queue");

   try(JMSContext context =
         connectionFactory.createContext()) {
       context.createProducer().send(queue,
                        "Text message sent at"+
                              new  Date());
   }
  }catch(NamingException e) {
      e.printStackTrace();
  }
 }
}
```

Listing 13.4

- Prende il contesto da JNDI
- Cerca per l'administered object di ConnectionFactory
- Preleva la coda dal JNDI

La classe Producer produce un Message in una Queue

33

Produttore fuori da un Container

```
public class Producer {
 public static void main(String[] args)
 {  try{
   Context jndiContext = new InitialContext();

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

   Destination queue = (Destination)
                       jndiContext.lookup("jms/javaee7/Queue");

   try(JMSContext context =
         connectionFactory.createContext()) {
       context.createProducer().send(queue,
                          "Text message sent at"+
                              new  Date());

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

Listing 13.4

- Prende il contesto da JNDI
- Cerca per l'administered object di ConnectionFactory
- Preleva la coda dal JNDI
- Cerca di creare il contesto (se fallisce chiude il contesto)

La classe Producer produce un Message in una Queue

34

Produttore fuori da un Container

```
public class Producer {
 public static void main(String[] args)
 {  try{
   Context jndiContext = new InitialContext();

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

   Destination queue = (Destination)
                       jndiContext.lookup("jms/javaee7/Queue");

   try(JMSContext context =
         connectionFactory.createContext()) {
       context.createProducer().send(queue,
                          "Text message sent at"+
                              new  Date());
   }
  }catch(NamingException e) {
      e.printStackTrace();
  }
 }
}
```

Listing 13.4

- Prende il contesto da JNDI
- Cerca per l'administered object di ConnectionFactory
- Preleva la coda dal JNDI
- Cerca di creare il contesto (se fallisce chiude il contesto)
- Crea il produttore dal contesto e invia un messaggio

La classe Producer produce un Message in una Queue

35

Writing Message Producers

36

- Con le nuove API di JMS 2.0, la scrittura di produttori e consumatori diventa meno verbosa
- Rimangono gli administered objects `ConnectionFactory` e `Destination`
- A seconda se si è fuori o dentro un container, si usa JNDI lookup oppure injection
- Tre esempi:
  - Produttore fuori da un container
  - Produttore in un container
  - Produttore in un container con CDI

36

Produttore in un Container

37

Tramutando il produttore in un EJB

Listing 13.5

```
@Stateless
public class ProducerEJB
{

 @Resource(lookup ="jms/javaee7/ConnectionFactory")
 private ConnectionFactory connectionFactory;

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

 public void sendMessage() {
   try(JMSContext context =
                    connectionFactory.createContext()) {
       context.createProducer().send(queue,
                  "Text message sent at"+ new Date());
   }
 }
}
```

Un EJB stateless

ProducerEJB in esecuzione all'interno di un Container usando @Resource

37

Produttore in un Container

38

Tramutando il produttore in un EJB

Listing 13.5

```
@Stateless
public class ProducerEJB {

 @Resource(lookup ="jms/javaee7/ConnectionFactory")
 private ConnectionFactory connectionFactory;

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

 public void sendMessage() {
   try(JMSContext context =
                        connectionFactory.createContext()) {
       context.createProducer().send(queue,
                    "Text message sent at "+new Date());
   }
 }
}
```

- Un EJB stateless
- La classe

ProducerEJB in esecuzione all'interno di un Container usando @Resource

38

Produttore in un Container

39

Tramutando il produttore in un EJB

Listing 13.5

```
@Stateless
public class ProducerEJB
{
 @Resource(lookup ="jms/javaee7/ConnectionFactory")
 private ConnectionFactory connectionFactory;

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

 public void sendMessage() {
   try(JMSContext context =
                        connectionFactory.createContext()) {
       context.createProducer().send(queue,
                     " Text message sent at "+new Date());
   }
 }
}
```

- Un EJB stateless
- La classe
- Annotazione di una risorsa da ricercare su JNDI per la connection factory

ProducerEJB in esecuzione all'interno di un Container usando @Resource

39

Produttore in un Container

40

Tramutando il produttore in un EJB

Listing 13.5

```
@Stateless
public class ProducerEJB {

 @Resource(lookup ="jms/javaee7/ConnectionFactory")
 private ConnectionFactory connectionFactory;

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

 public void sendMessage() {
   try(JMSContext context =
                        connectionFactory.createContext()) {
       context.createProducer().send(queue,
                   "Text message sent at "+new Date());
   }
 }
}
```

- Un EJB stateless
- La classe
- Annotazione di una risorsa da ricercare su JNDI per la connection factory
- …che viene dichiarata

ProducerEJB in esecuzione all'interno di un Container usando @Resource

40

Produttore in un Container

41

Tramutando il produttore in un EJB

Listing 13.5

```
@Stateless
public class ProducerEJB {

 @Resource(lookup ="jms/javaee7/ConnectionFactory")
 private ConnectionFactory connectionFactory;

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

 public void sendMessage() {
   try(JMSContext context =
                        connectionFactory.createContext()) {
       context.createProducer().send(queue,
                   "Text message sent at "+new Date());
   }
 }
}
```

- Un EJB stateless
- La classe
- Annotazione di una risorsa da ricercare su JNDI per la connection factory
- …che viene dichiarata
- Lo stesso per la coda

ProducerEJB in esecuzione all'interno di un Container usando @Resource

41

Produttore in un Container

42

Tramutando il produttore in un EJB

Listing 13.5

```
@Stateless
public class ProducerEJB
{
 @Resource(lookup ="jms/javaee7/ConnectionFactory")
 private ConnectionFactory connectionFactory;

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

 public void sendMessage() {
   try(JMSContext context =
                    connectionFactory.createContext()) {
       context.createProducer().send(queue,
                   "Text message sent at "+ new Date());
   }
 }
}
```

- Un EJB stateless La
- classe
- Annotazione di una risorsa da ricercare su JNDI per la connection factory
- …che viene dichiarata Lo
- stesso per la coda
- Rifattorizzazione del comportamento in un metodo di business

ProducerEJB in esecuzione all'interno di un Container usando @Resource

42

Writing Message Producers

43

- Con le nuove API di JMS 2.0, la scrittura di produttori e consumatori diventa meno verbosa
- Rimangono gli administered objects `ConnectionFactory` e `Destination`
- A seconda se si è fuori o dentro un container, si usa JNDI lookup oppure injection
- Tre esempi:
  - Produttore fuori da un container
  - Produttore in un container
  - **Produttore in un container con CDI**

43

Produttore in un Container con CDI

```
public class Producer {

    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    private JMSContext context;

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

    public void sendMessage() {
        context.createProducer().send(queue,
                "Text message sent at "+new Date());
    }
}
```

Listing 13.6

- Il container fa inject della factory di connessioni JMS

Managed Bean che produce un Message using @Inject

44

Produttore in un Container con CDI

```
public class Producer {

    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    private JMSContext context;

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

    public void sendMessage() {
        context.createProducer().send(queue,
                "Text message sent at "+new Date());
    }
}
```

Listing 13.6

- Il container fa inject della factory di connessioni JMS
- Specificando il tipo

Managed Bean che produce un Message using @Inject

45

Produttore in un Container con CDI

```
public class Producer {

    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    private JMSContext context;

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

    public void sendMessage() {
        context.createProducer().send(queue,
                "Text message sent at "+new Date());
    }
}
```

Listing 13.6

- Il container fa inject della factory di connessioni JMS
- Specificando il tipo
- E gestendone completamente il ciclo di vita

Managed Bean che produce un Message using @Inject

46

Produttore in un Container con CDI

```
public class Producer {

    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    private JMSContext context;

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

    public void sendMessage() {
        context.createProducer().send(queue,
                "Text message sent at "+new Date());
    }
}
```

Listing 13.6

- Il container fa inject della factory di connessioni JMS
- Specificando il tipo
- E gestendone completamente il ciclo di vita
- Coda ricercata su JNDI

Managed Bean che produce un Message using @Inject

47

Produttore in un Container con CDI

```
public class Producer {

    @Inject
    @JMSConnectionFactory("jms/javaee7/ConnectionFactory")
    private JMSContext context;

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

    public void sendMessage() {
        context.createProducer().send(queue,
                "Text message sent at "+ new Date());
    }
}
```

Listing 13.6

- Il container fa inject della factory di connessioni JMS
- Specificando il tipo
- E gestendone completamente il ciclo di vita
- Coda ricercata su JNDI
- Metodo di business

Managed Bean che produce un Message using @Inject

48

Organizzazione della lezione

49

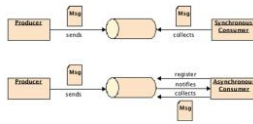
- Introduzione
- Messaging
- Java Messaging Service API
- Message Producers
- Message Consumers
- Conclusioni

49

Tipi di Consumer

50

- **Sincroni:** il ricevitore esplicitamente preleva il messaggio dalla destinazione, invocando `receive()`
- **Asincroni:** il ricevitore si registra all'evento di arrivo di un messaggio e implementa `MessageListener`, in modo che all'arrivo del messaggio viene invocato il suo metodo `onMessage()`



50

51

Consumer sincrono

51

Consumer sincrono

```
public class Consumer {
 public static void main(String[] args)
 {  try{
     Context jndiContext = new InitialContext();

    ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
    Destination queue = (Destination)
                        jndiContext.lookup("jms/javaee7/Queue");

    try(JMSContext context =
                        connectionFactory.createContext()) {
     while(true) {
      String message =
      context.createConsumer(queue).receiveBody(String.class);
     }
    }
  }catch(NamingException e) {
      e.printStackTrace();
  }
 }
}
```

- Preleva il contesto JNDI

Listing 13.7

La classe Consumer consuma Messages in modo sincrono

52

Consumer sincrono

```
public class Consumer {
 public static void main(String[] args)
 {  try{
    Context jndiContext = new InitialContext();

    ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
    Destination queue = (Destination)
                        jndiContext.lookup("jms/javaee7/Queue");

    try(JMSContext context =
                        connectionFactory.createContext()) {
     while(true) {
      String message =
      context.createConsumer(queue).receiveBody(String.class);
     }
    }
  }catch(NamingException e) {
      e.printStackTrace();
  }
 }
}
```

- Preleva il contesto JNDI
- Cerca gli oggetti amministrati: factory per le connessioni . . .

Listing 13.7

La classe Consumer consuma Messages in modo sincrono

53

Consumer sincrono

```
public class Consumer {
 public static void main(String[] args)
 {  try{
    Context jndiContext = new InitialContext();

    ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
    Destination queue = (Destination)
                        jndiContext.lookup("jms/javaee7/Queue");

    try(JMSContext context =
                        connectionFactory.createContext()) {

     while(true) {
      String message =
      context.createConsumer(queue).receiveBody(String.class);
     }
    }
  }catch(NamingException e) {
      e.printStackTrace();
  }
 }
}
```

Listing 13.7

- Preleva il contesto JNDI
- Cerca gli oggetti amministrati: factory per le connessioni …
- … e coda

La classe Consumer consuma Messages in modo sincrono

Consumer sincrono

```
public class Consumer {
 public static void main(String[] args)
 {  try{
    Context jndiContext = new InitialContext();

    ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
    Destination queue = (Destination)
                        jndiContext.lookup("jms/javaee7/Queue");

    try(JMSContext context =
                        connectionFactory.createContext()) {

     while(true) {
      String message =
      context.createConsumer(queue).receiveBody(String.class);
     }
    }
  }catch(NamingException e) {
      e.printStackTrace();
  }
 }
}
```

Listing 13.7

- Preleva il contesto JNDI
- Cerca gli oggetti amministrati: factory per le connessioni …
- … e coda
- Acquisisce il contesto (con ciclo di vita gestito da try-with-resources)

La classe Consumer consuma Messages in modo sincrono

Consumer sincrono

```
public class Consumer {
 public static void main(String[] args)
 {  try{
    Context jndiContext = new InitialContext();

    ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
    Destination queue = (Destination)
                        jndiContext.lookup("jms/javaee7/Queue");

    try(JMSContext context =
                        connectionFactory.createContext()) {

     while(true) {
      String message =
      context.createConsumer(queue).receiveBody(String.class);
     }
    }
   }catch(NamingException e) {
      e.printStackTrace();
   }
  }
}
```

Listing 13.7

- Preleva il contesto JNDI
- Cerca gli oggetti amministrati: factory per le connessioni …
- … e coda
- Acquisisce il contesto (con ciclo di vita gestito da try-with-resources)
- Ciclo infinito (*busy waiting*!!!)

La classe Consumer consuma Messages in modo sincrono

56

Consumer sincrono

```
public class Consumer {
 public static void main(String[] args)
 {  try{
    Context jndiContext = new InitialContext();

    ConnectionFactory connectionFactory = (ConnectionFactory)
          jndiContext.lookup("jms/javaee7/ConnectionFactory");
    Destination queue = (Destination)
                        jndiContext.lookup("jms/javaee7/Queue");

    try(JMSContext context =
                        connectionFactory.createContext()) {

     while(true) {
      String message =
      context.createConsumer(queue).receiveBody(String.class);
     }
    }
   }catch(NamingException e) {
      e.printStackTrace();
   }
  }
}
```

Listing 13.7

- Preleva il contesto JNDI
- Cerca gli oggetti amministrati: factory per le connessioni …
- … e coda
- Acquisisce il contesto (con ciclo di vita gestito da try-with-resources)
- Ciclo infinito (*busy waiting*!!!)
- Prova a ricevere

La classe Consumer consuma Messages in modo sincrono

57

58

Consumer asincrono

58

Consumer asincrono

```
public class Listener implements MessageListener {
 public static void main(String[] args) {
  try{

   Context jndiContext = new InitialContext();

   ConnectionFactory connectionFactory = (ConnectionFactory)
         jndiContext.lookup("jms/javaee7/ConnectionFactory");
   Destination queue = (Destination)
                     jndiContext.lookup("jms/javaee7/Queue");
   try(JMSContext context =
         connectionFactory.createContext()) {
    context.createConsumer(queue).setMessageListener(
                                        new Listener());
   }
  }catch(NamingException e) {
      e.printStackTrace();
  }
 }

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

Implementa interfaccia per listener

Listing 13.8

Il Consumer è un Message Listener

59

Consumer asincrono

```
public class Listener implements MessageListener {
 public static void main(String[] args) {
  try{

   Context jndiContext = new InitialContext();

   ConnectionFactory connectionFactory = (ConnectionFactory)
         jndiContext.lookup("jms/javaee7/ConnectionFactory");
   Destination queue = (Destination)
                  jndiContext.lookup("jms/javaee7/Queue");
   try(JMSContext context =
       connectionFactory.createContext()) {
    context.createConsumer(queue).setMessageListener(
                                        new Listener());
   }
  }catch(NamingException e) {
     e.printStackTrace();
  }
 }

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

- Implementa interfaccia per listener
- Metodo statico

Listing 13.8

Il Consumer è un Message Listener

60

Consumer asincrono

```
public class Listener implements MessageListener {
 public static void main(String[] args) {
  try{

   Context jndiContext = new InitialContext();

   ConnectionFactory connectionFactory = (ConnectionFactory)
         jndiContext.lookup("jms/javaee7/ConnectionFactory");
   Destination queue = (Destination)
                  jndiContext.lookup("jms/javaee7/Queue");
   try(JMSContext context =
       connectionFactory.createContext()) {
    context.createConsumer(queue).setMessageListener(
                                        new Listener());
   }
  }catch(NamingException e) {
     e.printStackTrace();
  }
 }

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

- Implementa interfaccia per listener
- Metodo statico
- Preleva il contesto JNDI

Listing 13.8

Il Consumer è un Message Listener

61

Consumer asincrono

```
public class Listener implements MessageListener {
 public static void main(String[] args) {
  try{

   Context jndiContext = new InitialContext();

   ConnectionFactory connectionFactory = (ConnectionFactory)
         jndiContext.lookup("jms/javaee7/ConnectionFactory");
   Destination queue = (Destination)
                  jndiContext.lookup("jms/javaee7/Queue");
   try(JMSContext context =
        connectionFactory.createContext()) {
    context.createConsumer(queue).setMessageListener(
                                        new Listener());
   }
  }catch(NamingException e) {
     e.printStackTrace();
  }
 }

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

Listing 13.8

- Implementa interfaccia per listener
- Metodo statico
- Preleva il contesto JNDI
- Lookup per gli oggetti administered

Il Consumer è un Message Listener

62

Consumer asincrono

```
public class Listener implements MessageListener {
 public static void main(String[] args) {
  try{

   Context jndiContext = new InitialContext();

   ConnectionFactory connectionFactory = (ConnectionFactory)
         jndiContext.lookup("jms/javaee7/ConnectionFactory");
   Destination queue = (Destination)
                  jndiContext.lookup("jms/javaee7/Queue");
   try(JMSContext context =
        connectionFactory.createContext()) {
    context.createConsumer(queue).setMessageListener(
                                        new Listener());
   }
  }catch(NamingException e) {
     e.printStackTrace();
  }
 }

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

Listing 13.8

- Implementa interfaccia per listener
- Metodo statico
- Preleva il contesto JNDI
- Lookup per gli oggetti administered
- Con la factory di connessioni JMS

Il Consumer è un Message Listener

63

Consumer asincrono

```
public class Listener implements MessageListener {
 public static void main(String[] args) {
  try{

   Context jndiContext = new InitialContext();

   ConnectionFactory connectionFactory = (ConnectionFactory)
         jndiContext.lookup("jms/javaee7/ConnectionFactory");
   Destination queue = (Destination)
                     jndiContext.lookup("jms/javaee7/Queue");
   try(JMSContext context =
        connectionFactory.createContext()) {
    context.createConsumer(queue).setMessageListener(
                                        new Listener());
   }
  }catch(NamingException e) {
     e.printStackTrace();
  }
 }

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

- Implementa interfaccia per listener
- Metodo statico
- Preleva il contesto JNDI
- Lookup per gli oggetti administered
- Con la factory di connessioni JMS
- Crea un oggetto di tipo Listener e lo registra

Listing 13.8

Il Consumer è un Message Listener

64

Consumer asincrono

```
public class Listener implements MessageListener {
 public static void main(String[] args) {
  try{

   Context jndiContext = new InitialContext();

   ConnectionFactory connectionFactory = (ConnectionFactory)
         jndiContext.lookup("jms/javaee7/ConnectionFactory");
   Destination queue = (Destination)
                     jndiContext.lookup("jms/javaee7/Queue");
   try(JMSContext context =
        connectionFactory.createContext()) {
    context.createConsumer(queue).setMessageListener(
                                        new Listener());
   }
  }catch(NamingException e) {
     e.printStackTrace();
  }
 }

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

- Implementa interfaccia per listener
- Metodo statico
- Preleva il contesto JNDI
- Lookup per gli oggetti administered
- Con la factory di connessioni JMS
- Crea un oggetto di tipo Listenere lo registra

All'arrivo di un messaggio, questa e` la callback

Listing 13.8

Il Consumer è un Message Listener

65

Organizzazione della lezione

66

- Introduzione
- Messaging
- Java Messaging Service API
- Message Producers
- Message Consumers
- Conclusioniv

Nelle prossime lezioni:

Ancora Java Messaging Service

66