JMS

JAVA MESSAGE SERVICE

Panoramica

JMS è una **specifica** parte di **J2EE**.

Descrive come programmi Java possano creare, spedire, ricevere, messaggi (enterprise) in un ambiente distribuito.

Comunicazione scarsamente accoppiata (tempo e spazio)

Consegna affidabile
- I messaggi sono consegnati integri ed esattamente una volta

Specifica aperta e integrabile
- Controllo
- Gestione/allocazione
- Sicurezza/permessi
- Etc…

Applicazione JMS

Client JMS
- Programmi java che spediscono/ricevono messaggi su «code» JMS

Messaggi

Oggetti amministrati
- Oggetti preconfigurati (dall'amministrazione del sistema) ed esposti dal sistema ai client
- «primitivi» ConnectionFactory, Queue, Topic

Provider JMS
- Sistema di messaggistica conforme a JMS
- Implementa funzionalità amministrative (creazione, gestione etc di oggetti amministrati)

Indirizzi e JNDI

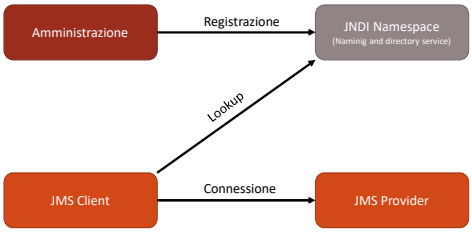
JMS non definisce una sintassi standard per gli indirizzi

Ma poggia su servizi di **naming** e **directory**

In particolare su Java Naming and Directory Interface

Vantaggi di JDNI:
- Informazioni amministrative sono oggetti java
- Astrae da specifiche implementazioni/provider
- Gestione uniforme delle risorse
- Ubiquità del servizio
- Semplificazione di deployement e configurazione

Amministrazione



Domini di messaggi

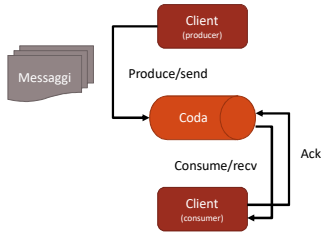
Punto-a-punto (PTP)
- Fondati sul concetto di coda di messaggi
- Ogni messaggi ha esattamente un consumatore

Publish-Subscribe (PPS)
- Organizzati in topic
- I messaggi hanno più consumatori
- I consumatori possono dichiarare filtri (message selector)

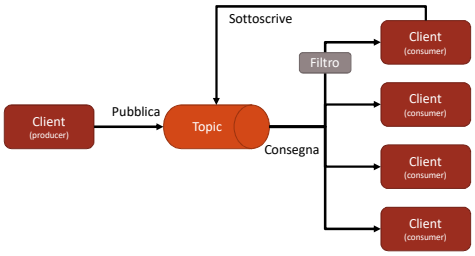
Messaggi possono contenere sia oggetti che xml

In entrambi i casi la ricezione può essere bloccante o meno:
- Metodi di ricezione (timeout opzionale)
- Ascoltatori (MessageListener.onMessage)

Punto-a-punto



Pub/Sub



Messaggi

Intestazioni
- Identificazione e routing
- Campi e valori proprietari (vendor-specific)
- Campi e valori applicazione (application-specific)
- Strutturati come dizionari (tipicamente)

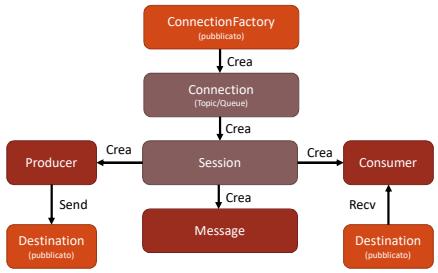
Proprietà
- Opzionali
- Metadati
- Definiti dall'utente

Corpo
- I veri dati (a vari livelli di opacità per JMS)
- Ci sono diversi tipologie di corpo (prossima slide)

Tipi di messaggi

Message Type	Contains	Some Methods
TextMessage	String	getText,setText
MapMessage	set of name/value pairs	setString,setDouble,setLong,getDouble,getString
BytesMessage	stream of uninterpreted bytes	writeBytes,readBytes
StreamMessage	stream of primitive values	writeString,writeDouble,writeLong,readString
ObjectMessage	serialize object	setObject,getObject

Produzione



Esempio: fase comune

Creazione del contesto:

```
InitialContext context = new InitialContext();
```

Ricerca del ConnectionFactory:

```
ConnectionFactory factory = context.lookup("pippo");
```

Creazione della connessione:

```
Connection connection = factory.createConnection();
```

Creazione della sessione:

```
Session session = connection.createSession(
    false, Session.AUTO_ACKNOWLEDGE);
```

Ricerca della destinazione:

```
Destination dest1 = (Queue)
    context.lookup("/jms/coda1");
Destination dest2 = (Topic)
    context.lookup("/jms/topic1");
```

Esempio: produttore

Setup (fase comune)
- Lookup, connection, session …

Creazione del consumeatore:
- `MessageProducer producer = session.createProducer(destination);`

Invio di un messaggio:
- `Message msg = session.createTextMessage();`
- `msg.setText("this the payload of a text message");`
- `producer.send(m);`

Chiusura della connessione:
- `connection.close();`

Esempio: consumatore

Setup (fase comune)
- Lookup, connection, session …

Creazione del produttore:
- `MessageConsumer consumer = session.createConsumer(destination);`

Avvio ricezione:
- `connection.start();`

Ricezione di un messaggio:
- `Message msg = consumer.receive();`

Sospensione ricezione:
- `connection.stop();`

Esempio: consumatore (async)

Setup (fase comune)
- Lookup, connection, session …

Creazione del consumatore:
- `MessageConsumer consumer = session.createConsumer(destination);`

Registrazione ascoltatore:
- `consumer.setMessageListener(listener);`

Un ascoltatore implementa l'interfaccia MessageListener:
-
```
MessageListener listener = new MessageListener(){
    public void onMessage(Message msg){
      …
    }
}
```

Funzionalità avanzate

Sottoscrizioni durevoli
- I sottoscrittori ricevono solamente messaggi pubblicati durante il loro ciclo di vita
- Le sottoscrizioni durevoli mantengono i messaggi in assenza del sottoscrittore
- Hanno una data di scadenza (per ovvie ragioni)

Request/Reply
- Basate su queue/topic temporanei
-
```
Session.createTemporaryQueue()
producer=session.createProducer(msg.getJMSReplyTo());reply=
session.createTextMessage("reply");
reply.setJMSCorrelationID(msg.getJMSMessageID);
producer.send(reply);
```


Funzionalità avanzate

Sessioni transazionali
- Operazioni miste topic/queue ma incapsulate in una transazione (atomica,…)
-
```
session=connection.createSession(true,0)
// … //
void onMessage(Message m) {
    try {
            Message m2=processOrder(m);
            publisher.publish(m2);
            session.commit();
     } catch(Exception e) {
            session.rollback();
     }
}
```


Funzionalità avanzate

Consegna persistente/non-persistente
- `producer.setDeliveryMethod(DeliveryMode.NON_PERSISTENT);`
- `producer.send(msg, DeliveryMode.NON_PERSISTENT ,3,1000);`

Selettori di messaggi/filtri
- Sintassi SQL-Like per accedere agli header (che sono dictionary-style)
-
```
subscriber  = session.createSubscriber(topic,
     "priority > 6 AND type = 'alert' ");
```
- Sia PTP che Pub/Sub (cf. filtri)

JMS e EJB

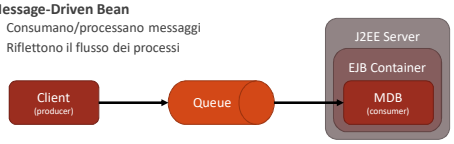
EJB è un componente server che incapsula la business logic di un'applicazione J2EE

EJB offre supporto allo sviluppo di applicazioni enterprise distribuite

- Container: forniscono servizi a livello di systema
  - autorizzazione, autenticazione, gestione delle transazioni etc…
- Beans: contengono la logica di controllo

Message-Driven Bean

- Consumano/processano messaggi
- Riflettono il flusso dei processi



Esempio di MDB

```
public class MB implements
    MessageDrivenBean, MessageListener{
  public void ejbCreate(){}
  public void ejbRemove(){}
  public void setMessageDrivenContext(
                    MessageDrivenContext mdc){…}
  pubic void onMessage(Message m){
    …
    try{
      if (m instanceof TextMessage)
       System.out.println("MDB: "+m.getText());
      }catch(JMSException e){
        …
      }
}}
```


Esercizi

1. Catene di processi e flussi di lavoro
2. Replicazione di servizi