JMS

RIO PR1

TKTL s2008

Tekijät:

Aki Valkama
Lauri Savolainen
Niklas Jahnsson

Sisällys

0 Johdanto

Tämä ohje on tehty rinnakkaisohjelmoinnin kurssiprojektina syksyllä 2008. Yritimme parhaamme mukaan havainnollistaa ja tuoda esiin Java Messaging Servicen eri osa-alueet. Annamme yliopistolle luvan käyttää ohjetta opetustarkoituksiin ja jatkokehittelyyn. Toivottavasti ohjeesta on apua muillekin kuin meille.

1 Menetelmän perusidea

Java Messaging Service eli JMS on javan versio viesti-orientoituneista väliohjelmista (message-oriented middleware tai MOM). Ensimmäinen versio julkaistiin vuonna 1998 ja wikipedian mukaan sen jälkeen on julkaistu vain yksi päivitys versio JMS 1.1 vuonna 2002, jonka dokumentaatioon ohjeemme pitkälti ottaen perustuu (Wikipedia, 2). Yleisesti ottaen tämän tyyppisten ohjelmien tarkoituksena on antaa ohjelmoijalle mahdollisuus erilaisten sovellusten tekemiseen tarjoamalla ominaisuuksia, jotka parantavat ohjelmien ”yhteistoiminnallisuutta, 'kannettavuutta' ja joustavuutta mahdollistamalla ohjelman jakamisen useille erilaisille alustoille ” (Wikipedia, 1). Kantavana ideana on eristää ja ”kapseloida” eri alustoissa ja käyttöjärjestelmissä toimivat ohjelmat saman paketin alle, jolloin osaa ohjelmien palveluista on mahdollista etäkäyttää.

Idea on siis tarjota ohjelmoijalle helppo työkalu viestejä hyödyntävien sovellusten kehittelyyn. Tarkemmin ajateltuna JMS tarjoaa yhteisen tavan java ohjelmille luoda, lähettää, vastaanottaa ja lukea yritysviestijärjestelmän viestejä (JMS specification, s. 13). Termi yritysviestijärjestelmä on ilmeisesti peräisin siitä tarpeesta, jota varten MOM:t on luotu. Alkuperäinen idea niiden kehittämiseen on lähtenyt tarpeesta luoda keinot kommunikaation vanhojen, uusien ja tulevaisuudessa kehitettävien järjestelmien välille.

JMS ei määrittele ominaisuuksia viestien eheyden ja turvallisuuden takaamiseksi, vaan JMS-toimittajat (provider) voivat tehdä oman implementoinnin niistä.

1.1 JMS:n komponentit

Seuraavassa on lueteltuna JMS:n eri komponenti, joita käyttämällä viestijärjestelmä voidaan kasata kokoon. Lista komponenteista on Wikipediasta, ja täydentävien tietojen osalta lähteet on mainittu erikseen (Wikipedia, 2).

1.1.1 Provider

Viestitysjärjestelmä, joka implementoi JMS:n ja toimittaa sekä pitää yllä valvovia ja kontrolloivia toimintoja. (Haase, s. 15)

1.1.2 Client

Kuvaa niitä yleisesti niitä ohjelmia, jotka tuottavat ja kuluttavat viestejä. Lisähuomautuksena todettakoon, että termi JMS client viittaa JMS:llä ja javalla toteutettuun clienttiin kun taas pelkkä client voi olla toteutettu myös muilla tavoin. Kuitenkin clientin täytyy toteuttaa tietyt rajapinnat JMS providerin kanssa, jotta se voisi toimia JMS-järjestelmässä.

1.1.3 Producer

Tuottaja-client, joka luo ja lähettää viestejä

1.1.4 Consumer

Kuluttaja-client, joka vastaanottaa viestejä.

1.1.5 Message

JMS-viestit koostuvat otsikosta, ominaisuuksista ja pääosasta. Otsikot ovat JMS:n määrittelemiä, joista järjestelmänvalvoja voi ohittaa joitain osia. Ominaisuuksilla voidaan tavallaan lisätä viestiin tarpeellisia otsikoita. Viestin pääosaan JMS antaa vielä viisi erilaista tietorakennetta/toteutustapaa.

1.1.6 Queue

Jono, jossa sijaitsevat lähetyt viestit, joita ei ole vielä luettu. Viestit luetaan jonosta siinä järjestyksessä, mikä on ensimmäisenä lähetetty jonoon. Kun viesti on luettu jonosta, se poistetaan. Jono pitää viestin tallessa, mikäli kuluttaja-client ei ole aktiivinen, joten sen ei tarvitse huolehtia viestien katoamisesta.

1.1.7 Topic

Viestien luokitteluun käytettävä yläkäsite. Seuraavaksi esiteltävässä viestejä topiccien avulla julkaisevassa Pub/Sub -mallissa käytettävä apuväline.

1.2 Viestitysmallit

JMS sovellus voi käyttää point-to-point (PTP), publish-and-subscribe (Pub/Sub) tai yhdistellä kahta juuri mainittua viestien välitystapaa.

PTP -mallilla tarkoitetaan jonoihin perustuvaa viestitysjärjestelmää, jossa client lähettää viestinsä aina johonkin tiettyyn jonoon (JMS specification, s. 75). PTP-mallin pääasiallinen ongelma onkin siis se, kuinka client löytää haluamansa jonon, lähettää viestin siihen ja ottaa viestejä vastaan. On hyvä huomata, että PTP-mallissa jokaisella viestillä on vain yksi kuluttaja, koska viesti lähetetään aina johonkin tiettyyn jonooon. Lisäksi viestin vastaanottaja aina ilmoittaa prosessuoituansa loppuun asti saamansa viestin, ja viestin vastaanottajan kulutusajankohdalla ei ole merkitystä. (Haase, s.17)

Pub/Sub -malli taas hoitaa viestien julkaisun (= produce) ja tilaamisen (=consume) solmujen tunnetun sisältö-pohjaisen hierarkian avulla. Nämä tunnetut solmut (=topic) hoitavat viestien välittämisen niitä tilanneille, ja sopeutuvat aina viestitystarpeen mukaan (JMS specification, s. 79). Ideana on siis se, että tuotettuaan viestin client lähettää sen julkaistavaksi, jolloin luodaan solmu viestin julkaisemista varten. Tämän jälkeen solmussa olevat viestit ovat olemassa vain siihen asti kunnes viesti on välitetty kaikille sen tilanneille clienteille. Tämän seurauksena syntyykin kaksi eroavaisuutta PTP-malliin: jokaisella viestillä voi olla useita kuluttajia ja kuluttaja voi kuluttaa tilaamansa solmun viestejä vain silloin kuin kun on tilannut solmun. JMS:ssä aikariippuvuutta voidaan hiukan heikentää käyttämällä ”kestäviä tilauksia” (durable subscriptions). (Haase, s. 17-18)

1.3 Viestien kuluttaminen

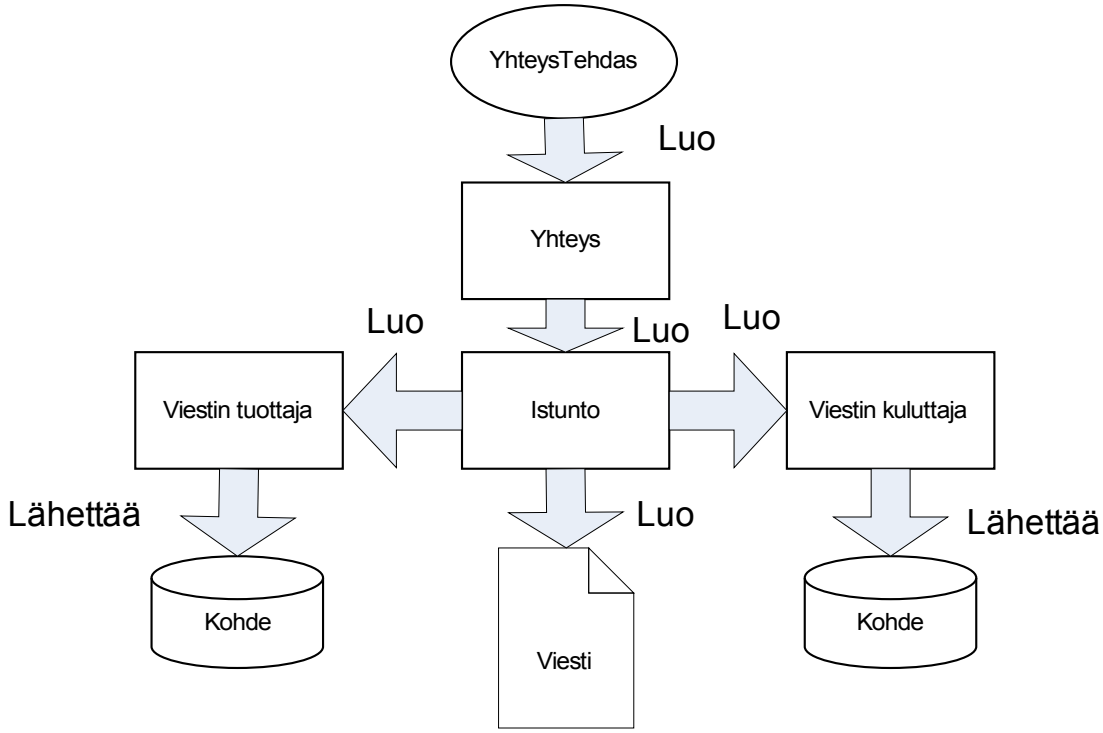
JMS mahdollistaa viestien kuluttamisen sekä asynkronisesti että synkronisesti. Näiden kahden ero on hyvin periaatteellinen kuluttaja-tuottaja -ongelman ratkaisuvaihtoehtojen ero: busy-wait versus viestikapulan välitys. Asiaa voidaan selventää seuraavasti kurssilta tutuilla käsitteillä. Synkroninen kuluttuja käyttää receive-metodia, jolloin kuluttaja on bysy-wait loopissa ja jatkuvasti yrittää hakea viestiä. Asynkroninen kuluttaja taas käyttää ”viestin kuuntelijaa” (message listener), joka sitten kertoo kuluttujalle, milloin kuluttaminen kannattaa aloittaa. (Haase, s. 19)

2 Käyttöohje esimerkkien avulla

Seuraavassa ovat kaksi esimerkkiä PTP-mallista: ensimmäinen synkroninen ja jälkimmäinen asynkroninen. Yleisellä tasolla JMS-sovellus koostuu clienteista, jotka on toteutettu JMS:llä tai jollain muulla MOM:lla, viesteistä, providerista ja clienttien käyttöön annetuista objekteista, joita ovat yhteystekijä (conncetionFactor) ja määränpäistä (destination) (JMS specification, s. 21 ja s. 23). Esimerkkejä, joista seuraavaksi hieman selvennetään yksinkertaisimpia, lukija voi löytää lisää esimerkiksi JMS specificationin luvusta 9. Seuraava esitys mukailee hyvin pitkälti luvun 9 esitystä tarkoituksenaan selventää ja erityisesti suomentaa käytettyjä käsitteitä. Suomennokset ovat pitkälti omien mieliemme tuotteita kuitenkin pitäen mielessä alalla yleensä käytössä olleet termit.

2.1 Perusesimerkki synkronisesta viestien kuluttamisesta

Tämän käyttöohjeen avulla kuka tahansa voi helposti luoda omat viestien synkroniset lähettäjät ja vastaanottajat. Oletamme kuitenkin, että sitä ennen järjestelmän ylläpitäjä on luonut mallit yhteystekijälle (ConnectionFactory) ja jonolle(Queue).



Aivan aluksi viestittämistä varten on luotava yhteystekijä clientille, joka toimii lähettäjän ja vastaanottajan yhdistävä tekijänä. Tarkemmin sanottuna ilman sitä ei voida luoda yhteyttä ja istuntoa tuottajan ja kuluttajan välille. Oletamme seuraavaksi, että ylläpitäjä on luonut yhteystekijän (YhteysTehdas) ja jonon (DataJono), jota JMS:n client voi käyttää.

```
ConnectionFactory yhteystekija;
Context viestitys = new InitialContext();
yhteystekija = (ConnectionFactory)messaging.lookup("YhteysTehdas");
Queue jono;
jono = (Queue)messaging.lookup("DataJono");
```

Kun yhteystekijä on luotu, client voi luoda tämän jälkeen yhteyden. Kun yhteys on luotu, client voi luoda istunnon. Istunnon boolean-arvo määrittää onko istunto päällä vai ei. Istunnossa myös luodaan tuottaja, eli Producer, joka lähettää viestejä.

```
Connection yhteys;
```

```
yhteys = yhteystekija.createConnection();
Session istunto;
istunto = yhteys.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

Tuottajan luonnissa määritellään kohde, eli se mihin viestit lähetetään.

```
MessageProducer lahettaja;
lahettaja = istunto.createProducer(jono);
```

Viestin kuluttaja on kohde johon lähetetyt viestit vastaanotetaan. Tässä tapauksessa kohde on ”jono”. On hyvä nyt huomata, että tällä tavalla luotu viestien kuluttaja kuluttaa viestinsä synkronisesti eli kyseessä on PTP-malli, joka esiteltiin aikaisemmin.

```
MessageConsumer vastaanottaja;
vastaanottaja = session.createConsumer(jono);
```

Viestien lähetys kuluttajalle voidaan aloittaa antamalla aloituskäsky Connectionille.

```
yhteys.start();
```

Tämän jälkeen tulee määritellä se minkälainen viesti halutaan lähettää. Yksinkertaisena esimerkkinä voidaan käyttää Stringiä, jolloin määrittely tapahtuisi seuraavasti:

```
String teksti = ”MOIKKA, MITES MENEE???”
TextMessage viesti

viesti = istunto.createTextMessage();
viesti.setText(teksti);
```

Sen jälkeen jäljellä on vain viestin lähettäminen eteepäin. Se tapahtuu yksinkertaisella käskyllä:

```
lahettaja.send(viesti);
```

Viestin vastaanottaja vastaanottaa viestin suorittamalla seuraavan rivin.

```
TextMessage viesti
viesti = (TextMessage)vastaanottaja.receive();
```

Jotta vastaanottajan ei tarvitsisi odottaa aina pelkästään loopissa, voidaan vastaanottajalle antaa parametrina aika kuinka kauan viestinsaapumista tulee odottaa. Tämä tapahtuu seuraavasti:

```
TextMessage = (TextMessage)vastaanottaja.receive(1000);
```

, jossa annettu parametri on millisekunneissa (1 sekunti = 1000 millisekuntia). Saatuaan viestin vastaanottaja purkaa sen seuraavalla komennolla:

```
String teksti
teksti = viesti.getText();
```

Olemme nyt luoneet vastaanottajan, lähettäjän ja viestin synkroniseen ympäristöön sekä lähettäneet ja vastaaottaneet yhden viestin. Nyt on paikallaan huomata, että oikeastihan vastaanottaja ja lähettäjä ovat kaksi täysin erillistä ohjelmaa. Sen takia suosittelemmekin tarkastelemaan Sunin tarjoamia esimerkkiohjelmia. Niissä samat asiast on tehty hiukan monimutkaisemmin ottamalle mukaan tarkistukset, mutta perusideat ovat aivan samoja kuin yllä. Kiinnostunut löytää kyseiset ohjelmat SimpleQueueReceiver.java ja SimpleQueueSender.java tarkastelemalla esimerkiksi lähteissä olevaa Kim Haasen JMS-tutorialia. Seuraavaksi katsomme kuitenkin myös hiukan asynkronista ympäristöä, ja sitä kuinka siinä toimiminen muuttaa ohjelmaa.

2.2 Asynkroninen viestien kuluttaminen

Olennaisena erona viestien asynkroniseen kuluttamiseen se, että otamme receive-metodin sijasta käyttöön javassa paljon käytetyt ”kuuntelijat” (*Listener). Nyt siis vastaanottajan on implementoitava MessageListener-luokka pitämällä sisällään metodin onMessage(Message viesti). Seuraava koodin pätkä yrittää konkretisoida saman asian.

```
public class tiedonKuuntelija implements MessageListener {
        …
        public void onMessage(Message viesti) {
                teksti = viesti.getText();
                …
        }
}
```

Tätä luokkaa sitten voidaan käyttää hyödyksi vastaanottaja ohjelmassa seuraavasti.

```
tiedonKuuntelija kuuntelija = new tiedonKuuntelija();
vastaanottaja.setMessageListener(kuuntelija);
```

Jonka jälkeen teksti voidaan hakea vaikka erillisellä luokkaan tiedonKuuntelija kirjoitetulla metodilla getText(). Nyt olemme käyneet läpi myös viestien asynkronisen vastaanottamisen.

4 Yhteenveto

JMS on ollut olemassa jo useamman vuoden, mutta viimeisin versio vuodelta 2002 näyttää ainakin aikajanalla olevan aika kaukana nykyhetkestä. Omasta mielestämme JMS vaikuttaa aivan käytettävältä välineeltä, mutta on kuitenkin vaikeampi alkaa arvioimaan sen todellista käytettävyyttä ilman oikeaa kokemusta sen käytöstä ”oikeissa” sovelluksissa. Se, että uusia versioita ei ole ilmestynyt vuoden 2002 jälkeen voi olla merkki siitä, että kaikki on kerrankin tehty kerralla oikein tai sitten käyttäjiä ei ole ollut tarpeeksi antamaan palautetta JMS:n ongelmista. Näyttäisi siltä, että tulevaisuudessa alemman tason AMQP (Advanced Message Queuing Protocol, http://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol) protokolla syrjäyttäisi JMS-implementoinnit, koska tällä tavoin järjestelmätoimittajien ohjelmien yhteentoimivuus paranee huomattavasti (Vinoski).

5 Lähteet

Haase, Kim, Java Message Service API tutorial, Sun Microsystems, Inc., California, 2002

Sun Microsystems, JMS specification, Sun Micosystems, April 12, 2002

Wikipedia: 1] http://en.wikipedia.org/wiki/Message_Oriented_Middleware, 3.12.2008, vapaa suomennus

2] http://en.wikipedia.org/wiki/Java_Message_Service, 3.12.2008

Vinoski: http://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol / References / Vinoski, .S.