비동기식 로그서비스 구현을 통한 Message-Driven POJO(MDP) 기술연구

장의진 O 백종현
대우정보시스템
{joomanba O, baegjh}@disc.co.kr

A Study on Message-Driven POJO(MDP) by Developing an Asynchronous Logging Service

Eui-Jin Jang O Baeg Jong Hyun
Daewoo Information Systems Co., Ltd.

요 약

효과적인 어플리케이션 로그처리는 어플리케이션 개발 및 운영 시에 매우 중요한 요소 중에 하나이다. 파일이나 콘솔을 이용하여 동기식으로 처리되는 일반적인 로그 서비스는 동시에 대량의 로그를 처리해야 할 경우 시스템에 많은 부하를 주게 되는 문제점이 있다. 이를 해결하기 위해서는 클라이언트가 지연 없이 로그를 남길 수 있는 비동기식 로그서비스가 필요하다. 이 논문에서는 비동기식 로그서비스 구현을 통해서 Message-Driven POJO(MDP) 구현 기술을 소개하고 최근에 소개된 EJB 3.0 Message Driven Bean(MDB) 기술과 비교 분석해 보도록 한다.

1. 서 론

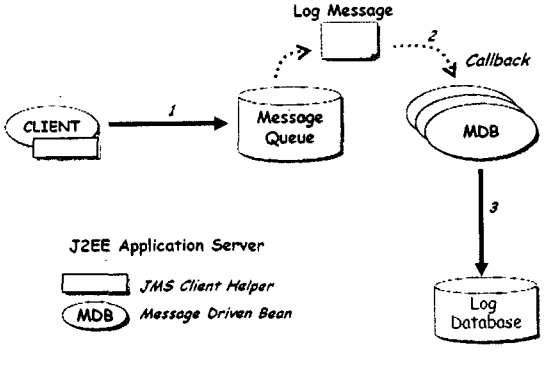
어플리케이션 개발 및 운영에 있어서 효과적인 로그처리의 중요성은 다시 강조해도 지나치지 않다. 현재 로그처리를 위한 Log4J, J2SE logging, Jakarta Commons Logging 등과 같은 다양한 프레임웍들이 존재하고 있다. 그러나 이들 프레임웍은 로그를 동기식으로 처리하도록 되어 있어서 대량의 로그처리가 동시에 진행될 경우 시스템 병목현상을 피할 수 없게 된다. 이를 해결하기 위해서 비동기 방식으로 로그를 처리해야 하는데 여기에는 Java Message Service(JMS) 기술이 적용될 수 있다. JMS는 비동기식으로 메시지를 처리하기 위하여 다양한 메시지 큐 서버와 연동할 수 있는 표준 API를 제공하는 기술이다. JMS 클라이언트는 JMS API를 이용하여 메시지큐로 메시지를 보내게 되고 이 메시지큐에 보내진 메시지는 Message Driven Bean(MDB)에 의하여 처리되어 진다. 그러나 Message Driven Bean는 표준 EJB 컴포넌트로 배포되어지기 위해서는 반드시 J2EE 어플리케이션 서버가 필요하게 된다. 그러나 웹로직 서버와 같은 J2EE 어플리케이션 서버가 필요 없는 어플리케이션에서 비동기식 로그처리를 위해서 J2EE 어플리케이션 서버가 필요한 MDB를 이용해야 한다는 것은 어플리케이션 아키텍처 상 큰 제약조건이 될 수 있다.

이 논문에서 MDB을 사용하지 않고 Plan Old Java Object(POJO) 을 이용하여 비동기식 메시지를 처리하는 Message-Driven POJO(MDP) 기술에 대하여 소개하고 최근 발표된 EJB 3.0 MDB와 기술비교를 통해서 두 기술을 장단점을 분석해 보도록 한다.

2. 비동기식 로그서비스

기존에 구현된 EJB 2.0 MDB 기반의 비동기식 로그서비스를 Spring Framework 기반의 Message-Driven POJO 기술과 EJB 3.0 MDB 기술을 이용하여 각각 구현해보기로 한다. 비동기식 로그서비스는 아래와 같이 동작하게 된다.

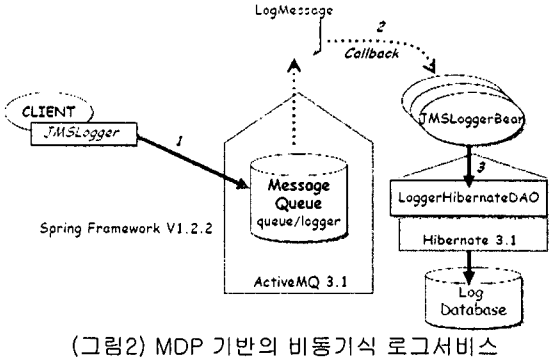
1. 로그 서비스 클라이언트는 JMS 클라이언트 클래스를 이용하여 메시지큐에 로그 메시지를 전달한다.
2. J2EE 어플리케이션 서버는 메시지큐에 전달된 로그 메시지를 callback 방식을 통해서 MDB에 전달한다.
3. MDB는 로그메시지 정보를 로그 데이터베이스에 저장한다.



(그림1) MDB 기반의 비동기식 로그서비스

3. Message-Driven POJO를 이용한 비동기식 로그서비스 구현

위에서 MDB 기반에서 구현된 비동기식 로그서비스를 POJO 기반으로 변환하기 위해서 Spring Framework 1.2.2 와 오픈소스 메시지큐 서버인 ActiveMQ 3.1를 사용하였다. Spring Framework은 EJB 컨테이너와 유사하게 POJO 객체에 트랜잭션, 보안, 비동기식 메시지 처리 같은 다양한 서비스 속성을 제공할 수 있는 IoC 기반의 경량 컨테이너를 제공하는 프레임웍이다. 아래 그림 2는 Spring Framework이 ActiveMQ 와 연동하여 POJO 기반에서 비동기 방식으로 로그메시지를 처리하는 모습이다.



(그림2) MDP 기반의 비동기식 로그서비스

소스 1 에서 보듯이 Message-Driven POJO 는 MDB 2.0 과 달리 javax.ejb.MessageDrivenBean 인터페이스를 구현해 줄 필요 없이 javax.jms.MessageListener 인터페이스의 onMessage() 메소드만을 구현해 주면 된다.

```
package jcf.logging.jms;

import javax.jms.*;

public class JMSLoggerBean implements MessageListener{

 public void onMessage(Message message) {

 try {
   ObjectMessage objMsg =  (ObjectMessage)message;
   LogMessage logMsg  = (LogMessage)objMsg.getObject();
   persistMessage(logMsg);
 }catch (Exception ex) {
```

(소스1) Message-Driven POJO

다음으로는 위에서 개발한 JMSLoggerBean POJO 객체을 ActiveMQ 서버와 연동해야 한다. ActiveMQ 와 연결하기 위해서는 ActiveMQ 에서 제공하는 JCAContainer 를 사용해야 한다. 소스 2 와 같이 spring context XML 파일에 JCAContainer 를 설정하면 된다. 여기서 serverUrl 값에는 ActiveMQ 가 구동되고 있는 서버 IP 와 접속포트 값을 적어주면 된다.

```
<bean id="activeMQContainer"
 class="org.activemq.jca.JCAContainer">
  <property name="workManager">
   <bean id="workManager"
    class="org.activemq.work.SpringWorkManager" />
  </property>

  <property name="resourceAdapter">
   <bean id="activeMQResourceAdapter"
    class="org.activemq.ra.ActiveMQResourceAdapter">
    <property name="serverUrl"
     value="tcp://localhost:61616" />
```

(소스2) Spring Context XML 파일

마지막으로 소스4와 같이 위에서 JMSLoggerBean 과 JCAContainer를 연결시켜 주면 Message-Driven POJO 개발이 완료된다.

```
<bean id="JMSLogger" factory-method="addConnector"
   factory-bean="activeMQContainer">
 <property name="activationSpec">
  <bean class="org.activemq.ra.ActiveMQActivationSpec">
   <property name="destination" value="queue/logger" />
   <property name="destinationType"
    value="javax.jms.Queue" />
  </bean>
 </property>
 <property name="ref" value="jmsLoggerBean" />
</bean>
```

(소스3) Spring Context XML 파일

4. EJB 3.0 MDB를 이용한 비동기식 로그서비스 구현

최근에 발표된 EJB 3.0 MDB 와 Message-Driven POJO 기술을 비교해 보기 위해서 위에서 개발된 POJO 기반의 비동기식 로그서버스를 MDB 3.0 스펙에 맞추어 재개발해 보았다. EJB 3.0 을 지원하는 JBoss AS 4.0R2 에 JBoss EJB 3.0R1 을 설치해서 MDB 를 개발하였다. 아래의 소스 5 는 EJB 3.0 MDB 스펙에 맞추어 작성한 JMSLoggerBean 구현한 모습이다.

```
@MessageDriven(activateConfig =
{
 @ActivationConfigProperty(propertyName="destinationType",
   propertyValue="javax.jms.Queue"),
 @ActivationConfigProperty(propertyName="destination",
   propertyValue="queue/logger")
})

public class JMSLoggerBean implements MessageListener{

 public void onMessage(Message message) {
```

(소스 4) Message-Driven Bean

MDB 3.0 를 구현하기 위해서는 먼저 MessageListener 의 onMessage() 메소드를 구현해 주면 된다. 메시지가 메시지큐에 도착하게 되면 MDB 컨테이너에서는 onMessage() 메소드를 callback 방식으로 호출하게 된다. EJB 3.0 에서 가장 특징적인 부분은 기존은 XML 방식의 Deployment Descriptor 를 대신하여 annotation 방식을 사용할 수 있다는 점이다. 위의 소스 4 에서 보는 것과 같이 @MessageDriven 이라는 annotation 을 이용하여 MDB 라고 설정하여 JBoss 에 배포하게 되면 JBoss 에서는 자동적으로 JMSLoggerBean 을 MDB 로 배포하게 된다.

5. 결론

지금까지 비동기식 로그서비스를 Spring Framework 기반의 Message-Driven POJO 방식과 EJB 3.0 기반의 MDB 방식으로 구현해 보았다. 과거 EJB 2.0 의 MDB 와 비교해 볼 때 위의 두 기술을 이용하면 매우 간편하게 비동기식 컴포넌트를 개발할 수 있게 되었다. 중요한 것은 두 기술의 차이점을 정확하게 인지하여 어플리케이션 개발 및 운영환경에 적합한 최적의 기술을 선택하는 것이다. 지금부터 두 기술을 차이점을 비교 분석해 본다.

* EJB Container 필요 여부

EJB 기반의 MDB 일 경우 반드시 EJB Container 가 필요하다. 만약 어플리케이션에서 MDB 이외의 다른 EJB 컴포넌트를 사용하지 않는다면 MDB 만을 사용하기 위해서 J2EE 어플리케이션 서버를 사용해야 하는 것은 부담이 될 수 있다. 이에 반해서 Spring Framework 기반의 Message-Driven POJO 일 경우에는 J2EE 어플리케이션 서버를 필요가 없다. EJB 를 전혀 사용하지 않는 일반 어플리케이션일 경우 Spring 프레임웍 기반의 Message-Driven POJO 기술을 사용하는 것이 합리적이다.

* 비동기식 컴포넌트 구현 편의성 여부

EJB 3.0 MDB 와 Spring 프레임웍 기반의 Message-Driven POJO 의 개발 편의성을 비교하면 EJB 3.0 기반의 MDB 가 개발 편의성이 더 높다고 할 수 있다. EJB 3.0 기반의 MDB 일 경우는 Deployment Descriptor XML 파일 대신에 Annotation 방식을 이용하여 간편하게 개발할 수 있다.

Spring 프레임웍 방식의 Message-Driven POJO 기술은 EJB 2.0 보다는 편리하게 MDB 를 구현할 수 있지만 개발자가 XML 을 이용하여 JCAContainer 와 Message-Driven POJO 를 선언해 주어야 한다. 하지만 JCAContainer 설정은 전체 어플리케이션에서 한번만 해주면 되기 때문에 실제로는 개발자에게 큰 부담을 주지 않는다.

* 유연한 컴포넌트 개발 여부

Spring 프레임웍 기반의 MDB 는 단순한 POJO 기반으로 되어 있어서 Spring 프레임웍 IoC 및 AOP 서비스를 완벽하게 사용할 수 있다. 이는 EJB 를 사용하지 않고 POJO 만을 가지고도 완벽한 트랜잭션 및 보안 서비스를 받을 수 있다는 것을 의미한다. EJB 3.0 기반의 MDB 도 IoC 와 유사한 Injection 기능이 제공되나 Injection 으로 제공되는 리소스는 EJBContext, DataSource 같이 EJB 3.0 스펙에 지정된 것에 제한적이다. 반면 Spring Framework 기반의 MDB 는 Spring Context 내에서 지정된 어떠한 POJO 객체도 참조가 가능하다.

" J2EE Development without EJB" 의 저자 Rod John 이 개발한 Spring Framework 이 많은 분야에서 활용됨에 따라서 POJO 기반 개발에 대한 관심이 높아지고 있다. POJO 기반 개발은 EJB 와 달리 개발과 테스팅이 용이하며 Spring Framework 같이 IoC 컨테이너와 AOP 서비스를 제공하는 프레임웍을 이용할 경우 EJB 와 같이 트랜잭션 및 보안이 보장되는 컴포넌트 개발이 가능하다는 장점을 가지고 있다. 최근 발표된 EJB 3.0 스펙도 POJO 기반 개발의 장점을 많은 부분 수용하여 개발 및 테스트 편의성을 향상시키려 했다. 이번 비동기식 로그서비스 구현에 적용된 두 기술 모두 POJO 기반 개발을 중심에 두고 있다. 어떤 기술을 사용하여 비동기식 컴포넌트를 개발 할 것인가는 어플리케이션 개발 및 운영환경을 분석하여 최적은 솔루션을 선택하는 것이 중요하다. 만약 J2EE 표준을 따르는 것이 중요하고 기존의 EJB 컴포넌트와 연동해야 할 경우는 EJB 3.0 MDB 를 사용하는 것이 합리적일 것이다.

6. 참고문헌

[1] James Strachan " Message-Driven POJOs" 2005

[2] Michael Juntao Yuan " POJO Application Frameworks: Spring vs. EJB 3.0" 2005

[3] Madhusudhan Konda " Capture the benefits of asynchrous logging" 2004

[4] Rod Johnson " J2EE Development without EJB" 2004

[5] Rod Johnson " Introduction to the Spring Framework" 2005

[6] JBoss.org " EJB 3.0 TrailBlazer" 2005

[7] Sun JavaOne " Enterprise JavaBeans™ 3.0" 2005