JCA(Java Connector Architecture)와 트랜잭션 최적화를 통한 데이터베이스 접근 방법

서범수*, 김중배**
한국전자통신연구원 인터넷컴퓨팅연구부

JCA(Java Connector Architecture) and the method to access databases using transaction optimization

Beom-Su Seo*, Joong-Bae Kim**
Internet Computing Department
Electronics and Telecommunications Research Institute
E-mail : *bsseo@etri.re.kr, **jjkim@etri.re.kr

Abstract

JCA(Java Connector Architecture)[3]는 다양한 형태의 데이터 리소스들을 하나의 J2EE[1] 서버에서 사용하기 위해 SUN 에서 제안한 리소스 연동 방안이다. JCA 는 J2EE 서버와 데이터베이스, ERP 등의 레거시 시스템을 연동하기 위해 필요한 다양한 인터페이스들을 제공한다. 이러한 인터페이스들은 데이터 소스로부터 커넥션을 얻기 위한 규약, 이때 필요한 접근 제어 및 인증 관련 규약, 그리고 커넥션과 연관된 트랜잭션에 대한 처리 방안을 제시하는 규약들로 구분된다. 본 논문에서는 JCA 의 아키텍처에 대한 설명과 함께 데이터베이스 커넥션에 초점을 맞추어, 트랜잭션의 유무와 상이한 데이터베이스 접근에 따른 3 가지 모델에 대해 논의한다.

1. JCA 란?

인터넷의 발전과 함께 거의 모든 시스템들이 웹 환경으로 옮겨가면서 많은 기업들은 보다 안정적이고 유지보수가 용이하며 서비스 컴포넌트의 작성 및 배포가 쉬운 웹 응용서버 기술을 도입하고 있다. MS 사의 .Net 프레임워크와 SUN 사의 J2EE 기술이 대표적인 기술로써 서비스를 구현하여 배포하고 구동하기 위해 필요한 IDE 툴, 배포 도구, 트랜잭션과 보안 및 외부 시스템 연동 기술 등의 거의 모든 미들웨어 기술들을 하나의 기반 프레임워크로 제공한다. JCA 는 SUN 사에서 제안한 J2EE 기술의 구성 기술로써 데이터베이스 등의 외부 시스템을 연동하기 위한 기술이다. J2EE 기술은 크게 JSP, Servlet 과 같은 표현 로직을 처리하는 웹 컨테이너 기술과 비즈니스 로직을 담고 있는 EJB 컨테이너[2] 기술로 나뉜다. SOAP 을 이용한 web service 와는 달리 JCA 는 EJB 컨테이너(EJB 서버)의 커넥션 매니저가 구현해야 하는 모듈과 EJB 컨테이너와 연동할 데이터 리소스들이 제공해야 할 어댑터를 표준화하고 각각의 리소스 벤더들이 이러한 표준 인터페이스의 구현할 것을 강요한다. 그림 1 은 JCA 를 사용할 경우 달라지는 시스템 구성을 나타낸다. JCA 를 적용하지 않는 경우 EJB 서버는 각각의 리소스마다 해당 리소스를 위한 어댑터를 구현하여야만 한다. 이 경우 N 개의 EJB 서버가 M 개의 리소스를 사용하려면, 각각의 EJB 서버마다 리소스를 관리하기 위한 방식 또한 상이하므로 N*M 개의 시스템 구성이 이루어진다. 그러나 JCA 를 사용할 경우 EJB 서버들은 JCA 에서 제안하는 표준화된 커넥션 매니저를 구현하고 각각의 리소스 벤더들은 리소스를 연동하기 위해 JCA 에서 제안하는 어댑터를 구현하여 제공하므로 N+M 개의 시스템 구성으로 줄어들게 된다. 그림 1 은 이러한 JCA 의 효과에 대해 나타낸다.

다양한 형태의 리소스들을 사용하기 위해서는 각각의 리소스들을 접근하는 커넥션 객체가 상이하고 각각의 리소스들에 대한 접근을 안전하게 보장하기 위한 보안 방법 및 트랜잭션 관리 방법 등이 상이하다. JCA 는 이러한 커넥션, 보안, 트랜잭션 관리 방법을 위한 EJB 컨테이너의 역할, 리소스 어댑터로 표현되는 리소스 측의 역할, 클라이언트(EJB)에게 보여지는 커넥션 객체에 대한 인터페이스 등을 정의한다. 그림 2 는 JCA 의 커넥션 요청 과정을 보다 자세히 표현한 것으로 어플리케이션 서버와 리소스 어댑터사이에 존재하는 다양한 메소드 호출을 나타낸다. 어플리케이션 컴포넌트가 커넥션 팩토리(JDBC 의 경우 javax.resource.DataSource)에 리소스에 대한 커넥션을 요청하면 커넥션 팩토리는 어플리케이션 서버에 존재하는 커넥션 매니저에게 커넥션을 요청한다. 커넥션 매니저는 커넥션 팩토리의 요청을 분석

하여 어떤 리소스 어댑터에 커넥션을 요청해야 하는지 판단하고 해당 리소스 어댑터의 ManagedConnectionFacotry 를 통해 ManagedConnection 을 요청한다. ManagedConnectionFactory 는 리소스에 종속적인 방법으로 리소스에 대한 물리적 커넥션을 생성하여 이를 랩핑한 ManagedConnection 을 생성한 후 커넥션 매니저에게 전달한다. 커넥션 매니저는 전달받은 ManagedConnection 을 트랜잭션과 연관시켜 보관하고, 공유 및 비공유 여부에 따라 동일 트랜잭션 내에서의 동일 리소스 어댑터에 대한 요청을 처리한다. ManagedConnection 내에 포함된 물리적 커넥션은 어플리케이션 서버의 트랜잭션이 커밋되거나 롤백되면서 자신을 관리하는 커넥션 매니저에게 커넥션의 커밋이나 롤백을 알리기 위한 콜백 매카니즘을 ConnectionEventListener 를 통해 제공한다. 실제 어플리케이션 컴포넌트에 전달되는 커넥션은 ManagedConnection 이 생성한 논리적 커넥션으로써 물리적 커넥션을 호출하기 전에 필요한 다양한 작업을 수행한다. 커넥션을 얻는 과정에서 해당 리소스를 접근하기 위해 id 나 password 와 같은 일반적인 접근 제어 방법을 사용하거나 Credential 이나 Principal 과 같은 JAAS(Java Authentication and Authorization Service)[5] 객체를 사용하여 접근 제어를 할 수 있다. 이것을 처리하기 위해 어플리케이션 서버는 Security Service Manager 를 제공한다. 어플리케이션에 존재하는 커넥션 매니저는 Security Service, Transaction Service 등을 이용하여 커넥션을 할당하며 저장하며 풀링한다.

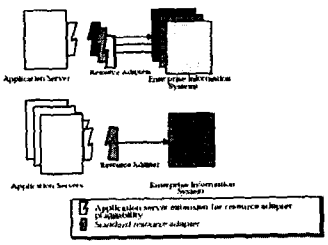

그림 1. JCA 의 효과

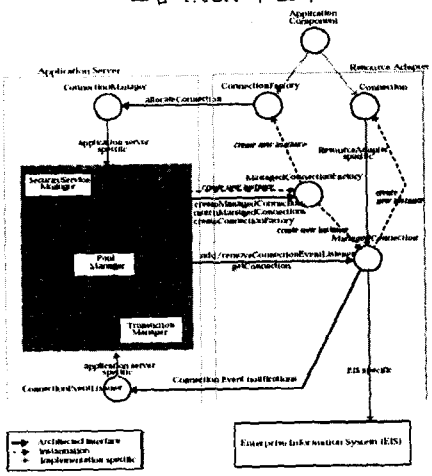

그림 2. Java Connector Architecture

2. JCA 에서 Connection 요청 프로세스

본 절에서는 JCA 를 구현한 EJB 컨테이너에서 데이터베이스에 대한 커넥션 요청을 처리하는 방법에 대해 논의한다.

그림 3 은 트랜잭션을 시작한 후 빈이 최초로 특정 DataSource 에 커넥션을 요청하였을 때 커넥션 매니저가 요청을 처리하는 과정을 나타낸다. 클라이언트가 DataSource 를 통해 커넥션을 요청하면 커넥션 매니저가 해당 요청을 받는다. 커넥션 매니저는 클라이언트의 커넥션 요청을 실제적으로 처리할 리소스 어댑터가 무엇인지를 검사하여 해당 리소스 어댑터의 ManagedConnectionFactory 에게 ManagedConnection 생성을 의뢰한다(3). 리소스 어댑터의 ManagedConnectionFactory 는 XADataSouce 로부터 XAConnection 을 생성하고 생성된 XAConnection 으로부터 XAResource 를 얻는다. 또한, XAConnection 으로부터 실제 물리적 커넥션을 생성한 후 XAResource, 물리적 커넥션 등을 가지고 있는 ManagedConnection 을 생성한다. 각각의 데이터베이스 드라이버마다 고유의 XADataSource 클래스를 가지고 있으므로 ManagedConnectionFactory 는 이 드라이버에서 제공하는 XADataSource 를 사용한다. XA[4,6] 관련 작업은 하나 이상의 데이터베이스를 접근할 경우 2PC[6] 메커니즘을 사용하여 데이터의 일관성을 유지하기 위한 것이다. 커넥션 매니저는 생성된 ManagedConnection 받은 후 ManagedConnection 내부에 존재하는 지역 트랜잭션을 시작시키고 지역 트랜잭션을 포함하는 동기화 객체를 생성하여 전역 트랜잭션에 등록한다(16). 지역 트랜잭션이란 커넥션 매니저와 리소스 어댑터 내부적으로 사용하는 트랜잭션이며 전역 트랜잭션이란 외부 트랜잭션 시스템이 제공하는 트랜잭션을 의미한다. 생성된 동기화 객체는 beforeCompletion()과 afterCompletion()이라는 두가지 메소드를 가지고 있으며 전역 트랜잭션 종료 직전과 직후에 등록된 동기화 객체들의 이 메소드들이 호출됨으로써 사용한 데이터를 초기화 하거나 물리적 커넥션을 커밋하는 것과 같은 작업이 이루어진다. 이것을 통해 개발자는 트랜잭션 완료 과정에 개입하여 다양한 기능을 추가할 수 있다. 이후 커넥션 매니저는 이전에 동일 트랜잭션에서 다른 ManagedConnectionFactory 에 대한 커넥션 요청이 존재하였는지를 검사하는데 이것은 동일 트랜잭션내에서 다중 데이터베이스를 접근할 경우 전역 트랜잭션에 XAResouce 를 등록하여 2PC 프로토콜을 사용하도록 하기 위함이다(17). 이와 같은 과정을 거쳐 ManagedConnectionStore 라는 자료구조에 ManagedConnection 을 저장하고 이것으로부터 클라이언트에게 보여질 논리적 커넥션을 생성하여 클라이언트에게 전달한다. ManagedConnection 은 내부적으로 가지고 있는 물리적 커넥션을 패러매터로 논리적 커넥션을 생성하여 벡터 형태로 보관하며, 실제 클라이언트가 논리적 커넥션의 메소드를 호출할 경우 트랜잭션과의 연관관계를 고려하여 물리적 커넥션에 메소드 호출을 전달함으로써 데이터베이스를 접근한다. EJB 가 다른 EJB 를 동일 트랜잭션 컨텍스트 안에서 호출하여 동일 데이터베이스를 접근한다면 이미 보관 중인 ManagedConnectionStore 로부터 이전에 생성되어 있던 ManagedConnection 객체를 가져와 이로부터 또 다른 논리적 커넥션을 만들어 해당 EJB 에 전달한다. 즉, 여러

클라이언트(EJB)가 한 트랜잭션 안에서 동일 데이터베이스를 접근할 경우, 물리적 커넥션 하나만을 공유하는 논리적 커넥션들을 각각의 클라이언트가 가지게 되며 각 커넥션들에게 이루어진 데이터 변경이나 입력, 삭제 등의 작업은 전역 트랜잭션의 종료에 따라 데이터베이스에 반영된다. 실질적인 커넥션 커밋/롤백은 전역 트랜잭션과 지역 트랜잭션의 상호 작용에 의해 이루어진다.

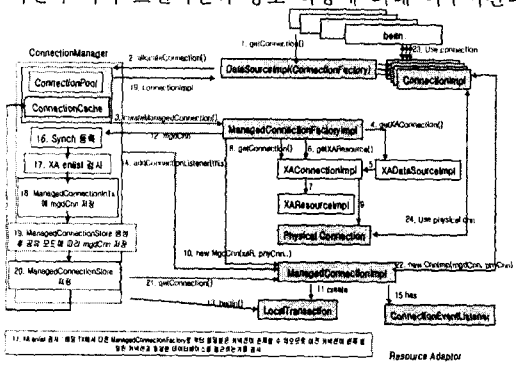

그림 3. 커넥션 요청 처리 프로세스

3. 전역 트랜잭션과 지역 트랜잭션

컨테이너에 배포되어 있는 빈이 트랜잭션 내에서 하나의 데이터베이스만을 사용한다면 커넥션 매니저는 다음 두가지 전략을 사용할 수 있다. 첫째, 빈의 커넥션 요청 시 해당 커넥션에 대한 XAResource 를 생성하여 트랜잭션 매니저에게 등록하는 방법이다. 동일 트랜잭션 내에서 상이한 데이터베이스를 접근하지 않으므로 트랜잭션 매니저는 트랜잭션 종료 시 데이터 리소스에 커밋 준비 요청을 내릴 필요가 없다. 이런 경우 트랜잭션 매니저는 1PC 기법을 사용하여 해당 데이터 리소스에게 커넥션을 커밋할 것을 요청한다. 두개 이상의 데이터베이스를 사용하는 경우 보다 2PC[6] 프로토콜에 의한 오버헤드는 감소하지만 XAResource 를 트랜잭션에 등록해야 하는 오버헤드는 여전히 존재한다. 두번째 방법은 이러한 오버헤드를 줄이기 위한 것으로 트랜잭션 매니저에 XAResource 를 등록하지 않고 커넥션 매니저가 내부적으로 지역 트랜잭션을 사용하여 트랜잭션 종료 시 물리적 커넥션을 종료하는 기법이다. 컨테이너는 배포된 빈이 실행 시간에 여러 개의 데이터 소스를 접근하는가에 대한 사전 정보를 얻을 수 없으므로 그림 3 의 (4, 5, 6, 7, 8, 9) 과정을 통해 XAResource 와 XAConnection, 물리적 커넥션을 생성한다. 이러한 과정은 빈 개발자가 다중 데이터 소스를 접근하여 개발한다는 것을 알고 있더라도 하나의 데이터 소스를 사용할 때와 차이가 없이 빈을 개발할 수 있도록 하기 위함이다. 로컬 트랜잭션의 경우 전역 트랜잭션이 없이 커넥션을 할당하였을 때 해당 커넥션의 초기화 및 재사용을 위한 다양한 기능을 제공한다.

전역 및 로컬 트랜잭션과 연관하여 다음 3 가지의 실행 모델이 가능하다.

3-1. 트랜잭션이 없이 커넥션이 할당되었을 경우

그림 4 는 트랜잭션이 없이 커넥션을 할당할 경우 로컬 트랜잭션의 처리 과정을 나타낸다. 시간에 따라 다음과 같은 처리 과정을 거친다.

T_0 : 빈이 DataSource 로부터 데이터베이스 커넥션을 요청한다.

T_1 : 커넥션 매니저는 ManagedConnection 을 그림 3 에 따라 할당받는다.

T_2 : ManagedConnection 이 지역 트랜잭션을 생성한다.

T_3 : 커넥션 매니저는 MangedConnection 이 가지고 있는 지역 트랜잭션을 시작한다. (그림 3 의 (13) 과정)

T_4 : 빈이 논리적 커넥션에 commit()을 호출한다.

T_5 : 연관된 트랜잭션이 없으므로 논리적 커넥션은 가지고 있는 물리적 커넥션의 commit()을 호출한다.

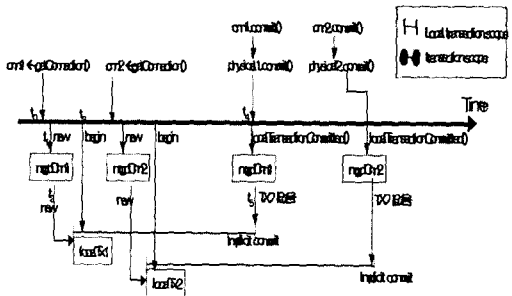

그림 4. 트랜잭션이 없는 경우의 커넥션 처리

EJB 가 논리적 커넥션에 commit 을 호출하면 지역 트랜잭션에게 이 사실이 통보되고 지역 트랜잭션은 커넥션 매니저에게 트랜잭션이 없이 할당된 커넥션을 보관하는 자료구조로부터 ManagedConnection 을 제거하여 풀에 반환하도록 요청한다.

3.2 전역 트랜잭션이 존재하고 하나의 데이터 소스만을 접근하는 경우

그림 5 는 전역 트랜잭션이 존재하고 해당 트랜잭션 내에서 하나의 데이터 소스만을 접근할 경우 시간에 따른 처리 과정을 나타낸다. 앞서 논의하였듯이 하나의 데이터 소스만을 접근하는 경우 전역 트랜잭션에 XAResource 를 등록하지 않고 지역 트랜잭션을 이용하여 물리적 커넥션을 커밋/롤백 시킴으로써 오버헤드를 감소시킬 수 있다. 시간에 따라 다음과 같은 처리 과정을 거친다.

T_0 : 빈이 DataSource 로부터 데이터베이스 커넥션을 요청한다.

T_1 : 커넥션 매니저는 ManagedConnection 을 그림 3 에 따라 할당받는다.

T_2 : ManagedConnection 이 지역 트랜잭션을 생성한다.

T_3 : 커넥션 매니저는 ManagedConection 이 가지고 있는 지역 트랜잭션을 패러매터로 하여 전역 트랜잭션에 등록할 동기화 객체를 생성한다.

T_4 : 전역 트랜잭션에 동기화 객체를 등록한다.

T_5 : 커넥션 매니저가 지역 트랜잭션을 시작한다.

T_6 : EJB 가 논리적 커넥션에 commit()을 호출한다. 이때 논리적 커넥션은 전역 트랜잭션이 시작되었기 때문에 자신이 가진 물리적 커넥션의 commit() 메소드를 호출하지 않는다.

T_7 : 전역 트랜잭션이 commit()을 호출한다.

T_8 : 등록되어 있던 동기화 객체들의 afterCompletion()이 호출된다.

T_9 : 동기화 객체들은 자신이 가지고 있는 지역 트랜잭션의 commit()을 호출한다.

T_{10} : 지역 트랜잭션은 물리적 커넥션의 commit() 메소드를 호출한다.

전역 트랜잭션에 등록되어 있던 동기화 객체가 지역 트랜잭션의 commit()을 호출하고 지역 트랜잭션이 물리적 커넥션에 대한 commit()을 수행하며 커넥션 매니저가 전역 트랜잭션과 연관된 ManagedConnection 을 적절한 자료구조에서 제거해 커넥션 풀로 이동시킨다.

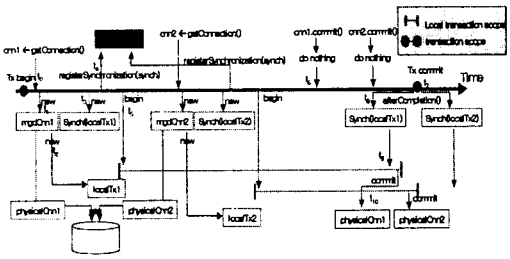

그림 5. 하나의 데이터 소스 접근에 대한 전역 트랜잭션과 지역 트랜잭션의 처리

3-3. 전역 트랜잭션이 존재하고 상이한 데이터 소스를 접근하는 경우

그림 6 은 동일 전역 트랜잭션 내에서 상이한 데이터 소스를 접근하는 경우, 시간에 따라 어떻게 처리되는지를 설명한다. EJB 스펙[1]에서는 분산 데이터 소스에 대한 처리를 JTS[4]를 이용하여 할 것을 권장한다. 분산 트랜잭션의 경우 2PC 프로토콜에 따라 커넥션의 커밋/롤백을 처리한다. 시간에 따라 다음과 같은 처리 과정을 거친다.

T_0 : 빈 A 가 DataSource 로부터 데이터베이스 커넥션을 요청하여 그림 3 의 과정을 거쳐 ManagedConnection 을 할당받고 동기화 객체를 등록한 후 로컬 트랜잭션을 시작한다.

T_1 : 빈 B 가 다른 DB 를 접근하는 DataSource 로부터 데이터베이스 커넥션을 요청하여 ManagedConnection 을 할당받고 동기화 객체를 등록한 후 로컬 트랜잭션을 시작한다.

T_2 : 커넥션 매니저는 빈들이 서로 다른 데이터베이스를 접근하므로 이들에게 할당된 ManagedConnection 이 가진 XAResource 를 트랜잭션에 등록한다.

T_3 : 빈 A 가 논리적 커넥션에 commit()을 호출하지만 어떤 작업도 수행되지 않는다.

T_4 : 전역 트랜잭션이 commit()을 호출한다.

T_5 : 전역 트랜잭션은 자신에게 등록되어 있는 XAResource 에 대해 commit()을 호출한다.

T_6, T_7 : XAResource 는 자신이 가지고 있는 물리적 커넥션에 대해 commit()을 호출한다. 물론 이 과정에서 트랜잭션 매니저와 데이터 소스들간에 2PC 에 따른 메시지가 교환된다.

T_8 : 트랜잭션 매니저는 등록되어 있던 동기화 객체들의 afterCompletion()이 호출된다.

T_9 : 동기화 객체들은 자신이 가지고 있는 지역 트랜잭션의 commit()을 호출한다.

T_{10} : 지역 트랜잭션은 이미 XAResource 들이 물리적 커넥션에 대해 commit() 메소드를 호출하였으므로 물리적 커넥션의 어떤 메소드도 호출하지 않는다. 단지 커넥션 매니저에게 해당 ManagedConnection 들이 재사용되도록 요청한다.

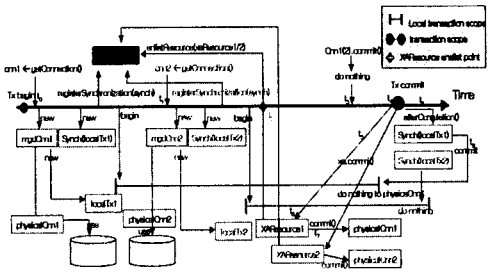

그림 6. 다중 데이터 소스 접근에 대한 전역 트랜잭션과 지역 트랜잭션 처리

4. 결론

본 논문에서는 JCA 의 구조 및 이를 구현한 리소스 어댑터를 소개하여 그 동작 과정을 설명하였다. 또한 트랜잭션의 유무와 데이터베이스 접근 방식에 따른 3 가지 가능한 모델을 제시하여 각각의 모델에서 전역 트랜잭션과 지역 트랜잭션과의 상호 관계에 대해 논의하였다. 하나의 데이터베이스를 접근할 경우 2PC 프로토콜을 위해 필요한 XAResource 등록 오버헤드와 2PC 프로세스 오버헤드를 감소시킴으로써 데이터베이스의 접근 속도를 향상시킬 수 있다. 본 논문에서는 제한된 지면 관계로 EJB 에서 데이터베이스를 사용하는 유형[7]] 과 커넥션을 공유하기 위한 알고리즘 등은 다른 논문을 통해 논의할 예정이다. 본 논문에서는 데이터베이스에 초점을 두어 논의하였지만 이를 JMS 나 ERP 와 같은 다양한 시스템으로 확장하여 연구할 필요가 있다.

5. 참고 문헌

[1] Java™ 2 Platform Enterprise Edition Specification, v1.4 Public Draft, SUN Microsystems, July 15, 2002

[2] Linda G Demichiel 외 2 인, "Enterprise Javabeans Specification, Version 2.1," Proposed Final Draft, Sun Microsystems, 2002

[3] Rahul Sharma, "J2EE Connector Architecture, Final Release version 1.0," Sun Microsystems, 2001

[4] Susan Cheung 외 1 인, "Java Transaction API (JTA) Version 1.0.1," Sun Microsystems, 1999

[5] "Java Authentication and Authorization Service(JAAS) Reference Guide," http://java.sun.com

[6] Andresas Vogel, Madhavan Rangararo, Programming with Enterprise JavaBeans, JTS and OTS, 1991, Wiley Computer Publishing

[7] 서범수, 김성훈, 장철수, 김중배, "E504 EJB 컨테이너 시스템의 데이터베이스 커넥션 관리 방법," 한국정보과학회 추계 학술발표논문집(3), pp.148-150, 2002.