EJB 환경에서 엔티티 빈의 성능향상을 위한 적용규칙

°최성만*, 김송주*, 이정열**, 유철중*, 장옥배*
*전북대학교 컴퓨터과학과
**정인대학 사무정보계열
sm3099@cs.chonbuk.ac.kr
songju@hannuri.net
ljy8383@hanmail.net
{cjyoo, okjang}@moak.chonbuk.ac.kr

Applicable Rules for Performance Improvement of Entity Bean in EJB Environment

°Seong-Man Choi*, Song-Ju Kim*, Jeong-Yeal Lee**,
Cheol-Jung Yoo*, Ok-Bae Chang*
*Dept. of Computer Science, Chonbuk National University
**Dept. of Office Automation, Chongin College

요 약

엔터프라이즈 환경에서 애플리케이션 개발에 적합한 J2EE(Java 2 Platform Enterprise Edition)의 서버측 컴포넌트 기술인 EJB(Enterprise JavaBeans)는 엔티티 빈(Entity Bean)과 세션 빈(Session Bean)으로 구분된다. 본 논문은 이러한 EJB를 구성하는 요소 중에서 데이터베이스의 효율적 액세스를 위한 DAO(Data Access Object)에 적용된 엔티티 빈의 성능을 향상시키기 위한 적용규칙을 제안한다.

1. 서론

엔터프라이즈 환경에서 애플리케이션 개발에 적합한 J2EE의 서버측 컴포넌트 기술인 EJB(Enterprise JavaBeans)는 컴포넌트 빈의 명세에 따라 엔티티 빈(Entity Bean)과 세션 빈(Session Bean)으로 구분된다. 엔티티 빈은 자료의 영속성을 고려하여 영구적으로 관리하는 경우이고, 세션 빈은 자료의 영속성이 일시적인 성격을 가지고 있는 경우이다[1]. 이렇게 EJB를 구성하는 요소 중에서 데이터베이스의 효율적 액세스를 위한 DAO(Data Access Object)에 적용된 엔티티 빈의 성능을 향상시키기 위한 적용규칙에 대해서 알아본다. 먼저 2장에서는 객체지향 분산 엔터프라이즈 애플리케이션의 개발 및 분산 배치를 위한 컴포넌트 아키텍처인 EJB의 개요에 대해서 알아보고, 3장에서는 실제 엔티티 빈으로 구현한 DAO에 대해서 알아본다. 또한 4장에서는 DAO의 구조 및 코드를 보면서 성능향상을 위해서 최적화하기 위한 적용규칙에 대해서 설명하고, 마지막 5장에서는 결론 및 향후 연구과제를 제시한다.

2. EJB의 개요

2.1 EJB의 정의 및 유형

EJB란 객체지향 분산 엔터프라이즈 애플리케이션의 개발 및 분산 배치를 위한 컴포넌트 아키텍처로서 EJB를 이용해서 만들어진 애플리케이션은 확장성이 있고, 트랜잭션을 보장하며, 다수의 사용자 환경에서도 안전하다[2]. 또한 한번 작성된 애플리케이션은 EJB 스펙을 지원하는 어떤 서버 플랫폼에서도 배치되고 운영될 수 있다. 이러한 EJB는 엔티티 빈과 세션 빈으로 분류된다. 먼저, 엔티티 빈은 클래스들이 가지고 있는 자료의 영속성을 고려하여 영구적으로 관리하는 경우를 의미하며, 빈 관리 퍼시스턴스(BMP)와 컨테이너 관리 퍼시스턴스(CMP)로 나누어진다. 세션 빈은 클래스들이 가지고 있는 자료의 영속성이 일시적인 성격을 가지고 있는데, 이는 서로 다른 빈들간의 상호작용을 표현해준다. 세션 빈은 무상태 세션 빈과 상태유지 세션 빈으로 나뉘어진다[1, 2, 3].

2.2 EJB 성능 측정

EJB의 성능 측정 방법으로는 첫째, 효율성(throughput) 측면으로 두 번째 빈의 평균 처리응답 시간에 대한 역이다. 즉, 접속한 클라이언트 수에 따른 컴포넌트가 초당 처리하는 시간이다. 둘째, 처리응답시간(response time) 측면으로 빈 하나를 생성하고 그 빈을 수행하는데 걸리는 총 시간을 의미한다. 또한 클라이언트 수 증감에 따른 컴포넌트를 처리하는데 소요되는 시간을 측정한다. 셋째, 메소드에 따른 초당 처리시간 측면으로 해당 메소드들을 수행하는데 소요되는 시간을 측정하는 것으로 메소드들의 처리시간을 측정한다[4].

3. 엔티티 빈의 적용 사례

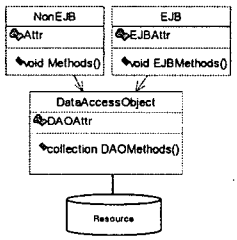
본 장에서는 엔티티 빈의 적용 사례로서 데이터베이스를 액세스하기 위한 DAO(Data Access Object)의 정의 및 구조, 적용 코드에 대해서 살펴본다[5].

3.1 DAO의 정의

실제적으로 엔티티 빈의 구현 성능향상을 위해서 여러 가지 방법을 이용하는데 그 중에서 하나가 DAO 패턴을 들 수 있다. DAO란 객체로서 XML 데이터를 표현하고 데이터베이스 벤더의 독립적인 목적을 위해 기존 시스템에서는 DAO를 이용한다. DAO는 데이터베이스 접근을 캡슐화하는데 이용되며, 대규모 배치에 적합하고, 독립적인 리소스 벤더와 독립적인 리소스 구현 및 빈 관리 퍼시스턴스에서 컨테이너 관리 퍼시스턴스까지의 이동을 용이하게 해주는 이점을 가진다.

3.2 DAO의 구조

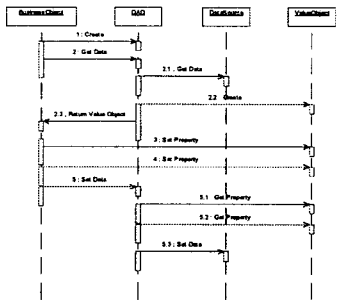
[그림 1]은 이러한 DAO 구조를 보여주고 있다.



[그림 1] DAO 구조

[그림 1]을 통해 알 수 있듯이 DAO의 참여자(participants)는 NonEJB, EJB, DataAccessObject, Resource가 있다. NonEJB 클래스는 EJB-required()를 제외한 비즈니스 메소드를 인스턴스화시켜 DAO를 이용하며, EJB 클래스는 EJB-required()를 담당하며 비즈니스 메소드가 호출될 수 있는 환경을 제공한다. DataAccessObject 클래스는 리소스에 대한 오퍼레이션 추상화와 특정한 타입의 데이터 처리와 접근하기 위한 표준 API를 제공한다. Resource는 임의적인 API 방법에 의한 검색과 영속적인 리소스 데이터를 Concrete DataObject에 제공한다.

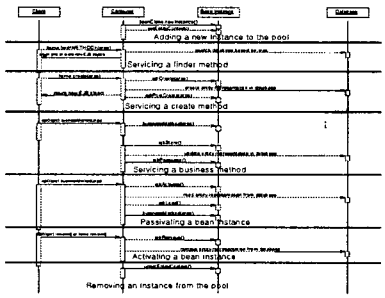
[그림 2]는 DAO에서의 많은 참여자들 사이에서의 상호작용을 시퀀스 다이어그램으로 보여주고 있다[6].



[그림 2] DAO 시퀀스 다이어그램

3.3 DAO의 적용 코드

실제로 DAO가 적용된 코드를 보면서 자세하게 살펴본다. [그림 3]은 [그림 4]의 DAO를 적용한 코드를 토대로 하여 EJB 컴포넌트 시스템에서의 적용 과정을 시퀀스 다이어그램으로 보여주고 있다.



[그림 3] CatalogDAO의 시퀀스 다이어그램

[그림 4]는 DAO를 적용한 코드의 예로서 Customer 정보를 표현하는 영속적인 객체를 위한 DAO 코드 예제로서 CloudscapeCustomerDAO가 findCustomer()를 호출하여 CustomerDAO를 구현한다. CloudscapeAccountDAO, CloudscapeOrderDAO, OracleCustomerDAO, OracleAccountDAO 등과 같은 다른 DAO들은 비슷하게 구현되어진다. 앞에서 언급하였던 DAO의 구조 및 적용 코드를 보면서 성능향상을 위해서 최적화시킬 때의 규칙이 있음을 알 수 있었다.

```
public class CloudscapeCustomerDAO implements
CustomerDAO {
  public CloudscapeCustomerDAO() {
  .....
  }
  public int insertCustomer() {
  .....
  }
  public boolean deleteCustomer() {
  .....
  }
  public Customer findCustomer() {
  .....
  }
  public boolean updateCustomer() {
  .....
  }
  public Collection selectCustomersVO() {
  .....
  }
  / 중략 /

  .....
  DAOFactory cloudscapeFactory = DAOFactory.
    getDAOFactory(DAOFactory.DAOCLOUDSCAPE);
  CustomerDAO custDAO = cloudscapeFactory.
    getCustomerDAO();
  int newCustNo = custDAO.insertCustomer();
  Customer cust = custDAO.findCustomer();
  cust.setAddress();
  cust.setEmail();
  custDAO.updateCustomer(cust);
  custDAO.deleteCustomer();

  Customer criteria = new Customer();
  criteria.setCity("New York");
  Collection customersList =
    custDAO.selectCustomersVO(criteria);

  / 중략 /
```

[그림 4] DAO를 적용한 코드의 예

4. 엔티티 빈을 최적화하기 위한 적용규칙

엔티티 빈은 애플리케이션과 설계에서 영속성 비즈니스 객체를 표현하기 위해 명확한 모델을 제공한다. 또한 엔티티 빈은 객체 모델에서 비즈니스 객체에 대한 같은 타입의 모델링과 판단을 허용할 뿐만 아니라 빈과 컨테이너 서비스 이면에서 모든 복잡성을 은닉하는 동안에 영속성 메커니즘을 캡슐화한다. 이러한 엔티티 빈은 Java 객체 레퍼런스로서 어떤 호출된 코드로부터 영속성 메커니즘과 영속성 폼의 은닉에 의해 배치될 때 결정하게 될 융통성과 데이터의 저장을 컨테이너에 의해 독립적인 최적화를 제공한다. 엔티티 빈의 이용으로 인해 대규모의 객체 지향 방법론의 이용을 가능하게 하였으며, EJB 기반 프로젝트 개발에서 다양하게 적용된 최적화를 위한 규칙은 다음과 같다.

(적용규칙 1) 프로젝트를 개발할 때 컨테이너 관리 영속성을 이용해야 한다.

컨테이너 관리 퍼시스턴스(CMP : Container-Managed Persistence)는 훨씬 적은 코딩라인으로 인해 작업을 쉽게 할 수 있을 뿐만 아니라 컨테이너에서 생성한 데이터베이스 액세스 코드와 컨테이너 내부에서 많은 최적화를 가능하게 한다. 컨테이너는 버퍼에서 변경을 위한 모니터를 빈의 메모리 버퍼에서 제공하기 때문에 트랜잭션이 완료(commit)되기 전에 데이터베이스 버퍼에 저장되는 것을 예방하여 불필요한 데이터베이스의 호출을 피한다. CMP는 하나의 데이터베이스 호출에서 레코드 데이터와 키 모두를 검색할 수 있는 판독을 가질 때 단일 데이터베이스로 두 개의 데이터베이스 접근을 최적화한다.

(적용규칙 2) 빈 관리 퍼시스턴스(BMP : Bean-Managed Persistence)와 컨테이너 관리 퍼시스턴스(CMP)를 동시에 지원하는 코드를 작성해야 한다.

많은 경우 EJB 개발자는 컨테이너가 CMP의 지원을 개발동안에 이용되는지 또는 EJB 배치방법에 대해서 관심을 갖지 않는다. 이것은 잠재적으로 CMP 메커니즘을 더욱더 최적화하기 위한 BMP 개발을 허용하는 빈을 구현하는 방법을 개발하기 위해 영속성 메커니즘을 완전히 비즈니스 로직으로부터 분리해야 한다. 만약 CMP를 선택하면 단독으로 배치되는 CMP 클래스에서 비즈니스 로직을 구현해야 한다. 그런 후에 CMP 클래스로부터 상속받은 BMP로 영속성 코드를 구현하면 BMP 클래스에서 데이터베이스 액세스 코드의 추가와 CMP 슈퍼 클래스의 모든 비즈니스 로직을 유지하게 된다.

(적용규칙 3) ejbStore에서 데이터베이스 접근을 최소화해야 한다.

CMP를 이용할 때, 빈은 컨테이너에게 모든 최적화를 유지하게 하고 ejbStore에 의해 절대적으로 관여하지 않는다. 컨테이너에 의해 제공되는 CMP 최적화는 빠른 컨테이너와 느린 컨테이너를 구별하게 하는 요소로서, BMP를 이용하여 배치할 때마다 버퍼를 위한 더티(dirty) 플래그를 유지하는데 매우 유용하다. 이러한 기술은 좀처럼 갱신되지 않는 쿼리에서 매우 유용하다. EJB 1.1을 이용하는 BMP는 최적화를 위해 어떠한 대안도 제공하지 않으므로 빈 작성자가 수작업으로 여전히 더티(dirty) 플래그를 설정해야 한다.

(적용규칙 4) lookup과 find 호출로부터 얻어진 레퍼런스를 항상 저장해야 한다.

저장된 레퍼런스는 엔티티 빈과 세션 빈 모두에서 유용하다. JNDI는 DataSource, 빈 레퍼런스, 심지어 환경 설정에 대한 EJB 리소스를 찾아준다. 이때 과

다한 lookup을 회피하기 위해서는 인스턴스 변수로서 레퍼런스를 정의하고, setEntityContext()를 이용하여 검색해야 한다. setEntityContext()는 단지 빈 인스턴스를 위해 한번 호출된다.

(적용규칙 5) 항상 SQL 문을 준비시켜야 한다.

이러한 최적화는 관계형 데이터베이스를 접근하기 위해 SQL을 이용하는 모든 코드에 대해서 유용하다. 관계형 데이터베이스를 이용하는 대부분의 EJB 구현 규칙은 빈 작성자가 데이터베이스 액세스 코드를 작성하는 동안 EJB 개발에서 매우 유용하다. 관계형 데이터베이스에서 최적화를 이용하기 위해서는 새로운 문장은 본래의 문장과 정확하게 조화되어야 한다. 규격화된 문장(Non-prepared statements)을 위해 데이터와 문장을 같은 스트링으로 전달한다. 비록 문장이 다음 호출로 인해 같이 탐색할 수 있지만 데이터는 일치하지 않는다. 그러므로 최적화의 기능을 억제시킨다. 비규격화된 문장(Prepared statements)을 위하여 데이터베이스에서 데이터 없이 단지 문장을 전달하며 저장된 결과를 얻는다. 이러한 테크놀러지는 컴파일된 문장의 총량을 최소화하여 매우 높은 문장 저장률을 촉진시킨다.

(적용규칙 6) 모든 문장을 정확히 닫아야 한다.

BMP 구현에서 데이터베이스 액세스 코드를 취급할 때 데이터베이스 액세스를 호출한 후 문장을 열어둔 상태로 두어서는 결코 안된다. 왜냐하면 각각의 열린 문장은 데이터베이스에서 열린 커서와 일치하기 때문에 열린 문장을 남겨두면 지나친 열린 커서를 가지는 데이터베이스를 야기하게 된다.

(적용규칙 7) 교착상태(Deadlock)를 회피해야 한다.

애플리케이션 코드는 ejbStore를 호출할 때, 직접적으로 제어하지 않는다. 컨테이너는 일반적으로 트랜잭션의 마지막으로 호출될 때 결정된다. 만약 다중 엔티티 빈 또는 다중 엔티티가 트랜잭션에 포함된다면, 엔티티를 표현하는 데이터베이스 레코드를 순서적으로 액세스 및 록킹(locking)에 대해 제어하지 않는다는 의미이다. 컨테이너 컨트롤 액세스와 EJB에서의 록킹은 교착상태를 위한 계정을 컨테이너에게 제공하므로 개발자 및 배치자의 할 일이 줄어들게 된다.

5. 결론 및 향후 연구과제

본 논문에서는 엔티티 빈의 성능을 향상시키기 위한 조건으로서 일곱 가지 규칙에 대해서 알아보았다. 이러한 규칙들은 집약적인 엔티티 개발과 배치에서 이용할 때, 근본적인 시스템과 컨테이너 로드를 최소화할 동안 다른 퍼시스턴스 저장소 타입을 제공하는 빈의 융통성과 실행속도의 증가를 두드러지게 향상시킬 것이다. 또한 이것은 제공된 기반구조의 이용을 효과적으로 작성해주는 빈을 제공하여 가장 적한한 배치 기반구조를 선택하기 위한 조건을 배치자에게 제공한다. 즉, 한번 작성되면 언제 어디서나 효율적으로 실행할 수 있다는 의미이다. 향후 연구과제로는 본 논문에서 제안한 엔티티 빈의 성능 향상을 위한 적용규칙에 대한 객관성을 증명하기 위해 실제 규칙을 적용한 경우와 적용하지 않은 경우를 테스트를 통해 정량적인 차이점을 보이는 것과 세션 빈에 대한 최적화 규칙에 대한 연구가 필요하다.

참고문헌

[1] Ed Roman, "Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition", John Wiley & Sons, Inc., 1999.

[2] Richard Monson-Haefel, "Enterprise JavaBeans", O'Reilly & Associates, Inc., 1999.

[3] Gilda Pour, "Enterpirse JavaBeans, JavaBeans & XML Expanding the Possibilities for Web-Based Enterprise Application Development", in Proceedings of Technology of Object-Oriented Languages and Systems, pp. 282-291, 1999.

[4] 오창남, 이긍해, "EJB 컴포넌트의 성능 측정 방법", 한국정보과학회 가을 학술발표논문집, 제27권 제2호, pp. 492-494, 2000.

[5] 최성만, 김정옥, 이정열, 유철중, 장옥배, "효율적인 EJB 컴포넌트화를 위한 Integrated DAO 패턴", 한국정보과학회 춘계학술발표논문집, 제28권 제1호, pp. 661-663, 2001.

[6] Deepak Alur, John Crupi, Dan Malks, "Core J2EE Patterns : Best Practices and Design Strategies", Prentice Hall PTR, 2001.