<JSTORM>

Dual Persistent Entity Bean

중앙대학교 컴퓨터공학과
자바 동호회
JSTORM
http://www.jstorm.pe.kr

Document Information

Document title:	Dual Persistent Entity Bean
Document file name:	Dual_Persistent_Entity_ pattern_jstorm_1.0_final
Revision number:	<1.0>
Issued by:	<김종윤, 이용원>
Issue Date:	<2002/2/10>
Status:	final

Content Information

Audience	EJB 중급
Abstract	EJB 개발자는 CMP와 BMP를 모두 지원하는 엔티티 빈 컴포넌트를 작성해야 할 때가 있다. 어떻게 하면 엔티티 빈을 디플로이먼트 할 때에 CMP와 BMP를 동시에 지원하도록 설계할 수 있을까?
Reference	http://www.theserverside.com/resources/patterns_review.jsp
Benchmark information	

Table of Contents

Dual Persistent Entity 4

Dual Persistent Entity

EJB 개발자는 CMP와 BMP를 모두 지원하는 엔티티 빈 컴포넌트를 작성해야 할 때가 있다.

어떻게 하면 엔티티 빈을 디플로이먼트 할 때에 CMP와 BMP를 동시에 지원하도록 설계할 수 있을까?

엔티티 빈 컴포넌트가 디플로이먼트 될 환경은 프로젝트에 따라 광범위하게 변화한다. 최상의 경우, 개발팀은 BMP로는 불가능한 현저한 성능의 향상을 얻을 수 있는 뛰어난 CMP가 구현된 어플리케이션 서버를 사용할 수 있을 것이다. 하지만 종종 개발팀은 CMP에 대한 지원이 부족하거나 서버가 사용하는 데이터베이스에 대한 지원이 빈약한 어플리케이션 서버를 사용해야 하는 경우도 있다. 이 경우가 바로 BMP가 필요한 상황이다. 이러한 상황은 엔티티 빈 개발자를 곤란한 상황에 처하게 한다. 어떻게 하면 엔티티 빈 개발자는 두 상황(역자 주 : CMP의 지원이 뛰어나서 BMP의 사용이 필요없는 어플리케이션 서버를 사용하는 경우와 CMP의 지원이 빈약하여 BMP를 써야하는 어플리케이션 서버를 사용하는 경우)을 모두 만족하는 컴포넌트를 제공할 수 있을 것인가?

이것을 해결하는 하나의 방법은 동일한 엔티티 빈 컴포넌트에 대해 CMP와 BMP를 위한 두개의 분리된 버전을 제공(ship)하는 것이다. 하나는 CMP를 지원하기 위해 패키지된 것이고, 다른 하나는 BMP를 지원하기 위한 패키지이다. 불행하게도, 이런 접근법은 컴포넌트 개발자로 하여금 두 개의 분리된 기반 코드와 컴포넌트를 유지하도록 요구하고, 테스팅과 디버깅 그리고 유지관리를 더욱 어렵게 한다.

진짜로 이식성이 뛰어난 EJB 컴포넌트라면 환경과 구성에 있어서 매우 광범위한 다양성을 지니는 어떠한 J2EE 호환 서버에서도 디플로이먼트가 가능해야 한다. 이식성이란 것은 컴포넌트가 어떠한 재프로그래밍이나 컴파일을 하지않고 커스터마이징 될 수 있어야 함을 의미한다. 변경될 수 있는 것은 오직 디플로이먼트 디스크립터 뿐이다.

그러므로,

비즈니스 로직을 CMP 규약에 만족하는 수퍼클래스와 BMP의 영속적인 로직을 담는 서브클래스로 구분함으로써 CMP와 BMP를 동시에 지원할 수 있는 엔티티 빈을 작성해야 한다. 디플로이먼트 할때 두 개중에서 어떤 것이 선택되느냐는 디플로이먼트 디스크립터의 설정에 의해 결정된다.

엔티티 빈은 엔티티 빈 로직을 두 개의 클래스로 분할함으로써 CMP와 BMP 양쪽 모두를 지원하도록 만들어 질 수 있다. 즉, 엔티티 빈 로직은 CMP 규약에 만족하는 수퍼클래스와 ejbStore(), ejbLoad() 그리고 다른 메소드들을 구현한 수퍼클래스를 상속하는 서브클래스로 나뉘게 된다. 이 새로운 컴포넌트는 표준 ejb-jar.xml 파일에 약간의 수정을 가하여 디플로이먼트 때에 자신의 영속성 모드를 선택할 수 있다.

Account 엔티티 빈의 예를 살펴보자. Account 엔티티 빈은 account id와 balance의 두 개의 속성을 가지고 있다. 또한, 3개의 비즈니스 메소드(deposit, withdraw, balance)와 하나의 특별한 find 메소드(findByBalance(int))를 포함한다. “Dual Persistent Entity Bean” 으로 적용한다면, Account 엔티티 빈은 그림 1. 과 같이 나타나게 된다.

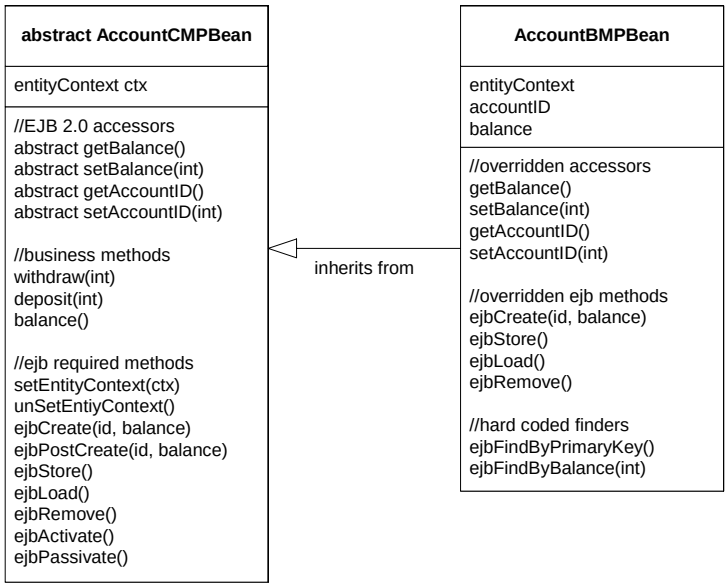

그림 1. Dual Persistent Entity Bean (이중 영속성 엔티티 빈)

CMP 수퍼클래스는 비즈니스 메소드와 추상 get/set 메소드들(abstract attribute accessor는 EJB 2.X CMP의 요구사항임), 그리고 set EntityContext()/unSetEntityContext()와 ejbCreate()와 같은 EJB 기본 메소드의 간단한 구현을 포함한다. ejbLoad(), ejbStore(), ejbRemove()의 구현사항은 비어있다는 것에 주의해야 한다. find 메소드는 CMP 클래스에서 구현할 필요가 없다. 왜냐하면, 디플로이먼트 디스크립터에서 따로 선언되어 있기 때문이다.

BMP 서브클래스는 accountID와 balance, 그리고 이들에 대한 get/set accessor의 구체적인 구현을 제공한다. 다른것 보다 이 클래스가 요구하는 오직 하나의 특별한 로직은 ejbCreate(), ejbLoad(), ejbStore()그리고 ejbRemove()의 영속성 관련 메소드의 실질적인 구현이다. CMP 수퍼클래스가 ejb-jar.xml 파일에 있는 쿼리(query) 정의에 의존하는데 반해, BMP 서브클래스에서는 find 메소드가 또한 반드시 구현되어야 한다. BMP에서 set EntityContext()/unSetEntityContext()나 ejbActivate()/Passivate()와 같은 비즈니스 로직은 재구현될 필요가 없다는 점에 주의해야 한다. 왜냐하면 이것들은 수퍼클래스에서 모두 상속받기 때문이다.

디플로이먼트 할때에, 우리는 간단히 ejb-jar.xml 파일을 변경함으로써 CMP 또는 BMP 클래스를 선택할 수 있다. 특별히, <ejb-class> 테그는 CMP 수퍼클래스나 BMP 서브클래스중 하나를 참조하는데 사용되게 될 것이다. 좀더 명백하게 하자면, <persistence-type> 태그가 컨테이너 관리(CMP)인가 빈 관리(BMP)인가에 대한 선택을 하게 된다. 만일 CMP를 선택하는 것이라면, ejb-jar.xml은 스키마나 속성들 그리고 find 메소드 등을 추가하기 위한 CMP에 특징적인 태그들로 구성되어야 할 것이다. 스키마는 또한 디플로이먼트가 될 어플리케이션 서버에 의해 제공되는 독점적인 메커니즘을 사용하여 기반 데이터 저장장치와 매핑된다. 반면에, 만일 BMP 기반으로 디플로이먼트를 한다면, ejb-jar.xml은 SQL DataSource를 <resource-ref> 태그들을 통해서 추가해야 한다.

좀더 이식성이 강한 엔티티 빈을 작성하는 것에 덧붙여, 이 패턴의 또 다른 사용이 있으니 그것은 바로 BMP 엔티티 빈들을 CMP 엔티티 빈으로 변경하는 것이다. EJB 1.X 스펙에 의해 제공되는 CMP 지원은 종종 중요한 어플리케이션의 요구에 충분하지 못할때가 있다. 더욱이, 시장에서 사용되는 많은 CMP 구현들은 좋지 않은 성능로 인해 나빠지고 있다. 이 모든 레가시 EJB 어플리케이션들은 EJB 1.X BMP에서 새롭고 좀더 정교해진 CMP로 전환되는 것을 통해 좋게 될 수 있다. 불행하게도, BMP에서 CMP로의 변경 프로세스는 매우 까다로운 일이 될 수 있다. 하나의 해결책은 CMP를 사용하여 완전하게 재작성하는 것이 될 수 있다. 이 선택은 많은 양의 더욱 증가된 작업을 필요로 하고, 필수적으로 하나의 엔티티 빈에서 다른 엔티티 빈으로 비즈니스

로직을 오려 붙이는 작업이 필요하다. 이것은 BMP 엔티티 빈을 CMP로 바꾸는데 있어서 매우 비효율적인 방법이다. “Dual Persistent Entity Bean” 패턴을 사용하여, 수퍼클래스를 만든후 코드를 수퍼클래스로 이동시키고 단지 속성들과 속성 접근자, 그리고 영속성 관련 메소드들만을 서브클래스에 남겨둠으로써 기존의 BMP 엔티티 빈을 CMP로 리펙토링 할 수 있다. 새로운 수퍼클래스를 디플로이먼트 및 테스트하고, 만일 필요하면 서브클래스는 나중에 제거할 수 있다.